

Univerzitet "Džemal Bijedić" u Mostaru

Fakultet informacijskih tehnologija



ZAVRŠNI RAD NAKON PRVOG CIKLUSA STUDIJA

Lasersko navođenje robota korištenjem computer vision tehnologija

elektronički bazirani projekat za lasersko upravljanje i navođenje vozila, kolica i robota

Profesor:

Doc. dr. Elmir Babović

Student:

Samir Habota IB160304

Akadska 2019/2020. godina

Mostar, mart 2020.

Sadržaj

IZJAVA O AUTORSTVU	1
SAŽETAK	2
ABSTRACT	3
1. Uvod	4
2. Obrada slike	5
2.1. Osnovni pojmovi za rad sa slikama	5
2.2. RGB spektar: red, green, blue	7
2.3. HSV spektar: hue, saturation, value	8
2.4. Binarna slika	9
2.5. Šum ili buka slike	11
2.5.1. Median filter	12
2.5.2. Gausovo zamagljenje	13
2.5.3. Matematička morfologija	14
2.5.3.1. Morfološka erozija	14
2.5.3.2. Morfološka dilatacija	15
2.6. Prevođenje slike u video	16
3. Implementacija softvera	17
3.1. Razvojno okruženje: Visual Studio	17
3.2. OpenCV	19
3.2.1. Dodavanje OpenCV biblioteke u projekat	20
3.2.2. Osnovni koncepti OpenCV biblioteke	23
3.2.2.1. Mat klasa	23
3.2.2.2. VideoCapture klasa	25
3.2.2.3. Funkcije ilustriranja	29
3.2.3. Definisanje vrijednosti prije praćenja laserske projekcije	32
3.2.3.1. HSV vrijednosti za binarnu sliku	33
3.2.3.2. Konstante laserske projekcije za binarnu sliku	36
3.2.3.3. Definisanje i upravljanje prozorima	37
3.2.4. Praćenje laserske projekcije	38
3.2.5. Pozicija laserske projekcije	51

3.2.6. Određivanje brzine kretanja	53
3.3. Značenje, otvaranje i komunikacija serijskog porta	56
4. Implementacija hardvera	59
4.1. Arduino Mega 2560	59
4.2. Programski jezik i kompajliranje Arduino koda	62
4.3. Arduino razvojna okruženja	63
4.3.1. Arduino IDE	63
4.3.2. PlatformIO	66
4.4. Arduino programski kod za upravljanje robota	67
4.5. Konstrukcija robota i spajanje komponenti	72
5. Testiranje robota	75
6. Budućnost projekta	76
7. Zaključak	77
8. Slike	78
9. Programski kodovi	79
10. Literatura	81

IZJAVA O AUTORSTVU

Ja, Samir (Smajo) Habota, student Fakulteta informacijskih tehnologija, Univerziteta „Džemal Bijedić“ u Mostaru, pod punom moralnom, materijalnom i krivičnom odgovornošću,

izjavljujem

da je rad pod naslovom:

„Lasersko navođenje robota korištenjem computer vision tehnologija“

u potpunosti rezultat sopstvenog istraživanja, gdje su korišteni sadržaji (tekst, ilustracije, tabele itd.) drugih autora jasno označeni pozivanjem na izvor i ne narušavaju bilo čija vlasnička ili autorska prava.

U Mostaru, 07.03.2020.

Samir Habota IB160304

SAŽETAK

Ovaj projekat je rezultat multidisciplinarnih vještina, prije svega iz oblasti razvoja softvera i hardverskih uređaja, koji za cilj ima kreirati softver i hardver koji će krajnjem korisniku omogućiti navođenje robota različitih veličina i svrha laserskim projekcijama.

Tok procesa se odvija u ugrađenom sistemu robota, softveru pisanom pomoću openCV biblioteke, softvera pisanom za arduino ploču, oboje koristeći programski jezik c++, a primarni učesnici u toj komunikaciji su robot i korisnik uz laser kao glavno sredstvo interakcije.

Projektna dokumentacija se sastoji od opšte analize slike i videa, dekompozicije njihovog načina rada i percepiranja okoline u kompjuterskom svijetu, te će u detalje razložiti proces i dočarati postupke koji su potrebni za krajnji cilj kompjuterskog praćenja objekta, koji će u ovom projektu biti predstavljen laserskom projekcijom.

Dokumentacija će pružiti obradu i objašnjenje computer vision openCV biblioteke, njenih funkcija i načina rada, uz dokumentaciju i primjere postupaka i preklapanja istih funkcija sa stvarnim svijetom.

Projekat će preuzeti odgovornost pojašnjenja svih razvojnih okruženja potrebnih za realizaciju rada, koristeći više alternativnih verzija okruženja, uključujući njihove prednosti i nedostatke.

Koristeći arduino ploču kao glavnu hardversku komponentu za donošenje odluka i pokretanje samog robota, projektna dokumentacija će u detalje pojasniti njenu svrhu, način rada, softver, logiku, komponente kao i spajanje elemenata u jednu cjelinu.

Kreirani robot će sam donositi odluke o praćenju eksterne laserske projekcije, brzini prilaženja, momentu zaustavljanja i skretanja koristeći computer vision tehnologije.

Ključne riječi: robot, softver, hardver, OpenCV, computer vision, motion tracking, lasersko navođenje, arduino.

ABSTRACT

This project is the result of multidisciplinary skills, primarily in the field of software and hardware development, which aims to create software and hardware that will allow the end user to lead robots of different sizes and purposes using laser projections.

The process flow takes place inside the embedded robot system, software written using the openCV library, software written for the arduino board, both using the c ++ programming language, the primary participants in this communication being the robot and the user with the laser as the main means of interaction.

The project documentation consists of a general analysis of image and video, the decomposition of their way of operation and the perception of the environment in the computer world, and will explain in detail the process and outline the procedures required for the ultimate goal of computer motion tracking of an object, which will be in this project represented as a laser projection.

The documentation will provide processing and explain computer vision of the openCV library, its functions and modes, along with documentation and examples of procedures and its overlapping of the functions with the real world.

The project will take on the responsibility of explaining all the development environments required to realize the work, using multiple alternative versions of the environment, including their advantages and disadvantages.

Using the arduino board as the main hardware component for making decisions and moving the robot itself, the project documentation will explain in detail its purpose, mode, software, logic, components, as well as fusing of all the elements into one unit.

The created robot will make its own decisions about monitoring and tracking the external laser projection, speed of approach, moment of stopping and turning, all using computer vision technologies.

Keywords: robot, software, hardware, OpenCV, computer vision, motion tracking, laser guidance, arduino.

1. Uvod

Tehnologija u današnjem modernom svijetu pušta svoje korijenje u skoro pa svaku oblast čovječanstva, gledajući da poboljša, unaprijedi i olakša svakodnevne poslove, rutine i ljudske obaveze.

Živimo u svijetu i okruženju gdje je tehnološki napredak na zavidnom nivou i gdje se nove tehnologije i pomagala rađaju svakim danom.

Fizionomija ljudi nije nikada bila namijenjena za teške fizičke poslove, koji su potrebni da bi se naša civilizacija održavala i napredovala prirodom i tehnologijom. Zbog toga je čovjek izumio mašine za prenos teških, krupnih i nepredvidivih stvari, kako bi sebi olakšao i ubrzao proces daljeg napredka.

Međutim, ponuđena rješenja su se rodila u vremenu kada tehnologija nije bila na nivou na kojem je danas, što dovodi do toga da se teška kolica tokom kupovine i dalje moraju gurati i vući za sobom, da komunalni radnici i dalje moraju sami vući ogromne kontejnere na dizalice, radnici na brodovima i skladištima prenose robu na osnovu svoje fizičke sposobnosti.

Dosadašnja rješenja kao što su lebdeća kolica ili kolica za smanjenje težine sadržaja, se i dalje oslanjaju na fizičku snagu ljudi u takvim okruženjima, ali nijedno ne eliminiše taj faktor u potpunosti ili nudi priliku za otklanjanje ljudskog faktora ikako iz takvih poslova, što bi se u kombinaciji sa dosadašnjim rješenjima moglo positići veoma lahko.

Uz realizaciju i implementaciju uređaja koji bi isključio faktor ljudske fizičke snage bi se znatno smanjile instance povreda, opasnih radnih mjesta, ili ozlijeđa i otvorio bi se spektar novih radnih pozicija upravljanja, programiranja i korištenja takvih robota unutar vlastitog okruženja, uslova i radnog prostora.

Ovaj rad upravo predstavlja koncept kreiranja robota koji će učiniti tačno to: otkloniti potrebu ljudske fizičke snage, koja ograničava rad pojedinih mašina i stvara opasne scenarije za koriniske, koji će sam, zajedno sa sadržajem promjenjive težine, pratiti lasersku projekciju, projektovanu od korisnika ili druge mašine.

Ovakvo rješenje je zbog načina navođenja laserom, maksimalno modularno i skalabilno, te može u zavisnosti od motora prenositi težine koje su do sada bile ograničene nemoći ili opasnošću, te ubrzati rast tehnologije i osigurati sigurnija radna okruženja.

Radnici u skladištima bi mogli laserom navoditi kolica koja sami ne bi mogli ni pomjeriti, bez da sebe dovode u opasnost, kupci u supermarketima navoditi svoja kolica bilo koje težine bez da spriječavaju druge kupce, dok bi velike korporacije i agencije mogle automatizirati u potpunosti rad svojih skladišta uvođenjem automatskih laserskih projekcija, njihovih programabilnih ruta itd.

U nastavku će biti govora o implementaciji i realizaciji softvera i hardvera za robota koji će pomoću computer vision tehnologija moći biti navođen laserskim projekcijama.

2. Obrada slike

Da bi uopšte započeli razvijati koncept navođenja robota computer vision tehnologijom, jedan od osnovnih koncepata koji je krucialan za razumjeti je način na koji računar ili robot vizuelno percipira svijet oko sebe.

Potrebno je sintetički simulirati i implementirati čulo vida, kako bi se robot mogao orijentisati u prostoru, što je neophodno za bilo kakvo dalje upravljenje i navođenje.

Implementacijom vida ćemo robotu dati osnovni temelj za praćenje specifičnih objekata koje vidi, te mu u stvarnom vremenu moći slati signale za kretanje, na osnovu njegove kalkulacije objekata koje prvo vidi, pa onda prati.

Kako bi simulirali ikakav vid, potrebna nam je kamera, kao jedan od osnovnih hardverskih komponenti, koja će se nalaziti na prednjem dijelu robota, čekajući input sliku, koju će obrađivati do tog nivoa dok ne ustanovi objekat koji treba da prati.

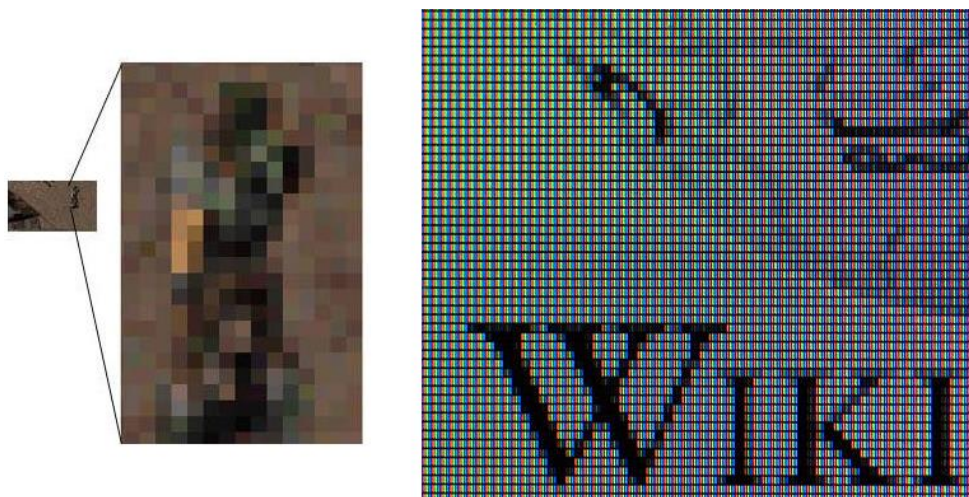
Međutim, kako bi došao do tog nivoa da sa slike ustanovi objekat koji je potrebno pratiti, potrebno je input sliku obraditi na specifičan način.

2.1. Osnovni pojmovi za rad sa slikama

Računar bilo koje vrste, input sliku koju prikazuje na ekran, prati ili snima vidi kao skup piksela u 2D koordinatnom sistemu.

Jedan piksel (riječ kreirana od engl. „picture element“) je osnovna i najmanja jedinica programibilne boje na ekranu računara ili kompjuterkoj slici [1].

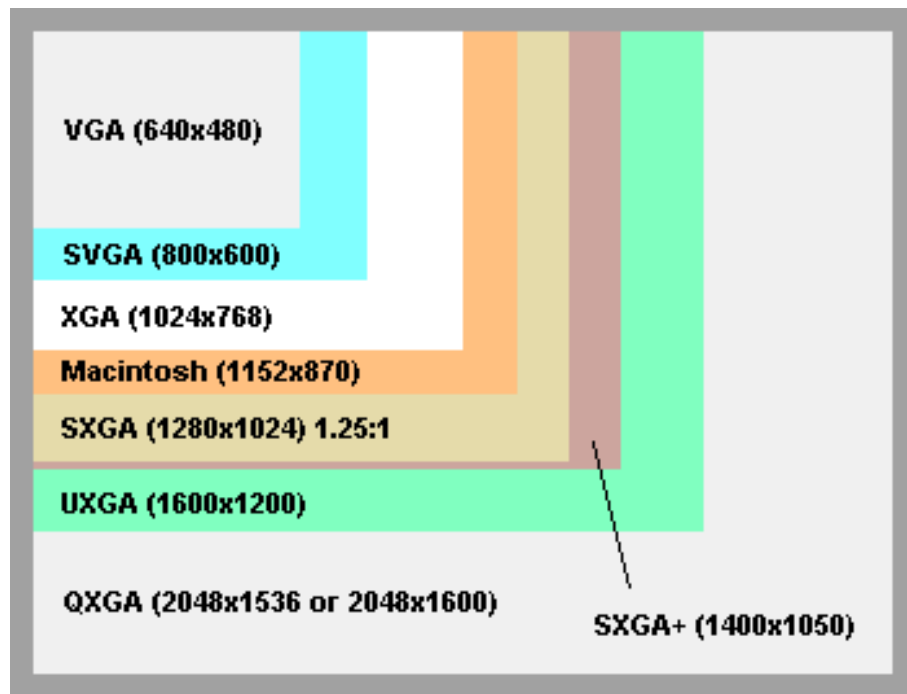
Riječ je o logičkoj jedinici, dok njena fizička reprezentacija zavisi od rezolucije ekrana na kojoj se slika prikazuje.



Slika 1 Lijevo: Logička reprezentacija piksela, Desno: Fizička reprezentacija piksela

Rezolucija predstavlja broj piksela (individualnih tačaka boje) sadržanih na ekranu monitora, izraženih u obliku broja piksela po horizontalnoj i vertikalnoj osi.

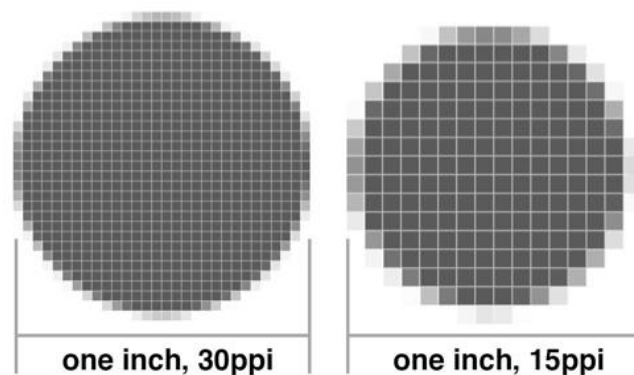
Oštrina slike zavisi od rezolucije slike i veličine ekrana ili prozora unutar kojeg se slika prikazuje. Zbog toga, identična rezolucija piksela daje oštriju sliku na manjim ekranima, dok se oštrina postepeno gubi povećavanjem ekrana ili prozora koji prikazuje sliku, jer se isti broj piksela širi na veću fizičku površinu.



Slika 2 Standardne rezoucije ekrana

PPI (engl. pixel per inch) ili broj piksela po jedom inču predstavlja mjeru oštine odnosno gustoće osvijetljenih tačaka (piksela) na ekranu.

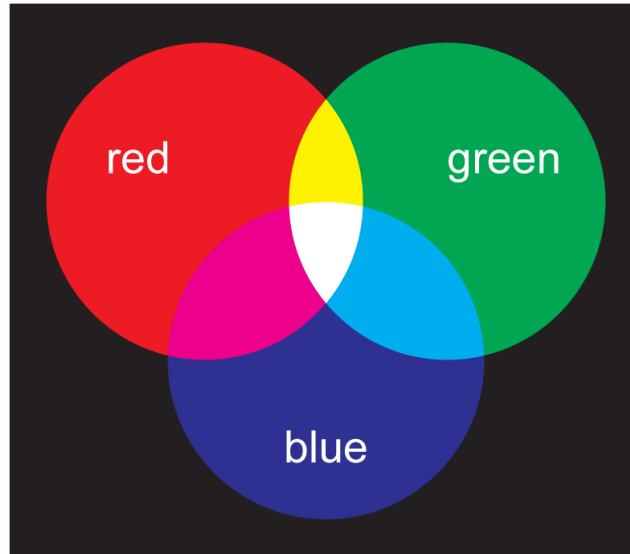
Pojam **DPI** (engl. dots per inch) ili broj osvijetljenih tačaka po jednom inču je termin za istu karakteristiku gustoće korišten za printanje. Pojam DPI u svijetu kompjutera, ekrana, kompjuterske grafike ili computer vision tehnologija se često koristi umjesto pojma PPI zbog veoma sličnog značenja unutar ovog konetksta.



Slika 3 Vizuelni prikaz različitih PPI vrijedosti

2.2. RGB spektar: red, green, blue

U svijetu likovne umjetnosti, paleta primarnih boja se sastoji od crvene, zelene i plave boje. Naziv „paleta ili spektar primarnih boja“ je dobila zbog karakteristike da se bilo koja druga boja unutar vidljivog spektra boja može dobiti kombinacijom ove tri boje.

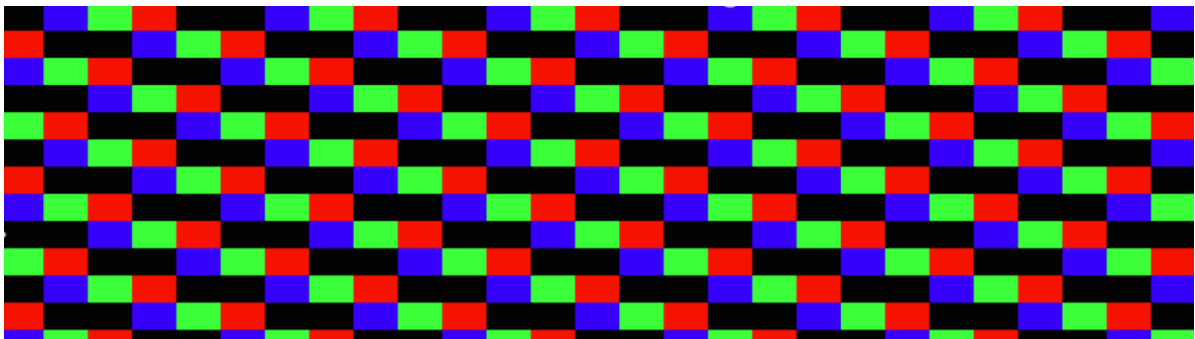


Slika 4 RGB spektar boja

U kompjuterskoj grafici, svaki piksel na ekranu je sastavljen od samo ove tri primarne boje, u različitim primjesama, te u zavisnosti od intenziteta svake pojedinačne boje, se kreira finalna boja, vidljiva oku u kombinaciji sa drugim pikselima koji čine sliku, kada se pikseli izoštre i udalje.

Unutar kompjuterske slike, intenzitet crvene, zelene i plave boje se može kretati od 0 do 100 posto intenziteta. Svaki nivo, procenat se može predstaviti rasponom deimclanih brojeva od 0 do 255, što čini 256 razina svake boje, što je u binarnom sistemu brojeva raspon od 00000000 do 11111111, ili u hexadecialnom sistemu od 00 do FF.

Ukupan broj boja mogućih ovom kombinacijom brojeva je $256 \times 256 \times 256$ ili 16,777,216 ukupnih boja koje je moguće prikazati na ekranu.



Slika 5 Mapiranje RGB boja po pikselu

Radi izrazito velike kombinacije boja, čiji obim pokriva svaku boju vidljivog spektra, se input slika dobijena kamerom prirodno prikazuje u primarnom RGB spektru boja.

Međutim, za praćenje objekata koje želimo postići, prirodna slika u RGB spektru nam je dovoljna samo za korisnika, kako bi on vidio prirodno stanje vida robota.

Da bi postigli praćenje i navođenje, moramo RGB sliku dalje obraditi i opstruisati vid robota, kako bi on iz čitave RGB slike vidio samo dijelove koji su njemu bitni i potrebni, te na osnovu tih dijelova slike dalje kalkulisao svoje kretanje i praćenje.

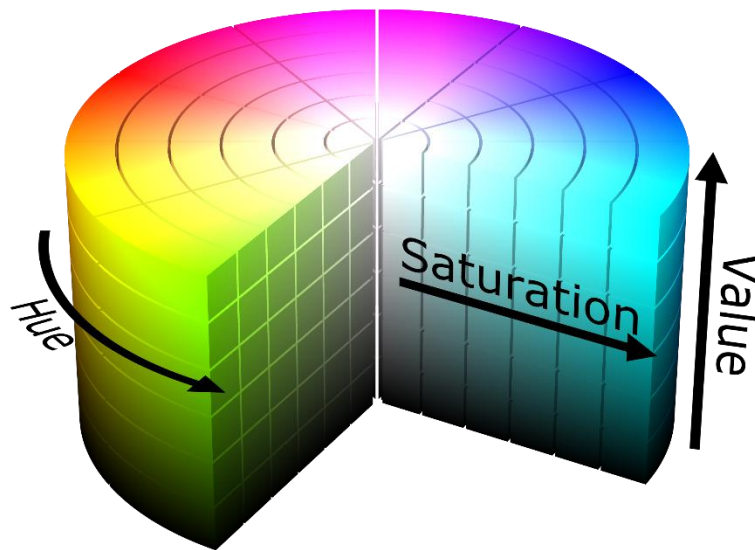
Da bi postigli takvu opstrukciju slike unutar vida robota, moramo je obraditi u par koraka, mijenjajući spektar, izgled i attribute input slike.

2.3. HSV spektar: hue, saturation, value

Prvi korak pri opstruisanju slike kako bi je prilagodili za praćenje određenih objekata i projekcija jeste konvertovati je iz RGB spektra boja u HSV spektar.

HSV model (engl. hue, saturation, value), je također tročlana jedinica koja definiše boju na osnovu nijanse (hue), zasićenosti (saturation) i vrijednosti svjetline (value ili brightness).

Model je baziran na 3D cilindričnim polarnim koordinatama, gdje se vizuelna reprezentacija može demonstrirati prikazom cilindra unutar RGB modela moja.



Slika 6 HSV model

Hue ili **nijansa** predstavlja čistu boju odnosno nijansu boje unutar RGB modela koji je predstavljen tačkom unutar 360 stepeni kruga boja. Cilindar je moguće otvoriti i posmatrati ga kao spektar boja u 2D koordinatnom sistemu, gdje se nijanse boje mijenjaju kretanjem niz traku 2D cilindra u pravcu lijevo ili desno.

Saturation ili zasićenost predstavlja intenzitet boje odnosno domet sive boje unutar nijanse odabrane boje. U većini slučajeva je prikazana u obliku procenata od 0% do 100%, gdje 100% predstavlja najveći intenzitet određene boje.

Value ili vrijednost svjetline predstavlja nivo svjetline određene boje. Također je prikazana u obliku procenata od 0% do 100%, gdje 0% predstavlja crno, a 100% najsvjetlije.

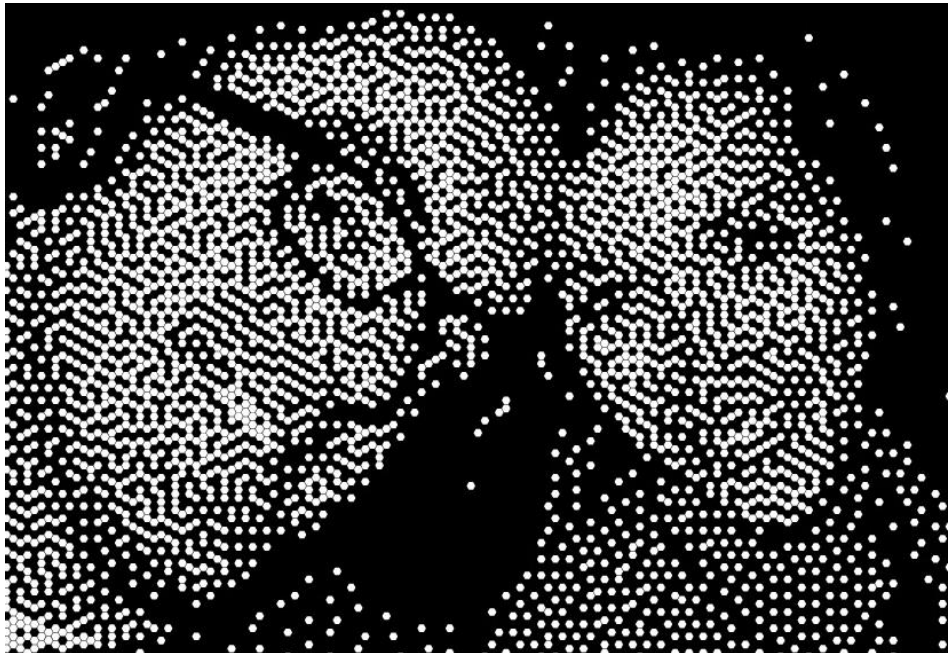
U kompjuterkoj grafici je moguće konvertovati iz RGB spektra boja u HSV spektar boja, gdje se svaki piksel sa svoje tri cifre RGB vrijednosti, konvertuje u tri cifre HSV vrijednosti, decimalni broj RGB nijanse početne boje, procenat intenzitet, te procenat svjetline početne boje.

Pretvaranjem slike koju smo dobili kamerom iz RGB spektra u HSV spektar smo završili prvi korak obrađivanja i prilagođavanja slike vidu robota, kako bi ostavili praćenje tačnih objekata odnosno laserskih projekcija.

Sljedeći korak predstavlja konverotvanje kreirane HSV slike u binarnu sliku, na osnovu pojedinačnih HSV vrijednosti svakog piksela slike, od kojeg će zavisiti pikseli binarne slike.

2.4. Binarna slika

Binarna (engl. binary ili bi-level, two-level) slika je slika koja se sastoji od piksela koji mogu biti jedne od tačno i samo dvije boje, najčešće bijela i crna. To znači da je svaki piksel sačuvan unutar jendog bita, koji može odgovarati tačno i samo jendom od dvije vrijednosti: 0 i 1, gdje se nulom najčešće notira crni piksel, a jedinicom bijeli.



Slika 7 Prikaz binarne slike

Binarne slike se često pojavljuju u digitalnoj obradi kao maske, pragovi podešavanja (engl. threshold) ili slike drhtanja (engl. dithering, image noise).

Neki input ili output uređaji kao što su laserski printeri, fax mašine ili čak bi-level računarski monitori podržavaju samo binarne slike i prikaze.

Prednosti binarne slike prilikom njene obrade je ta što se u memoriji može pohraniti i čuvati kao bit mapa (engl. bitmap), koja predstavlja dinamički niz popunjen sa bitovima odgovarajućih boja, predstavljenim kao nule i jedinice.

U svijetu programiranja, rad sa nizovima elemenata tako niskog nivoa znatno olakšava generalnu manipulaciju sadržaja i finalnog produkta.

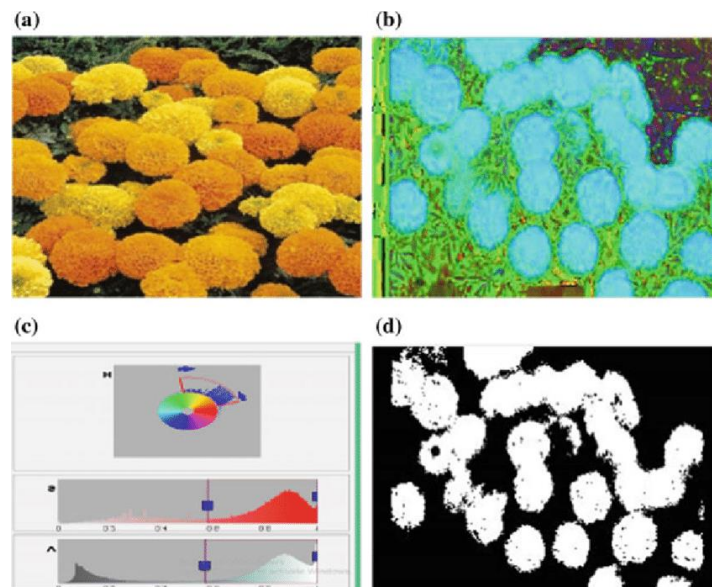
Dodatno, jedna binarna slika dimenzija 640x480 zahtijeva samo 37.5 KB memorije, zbog čega ovaj format veoma često koriste fax mašine i druga rješenja za menadžment dokumenata.

Svaka slika prikazana unutar HSV spektra boja, se može konvertovati u binarnu sliku koristeći prag date HSV slike (engl. thresholding an HSV image).

Različite pojedinačne HSV vrijednost, ili minimalna i maksimalna dozvoljena pojedinačna HSV vrijednost po pikselu slike, kreira različitu binarnu sliku, odnosno postavlja bijelu boju, označenu sa binarnom jedinicom i crnu boju, označenu sa binarnom nulom na različita mjesta.

Pomoću ovog koncepta je moguće konvertovati HSV sliku i kreirati binarnu sliku koja će pozicije tačno određene boje popunjavati bijelim pikselima, a sve ostalo crnim, te manipulisanjem pojedinačnih HSV vrijednosti mijenjati praćenu boju koju želimo konvertovati.

Ovaj koncept će dalje biti razrađen prilikom praćenja objekata na binarnoj slici, manipulacijom minimalnih i maksimalnih HSV vrijednosti input slike.



Slika 8 Proces konverzije RGB slike u binarnu sliku za žutu boju: a) RGB input slika
b) HSV slika c) Podešavanje HSV vrijednosti za žutu boju d) Binarna slika

Međutim, nakon obrade slike i konvertovanja iste iz RGB modela u HSV model, modifikovanja HSV vrijednosti kako bi se dobila binarna slika koja postavlja bijelu boju na određene objekte, je potrebno dodatno optimizirati binarnu sliku, kako bi robot kasnije mogao sa što većom sigurnošću i preciznošću pratiti lasersku projekciju.

Njaveća optimizacija se ogleda u što boljem otklanjanju šuma ili buke unutar same slike.

2.5. Šum ili buka slike

Šum slike je slučajna ili nenamjerna varijacija svjetline ili informacija o boji određene slike i najčešće je aspekt elektronskog šuma. Može ga proizvesti senzor i sklop skenera ili digitalne kamere.

Šum slike je generalno nepoželjna nus pojava prilikom bilo kojeg snimanja i prikaza fotografije, koja opstruira informacije na samoj slici.

Može se pojaviti u različitim veličinama i nivoima opstruisanja slike, od najmanjih mrlja ili zrna na digitalnoj fotografiji, do tolike razmjere da se glavni objekat prikazan na slici više ne može jasno razaznati.



Slika 9 Prikaz šuma ili buke na slici

Postoje različiti tipovi šuma slike, svaki specifičan po svom izgledu. Neki od njih su: Gausov šum (engl. Gaussian noise), sol i biber šum (engl. Salt-and-pepper noise), anizotropni šum (engl. Anisotropic noise), periodični šum (engl. Periodic noise) itd.

Za otklanjanje šuma sa RGB ili HSV input slike postoje razne tehnike, od kojih su najefikasnije ugrađivanje median filtera (engl. Median filter) i Gausovog zamagljenja (engl. Gaussian blur), dok je za binarne fotografije najefikasnije koristiti morfološke operacije tipa erozije i dilatacije.

2.5.1. Median filter

Median filter je nelinearna tehnika digitalnog filtriranja, najčešće korištena za uklanjanje šuma sa fotografija ili signala. Smanjenje buke korištenjem median filtera je najčešće korak predobrade slike, ili pripreme slike manjom obradom, koja će spremiti fotografiju za dalje, veće obrade.

Median filter se veoma često koristi u digitalnoj obradi fotografija jer, pod ispravnim uslovima, sačuva ivice fotografije dok otkloni sav ostali šum unutar nje.



Slika 10 Poboljšanje slike koja boluje od nedostatka piksela koristeći median filter

Glavna ideja median filtera se nalazi u njegovom algoritmu, koji prolazi kroz fotografiju dio po dio, mijenjajući svaki dio ili inkrement slike sa medijanom susjdenih inkremenata, pri čemu medijana predstavlja statističku centralnu vrijednost ta dva inkrementa fotografije.

Šablon susjednih inkremenata slike se naziva prozor (engl. window), koji se pomjera inkrement po inkrement tokom cijele fotografije.

Za 2D signale ili fotografije za unutar prozora se moraju uključiti svi inkrementi datog prečnika ili elipsoidne regije.

Median filter se može implementirati u skoro pa svakom programskom jeziku i okruženju koji dozvoljava rad sa fotografijama, signalom ili videom, gdje je najbolja praksa, u zavisnosti od inputa, programskog okruženja ili manjka ugrađenih funkcija, koristiti sljedeći pseudo kod:

```
allocate outputPixelValue[image width][image height]
allocate window>window width * window height
edgex := (window width / 2) rounded down
edgey := (window height / 2) rounded down
for x from edgex to image width - edgex do
  for y from edgey to image height - edgey do
    i = 0
    for fx from 0 to window width do
      for fy from 0 to window height do
        window[i] := inputPixelValue[x + fx - edgex][y + fy - edgey]
        i := i + 1
      sort entries in window[]
      outputPixelValue[x][y] := window>window width * window height / 2]
```

Programski kod 1 Pseudo kod median filtera

2.5.2. Gausovo zamagljenje

U obradi slike, Gausovo zamagljenje ili izgladivanje (engl. Gaussian blur, Gaussian smoothing) je rezultat zamagljenja fotografije Gausovom funkcijom. Ova tehnika je široko i često korištena u grafičkom softveru i obradi slika u svrhu smanjenja šuma i detalja fotografije.

Vizuelni efekat koji se dobije tehnikom zamagljenja slike je sličan kao posmatranje određene fotografije kroz prozirni ekran. Zamagljenje otklanja manje iregularitete, šum i mrlje slike tako što ih stopi u preostalu kompoziciju fotografije.

Gausovo zamagljenje se također koristi kao predobrada fotografije, za dalju obradu kroz druge computer vision algoritme.



Slika 11 Poboljšanje bučne slike koristeći Gausovo zamagljenje

2.5.3. Matematička morfologija

Matematička morfologija (MM) je teorija i tehnika za analizu i procesiranje geometrijskih struktura zasnovana na teoriji skupova, teoriji rešetke, topologiji i nasumičnim funkcijama. Najčešće se primjenjuje na digitalnim slikama, ali primjenu može naći i u grafovima, površinskim mrežama, tvrdom materijom i mnogim drugim prostornim strukturama.

Topološki i geometrijski koncepti neprekidnog prostora kao što su veličina, oblik, konveksnost, povezanosti i geodetska udaljenost je matematička morfologija uvela na kontinuirane kao i diskretne prostore.

Matematička morfologija je temelj morfološke obrade slike, koja se sastoji od skupa operatora koji transformišu slike prema navedenim karakteristikama.

Prvobitno je razvijena za obradu binarnih slika, u kojem kontekstu će biti korištena u ovom radu, ali se kasnije proširila na funkcije sivih tonova slike.

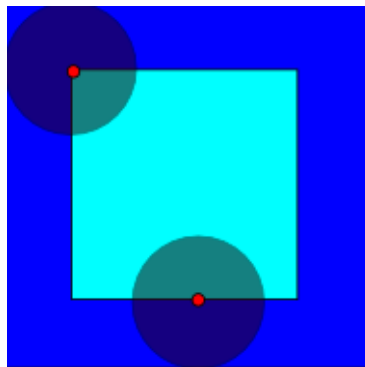
Osnovni morfološki operatori su erozija, dilatacija, otvaranje i zatvaranje, pri čemu će posebna pažnja biti okrenuta prema eroziji i dilataciji binarne slike.

2.5.3.1. Morfološka erozija

Erozija je jedna od dvije temeljne operacije u obradi morfološke slike, na kojoj se temelje sve ostale morfološke operacije.

U binarnoj morfologiji, slika se gleda kao podskup Euklidiskog prostora¹. Osnovna ideja u binarnoj morfologiji je ispitivanje slike jednostavnim, unaprijed definiranim oblikom, izvlačeći zaključke o tome kako se ovaj oblik uklapa ili promašuje oblike na slici.

U pojednostavljenom obliku, erozija binarne slike prolazi kroz sve vidljive granice slike koje su predstavljanje sa jedincama ili bijelom bojom, uključujući bilo kakav šum ili buku na slici, te poredi predefinisane oblike da li se mogu uklopiti i popuniti u granice negativnog prostora nula ili crne boje sa šumom ili bijelom bojom slike koji je pronađen.



Slika 12 Erozijski tamno plavog kvadrata u svijetlo plavi kvadrat koristeći predefinisane granice

¹Matematički prostor kojeg intuitivno svakodnevno zamišljamo. Naziv je dobio po starogrčkom matematičaru Euklidu.

Ako se ne ispune kriteriji za popunjavanje, odnosno ako nisu jasno definisane granice oblika unutar binarne slike, gdje se može popuniti erodni oblik, što se najčešće dešava sa šumom na slici, jer on nema tačno definisane granice, onda se logičke jedinice pretvaraju u nule, odnosno bepostrepan bijeli prostor binarne slike uzrokovan šumom se smanjuje, stvarajući jasniju sliku sa manje buke i jasnije definisanim granicama.

Ova morfološka operacija erodira bijeli prostor, čineći ga manjim ili nepostojećim, testirajući da li predefinisani erodski oblici mogu popuniti prostor u određenim granicama.

Kriteriji su najčešće manji oblici, ili granice u koje se predefinisani oblik može smjestiti.

2.5.3.2. Morfološka dilatacija

Dilatacija je druga od dvije temeljne operacije u obadi morfološke slike.

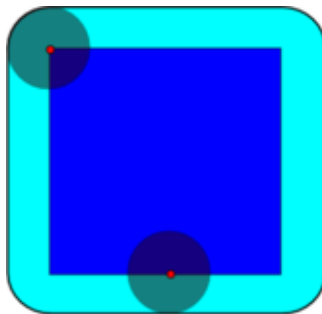
Nakon erozije binarne fotografije, bijeli prostor, prostor označen logičkim jedinicama, je smanjen unutar preciznih granica koje izbjegavaju šum slike.

Međutim, samo erozijom je moguće izgubiti tačne granice oblika na fotografiji, pogotovo ako je u pitanju veći oblik, koji će po prirodi algoritma izgubiti bijelog prostora u svom centru, kako se što više udaljava od granica oblika.

Također se mora uzeti u obzir da morfološke operacije ne daju 100% precizan prikaz oblika koji se prati na binarnoj slici, zbog raznih vanjskih faktora kao što su osvjetljenje, kvalitet kamere i krajnje količine buke na početnoj fotografiji.

Zbog toga se nakon procesa erozije, koji smanjuje šum binarne fotografije, ali je ostavlja u nestabilnom stanju, slika provlači kroz proces dilatacije koji treba dodatno ustabiliti sliku i oblik koji se prati, koristeći suprotni algoritam i logiku od erozije, ali ovaj put bez šuma na fotografiji.

Kako erozijom vidimo tačno posmatrane objekte na slici bez šuma, dilatacija bijeli prostor slike povećava do određenih predefinisanih granica, koristeći predefinisane oblike, kako bi popunila sve eventualne praznine ili negativan prostor označen logičkim nulama i dovela binarnu fotografiju u stabilno stanje, gdje se tačno mogu razaznati oblici sa početne slike.



Slika 13 Dilatacija tamno plavog kvadrata u svijetlo plavi kvadrat, popunjavanjem prostora u određenim granicama za zaobljenim oblikom

Ova morfološka operacija dilatira bijeli prostor, čineći ga većim i stabilnijim, u određenim granicama, sa predefinisanim oblikom.

Procesi erozije i dilatacije, kao i primjeri na binarnoj slici će biti detaljnije prikazani u softverskoj implementaciji ovog rada.

2.6. Prevođenje slike u video

Video je elektornički medij za snimanje, kopiranje, reprodukciju, emitovanje i prikaz pokretnih vizuelnih medija.

Video je najprije razvijen za mehaničke televizisjke sisteme, koji su brzo zamjenili sistemi katodnih cijevi (CRT), te u današnjici znamo za reprodukciju videa preko „pljosnatih“ TV ekrana, računara, laptopa, monitora ili mobilnih uređaja.

Video sistemi se razlikuju u rezoluciji prikaza, omjeru slike, brzini osvježavanja (engl. refresh rate), mogućnostima boje i drugim kvalitetama i osobinama već opisanim u ovom radu.

Postoje digitalna i analogna varijanta videa, koji se mogu prenositi po raznim medijima, uključujući radio prenos, magnetne vrpce, optičke diskove, računarske datoteke i mrežno strujanje.

Za implementaciju i realizaciju ovog rada je neophodno shvatiti logički koncept šta zapravo znači video u programskom smislu.

Bilo koji input videa kamerom se može tretirati kao niz slika, koje se prikazuju jedna za drugom, određenom brzinom.

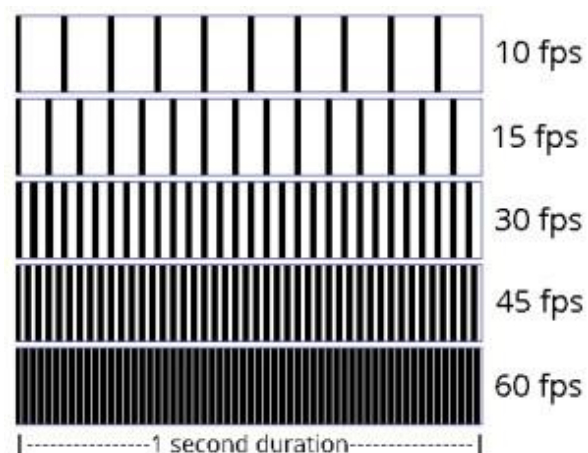
Ovaj koncept je nastao mnogo prije same reprodukcije videa, kada su animatori, ilustratori i slikari brzom zamjenom crteža ili fotografija jedno za drugom kreirali animacije.

U ovom kontesktu se jedna slika naziva frame, a njihovo mijenjanje jednim za drugim kreira termin frame po sekundi ili FPS (engl. frame per second), koji upravo predstavlja frekvenciju, brzinu u kojoj se slike uzastopno pojavljuju i mijenjaju na ekranu, jedna za drugom.

Za implementaciju robota koji će pratiti laserku projekciju je neophodno svaku sliku ili frame, koji robot primi simulacijom čula vida, modifikovati na gore objašnjeni način, korak po korak.

Modifikovanjem svake slike se efektivno modifikuje i video prenos, koji nije ništa više nego suma svih slika, poredanih jedna za drugom.

U nastavku će biti prikazano softversko rješenje, koje će koristiti ovo znanje kako bi simuliralo vid robota i kreiralo algoritam za praćenje laserskih projekcija.



Slika 14 Prikaz različitih "frames per second" projekcija slike

3. Implementacija softvera

Kako bi se u potpunosti realizovalo rješenje robota koji prati lasersku projekciju, potrebno je svo navedeno znanje iz poglavlja dva o obradi slike implementirati softverski na video inputu koji će robot primati, simuliranjem svog čula vida kroz kameru.

Hardverska komponenta potrebna za implementaciju i testiranje softvera je kamera, u formi ili ugrađenje web kamere na robotu, računaru ili laptopu, ili eksterne kamere priključene na sistem na način da robot i softver mogu doći do i očitati video.

Detaljnije informacije o radu kamere će biti naknadno navedene.

Razvojno okruženje korišteno za implementaciju softvera je Visual Studio 2019 Enterprise, dok je programski jezik korišten za pisanje softvera c++.

Srž softvera se oslanja na computer vision biblioteci OpenCV, koja nudi većinu funkcija potrebnih za obradu slike i videa predhodnog poglavlja.

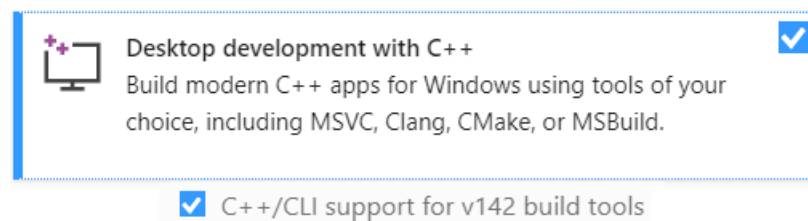
Detalji i intezivna obrada ove biblioteke slijede u nastavku.

3.1. Razvojno okruženje: Visual Studio

Već spomenuto razvojno okruženje korišteno za implementaciju softvera je Visual Studio 2019 Enterprise [2].

Visual Studio nudi razne šablone različitih programskih jezika i okvira (engl. frameworks), koji dolaze sa unaprijed definisanim datotekama potrebnim za rad na svakom pojedinačnom šablonu.

U svrhu izrade rada za razvoj softvera robota koji prati laserse projekcije, potrebno je prije svega unutar Visual Studio instalera omogućiti „desktop development with c++“, te označiti „c++/cli support“ u dodatnim opcijama za instalaciju, korištene verzije visual studia.



Slika 15 Postavke Visual Studio instalera prije kreiranja projekta

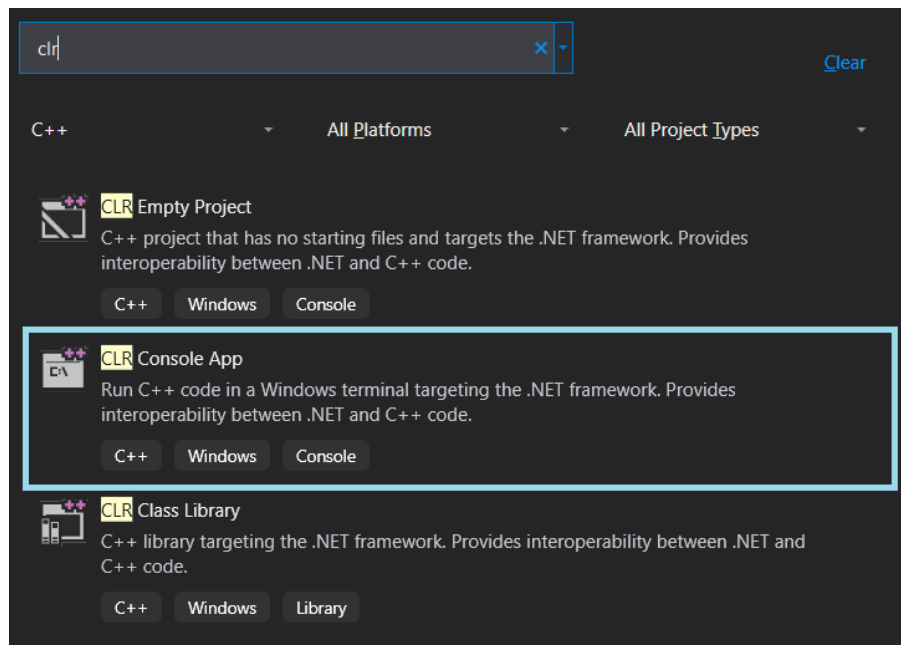
C++/CLI je c++ programski jezik modificiran sa cli odnosno „Common Language Infrastructure“. Radi se o specifikaciji programskog jezika c++, napravljenoj od strane Microsoft-a, koja zamjenjuje managed extenzije za c++. Riječ je o potpunoj reviziji koja pojednostavljuje zastarjelu managed c++ sintaksu i pruža interoperabilnost i kompatibilnost sa Microsoft .Net programskim jezicima kao što su c#.

Microsoft .Net framework je potreban kako bi budući projekat ostvarivao kontakt sa hardverskim komponentama, kao i omogućio rad u c++ šablonima potrebnim za implementaciju softvera.

Nakon modifikacije Visual Studio instalera, potrebno je otvoriti novi Visual C++ CLR konzolni projekat, u kojem će biti pisan softver za ovaj rad.

Instalacijom c++/cli support-a, dobijamo mogućnost kreiranja CLR projekta koji ostvaruje kompatibilnost i suradnju c++ i .Net koda.

CLR ili „Common Language Runtime“ je dio Microsoft .Net framework-a, koji ostvaruje izvršenje programa pisanih u drugačije podržavanim programskim jezicima, kao što je u ovom slučaju c++.



Slika 16 Keiranje CLR konzolne aplikacije unutar Visual Studia

Šabloni (engl. templates) unutar Visual Studia sa sobom donose već ugrađenje datoteke, biblioteke i zavisnosti (engl. dependencies), koje su potrebne za realizaciju vrste programa kreiranog odgovarajućim šablonom.

CLR konzolni projekat dolazi sa preodređenom sintaksom, datotekama i bibliotekama pomoću kojih će se kasnije moći uspostaviti serijska komunikacija sa hardverom.

Nakon postavki razvojnog okruženja, potrebno je dodati OpenCV biblioteku, koja predstavlja srž koda pisanog za implementaciju softvera.

3.2. OpenCV

OpenCV (Open Source Computer Vision Library) je open source computer vision i machine learning biblioteka [3]. Biblioteka i čitav projekat je izgrađen kako bi pružio zajedničku infrastrukturu za aplikacije računarskog vida i ubrzao upotrebu mašinske percepcije u komercijalnim proizvodima.

Budući da je BSD licenciran² produkt, OpenCV olakšava kompanijama upotrebu i izmjenu koda. Biblioteka posjeduje preko 2500 optimiziranih algoritama u koje spada sveobuhvatan skup klasičnih kao i najsavremenijih algoritama računarskog vida i mašinskog učenja.

Ovi algoritmi se mogu koristiti za prepoznavanje lica, objekata, klasificiranje ljudskih radnji u videima, praćenje pokreta kamere, praćenje objekata koji se kreću, ekstraktovanje 3D modela objekata, praćenje očiju i u mnoge druge svrhe.

Zajedno sa velikim firmama kao što su Google, Yahoo, Microsoft, Intel, IBM, Sony i Honda koje već koriste biblioteku, drugi manji projekti unutar uzlaznih kompanija kao što su Applied Minds, VideoSurf i Zeitera intenzivno koriste mogućnosti OpenCV biblioteke, šireći je na još veće tržište.

Posjeduje C++, Python, Java i MATLAB interfejs, a podržava Windows, Linux, Android i MAC OS. OpenCV uglavnom naginje ka aplikacijama za implementaciju vida u stvarnom vremenu i koristi MMX³ i SSE⁴ upute kada su dostupne.

OpenCV je izvorno pisan u C++ programskom jeziku i ima šablosnki predložen interfejs koji savršeno funkcioniše sa STL kontejnerima⁵.



Slika 17 Oficijelni logo OpenCV biblioteke

² BSD licence su skup licenci za dopušten besplatni softver, koji nameće minimalna ograničenja za upotrebu i distribuciju pokrivenog softvera.

³ MMX je "jedna instrukcija, više podataka" (SIMD) set instrukcija, dizajniran od strane Intel-a

⁴ SSE je nova SIMD eksenzija od Intel Pentium III

⁵ STL kontejner je kolekcija kompleksnih objekata istog tipa podataka

3.2.1. Dodavanje OpenCV biblioteke u projekat

Nakon postavljanja razvojnog okruženja, potrebno je dodati OpenCV u kreirani projekat, kako bi se mogle koristiti sve funkcije koje biblioteka nudi.

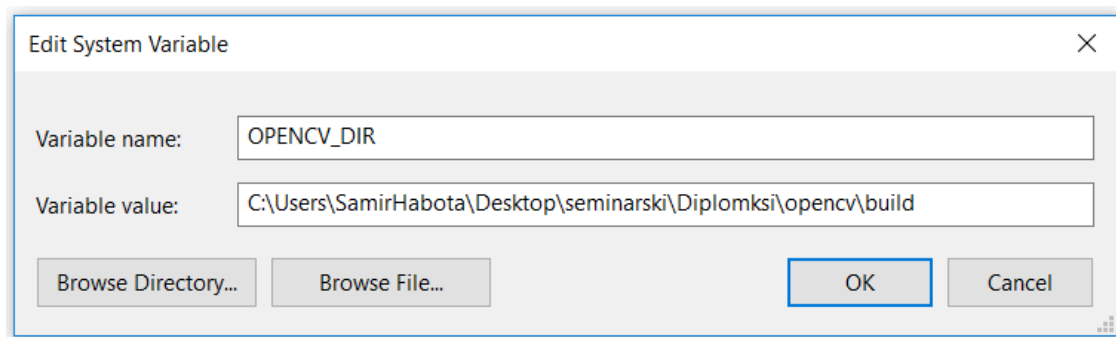
Za svrhu realizacije ovog rada će biti korištena OpenCV biblioteka verzije 3.1.0. Svaka verzija se može preuzeti sa oficijelne stranice biblioteke⁶.

Nakon download-a će se preuzeti jedan folder koji će u sebi imati pre-buildani build folder, sa svim potrebnim datotekama, kao i source folder.

Glavni OpenCV folder treba smjestiti na jedno mjesto, te unutar računara postaviti globalne varijable, koje predstavljaju putanje do određenih foldera.

Postavljanje putanje se vrši na sljedećoj lokaciji: desni klik na „my computer“, „properties“, gore desno na „advanced system settings“, te pri dnu kliknuti na „Environment Variables“.

U „Environment Variables“ je potrebno dodati novu „System Variable“ klikom na dugme „New“, te zapisati putanju do build foldera OpenCV biblioteke i dati joj jedinstveno ime.



Slika 18 Dodavanje globalne putanje za build folder OpenCV biblioteke

Nakon dodavanja putanje za build folder, potrebno je navigirati do „System Variable“ pod nazivom „Path“, te klikom na „Edit“ u nju dodati putanju do binarnih datoteka, klikom na dugme „New“. Unutar ove putanje će se već koristiti ime predhodno unešene globalne putanje do build foldera, kako se binarne datoteke nalaze u build folderu biblioteke. Znak „%“ služi kao separator globalnih sistemskih varijabli.

`%OPENCV_DIR%\x64\vc14\bin`

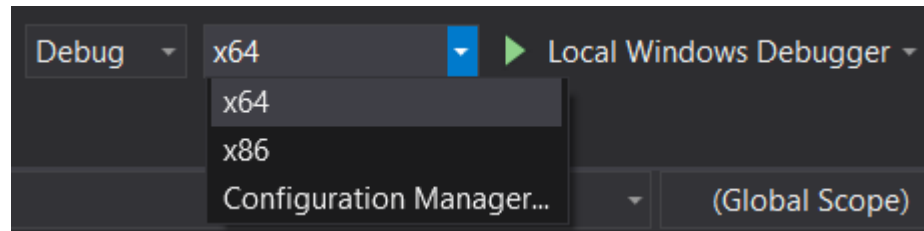
Slika 19 Dodavanje putanje za binarne datoteke

Na ovaj način postavljanja varijabli za putanje do osnovnih datoteka biblioteke, nije dalje potrebno mijenjati putanje u razvojnom okruženju, nego samo zamijeniti foldere, kako su putanje sada same po sebi dinamične. Sa time smo izbjegli probleme oko unaprjeđenja ili prebacivanja na novije verzije biblioteke, sa drugačijim sadržajem foldera, ali i dalje istim putanjama do njih.

⁶ <https://opencv.org/releases/>

Nakon definisanja i dodavanja putanja do foldera biblioteke, potrebno je iste dodati u razvojno okruženje.

Prije samog dodavanja biblioteke, potrebno je na vrhu prozora razvojnog okruženja Visual Studio, unutar „Configuration Manager“ sa padajuće liste odabrati x64 bitnu verziju okruženja, kako OpenCV podržava samo 64 bitno okruženje.

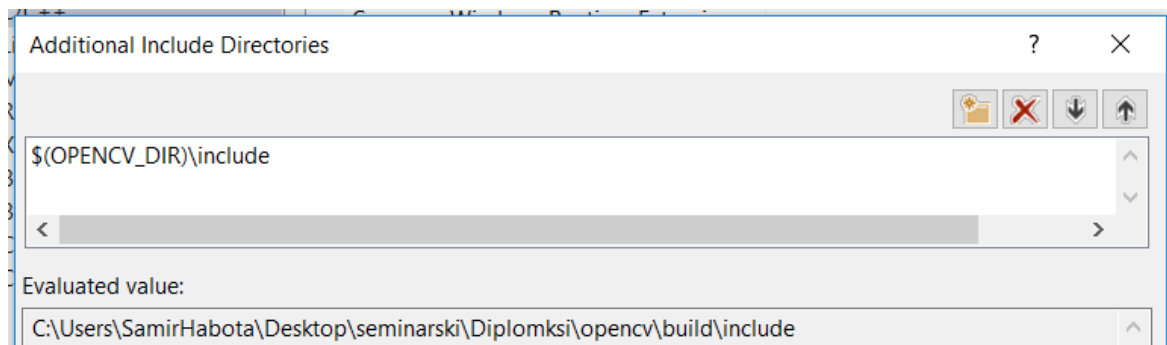


Slika 20 Configuration manager za x64 bitno okruženje

Da bi se dodale sve potrebne datoteke i putanje, potrebno je navigirati na tab „View“, „Solution Explorer“, desni klik na ime projekta, „Properties“, te na lijevoj strani kliknuti na tab „C/C++“.

Zatim, na desnoj strani navigirati do „Additional Include Directories“, te kliknuti na „Edit“, kako bi se dodala putanja koja sadrži sve include datoteke.

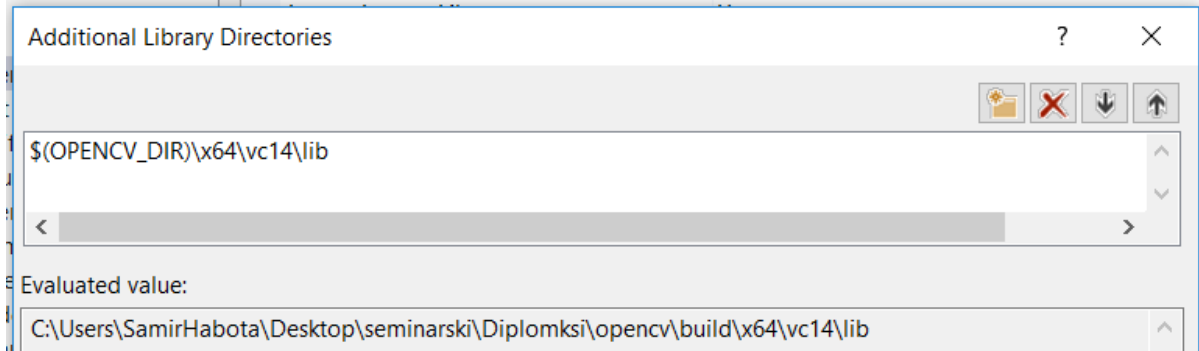
Include datoteke se nalaze u subfolderu „include“ unutar foldera build, te će se iskoristiti globalno unešena sistemaska varijabla do putanje build foldera. Znak „\$“ praćen sa zagradama mijenja globalnu varijablu dinamički, a njen puni naziv se može vidjeti ispod.



Slika 21 Dodavanje include datoteka u razvojno okruženje koristeći globalnu putanju do build foldera

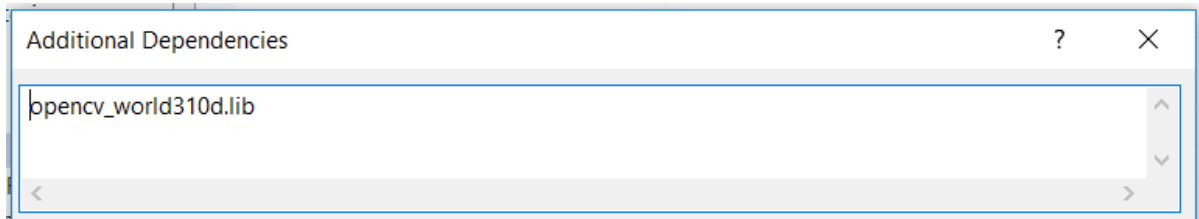
Nakon dodavanja include foldera, potrebno je opet sa lijeve strane, navigirati na tab „Linker“, koji se nalazi ispod „C/C++“ taba, te unutar „Linker“ taba, ispod odabrati tab „General“.

Desno je potrebno navigirati na „Additional Library Directories“, te klikom na „Edit“, dodati lib folder, koji sadrži sve lib datoteke, na isti način kao include folder, koristeći globalnu putanju do build foldera.



Slika 23 Dodavanje lib datoteka u razvojno okruženje koristeći globalnu putanju do build foldera

Verzija 3.1.0 sve funkcije drži u jednoj pre-buildanoj .lib datoteci unutar lib foldera, koja nosi naziv „opencv_world310d.lib“. Ovu datoteku je potrebno dodati navigirajući lijevo u tab „Input“, unutar „Linker“ taba, te desno na „Additional Dependencies“ kliknuti „Edit“ i dodati „opencv_world310d.lib“ datoteku.



Slika 22 Dodavanje "opencv_world310d.lib" datoteke iz lib foldera biblioteke

Po završetku procesa dodavanja OpenCV biblioteke u razvojno okruženje, možemo koristiti sve funkcije biblioteke, dodavanjem direktorija na vrhu cpp file-a. Također intellisense⁷ će podržavati sve ključne riječi i funkcije biblioteke.

Sa ovim smo efektivno dodali biblioteku u razvojno okruženje, te možemo započeti pisanje koda koristeći c++ programski jezik uz OpenCV biblioteku.

Također, za lakše korištenje svih funkcija unutar sada već dodane OpenCV biblioteke, potrebno je na vrh dokumenta dodati **namespace** „cv“, kako se sve funkcije biblioteke nalaze u cv:: namespace-u.

```
#include "pch.h"
#include <sstream>
#include <string>
#include <iostream>
#include <opencv\cv.h>
#include <opencv2\imgproc.hpp>
#include <opencv2\highgui.hpp>

using namespace cv;
```

Programski kod 2 Spisak svih pojedinačnih biblioteka i namespace-a za c++ i OpenCV na vrhu projekta

⁷ Pomoć za automatsko dovršavanje koda

3.2.2. Osnovni koncepti OpenCV biblioteke

OpenCV biblioteka nudi ugrađene klase, funkcije i logiku rada sa kojom se potrebno upoznati prije pisanja finalnog koda. Sve funkcije i klase će biti prikazane u programskom jeziku C++, dok slična sintaksa istih funkcija i objekata postoji i u drugim, već navedenim, programskim jezicima.

Svi koncepti nabrojani u ovom radu nisu jedini kojima OpenCV biblioteka vlada, ali spadaju u osnovne i biti će korištene u pisanju softvera za robota koji prati laserske projekcije[4].

3.2.2.1. Mat klasa

Mat klasa predstavlja osnovni kontejner za čuvanje slike ili frame-a. Objekat tipa Mat u sebi sadrži sve informacije potrebne za skladištenje i rad sa slikama.

Prva karakteristika klase Mat je ta da se za objekat tipa Mat ne mora ručno alocirati memorija za njen sadržaj, niti dealocirati nakon izvršenja upotrebe, kao što je to običaj sa ostalim klasama unutar programskog jezika C++. Iako je ovaj pristup i dalje moguć, većina funkcija i klasa unutar OpenCV biblioteke podrazumijeva samostalnu alokaciju i dealokaciju memorije.

Dodatno, ako se u bilo koju funkciju proslijedi već kreirani Mat objekat, sa već alociranom memorijom za njegov sadržaj, neće se praviti nova instanca prilikom preuzimanja objekta, nego će se iskoristiti već alocirani prostor, što znači da se nikada neće koristiti više memorije nego što je potrebno. Zbog toga je rad sa Mat objektima veoma zahvalan za memorijski prostor i pisanu sintaksu.

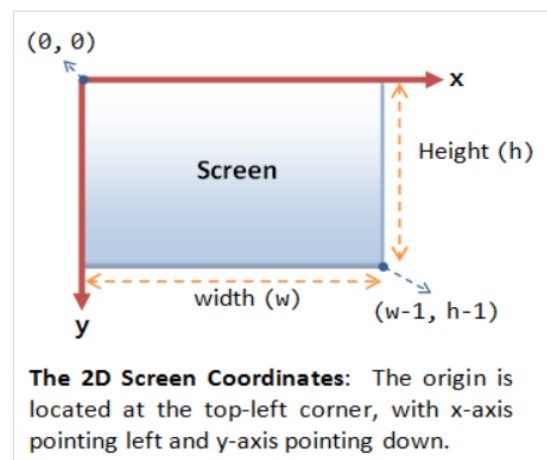
OpenCV je prvenstveno biblioteka za obradu slika. Zbog toga je prirodno da se objekti tipa „slika“ šalju u razne funkcije i provlače kroz mnoge algoritme, koji u zavisnosti na veličinu slike mogu kompjuterski biti veoma teški. Zbog toga OpenCV biblioteka nudi klasu koja u suštini čuva informacije jedne slike.

U jednom Mat objektu je slika, koja je učitana u njega, predstavljena kao matrica piksela, u 2D koordinatnom sistemu, gdje se prvi piksel nalazi u gornjem lijevom ćošku prozora koji prikazuje silku, na koordinatama matrice nula, nula. Svaki piksel slike u boji je sastavljen od svojih RGB vrijednosti, i kreće se u obliku matrice niz x i y koordinate slike.

Međutim ovo je samo slikovita reprezentacija Mat objekta. U stvarnosti se Mat objekat sastoji iz dva dijela: zaglavlje matrice (engl. matrix header) koje sadrži informacije o veličini matrice, načina kako je učitana, na kojoj adresi se nalazi itd i pokazivača koji pokazuje na matricu sa sadržajem piksela.

Zaglavlje matrice je konstantno, ali veličina može varirati od slike do slike i obično je veća za redoslijed veličine.

Mat objekti predstavlja osnovu za rad sa OpenCV bibliotekom i koriste se u skoro svakom algoritmu.



Slika 24 Prikaz slike kao matrice unutar Mat objekta

Učitavanje slike unutar Mat objekta se vrši preko ugrađene funkcije **imread**, koja kao rezultat vraća Mat objekat (pokazivač na novonastali Mat objekat) sa popunjenom slikom koja se predala kao parametar u funkciju imread. Tačnije, u prvi parametar se upisuje putanja i ime slike. Ako se slika nalazi u folderu samog projekta, onda je dovoljno napisati samo ime slike, pri čemu se putanja podrazumjeva.

Funkcija kao drugi parametar prima predefinisanu OpenCV zastavu (engl. flag), koja definiše na koji način će slika biti obrađena i prikazana. Dvije najčešće korištene zastave pri učitavanju slike su „**CV_LOAD_IMAGE_COLOR**“ koja učitava sliku u koloru i „**CV_LOAD_IMAGE_GRAYSCALE**“ koja učitava sivu sliku bez boje.

Ove zastave obrađuju vrijednosti svakog pojedinačnog piksela, davajući mu RGB boju ili ton sive boje.

Također je bitno naglasiti da unutar OpenCV biblioteke, RGB spektar koji zadano ide redoslijedom „red, green, blue“, mijenja svoj redoslijed u BGR ili „blue, green, red“.

Mat klasa ima ugrađen konstruktor, konstruktor kopije kao i operator dodijele, uz dodatnu funkciju članicu **.copyTo(Mat)**, koja pravi duboku kopiju u Mat objekat predan kao parametar.

Za prikaz slike koja se nalazi unutar Mat objekta se koristi funkcija **imshow**. Funkcija prima dva parametra. Prvi parametar je string koji predstavlja ime prozora u kojem će se smjestiti slika. Prozori se dodjeljuju i otvaraju sami od sebe unutar konteksta OpenCV biblioteke prilikom pokretanja softvera u Visual Studio okruženju.

Drugi parametar je sam Mat objekat koji se želi prikazati u definisanom prozoru.

```
//declaring a Mat object
Mat myImage;
//filling a Mat object
myImage = imread("image_name.png", CV_LOAD_IMAGE_COLOR);
//displaying a Mat object
imshow("myWindowName", myImage);

//copy constructor
Mat myCopy(myImage);
//declaration
Mat secondCopy, thirdCopy;
//assignment operator
secondCopy=myImage;
//copyTo method
myImage.copyTo(thirdCopy);
```

Programski kod 3 Osnovne funkcionalnosti objekata tipa Mat

Kako bi kreirali softver za robota koji prati laserske projekcije, potrebno je da simuliramo čulo vida, koje se ogleda u videu, odnosno brzom smjenom slika, frame-ova ili Mat objekata jednim za drugim.

3.2.2.2. VideoCapture klasa

Do sada je bilo govora striktno o obradi slike ili jednog frame-a, te o korelaciji između slike i videa i kako se od niza slika dobija video.

Međutim, kako bi se kreirao softver koji će simulirati čulo vida, potrebno je obezbijediti učestalan i neprekidan tok videa.

U tu svrhu će biti korištena klasa VideoCapture, koja je ugrađena u OpenCV biblioteku. VideoCapture klasa je klasa namjenjena za snimanje video zapisa iz video datoteka, nizova slika ili kamera. Rad sa VideoCapture klasom počinje sa deklarisanjem objekta tipa te klase.

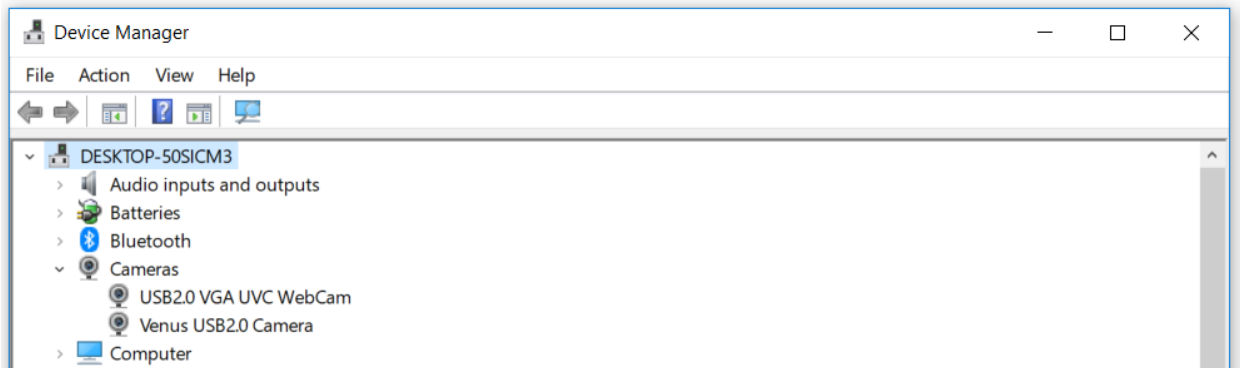
```
VideoCapture capture; //declare a VideoCapture object for further use
```

Programski kod 4 Deklaracija video capture objekta

Nakon deklaracije objekta, potrebno je uspostaviti konekciju ili otvoriti jednu od dostupnih kamera na računaru ili laptopu. Za razvoj ovog softvera će biti korišten laptop. Svaki laptop u sebi ima ugrađenu web kameru, koja se nalazi iznad ekrana mašine. Web kamera se na laptopima tretira kao prednja kamera. Bilo koja druga, eksterna kamera, priključana na laptop preko USB ili druge konekcije se tretira kao stražnja (engl. rear) kamera. Laptop sa više priključenih kamera ima više stražnjih kamera i uvijek jednu prednju web kameru.

OpenCV biblioteka informacije o dostupnim kamerama čuva u jednodimenzionalnom nizu čija indeksacija počinje od broja nula. Taj niz kreira direktno preuzimajući sve uređaje tipa kamera sa mašine. Kako se radi o jednodimenzionalnom nizu, redoslijed kamera registrovanih na laptop je veoma bitan.

Spisak i redoslijed svih dostupnih kamera se može pronaći unutar „Device Manager“ prozora računara (windows, control panel, device manager).



Slika 25 Spisak svih kamera mašine unutar "Device Manager"-a

Unutar taba „Cameras“ možemo pronaći sve registrovane kamere, pri čemu se prva kamera čuva u jednodimenzionalnom nizu na indeksu nula, druga na indeksu jedan itd. Generalno je web kamera laptopa uvijek na poziciji nula, jer predstavlja glavnu i osnovnu kameru svakog laptopa, koja se fizički ne može ukloniti. Ovo pravilo nije tačno u svakom slučaju, jer se web kamera može softverski ukloniti sa mašine.

U svrhu ovog rada će biti korištena eksterna, stražnja kamera, konektovana na laptop preko USB konekcije. Kameru je moguće vidjeti na spisku kamera kao „Venus USB2.0 Camera“, te zaključiti da se nalazi na drugoj poziciji spiska, odnosno na indeksu broj jedan.

Kako bismo spojili kameru sa VideoCapture objektom, potrebno je nad objektom pozvati funkciju članicu **.open(value)**, te joj kao parametar predati redni broj indeksa kamere sa kojom želimo ostvariti kontakt. Za stražnju „Venus“ kameru u ovom radu je to indeks broj jedan.

```
capture.open(1); //opening the rear camera, second on the list
```

Programski kod 5 Otvaranje stražnje kamere, druge na spisku

Nakon uspostavljanja konekcije i otvaranja kamere, dobra praksa je provjeriti da li je konekcija bila uspješna, te prekinuti dalje izvršenje programa u suprotnom. Provjeru možemo izvršiti funkcijom članicom **.isOpened()** nad VideoCapture objektom, koja kao rezultat vraća logičko tačno (engl. true) ili netačno (engl. false) u zavisnosti od toga da li je konekcija upješno uspostavljena.

```
// check if the connection succeeded, or terminate the program  
if(!capture.isOpened()) return -1;
```

Programski kod 6 Provjera uspostavljanja konekcije sa kamerom

Nakon uspješnog uspostavljanja konekcije i otvaranja videa sa kamere, također je dobra praksa ograničiti visinu i širinu videa, kako bi uvijek dobijali iste i konzistentne rezultate prilikom obrade videa. Postavljanje visine i širine VideoCapture objekta se postiže funkcijom članicom **.set(flag, value)**, koja prima dva parametra. Prvi parametar je OpenCV zastava koja definiše da li se radi o postavljanju visine ili širine videa, a drugi je broj koji predstavlja vrijednost same visine i širine. Visina i širina korištena u ovom radu će biti standardna 640x480 piksela.

```
capture.set(CV_CAP_PROP_FRAME_WIDTH, 640); //seting the width  
capture.set(CV_CAP_PROP_FRAME_HEIGHT, 480); //seting the height
```

Programski kod 7 Definisane visine i širine videa

Kao što je već bilo govora, svaki video uhvaćen digitalnom kamerom predstavlja niz slika ili frame-ova, koji se prikazuju jedan za drugim, po određenoj brzini i određenom vremenu. Kako bi se primijenila bilo kakva obrada videa, potrebno je uraditi dekompoziciju videa na njegove pojedinačne frame-ove, te obrađivati svaki frame zasebno prije njegovog prikaza na ekran.

Da bi se postiglo prikazivanje svakog frame-a jednog za drugim u kontinuitetu videa, potrebno je svaki frame prikazati kroz beskonačnu petlju, petlju koja će prikazivati svaki frame dok postoji dotok videa, ili korisnik ne prekine program.

Postoji više načina zapisa beskonačne pretlje u programskom jeziku C++, ali će u ovom radu biti korištena petlja **while(1){}**, koja će pokretati tijelo petlje, određeno velikim vitičastim zagradama, sve dok je broj jedan logički tačan. Kako je ta tvrdnja uvijek tačna, petlja će se izvršavati beskonačno.

Nakon otvaranja videa pomoću VideoCapture klase, potrebno je unutar petlje preuzeti svaki frame iz objekta tipa VideoCapture u jedan Mat objekat. Kako će se svakom iteracijom petlje učitavati nova, odnosno sljedeća slika, dobra praksa je glavni Mat objekat deklarirati prije početka petlje.

Učitavanje pojedinačnog framea iz VideoCapture objekta u Mat objekat se može vršiti na dva načina, sa dvije različite funkcije. Prva funkcija članica nad VideoCapture objektom, koja će biti korištena prilikom izrade ovoga rada, je **.read(Mat)**, koja će iz videa frame učitati u Mat objekat koji prima kao parametar.

Druga opcija je koristeći preklopni operator unosa (VideoCapture >> Mat), koji zahtjeva i učitava novi frame iz videa.

Kada je frame videa učitao u Mat objekat, predstavlja jednu sliku, nad kojom je moguće raditi sve operacije i koristiti sve funkcije koje podržava jedan Mat objekat.

Jedna od tih funkcionalnosti je ipis slike na ekran koristeći **imshow** funkciju, kao što je već obrađeno.

Dakle, nakon deklarisanja Mat objekta i započinjanja beskonačne petlje, prvi korak je preuzimanje frame-a iz videa, onda njegovo prikazivanje, te započinjanje nove iteracije petlje, koja uzima sljedeći frame, pa ga prikazuje na ekran, pa ponovno sljedeći frame na ekran itd.

Sa time smo postigli prikaz cijelog videa na ekran, kao i omogućili korištenje svih tehnika za obradu slike, na video, obrađujući svaku sliku pojedinačno prije prikazivanja.

Međutim, priroda beskonačne petlje je veoma brza i nepredvidiva, te je potrebno ugraditi mehanizam za prekid petlje kao i dovoljno vremena za prikaz svakog frame-a, kako je brzina iteracije petlje puno brža od preuzimanja i učitavanja frame-a videa. Kada bi se ovakva petlja pustila da radi, krajnji rezultat bi bila siva slika na monitoru, kako frame-ovi ne bi imali dovoljno vremena da se učitaju na ekran.

Zbog toga je neophodno obezbijediti pauzu između svake iteracije petlje i učitavanja novog frame-a, što se veoma lahko može postići funkcijom **waitKey(time)**.

Funkcija **waitKey** kao parametar prima vrijeme u milisekundama i uvijek će tačno toliko vremena čekati na neki input sa tastature od korisnika. Ako se taj input ne ostvari, nastaviti će se sljedeća iteracija petlje. Sa time se uspješno ostvarila pauza od definisanih milisekundi između započinjanja nove iteracije petlje, odnosno osvježavanje prikaza ekrana pod uslovom da korisnik nije unio ništa na tastaturi.

Također, funkcija **waitKey** očitava unos tipke sa tastature te kao rezultat vraća broj ili ASCII⁸ vrijednost slova koji je unesen, te ga može manipulirati u svrhu prekida petlje.

Ako primejra radi korisnik unese broj veći od nula, u roku određenih milisekundi, petlja se završava, a ako ne, vrijeme od tih milisekundi će se ispoštovati pauzom, te dati dovoljno vremena da se učitani frame prikaže na ekran, prije prelaska na sljedeći.

⁸ "American Standard Code for Information Interchange", standard za kodiranje znakova elektronske komunikacije.

Radi prirode i mehanizma waitKey funkcije, se ista najčešće piše na dnu petlje, nakon obrade i prikaza pojedinačnih frame-ova videa.

```
VideoCapture capture; //defining VideoCapture object
capture.open(1); //opening camera on index one
if (!capture.isOpened()) return -1; //checking the connection

//setting the height and the width of the capture frame
capture.set(CV_CAP_PROP_FRAME_WIDTH, FRAME_WIDTH);
capture.set(CV_CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT);

Mat cameraFeed; //Mat object to store each frame of the camera feed

while(1){ //starting the infinite loop
capture.read(cameraFeed); //store image to Mat object
imshow(windowName, cameraFeed); //displaying the current frame
waitKey(10); //delay of 10 milliseconds for the screen to refresh
}
```

Programski kod 8 Kompletni postupak prikazivanja videa na ekran pomoću OpenCV biblioteke

```
capture >> cameraFeed; // get a new frame from camera by input operator
```

Programski kod 9 Druga opcija preuzimanja frame-a: preklopni operator unosa

```
//break infinite loop if the user enters a number higher than zero
//wait 30 milliseconds for his input
if(waitKey(30) >= 0) break;
```

Programski kod 10 Terminiranje petlje koristeći unos korisnika i waitKey funkciju

Bitno je naglasiti da je jedan frame videa moguće kopirati, modifikovati i prikazati više puta i na više načina unutar petlje. Modifikacije jednog frame-a se mogu čuvati u drugim Mat objektima i u isto vrijeme prikazivati na ekran.

Unutar razvojnog okruženja Visual Studio se pokretanjem softvera, otvaraju i prikazuju prozori sami, za svaki frame, odnosno video.

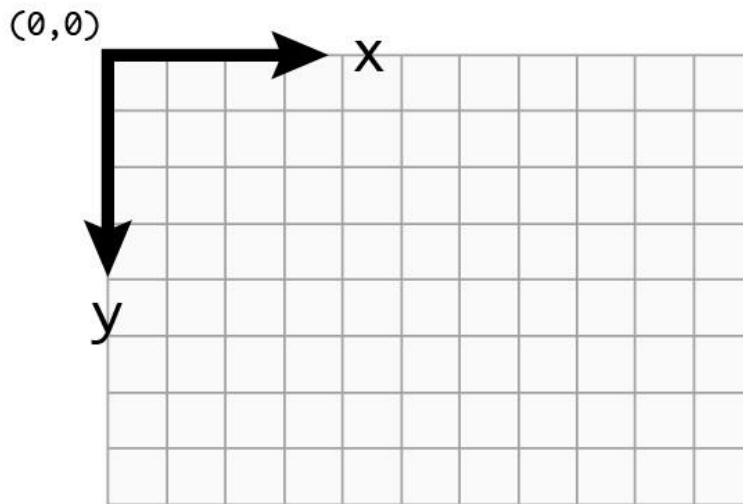
Modifikacija svakog frame-a će u zbiru modifikovati čitav video tehnikama obrađenim za modifikaciju slike odnosno frame-a.

Na kraju opisa VideoCapture klase, potrebno je istaknuti da klasa posjeduje vlastiti destruktork, pri čemu objekat tipa VideoCapture sam oslobađa svoju memoriju prilikom završenja programa.

3.2.2.3. Funkcije ilustriranja

OpenCV biblioteka posjeduje veliki broj funkcija koje za cilj imaju ilustrovati različite 2D objekte na definisanoj slici odnosno Mat objektu. Ove funkcije također služe za obradu slika ili pojedinačnih frame-ova videa, koja će korisniku bolje predložiti ponašanje i akcije robota. Za razumijevanje funkcija za ilustrovanje u OpenCV biblioteci, bitno je podsjetiti se da je jedna slika označena sa 2D koordinatnim sistemom, čije se početne koordinate nalaze u gornjem lijevom ćošku.

Svako pomjeranje u desnu stranu matrice povećava horizontalnu x koordinatu, dok svako pomjeranje prema dolje povećava vertikalnu y koordinatu. To znači da je svaki piksel unutar jednog frame-a definisan dvodimenzionalnom tačkom x i y koordinata.



Slika 26 Matrica slike određena dvodimenzionalnim koordinatnim sistemom

Svaki oblik nacrtan na dvodimenzionalnom koordinatnom sistemu je zapravo skup spojenih tačaka, koje u kombinaciji čine finalnu ilustraciju. Generalno je jedna tačka određena sa x i y koordinatom osnovni temelj za ilustrovanje u programiranju, kako može predstavljati početak i kraj linije, centar kruga, žarišta elipse vrhove kvadrata ili trokuta itd.

Zbog toga, OpenCV biblioteka dolazi sa ugrađenom klasom koja određuje tačku na ekranu, te se ova klasa koristi u konstrukciji drugih, kompleksnijih ilustracija. Klasa **Point(x,y)** kao parametre u svom konstruktoru prima x i y koordinatu, koje određuju poziciju određenog piksela na slici. Klasa dolazi sa svojim konstruktorom kopije, operatorom dodjele kao i destruktorom, što znači da sama dealocira memoriju iza sebe nakon korištenja.

Također je bitno napomenuti da „Point“ nije jedina klasa za rad sa tačkama, kako OpenCV biblioteka nudi rad sa 3D koordinatnim sistemom ili tačkama u imaginarnom sistemu, decimalnim vrijednostima i mnogim drugim. U svrhu izrade ovog rada će biti korištena početna i najosnovnija klasa Point u svrhu funkcija ilustrovanja na ekran u dvodimenzionalnom koordinatnom sistemu.











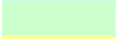









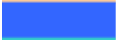







Također, koordinatama x i y je moguće pristupiti kao atributima nakon deklarisanja objekta tipa Point.

```
Point pt = Point(10, 8); //creating a point using the constructor

//creating a point by accessing x and y attributes
Point pt;
pt.x = 10;
pt.y = 8;
```

Programski kod 11 Kreiranje objekta tipa Point

Druga osnovna klasa koja se koristi u kombinaciji sa funkcijama za ilustrovanje je **Scalar(b,g,r)**. Svakom nacrtanom objektu je moguće odrediti boju unutar već obrađenog RGB spektra boja. Svaka boja se može kreirati miješanjem crvene, zelene i plave boje. Klasa skalar vraća tačno određenu boju koja predstavlja kombinaciju zadane tri vrijednosti intenziteta za crvenu, zelenu i plavu.

	RGB(128,0,128)		RGB(153,204,0)
	RGB(128,0,0)		RGB(255,204,0)
	RGB(0,128,128)		RGB(255,153,0)
	RGB(0,0,255)		RGB(255,102,0)
	RGB(0,204,255)		RGB(102,102,153)
	RGB(204,255,255)		RGB(150,150,150)
	RGB(204,255,204)		RGB(0,51,102)
	RGB(255,255,153)		RGB(51,153,102)
	RGB(153,204,255)		RGB(0,51,0)
	RGB(255,153,204)		RGB(51,51,0)
	RGB(204,153,255)		RGB(153,51,0)
	RGB(255,204,153)		RGB(153,51,102)
	RGB(51,102,255)		RGB(51,51,153)
	RGB(51,204,204)		RGB(51,51,51)

Slika 27 RGB vrijednosti za najčešće korištene boje

Međutim, unutar OpenCV biblioteke, RGB spektar ima neznanto drugačiji redoslijed BGR, te konstruktor klase Scalar kao parametre prima prvo vrijednosti za plavu, zelenu pa crvenu boju. Kombinacija vrijednosti, iako u drugačijem redoslijedu, vraća identične rezultate tj. boje. Maksimalna vrijednost za svaki boju iznosi 255, što predstavlja najjači intenzitet te boje, dok je najmanji nula.

```
Scalar(255, 0, 0); //blue
Scalar(0, 255, 0); //green
Scalar(0, 0, 255); //red
Scalar(255, 255, 255); //white
Scalar(0, 0, 0); //black
```

Programski kod 12 Primjer klase Scalar za različite boje

Nakon obrade osnovnih elemenata za korištenje funkcija ilustriranja, moguće je pisati konstrukcije koje crtaju kompleksnije objekte na ekran. Tri funkcije koje će biti korištene u ovom radu su funkcije za liniju, krug i tekst. OpenCV biblioteka nudi još mnoge funkcije za ilustriranje, sa čijim kombinacijama se mogu crtati znatno kompleksniji objekti.

Crtanje linije na ekran koristeći OpenCV biblioteku se postiže funkcijom **line**. Funkcija line je složena funkcija koja prima znatno više parametara. Prvi parametar je objekat tipa Mat, odnosno slika sačuvana u Mat objektu, na kojoj će se crtati linija. U kontekstu videa, svaki Mat objekat u petlji moramo provući kroz funkciju line, ako želimo ilustrovati liniju duž cijelog videa. Drugi parametar predstavlja objekat tipa Point, koji označava početnu tačku crtanja linije, dok treći parametar, također objekat tipa Point, određuje završnu tačku. Objekat tipa tačka, kao što je već rečeno, prima x i y koordinatu koje određuju poziciju tačno određenog piksela.

Osnovna konstrukcija je sa time nacrtana: ilustruj liniju na slici od tačke A do tačke B. Sljedeći parametar je objekat tipa Scalar, koji određuje boju linije, i peti parametar integer broj, koji određuje debljinu. Dodatni parametri su integer broj za tip linije (spojena, isprekidana), te integer broj za pomak (engl. shift), koji predstavlja broj frakcijskih bitova u koordinatama tačke.

Samo prva tri parametra (Mat objekat, početna i krajnja tačka) su obavezna, dok ostali parametri imaju podrazumijevane vrijednosti i mogu se izostaviti. Podrazumijevane vrijednosti su crna boja, spojena i popunjena linija, pomaka i debljine jedan.

```
//pseudo parameters
void line(Mat& img, Point pt1, Point pt2, const Scalar& color, int
thickness=1, int lineType=8, int shift=0);

//a white line from (0,0) to (25,25) on the 'img' image with thickens 1
line(img, Point(0,0), Point(25,25), Scalar(255,255,255), 1);
```

Programski kod 13 Parametri i primjer crtanja linije na ekran

Crtanje kruga koristeći OpenCV biblioteku se postiže funkcijom **circle**. Funkcija circle radi na isti način kao i funkcije line, sa razlikom što prva tačka na poziciji drugog parametra, predstavlja centar kruga, dok treći parametar više nije tačka nego integer broj, čija vrijednost definiše prečnik kruga. Podrazumijevani parametri funkcionišu na isti način.

```
//pseudo parameters
void circle(Mat& img, Point center, int radius, const Scalar& color, int
thickness=1, int lineType=8, int shift=0);

//a blue circle on the image 'img' of radius 20, with the center in
(20,20) and a thickness of 1
circle(img, Point(20,20), 20, Scalar(255,0,0), 1);
```

Programski kod 14 Parametri i primjer crtanja kruga na ekran

U određenim situacijama je potrebno na sliku ispisati tekst koji će olakšati praćenje programa i čula vida robota. Ispis teksta na ekran unutar OpenCV biblioteke se može postići korištenjem funkcije **putText**. PutText funkcija kao prvi parametar prima objekat tipa Mat, kao i funkcije opisane prije nje. Drugi parametar je objekat tipa string, odnosno sami tekst koji se želi ispisati na ekran, te parametar tipa Point koji određuje početnu poziciju teksta na ekranu.

Ostali parametri redom označavaju vrijednosti za font, veličinu, boju, debljinu, te tip linije. Font se može odrediti korištenjem OpenCV zastava, ili integer brojem koji zamjenjuje jednu od ovih zastava koristeći njihov redoslijed.

```
//pseudo parameters
void putText(Mat& img, const string& text, Point org, int fontFace,
double fontScale, Scalar color, int thickness=1, int lineType=8);

//Red text on 'img' image at position (0,0) saying 'Message' with
thickness 1, fontface 1 and fontscale 1
putText(img, "Message", Point(0,0), 2, 1, Scalar(0,0,255), 1);
```

Programski kod 15 Parametri i primjer crtanja teksta na ekran

Neke od dodatnih funkcija za crtanje⁹ unutar OpenCV biblioteke su elipsa, linija sa strelicom, pravougaonik itd. Funkcije ilustriranja će prvenstveno biti korištene za vizualiziranje granica i toka vještačkih misli koje će robot vidjeti i o kojima će razmišljati.

3.2.3. Definisanje vrijednosti prije praćenja laserske projekcije

Svako stanje modifikovane slike koju robot prima digitalnom kamerom služi za različito filtriranje objekta kojim se robot navodi. Kao što predhodna poglavlja pojašnjavaju, HSV slika filtrira objekte tačno određene boje u zavisnosti od HSV vrijednosti prilikom prevođenja u binarnu sliku, dok onda binarna slika filtrira stabilnost objekta, broj objekata određene boje na ekranu, te njenu minimalnu i maksimalnu veličinu koju će robot pratiti.

Pojednostavljeno, za praćenje laserske projekcije crvene boje, HSV slika filtrira sve crvene objekte prilikom pretvaranja u binarnu sliku, a zatim binarna slika određuje koji će se od tih crvenih objekata pratiti, po parametrima da li je objekat stabilan (da li je objekat zaista objekat na slici ili šum slike), da li zadovoljava specificiranu veličinu laserske projekcije (da li je crveni objekat prevelik ili premal da bi bio laser), te količinu specificiranih objekata (ponašanje robota ako se nađe više laserskih projekcija na slici i koliko projekcija može maksimalno registrovati dok ne zaključi da je riječ o šumu slike) i ako je laser pronađen i zadovoljio sve uslove, da ga robot prati.

Sve ove vrijednosti je potrebno definisati i prilagoditi za crvenu lasersku projekciju. Također, bitno je naglasiti da video i slike mogu varirati u zavisnosti od osvjetljenja, sjena, kvaliteta kamere, šuma i drugih parametara, te da će se u ovom radu nastojati minimizirati vanjske

⁹ https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html#puttext

nepogode i faktori koji utiču na kvalitet praćenja lasera, simulacijom što boljeg perfektnog okruženja, uz već definisane i obrađene tehnike (median filter, morfološka erozija, dilatacija itd.).

3.2.3.1. HSV vrijednosti za binarnu sliku

Kako bi se postiglo praćenje laserske projekcije, potrebno je prvo modifikovati RGB sliku kamere, u sliku koja se nalazi u HSV spektru. Nakon toga je potrebno HSV sliku modifikovati u binarnu sliku. HSV vrijednosti prilikom prevođenja određuju koju će boju robot vidjeti, odnosno objekte koje boje će zapravo pratiti unutar krajnje binarne slike.

Određivanje koja boja će biti primarna kada se proslijedi u binarnu sliku se postiže modifikovanjem minimalnih i maksimalnih vrijednosti za nijansu, zasićenje i vrijednost svjetline.

Svaka boja ima domet nijanse, zasićenja i vrijednosti svjetline te modifikovanjem maksimalnih i minimalnih vrijednosti se sužava spektar boja koji pripadaju tačno tom dometu. Što je domet manji, odnosno razlika između maksimuma i minimuma svake komponente manja, to je prikaz i filter tačno određene boje precizniji.

Prilikom prevođenja HSV slike u binarnu sliku, minimalne i maksimalne vrijednosti pojedinačnih HSV komponenti diktiraju koja će boja biti prevedena u bijelo ili logičke jedinice, a koja u crno ili logičke nule, unutar binarne slike.

Ako se HSV vrijednosti prilikom prevođenja modifikuju za crvenu boju, crveni objekti na binarnoj slici će biti popunjenim bijelim pikselima, dok će sve ostalo biti ugašeno crnom bojom.

Krajnja minimalna i maksimalna vrijednost HSV spektra je od 1 do 255, što znači da se granice mogu postaviti u inetrvalu od 0 do 256 za minimalnu i maksimalnu vrijednost svake HSV komponente.

```
//initial min and max HSV filter values
int H_MIN = 0;
int H_MAX = 256;
int S_MIN = 0;
int S_MAX = 256;
int V_MIN = 0;
int V_MAX = 256;
```

Programski kod 16 Inicijalne minimalne i maksimalne vrijednosti za HSV filter u binarnu sliku

Za lasersku projekciju jake crvene sa primjesama bijele boje svjetla je potrebno sve HSV vrijednosti ostaviti na samom minimumu i maksimumu osim vrijednosti maksimalnog zasićenja (S_MAX) koje je neophodno postaviti što niže moguće i u ovom radu će biti spušten na 10.

Međutim, u svrhu kalibriranja i stvaranja što optimalnijeg okruženja, kao i u svrhu demonstracije filtriranja različitih boja, će se implementirati dinamičko mijenjanje svih minimalnih i maksimalnih HSV vrijednosti. U prevodu, sve vrijednosti će inicijalno biti postavljene na minimum i maksimum, te nakon pokretanja softvera i početka registrovanja videa, će se za vrijeme izvršavanja programa moći u stvarnom vremenu mijenjati sve

vrijednosti. Takvo rješenje će se postići implementacijom klizača, koji će se prikazati u zasebnom prozoru prilikom pokretanja programa i koji će u stvarnom vremenu modifikovati sve dostupne HSV vrijednosti.

Nakon definisanja inicijalnih vrijednosti, sljedeća funkcija će kreirati klizače koji će ih moći modifikovati:

```
//naming the trackbars window
const string trackbarWindowName = "Trackbars";
//create window for trackbars
void createTrackbars() {

    namedWindow(trackbarWindowName, 0);

    char TrackbarName[50];
    sprintf_s(TrackbarName, "H_MIN");
    sprintf_s(TrackbarName, "H_MAX");
    sprintf_s(TrackbarName, "S_MIN");
    sprintf_s(TrackbarName, "S_MAX");
    sprintf_s(TrackbarName, "V_MIN");
    sprintf_s(TrackbarName, "V_MAX");

    createTrackbar("H_MIN", trackbarWindowName, &H_MIN, H_MAX, on_trackbar);
    createTrackbar("H_MAX", trackbarWindowName, &H_MAX, H_MAX, on_trackbar);
    createTrackbar("S_MIN", trackbarWindowName, &S_MIN, S_MAX, on_trackbar);
    createTrackbar("S_MAX", trackbarWindowName, &S_MAX, S_MAX, on_trackbar);
    createTrackbar("V_MIN", trackbarWindowName, &V_MIN, V_MAX, on_trackbar);
    createTrackbar("V_MAX", trackbarWindowName, &V_MAX, V_MAX, on_trackbar);
}
```

Programski kod 17 Funkcija za kreiranje klizača

Funkcija **namedWindow()** prima dva parametra, objekat tipa string koji označava ime prozora i OpenCV zastavu, ili njen redni broj, koji definiše vrstu prozora (normal ili resizable).

Zatim se u buffer niz učitavaju imena svih varijabli za modifikovanje, koristeći funkciju **sprintf_s()**, koja prima buffer i objekat tipa string. Nakon toga se, koristeći OpenCV funkciju **createTrackbar()**, kreiraju klizači na zadanom prozoru, čije pomjeranje dinamički mijenja vrijednosti unesjenih varijabli.

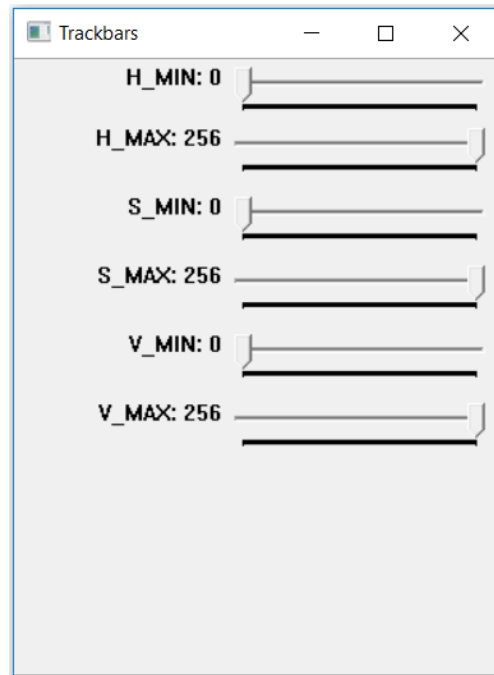
Dodatno, kao posljedni parametar **createTrackbar** funkcije, se proslijeđuje funkcija, koja se poziva svaki put kada se klizač pomjeri. Ova funkcija je ručno pisna i u implementaciji ovog rada prazna, ali se može iskoristiti za buduće promjene i modifikacije.

Pozivom ove funkcije bilo gdje u kodu se, prilikom pokretanja programa, sam otvara prozor imena „Trackbars“, sa ugrađenim klizačima.

```
//create slider bars for HSV filtering
createTrackbars();
```

Programski kod 18 Poziv funkcije za kreiranje klizača

Pomjeranjem ovih klizača se dinamički mijenjanju vrijednosti zadanih varijabli po klizaču odgovornom za tu varijablu. Prozor sa klizačima će biti pozvan prije petlje koja će da registruje video, te prilikom pretvaranja HSV slike u binarnu će klizači diktirati koje su to minimalne i maksimalne vrijednosti HSV komponenti, odnosno koja će se boja prikazivati i biti primarna na binarnoj slici.



Slika 28 Prikaz prozora sa klizačima minimalnih i maksimalnih vrijednosti

Ponovno, za lasersku projekciju crvene sa primjesama bijele boje, je potrebno maksimalnu zasićenost (S_MAX) posaviti na 10.

Implementacija klizača nije obavezna za realizaciju softvera, služi za demonstraciju i eksperimentisanje filtriranja različitih boja, te kao osigurač pri nemogućnosti robota da uz zadane minimalne i maksimalne HSV vrijednosti detektuje određenu boju, zbog promjene okruženja, svjetlosti ili šuma slike, pa uz pomoć klizača može u stvarnom vremenu riješiti taj problem, modifikovanjem vrijednosti.

U približno perfektnom okruženju koje će se nastojati postići u daljoj implementaciji rada, je moguće fiksno ostaviti maksimalnu zasićenost (S_MAX) na 10, te izostaviti implementaciju klizača.

Međutim, klizači su ostavljeni u radu ne samo radi demonstracije, nego i kao potencijalni pokazatelj budućih ideja, planova i kocepata kreiranog projekta, o kojem će biti govora u budućim poglavljima rada.

3.2.3.2. Konstante laserske projekcije za binarnu sliku

Nakon što se HSV slika konvertovala u binarnu sliku sa konkretnim vrijednostima minimuma i maksimuma HSV komponenti, će binarna slika na osnovu tih vrijednosti prikazivati objekte tražene boje bijelom bojom, dok će sve ostalo isključiti i prikazati crnu boju. U ovom stanju je bitno binarnu sliku što više optimizovati, koristeći tehnike za smanjenja šuma slike, kako bi objekti bili što preciznije prikazani na slici. Primjenjivanje tehnika za smanjenje šuma na binarnu sliku unutar koda će biti detaljnije pojašnjeno naknadno.

Trenutno je potrebno definisati koliko tačno objekata na binarnoj slici se može tretirati kao pojedinačni objekat, a preko koliko objekata se slika i dalje smatra pod šumom. Ako se na slici nalazi mnogo bijelih objekata, većinom je u pitanju šum slike.

Pojednostavljeno, ako binarna slika prikazuje preko npr. pet crvenih objekata, slika se smatra pod šumom i potrebno je smanjiti broj objekata. Ako slika prikazuje deset laserskih projekcija, a definisano je da je maksimum objekata na slici pet, slika će se tretirati pod šumom i neće započeti proces praćenja.

Naravno da binarna slika treba registrovati samo jednu lasersku projekciju, što će biti riješeno kasnije, ali je za početak potrebno dovesti sliku u takvo stanje da se sa sigurnošću može reći da slika precizno prikazuje do pet objekata, koji mogu biti laserske projekcije ili drugi objekti crvene boje.

```
//max number of objects to be detected in frame  
const int MAX_NUM_OBJECTS = 5;
```

Programski kod 19 Definisane broja maksimalnih objekata na ekranu

Nakon definisanja broja objekata kojih jedan frame može prikazivati bez da se slika smatra pod šumom i rješavanja pitanja boje, potrebno je definisati šta zapravo predstavlja jedan objekat, odnosno definisati njegove dimenzije. Idealno se želi postići da binarna slika registruje samo objekte laserske projekcije crvene boje. U prevodu, tih pet objekata moraju biti tačno određenih dimenzija, koje će biti definisane. Minimalna površina objekta potrebna, da bi se objekat smatrao laserskom projekcijom je površina od 10*10 piksela. Sve manje od 10*10 ili ukupno 100 piksela površine se ne smatra laserskom projekcijom.

```
//minimum object area  
const int MIN_OBJECT_AREA = 10 * 10;
```

Programski kod 20 Definisane minimalne površine objekta

Kako bi definisali maksimalnu dozvoljenu površinu laserske projekcije, potrebno je prvo fiksno definisati dimenzije slike Kao što je već rečeno, slika će biti standardnih dimenzija od 640*480 piksela.

```
//default capture width and height  
const int FRAME_WIDTH = 640;  
const int FRAME_HEIGHT = 480;
```

Programski kod 21 Dimenzije prikazne slike

Dakle, visina puta širina slike je zapravo površina cijele slike, te će se ta površina dijeliti sa 100, kako bi definisali makismlanu dozvoljenu površinu laserske projekcije. Svaki objekat veći od ove površine se ne smatra laserskom projekcijom.

```
//maximum object area  
const int MAX_OBJECT_AREA = FRAME_HEIGHT * FRAME_WIDTH / 100;
```

Programski kod 22 Definisane maksimalne površine objekta

Zaključak ovog poglavlja je da nakon filtriranja svih crvenih objekata na binarnu sliku, želimo prikazati do pet laserskih projekcija. Laserska projekcija je onaj objekat čija površina spada u dozvoljeni spektar veličine projekcije. Prikaz više objekata od dozvoljenog će sliku staviti u stanje šuma, te se algoritam za praćenje neće ni pokretati. Ako je nađeno ispod pet objekata, ali oni nisu laserske projekcije, algoritam također neće pratiti ništa dalje. Samo ako pronađe do pet validnih laserskih projekcija, algoritam za praćenje ima cilj, te ga je moguće pokrenuti.

Ovaj proces predstavlja kalibraciju i podešavanje ponašanja binarne slike prije praćenja. Ako uspješno prođu svi filteri do pet laserskih projekcija, softver će tražiti da se uklone sve projekcije sem jedne, koju će krajnji robot pratiti.

Dvije ili do pet laserskih projekcija i algoritam zahtijeva uklanjanje dodatnih projekcija, dok preko pet laserskih projekcija stavlja sliku u stanje šuma. Ako se na binarnoj slici i dalje nalazi više od jedne ispravne projekcije, ili projekcija naknadno uđe u kadar vida robota, robot će se zaustaviti i zahtijevati otklanjanje drugih projekcija.

Međutim, ako se pri praćenju detektuje drugi crveni objekat koji nije laserska projekcija, odnosno ako je površinom veći ili manji od zadanih vrijednosti, robot će ili ignorisati taj objekat ako je dovoljno mal ili staviti sliku pod stanje šuma ako objekat zauzima previše prostora, te zahtjevati dalju kalibraciju.

Zbog toga je bitno naglasiti da, iako robot posjeduje mehanizme odbrane za svaki scenarij i nastoji binarnu sliku, koja predstavlja primarni izvor vida robota, optimizovati erozijom i dilatacijom, da su podloga i okruženje robota veoma bitne. Finalni produkt ovog rada je namijenjen da se koristi u skoro perfektnim uslovima, jednoboje podloge, konstantnog osvijetljena i bez dodatnih objekata koji bi mogli ometati robota prilikom praćenja projekcije. Ali zbog nivoa greške svakog projekta i softvera su ugrađeni ovi mehanizmi odbrane u svrhu izbjegavanja većih problema ili katastrofa.

3.2.3.3. Definisane i upravljanje prozorima

U svrhu lakšeg praćenja i demonstracije različitih slika i polja vida robota, kao i načina na koji robot vidi i prati lasersku projekciju, će biti prikazana tri prozora: za čisti video sa kamere, video konvertovan u HSV spektar, kao i video konvertovan u binarnu sliku. Dodatno će biti pokrenuti klizači u svrhu mijenjanja minimalnih i maskimalnih HSV vrijednosti po potrebi u zasebnom prozoru.

Prozori se prikazuju pomoću već obrađene funkcije imshow, koja prima jedinstven naziv prozora, kao i sliku odnosno objekat tipa Mat koji se prikazuje u tom prozoru.

Radi skalabilnosti i bilo kakvih izmjena, funkcije prozora, vrste prozora ili daljeg modifikovanja softvera u budućnosti, će se jedinstvena imena prozora čuvati u zasebnim konstantama, kao objekti tipa string.

```
//names that will appear at the top of each window
const string windowName = "Original image";
const string HSVWindowName = "HSV image";
const string binaryWindowName = "Binary image";
const string trackbarWindowName = "Trackbars";
```

Programski kod 23 Konstante imena svih prozora

Kroz dalju implementaciju algoritma za praćenje laserske projekcije će biti korišteni nazivi gore definisanih prozora.

3.2.4. Praćenje laserske projekcije

Predhodna poglavlja služe kao uvod i priprema za pisanje softvera koji će pratiti lasersku projekciju pomoću eksterne USB kamere, detaljno prikazivati simulirano čulo vida budućeg robota i sve koristeći OpenCV biblioteku. Sve tehnike i principi koji su do sada objašnjeni će biti implementirani redom koristeći konkretne funkcije i algoritme za praćenje projekcije.

Radi lakšeg razumijevanja i demonstracije koda, su uvedene dvije boolean zastave, čije se vrijednosti mogu mijenjati prije pokretanja softvera, a koje označavaju da li će se na binarnu sliku primjeniti matematička morfologija, te da li je potrebno započeti algoritam praćenja laserske projekcije. Ako se ove zastave postave na „true“, softver će primjeniti morfologiju kako bi otklonio i ublažio šum slike, te započeti algoritam koji prati projekciju. Zastave su ostavljene u kodu radi demonstracije prikaza videa u obliku prije morfologije, ili prije početka praćenja.

```
//flags for tracking and modification
bool trackObjects = true;
bool useMorphOps = true;
```

Programski kod 24 Zastave za korištenje morfologije i praćenja projekcije

Prvi korak predstavlja deklarisanje svih varijabli koje će se koristiti prilikom izvršenja programa. Potrebno je deklarirati ukupno tri objekta tipa Mat, odnosno rezervirati tri matrice koje će čuvati različite slike. Te slike, kao i imena varijabli će biti rezervirane za prirodnu RGB sliku koju robot prima sa kamere, HSV sliku koja će se dobiti iz RGB prirodne slike, te binarnu sliku nastalu modifikacijom minimalnih i maksimalnih HSV vrijednosti.

```
//Matrix to store each frame of the webcam feed  
Mat cameraFeed;  
  
//matrix storage for HSV image  
Mat HSV;  
  
//matrix storage for binary threshold image  
Mat binary;
```

Programski kod 25 Deklaracija svih Mat objekata za projekat

Glavni cilj praćenja laserske projekcije je zapravo odrediti x i y koordinatu pozicije, gdje se laserska projekcija nalazi na slici. Na osnovu x i y koordinata praćene projekcije će robot znati kuda i na koji način da se kreće. Pronaći precizne x i y vrijednosti laserske projekcije na slici je zapravo najveći izazov ovog projekta. Zbog toga je u startu potrebno deklarirati i rezervirati varijable za čuvanje x i y koordinate pronađenja projekcije, koje će se kasnije slati robotu u svrhu odlučivanja u kojem pravcu i kojom brzinom se treba kretati. Inicijalne vrijednosti koordinata će biti postavljene na (0,0), odnosno u gornjem lijevom ćošku prozora.

```
//x and y values for the location of the object  
int x = 0, y = 0;
```

Programski kod 26 Deklarisanje i inicijalizacija varijabli za čuvanje vrijednosti koordinata laserske projekcije

Nakon toga je u projektu potrebno deklarirati objekat tipa VideoCapture, te otvoriti konekciju sa eksternom kamerom. Nakon provjere da je konekcija uspješna, potrebno je postaviti visinu i širinu prozora te započeti beskonačnu petlju unutar koje će se u objekat tipa Mat naziva „cameraFeed“ učitavati svaki frame input videa. Dakle cameraFeed varijabla će sadržavati prirodnu RGB sliku sa kamere.

Sva logika, funkcije i postupci ovog koraka su detaljno objašnjeni u predhodnim poglavljima, te se zasnivaju na znanju OpenCV biblioteke, njenih klasa i funkcija.

Također, ako se u implementaciji softvera koriste klizači, potrebno ih je pokrenuti i pozvati prije početka beskonačne petlje.

Dalja procedura se vrši unutar beskonačne petlje, te će koristiti konkretne funkcije za konverziju i smanjenje šuma slike, koje su do sada bili obrađeni.

```
//create slider bars for HSV filtering
createTrackbars();

//video capture object to acquire webcam feed
VideoCapture capture;
//open capture object at location
capture.open(1);

// check if the connection succeeded, or terminate the program
if (!capture.isOpened()) return -1;

//set height and width of capture frame
capture.set(CV_CAP_PROP_FRAME_WIDTH, FRAME_WIDTH);
capture.set(CV_CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT);

//start an infinite loop where webcam feed is copied to cameraFeed matrix
while (1) {
    //store image to matrix
    capture.read(cameraFeed);
```

Programski kod 27 Postupak deklarisanja i započinjanja video prenosa

Nakon učitavanja svake prirodne RGB slike videa u Mat objekat, je potrebno konvertovati taj frame iz RGB u HSV spektr. U svrhu konverzije se koristi OpenCV funkcija **cvtColor(Mat izvor, Mat destinacija, flag)**, koja prima tri parametra. Prvi parametar predstavlja objekat tipa Mat koji se želi konvertovati, dok je drugi parametar Mat objekat u koji se želi učitati novonastala slika. U ovom slučaju je prvi parametar „cameraFeed“ objekat, a destinacijski je „HSV“ matrica. Međutim, kako bi funkcija znala u koji spektar da konvertuje početnu sliku, potrebno je kao treći parametar funkcije proslijediti OpenCV zastavu koja određuje željeni spektar konverzije, u ovom slučaju **COLOR_BGR2HSV** zastavu, koja konvertuje frame specifično iz BGR u HSV spektar.

```
//convert frame from BGR to HSV colorspace
cvtColor(cameraFeed, HSV, COLOR_BGR2HSV);
```

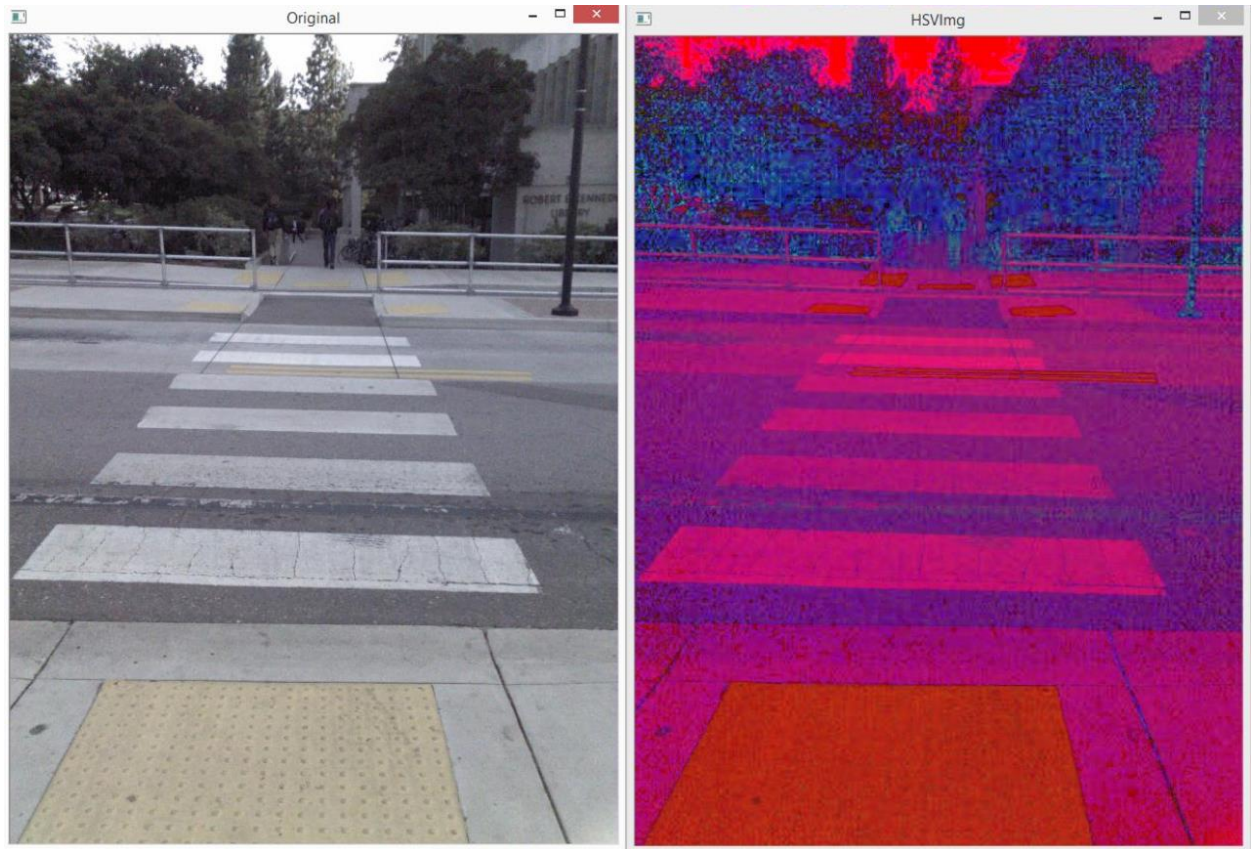
Programski kod 28 Konvertovanje BGR u HSV sliku koristeći cvtColor() funkciju

Nakon konverzije i kreiranja HSV slike, će se pokrenuti prvi korak u smanjenju šuma slike, gdje će se novonastali HSV frame provući kroz median filter, koji je detaljno objašnjen u prethodnim poglavljima. OpenCV biblioteka posjeduje **medianBlur(Mat izvor, Mat destinacija, int veličina)** funkciju, koja također prima tri parametra: izvornu sliku, destinacijsku sliku i veličinu otvora filtera. U ovom kontekstu je izvorna i destinacijska slika identična, kako se samo ona želi provući kroz filter ali i dalje sačuvati u tom istom Mat objektu.

```
//add median blur to HSV image
medianBlur(HSV, HSV, 3);
```

Programski kod 29 Median filter nad HSV slikom

Nakon konverzije, se RGB slika unutar cameraFeed objekta ne mijenja, rezultat modifikacije se upisuje unutar HSV Mat objekta. Ispis HSV slike na ekran, pomoću funkcije imshow, bi prikazao originlanu RGB sliku u HSV spektru boja.



Slika 29 Prikaz BGR slike u HSV spektru

Nakon prve konverzije i koraka smanjenja šuma, je potrebno HSV sliku konvertovati u binarnu, kako će se finalno praćenje odvijati upravo na njoj, prateći logičke jedinice ili bijeli prostor laserske projekcije. OpenCV posjeduje funkciju koja radi na modifikovanju HSV slike, uz definisane minimalne i maksimalne HSV vrijednosti kako bi je konvertovala u binarnu sliku. Proces se naziva pronalaženje praga (engl. thresholding) slike.

Funkcija **inRange(Mat, Scalar, Scalar, Mat)** prima četiri parametra. Prvi parametar je ponovno izvorna, u ovom konetkstu HSV slika, dok je posljednji parametar destinacijska slika, objekat tipa Mat naziva „binary“, koji je deklarisan prije petlje, kao i svi drugi Mat objekti. Drugi i treći parametar funkcije predstavljaju skalarnu kombinaciju posebno minimalnih i maksimalnih vrijednosti svake HSV komponente pojedinačno.

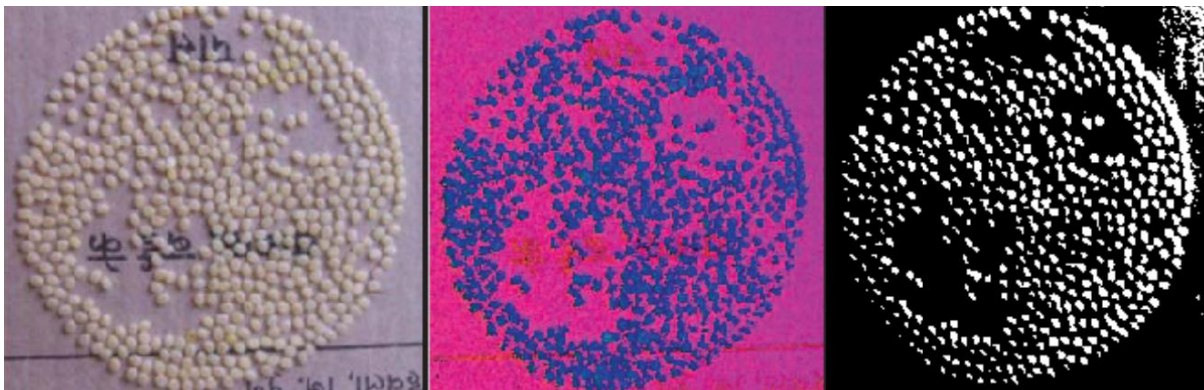
Ove vrijednosti su definisane na početku projekta i po želji se mogu modifikovati ukoliko se koristi implementacija projekta sa klizačima.

Vrijednosti za crvenu, sa primjesama bijele boje laserske projekcije, kao i detaljan opis Scalar funkcije su postavljeni i detaljno objašnjeni u poglavlju definisanja vrijednosti prije praćenja laserske projekcije.

```
//filter HSV image between values and store filtered image to binary matrix  
inRange(HSV, Scalar(H_MIN, S_MIN, V_MIN), Scalar(H_MAX, S_MAX, V_MAX),  
binary);
```

Programski kod 30 Konvertovanje HSV slike u binarnu sliku koristeći inRange() funkciju

Ponovno, nakon konverzije, originalna slika unutar cameraFeed Mat objekta, kao i HSV slika unutar HSV Mat objekta ostaju nepromijenjene, modifikacija se učitava u Mat objekat „binary“, tako da nakon ove konverzije postoje tri različite slike, ili frame-a videa, koje je moguće pojedinačno i zasebno prikazati na ekran pomoću imshow funkcije.



Slika 30 Konverzija od BGR, preko HSV do binarne slike

Nakon posljednje konverzije slike, potrebno je uraditi i drugi korak smanjenja šuma sada binarne slike, koristeći matematičku eroziju i dilataciju. U ovom projektu će zastava za morfološke operacije biti postavljena na true, te će se uvijek pozivati posebno pisana funkcija, koja kao parametar prima referencu na binarnu sliku, te je provlači kroz morfološke operacije.

```
//perform morphological operations on binary image to eliminate noise  
//and emphasize the filtered object  
if (useMorphOps) morphOps(binary);
```

Programski kod 31 Poziv funkcije za morfološke operacije nad binarnom slikom

Prvi korak pri provođenju morfoloških operacija nad binarnom slikom jeste definisati morfološki objekat za smanjenje šuma na slici. Morfološki objekti ili elementi se čuvaju u objektima tipa Mat, a mogu se kreirati OpenCV funkcijom **getStructuringElement(object_flag, size)**. Kako je riječ o cirkularnoj laserkoj projekciji, koju je sa vremena na vrijeme moguće dovesti u stanje elipse, element odabran za morfološke operacije je elipsa. Svaki objekat, uključujući elipsu, ima svoju OpenCV zastavu, koja se predaje kao prvi parametar u funkciju za kreiranje objekta.

Drugi parametar predstavlja veličinu objekta, definisanu preko površine u pikselima, kortseći funkciju **Size(int piksel, int piksel)**. Kako prilikom morfološke dilatacije želimo postići jasnu vidljivost laserske projekcije, će objekat specifično za dilataciju biti površinski veći.

```
void morphOps(Mat& binary) {  
  
    //create structuring element that will be used to "dilate" and "erode"  
    image.  
  
    Mat erodeElement = getStructuringElement(MORPH_ELLIPSE, Size(3, 3));  
  
    //dilate with larger element to make sure object is nicely visible  
    Mat dilateElement = getStructuringElement(MORPH_ELLIPSE, Size(8, 8));
```

Programski kod 32 Funkcija morfoloških operacija: kreiranje morfoloških objekata za eroziju i dilataciju

Nakon određivanja morfoloških objekata i definisanja da se erozija i dilatacija ograničavaju na elipse određenih površina, koje se podudaraju sa približnom površinom jedne laserse projekcije po pikselima, je potrebno konkretno sprovesti te operacije nad binarnom slikom. Biblioteka nudi ugrađene funkcije za eroziju i dilataciju, koje primaju po tri parametra: izvornu sliku, destinacijsku sliku, te morfološki element ili objekat. Izvor i destinacija će biti ista binarna slika, kako želimo samo nju modifikovati. Radi boljih i preciznijih rezultata se preporučuje provesti sliku dva puta kroz eroziju i dilataciju, kako bi se uklonilo što više šuma sa slike.

```
erode(binary, binary, erodeElement);  
erode(binary, binary, erodeElement);
```

Programski kod 33 Funkcija morfoloških operacija: erozija binarne slike koristeći erode() funkciju



Slika 31 Prikaz binarne slike prije i poslije morfološke erozije

Redoslijed je također veoma bitan, gdje se slika prvo provlači kroz dva sloja erozije, te zatim kroz dva sloja dilatacije.

Erozija smanjuje nedefinisan bijeli prostor, te ostavlja samo čiste i definisane objekte, dok dilatacija povećava i popunjava sve preostale objekte bijele boje, koji su precizno definisani kao objekti od strane erozije, koristeći elemente elipse zadane površine kao vodič.

```
dilate(binary, binary, dilateElement);  
dilate(binary, binary, dilateElement);  
}
```

Programski kod 34 Funkcija morfoloških operacija: dilatacija binarne slike koristeći dilate() funkciju



Slika 32 Prikaz binarne slike iz stanja erozije lijevo, do stanja dilatacije desno

Sa ispravno postavljenim minimalnim i maksimalnim HSV vrijednostima za lasersku projekciju, prihvatljivim okruženjem, kvalitetom kamere i definisanim vrijednostima za projekciju će softver uz ove konverzije i filtriranja na binarnoj slici prikazivati samo lasersku projekciju tačno određene boje i dimenzija, dok će sve ostalo ignorisati, uz već objašnjene mehanizme odbrane.

Postignuto je da budući robot vidi samo lasersku projekciju u obliku bijelih ili upaljenih piksela na binarnoj slici. Međutim, nije dovoljno da softver i robot vide projekciju, potrebno je da u svakom trenutku znaju gdje se projekcija nalazi na ekranu, odnosno koje su joj x i y koordinate na slici. Pomoću tih koordinata će kasnije robot donositi odluke o tome da li da se kreće, na koji način da se kreće kao i kojom brzinom.

Prirodno, sljedeći korak predstavlja pronalaženje x i y vrijednosti koordinata praćene projekcije sa binarne slike.

Nazad u beskonačnoj petlji, nakon završetka primjene morfoloških operacija nad referencom binarne slike, je potrebno pozvati posebno pisanu funkciju za praćenje projekcije. Ponovno, zastava koja omogućava poziv ove funkcije će unutar projekta biti inicijalno postavljena na true, te će se za svaki frame input videa pozivati funkcija koja kalkuliše koordinate i određuje poziciju laserske projekcije za praćenje.

Funkcija za sada prima četiri parametra. Prvi i drugi paramater predstavljaju x i y koordinate, koje u startu imaju vrijednosti (0,0), te će se popuniti aktuelnim vrijednostima koordinata za svaki frame koji bude proslijeđen funkciji za praćenje projekcije. Treći parametar predstavlja

obrađenu binarnu sliku, na osnovu koje će se kalkulisati pozicija projekcije, te se koordinate čuvati u proslijeđenim x i y koordinatama.

Četvrti parametar je čista RGB slika sačuvana u „cameraFeed“ Mat objektu, na kojem će kasnije biti izvršene modifikacije za lakše praćenje i bolje razumijevanje vida robota. X i y koordinate, kao i RGB input slika se u funkciji primaju po referenci, kako će se njihov inicijalni sadržaj (vrijednosti koordinata, kao i izgled slike) mijenjati.

```
//pass in modified binary frame to the object tracking function  
//this function will return the x and y coordinates of the  
//filtered laser projection  
if (trackObjects) trackFilteredObject(x, y, binary, cameraFeed);
```

Programski kod 35 Poziv funkcije za praćenje lasera nad modifikovanom binarnom slikom

Kako će se nad binarnom slikom vršiti određene modifikacije i testiranja, prvi korak predstavlja napraviti duboku kopiju binarne slike, unutar zasebne varijable tipa Mat. U ovu pomoćnu (engl. temporary) Mat varijablu će se kopirati binarna slika koristeći funkciju članicu **copyTo(Mat)**, koja je u detalje obrađena u predhodnim poglavljima. Nakon kopiranja binarne slike, potrebno je deklarirati dva vektora: vektor kontura i vektor hijerarhije.

```
//copying the binary image to a temporary Mat object for modification  
Mat temp;  
binary.copyTo(temp);
```

Programski kod 36 Kopiranje binarne slike u pomoćni objekat

Vektor kontura predstavlja vektor vektora 2D tačaka. Pojednostavljeno, radi se o 2D matrici u kojoj je svaki pojedinačni element tačka sa svojim x i y koordinatama. Ovaj vektor će u sebi sadržavati sve tačke na granici pronađenog objekta, odnosno vrijednosti kontura objekta. Način pronalaska kontura slijedi u nastavku.

Vektor hijerarhije predstavlja vektor **Vec4i** objekata. Vec4i objekat je definisan u OpenCV biblioteci i predstavlja vektor sa četiri dimenzije, gdje je svaka dimenzija sačuvana u integer broju. Ovaj vektor od tačno četiri elementa sadrži tačke x1, y1, x2, y2, gdje prve dvije koordinate označavaju početnu tačku linije a zadnje dvije završnu tačku linije, koja se prostire od početka do kraja površine nekog objekta. Jedan Vec4i objekat predstavlja jedan objekat pronađen na binarnoj slici. Vektor hijerarhije je prema tome vektor pronađenih objekata, odnosno matrica Vec4i objekata u kojem je svaka kolona jedna od koordinata početne i zadnje tačke linije koja se prostire širom objekta. Veličina vektora hijerarhije (**.size()**) ujedno predstavlja broj objekata pronađenih na binarnoj slici.

Na istom indeksu vektora kontura i vektora hijerarhije će se čuvati podaci u jedom objektu pronađenom na slici. Objekat pronađen na indeksu nula unutar vektora hijerarhije, svoje konture može naći u vektoru kontura na istom indeksu nula.

Za popunjavanje vektora kontura kao i vektora hijerarhije se koristi funkcija **findContours(Mat, countours, hierarchy, flag mode, flag method)**, koja je ugrađena u OpenCV biblioteku. Funkcija kao prvi parametar prima Mat objekat, u ovom kontekstu pomoćnu varijablu modifikovane binarne slike, te vektore kontura i hijerarhije za popunjavanje kao drugi i treći parametar.

Kao četvrti parametar funkcija prima OpenCV zastavu koja označava način (engl. mode) pronalaska kontura, te će se u ovom kontekstu koristiti zastava **CV_RETR_CCOMP**. CV_RETR_CCOMP zastava za način pronalaska kontura, vraća sve konture i organizuje ih u hijerarhiju na dva nivoa. Na prvom, gornjem, nivou se nalaze vanjske granice komponente, dok se na drugom, donjem, nivou nalaze unutrašnje granice otvora.

Peti parametar predstavlja OpenCV zastavu koja označava metodu pronalaska kontura. Zastava za metodu korištena u ovom projektu je **CV_CHAIN_APPROX_SIMPLE** koja kompresuje horizontalne, verikalne i dijagonalne segmente i ostavlja samo njihove krajnje tačke.

Ove dvije zastave predstavljaju idealnu kombinaciju za pronalazak kontura laserske projekcije, mada je moguće kombinovati i koristiti druge zastave po različitim potrebama objekta kojeg se želi locirati na binarnoj slici.

Kao posljednji objekat i koncept za razumjeti, prije konkretnog kalkulisanja x i y koordinata i praćenja objekta je koncept trenutaka ili momenata (engl. moments) slike. U suštini, momenti slike kalkulišu matematičke površine i težine objekata pronađenih na slici, te razčlanjuju njihove sličnosti odnosno razlike. Unutar OpenCV biblioteke je ugrađen **Moments** tip podatka, koji u sebi sadrži sve dostupne pojedinačne momente slike. Momenti će biti korišteni u kombinaciji sa konturama, kako bi se izračunale x i y vrijednosti koordinata. Jedan momenat će se kreirati za svaku konturu traženog objekta, te ako je objekat prošao sve uslove laserske projekcije, će izračunati x i y koordinatu centra objekta. Kako objekat tipa Moments sadrži konstruktor, kao i attribute za m00, m10 i m01 pojedinačni momenat, koji predstavljaju površine objekta zadanog konturama po x i y osi, se koordinate mogu matematički izračunati po formuli $x = m_{10}/m_{00}$ i $y = m_{01}/m_{00}$.

$$\bar{x} = \frac{M_{10}}{M_{00}} \quad \bar{y} = \frac{M_{01}}{M_{00}}$$

Slika 33 Formule za izračunavanje x i y koordinata koristeći momente objekta na slici, određenih kontura

Također, potrebno je čuvati referencu na svaku površinu objekta vraćenu momentom u posebnoj varijabli, u slučaju da se slika opstruira sa minimalnim šumom, kako bi najveća laserka projekcija, odnosno objekat na binarnoj slici, uvijek bio praćen što preciznije. Dodatno će se definisati i varijabla u kojoj će se čuvati informacija da li je objekat uspješno pronađen.

```
void trackFilteredObject(int& x, int& y, Mat binary, Mat& cameraFeed) {  
    //two vectors needed for output of findContours  
    vector<vector<Point>> contours;  
    vector<Vec4i> hierarchy;  
  
    //find contours of filtered image using openCV findContours function  
    findContours(temp, contours, hierarchy, CV_RETR_CCOMP,  
    CV_CHAIN_APPROX_SIMPLE);  
  
    //use moments method to find our filtered object  
    double refArea = 0;  
    bool objectFound = false;
```

Programski kod 37 Deklarisanje potrebnih vektora, varijabli i pronalazak objekata i kontura objekata na binarnoj slici

Nakon definisanja svih vrijednosti i pronalaska svakog objekta prikazanog na binarnoj slici, zajedno sa njegovim konturama, se može započeti algoritam za računanje x i y koordinata.

Ako se unutar hijerarhijskog vektora nalazi preko nula objekata, odnosno ako je minimalno jedan objekat pronađen, preuzima se vrijednost veličine hijerarhijskog vektora, koja predstavlja broj pronađenih objekata, pomoću funkcije `size()`. U suprotnom nijedan objekat nije pronađen na binarnoj slici, te se korektna poruka može ispisati na ekran originalne cameraFeed slike koristeći funkciju `putText()`. Kako je originalna slika primljena u funkciju po referenci, će se na slici unutar petlje, iz koje se pozvana funkcija za praćenje, pojaviti svaka poruka nacrtana na sliku unutar funkcije. Nakon toga se uvodi prvi mehanizam odbrane robota, definisan u prethodnim poglavljima. Ako je pronađen više od jednog objekta na slici, odnosno više laserskih projekcija, funkcija se terminira uz odgovarajuću poruku. U suprotnom, ako se broj pronađenih objekata nalazi ispod definisanog broja, prolazi se petljom kroz svaku tačku `Vec4i` objekta, elementa hijerarhijskog vektora. Kako se konture objekta nalaze na istom indeksu kao i sami objekat unutar hijerarhijskog vektora, se definiše momenat nad tim konturama i poziciji objekta. Za kreiranje momenta se koristi funkcija **`moments(Mat)`**, kojoj se predaje kontura na tom indeksu, konvertovana u Mat objekat. Površina pronađenog objekta je `m00` nađenog momenta nad konturama objekta.

Posljednji korak predstavlja izračunati x i y koordinate objekta koristeći momenat koji se nalazi na njemu. Međutim, ovdje se objekat, odnosno laserska projekcija, provlači kroz sve ostale filtere, uslove i mehanizme odbrane. Ako je površina u pikselima laserske projekcije manja od definisane minimalne površine ili veća od definisane maksimalne površine, onda projekcija i dalje nije pronađena, jer ne ispunjava uslove. Također, ako je površina projekcije manja od referentne površine u bilo kojem trenutku, objekat nije dovoljno precizan da bi zadovoljio sve uslove.

U suprotnom je laserska projekcija uspješno pronađena, te joj se računaju x i y koordinate pomoću momenata njenih kontura, varijabla za praćenje da li se objekat uspješno pronašao se postavlja na „true“, te referentna površina postaje površina uspješno pronađene projekcije. U svim ostalim slučajevima se na ekran crta odgovarajuća poruka za prevelik šum slike.

U momentu kada je laserska projekcija pronađena i kada se izračunaju koordinate pozicije, se poziva funkcija za crtanje okvira oko projekcije, koja također prikazuje koordinate lasera na ekranu.

```
if (hierarchy.size() > 0) {
    int numObjects = hierarchy.size();

    if (numObjects > 1) {
        putText(cameraFeed, "Please remove other projections", Point(0, 350), 2,
            1, Scalar(0, 0, 255), 1);
        return;
    }

    //if number of objects greater than MAX_NUM_OBJECTS we have a noisy
    filter
    if (numObjects < MAX_NUM_OBJECTS) {

        for (int index = 0; index < numObjects; index = hierarchy[index][0]) {

            Moments moment = moments((cv::Mat) contours[index]);
            double area = moment.m00;

            if (area > MIN_OBJECT_AREA && area < MAX_OBJECT_AREA && area > refArea) {
                x = moment.m10 / area;
                y = moment.m01 / area;
                objectFound = true;
                refArea = area;
            }
            else objectFound = false;
        }

        if (objectFound) {
            drawObject(x, y, cameraFeed);
        }
        else putText(cameraFeed, "Too much noise", Point(0, 350), 2, 1, Scalar(0,
            0, 255), 1);
        }
        else {
            putText(cameraFeed, "No laser detected", Point(0, 350), 2, 1, Scalar(0,
            0, 255), 1);
        }
    }
}
```

Programski kod 38 Algoritam za pronalaženje i praćenje laserske projekcije

Funkcija **drawObject(int x, int y, Mat cameraFeed)** prima nađenu x i y koordinatu, kao i referencu na frame originalne input slike. Funkcija na lokaciji x i y crta krug sa nišanima preko pronađene laserske projekcije koja se nalazi na tim koordinatama, koristeći funkcije za ilustrovanje OpenCV biblioteke. Također, ispisuje x i y koordinate ispod nišana kako bi korisnik mogao pratiti njihovo mijenjanje u pravom vremenu. Funkcija dodatno koristi pomoćnu funkciju za pretvaranje integer brojeva u string, kako bi mogla ispisati trenutne koordinate na ekran koristeći **putText()** funkciju.

```
string intToString(int number) {  
    std::stringstream ss;  
    ss << number;  
    return ss.str();  
}
```

Programski kod 39 Funkcija za konvertovanje integer broja u string

```
void drawObject(int x, int y, Mat& frame) {  
  
    //drawing crosshairs on the image  
  
    circle(frame, Point(x, y), 20, Scalar(0, 255, 0), 2);  
    if (y - 25 > 0)  
        line(frame, Point(x, y), Point(x, y - 25), Scalar(0, 255, 0), 2);  
    else line(frame, Point(x, y), Point(x, 0), Scalar(0, 255, 0), 2);  
    if (y + 25 < FRAME_HEIGHT)  
        line(frame, Point(x, y), Point(x, y + 25), Scalar(0, 255, 0), 2);  
    else line(frame, Point(x, y), Point(x, FRAME_HEIGHT), Scalar(0, 255, 0),  
        2);  
    if (x - 25 > 0)  
        line(frame, Point(x, y), Point(x - 25, y), Scalar(0, 255, 0), 2);  
    else line(frame, Point(x, y), Point(0, y), Scalar(0, 255, 0), 2);  
    if (x + 25 < FRAME_WIDTH)  
        line(frame, Point(x, y), Point(x + 25, y), Scalar(0, 255, 0), 2);  
    else line(frame, Point(x, y), Point(FRAME_WIDTH, y), Scalar(0, 255, 0),  
        2);  
  
    putText(frame, intToString(x) + ", " + intToString(y), Point(x, y + 30),  
        1, 1, Scalar(0, 255, 0), 2);  
}
```

Programski kod 40 Funkcija za crtanje nišana i ispis vrijednosti koordinata na sliku

Kako se se ovaj proces lociranja, validiranja i određivanja laserske projekcije, poziva za svaki frame input videa, x i y koordinate kao i pozicija nišana će se mijenjati i pomjerati u stvarnom vremenu, kako se laserska projekcija ili robot bude pomjerao. Sa ovim smo uspješno postigli da budući robot posjeduje čulo vida koje vidi samo jedn lasersku projekciju i nju uvijek prati „pogledom“. Naknadno će biti objašnjeno ponašanje robota u zavisnosti od toga gdje se laserska projekcija nalazi na ekranu.



Slika 34 Prikaz originalne slike sa nišanom i koordinatama, HSV slike, te binarne slike sa laserskom projekcijom

Krajnje, nazad u petlji, nakon poziva funkcije za praćenje objekta, se sve kreirane i modificirane slike otvaraju u prozorima i njihov sadržaj se prikazuje korisniku. Početna originalna cameraFeed slika će biti modificirana nišanom ako je projekcija pronađena, koji će uvijek da prati projekciju i ispisuje pronađene x i y koordinate. Dodatno će se pozvati waitKey() funkcija koja daje vremena da se sljedeća slika videa učitava na vrijeme.

```
//show frames
imshow(binaryWindowName, binary);
imshow(windowName, cameraFeed);
imshow(HSVWindowName, HSV);

//delay 10ms so that screen can refresh.
waitKey(10);
```

Programski kod 41 Prikaz svih prozora

Nakon pronalazaženja x i y koordinata je potrebno definisati kako robot da se ponaša u zavisnosti od pozicije projekcije. Njegovo ponašanje se svodi na to da li će se kretati, kojom brzinom, kao i u koju stranu će skretati, odnosno kojim intenzitetom će se budući točkovi vrtjeti. Unutar implementacije softvera će sve ove varijable biti izračunate prije prelaska na implementaciju hardvera.



Slika 35 Detaljniji prikaz nišana i koordinata na originalnoj slici

3.2.5. Pozicija laserske projekcije

Nakon uspješnog kalkulisanja x i y koordinata smo postigli praćenje laserske projekcije, gdje god se ona nalazila na slici, u stvarnom vremenu. Međutim, same koordinate nemaju dalju značajnost ako se nad njima ne uvede referentna pozadina koja će diktirati značenje te pozicije koordinata, odnosno šta robot da radi i kako da se ponaša u zavisnosti od pozicije laserske projekcije na slici.

Zbog toga će se početna RGB slika podijeliti prvo u četiri zasebna kvadranta duž vertikalne y ose. U zavisnosti od toga u kojem se od ova četiri kvadranta projekcija nalazi, će robot prikazivati različita ponašanja. Prvi kvadrant se nalazi na y osi od 0 do 120 piksela i nalazi se pri samom vrhu slike: od vrha do 120-og piksela po y osi slike. Drugi kvadrant se nastavlja od 120 do 220 piksela, treći od 220 do 320 piksela i posljednji, četvrti kvadrant, je definisan od 320 do 480 piksela, odnosno od 320-og piksela po vertikalnoj y osi do kraja slike koja je definisana sa 480 piksela.

Od ova četiri kvadranta, posljednji kvadrant, odnosno prvi sa donje strane slike, predstavlja zonu u kojoj se robot nalazi u ispravnoj poziciji. Odnosno ako se laserska projekcija nalazi na dnu ekrana, između 320 i 480 piksela, robot je na ispravnoj poziciji i ne mora se kretati dalje, te će se zaustaviti. Ako se projekcija nalazi iznad donjeg, posljednjeg kvadranta, u bilo kojem od gornja tri kvadranta, robot se mora kretati naprijed, kako bi projekciju spustio u najniži, posljednji kvadrant.

Na horizontalnoj x osi je potrebno pronaći centar slike. Kako je širina slike definisana sa 640 piksela, sredina se nalazi tačno na 320-om pikselu ekrana. Od vrha slike je potrebno povući liniju do početka posljednjeg kvadranta na dnu slike. Ova vertikalna linija predstavlja vodič koji pokazuje li se robot treba kretati desno ili lijevo. Ako je projekcija sa desne strane centra slike robot se mora kretati udesno i obrnuto. Tako da, u zavisnosti od laserske projekcije, uz kretanje naprijed, robot se također u isto vrijeme može kretati ulijevo ili udesno, povećavajući brzinu u određenom točku.

Međutim, samo jedna vertikalna linija, nije površinski dovoljna da bi se laserska projekcija, veće površine, ikada nalazila tačno na njoj, odnosno horizontalno tačno na sredini slike. Zbog toga je potrebno lijevo i desno od glavne vertikalne linije dodati dvije pomoćne linije, koje u zbiru između sebe čine prostor dovoljno širok, da se za laserku projekciju može reći da se nalazi horizontalno na sredini. Ove pomoćne linije se nalaze na 30 piksela udaljenosti od glavne vertikalne linije, sa lijeve i desne strane, dakle na 290 lijevo i 350 piksela desno, sa 320-im pikselom u sredini. Sada, ako se projekcija nalazi više desno od krajnje desne pomoćne linije, robot se mora kretati udesno, kako bi pratio projekciju, odnosno okretati lijevi točak jače. Dakle, ako se laserska projekcija nalazi iznad donjeg kvadranta, robot se kreće naprijed i ako nije u sredini, odnosno lijevo ili desno od pomoćnih centralnih linija, robot se kreće lijevo ili desno, dok projekcija ne dođe u posljednji kvadrant, gdje se robot zaustavlja. Ako se pri kretanju robot nađe ponovno u sredini horizontalno, nastavlja se kretati linearno pravo, dok projekcija ne dođe u posljednji kvadrant.

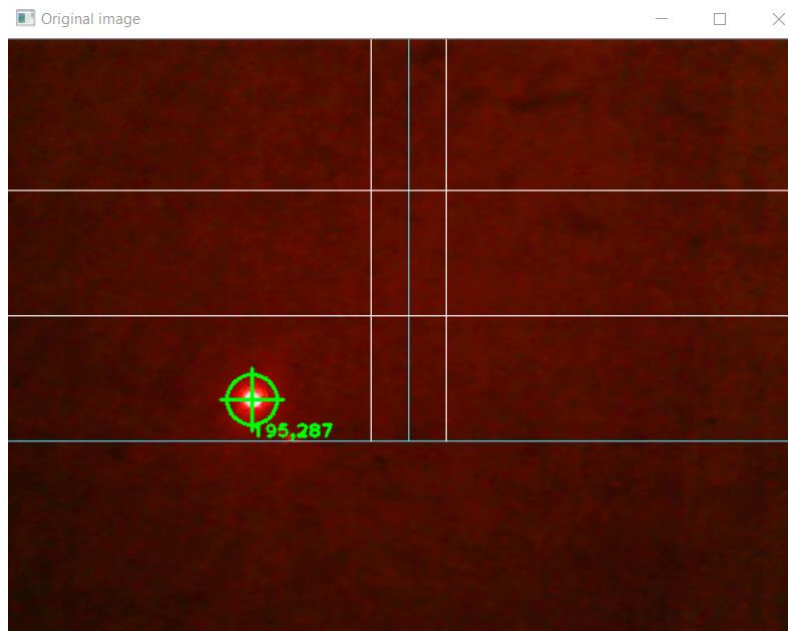
Kako bi vizuelno realizovali ove smjernice, potrebno ih je na samom početku beskonačne petlje nacrtati na originalni RGB frame, koristeći OpenCV funkcije za ilustrovanje.

```
//draw grid
//vertical guides
line(cameraFeed, Point(FRAME_WIDTH / 2, 0), Point(FRAME_WIDTH / 2,
FRAME_HEIGHT - (FRAME_HEIGHT / 3)), Scalar(243, 226, 68), 1);
line(cameraFeed, Point(FRAME_WIDTH / 2 - 30, 0), Point(FRAME_WIDTH / 2 - 30,
FRAME_HEIGHT - (FRAME_HEIGHT / 3)), Scalar(255, 255, 255), 1);
line(cameraFeed, Point(FRAME_WIDTH / 2 + 30, 0), Point(FRAME_WIDTH / 2 +
30, FRAME_HEIGHT - (FRAME_HEIGHT / 3)), Scalar(255, 255, 255), 1);

//horizontal guides
line(cameraFeed, Point(0, FRAME_HEIGHT - (FRAME_HEIGHT / 3)),
Point(FRAME_WIDTH, FRAME_HEIGHT - (FRAME_HEIGHT / 3)), Scalar(243, 226,
68), 1);
line(cameraFeed, Point(0, 120), Point(FRAME_WIDTH, 120), Scalar(255, 255,
255), 1);
line(cameraFeed, Point(0, 220), Point(FRAME_WIDTH, 220), Scalar(255, 255,
255), 1);
```

Programski kod 42 Crtanje smjernica za poziciju laserke projekcije

Kako se u funkciju praćenja i crtanja nišana originalna slika prosljeđuje po referenci, će smjernice biti pristune uvijek na originalnoj input slici, pa time i input videu.



Slika 36 Prikaz smjernica za definisanje pozicije projekcije

3.2.6. Određivanje brzine kretanja

Ako se laserska projekcija nalazi u donjem, posljednjem kvadrantu, robot treba da se zaustavi, dok, ako se nalazi u bilo kojem od gornja tri kvadranta, se treba kretati unaprijed. Međutim, isto ponašanje bi se moglo postići sa samo dva kvadranta, gornjim i donjim, gdje robot na jednom staje, a na drugom se kreće. Razlog zašto su uvedena tri dodatna kvadranta je simuliranje različitih brzina kretanja.

Željeni efekat je simulirati ljudsko ponašanje pri donošenju odluke za zaustavljanje hoda. Naime, prilikom kretanja, što se tijelo više primiće željenoj destinaciji ili objektu, to se hod postepeno usporava, dok tijelo finalno ne dođe do stanja mirovanja. Pojednostavljeno, ako se želi pomjeriti za jedan metar, nije potrebna velika brzina i napor, kao kada se što prije želi pomjeriti na lokaciju udaljenu deset metara. U kontekstu projekta je željeni objekat kojem se želi pristupiti laserska projekcija, a tri kvadranta simuliraju udaljenost projekcije, odnosno brzinu kretanja prema projekciji.

Definisane su ukupno tri brzine kretanja robota: brzo kretanje unaprijed, umjereno kretanje unaprijed i sporo kretanje unaprijed. Ako se laserska projekcija nalazi u prvom, gornjem kvadrantu, robot se kreće najbržom definisanom brzinom unaprijed. Kada se projekcija spusti u drugi, srednji kvadrant, brzina kretanja robota unaprijed se smanjuje na umjerenu i finalno, kada projekcija pređe granicu trećeg kvadranta, kretanje prelazi u sporo kretanje, sve dok projekcija ne dođe u četvrti kvadrant, gdje se robot zaustavlja. Na putu, ako projekcija izađe van vertikalnih smjernica, robot će se istom brzinom kretati ulijevo ili udesno, u zavisnosti od toga u kojem se kvadrantu nalazi projekcija. Same numeričke vrijednosti brzine kretanja će biti definisane i prikazane u hardverskoj implementaciji projekta.

Ovom implementacijom se postiglo da se robot, kada ugleda lasersku projekciju u prvom, njemu najdaljem kvadrantu, počinje kretati unaprijed, prema projekciji, visokom brzinom, te što joj se više približava, odnosno se laserska projekcija spušta niz kvadrante slike, to robot usporava dok ne dođe do prirodnog zaustavljanja.

Unutar funkcije praćenja, nakon što se laserska projekcija uspješno otkrila i nakon što su se definisale vrijednosti njene x i y koordinate, će na ekran biti ispisane poruke u zavisnosti od toga da li je robot u ispravnoj poziciji ili se treba kretati unaprijed, ulijevo ili udesno i kojom brzinom, sve u zavisnosti od pozicije laserske projekcije.

Pomoću y koordinate projekcije se može definisati u kojem se vertikalnom kvadrantu projekcija nalazi, te kojom brzinom se robot treba kretati unaprijed. Pomoću x koordinate se može definisati da li se projekcija nalazi horizontalno u sredini, ili ulijevo ili udesno od pomoćnih linija, te da li je potrebno da robot skreće lijevo ili desno. Sve odluke se donose u zavisnosti od pozicije trenutnog x i y piksela laserske projekcije u odnosu na ostale piksele slike i videa, čije su granice kvadranta definisane u predhodnom poglavlju, koristeći globalne vrijednosti za visinu i širinu ekrana.

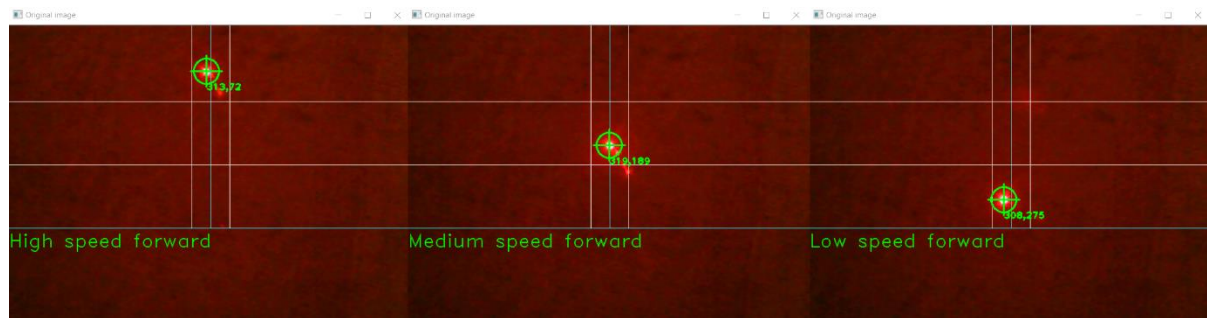
Poruke će se crtati na originalnu sliku unutar funkcije za praćenje projekcije, nakon što je objekat uspješno pronađen, te se preko njega nacrtao nišan. Ponovno će se koristiti funkcije za ilustrovanje iz OpenCV biblioteke.

Za svaku novu vrijednost x i y koordinate laserske projekcije se prvo provjerava u kojem se vertikalnom kvadrantu nalazi, kako bi se ispisala poruka o brzini kretanja, te se provjerava, da li je potrebno skretati ulijevo ili udesno.

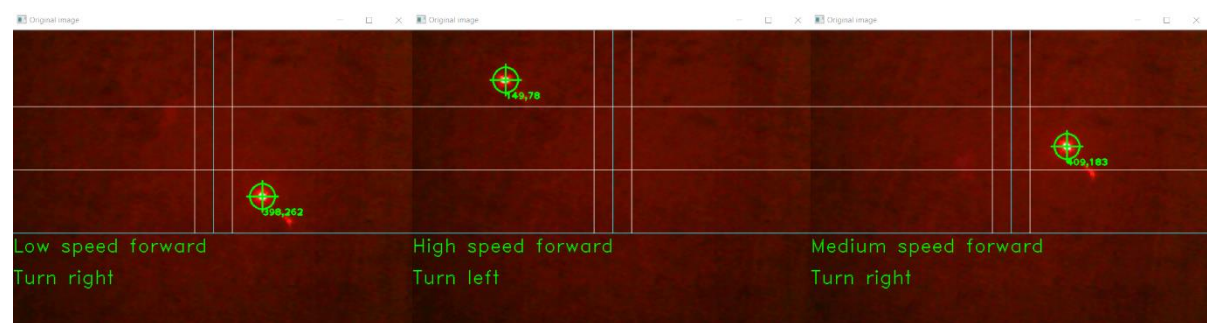
Ako i kada projekcija dođe u ispravnu poziciju, donji kvadrant gdje se robot zaustavlja, će se na ekran također ispisati odgovarajuća poruka.

```
if (objectFound) {
drawObject(x, y, cameraFeed);
if (y < 120) {
putText(cameraFeed, "High speed forward", Point(0, 350), 2, 1, Scalar(0,
255, 0), 1);
if (x < (FRAME_WIDTH / 2) - 30) {
putText(cameraFeed, "Turn left", Point(0, 400), 2, 1, Scalar(0, 255, 0),
1);
}
if (x > (FRAME_WIDTH / 2) + 30) {
putText(cameraFeed, "Turn right", Point(0, 400), 2, 1, Scalar(0, 255, 0),
1);
}
}
else if (y > 120 && y < 220) {
putText(cameraFeed, "Medium speed forward", Point(0, 350), 2, 1,
Scalar(0, 255, 0), 1);
if (x < (FRAME_WIDTH / 2) - 30) {
putText(cameraFeed, "Turn left", Point(0, 400), 2, 1, Scalar(0, 255, 0),
1);
}
if (x > (FRAME_WIDTH / 2) + 30) {
putText(cameraFeed, "Turn right", Point(0, 400), 2, 1, Scalar(0, 255, 0),
1);
}
}
else if (y > 220 && y < 320) {
putText(cameraFeed, "Low speed forward", Point(0, 350), 2, 1, Scalar(0,
255, 0), 1);
if (x < (FRAME_WIDTH / 2) - 30) {
putText(cameraFeed, "Turn left", Point(0, 400), 2, 1, Scalar(0, 255, 0),
1);
}
if (x > (FRAME_WIDTH / 2) + 30) {
putText(cameraFeed, "Turn right", Point(0, 400), 2, 1, Scalar(0, 255, 0),
1);
}
}
else {
putText(cameraFeed, "Robot in position", Point(0, 350), 2, 1, Scalar(243,
226, 68), 1);
}...
}
```

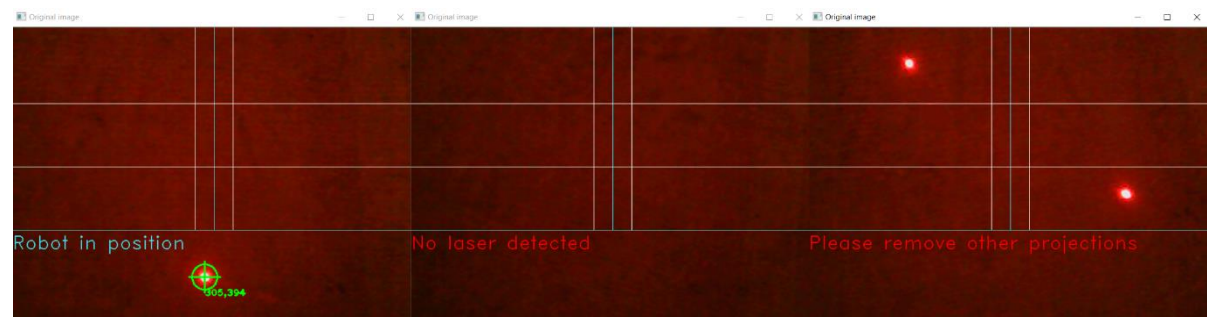
Poruke za kretanje unaprijed određenom brzinom, te skretanje ulijevo ili udesno se na ekranu mogu nalaziti u isto vrijeme, ili pojedinačno samo za brzinu ako se laserska projekcija nalazi na horizontalnom centru. Također, u slučaju da algoritam pronađe više od jedne validne laserske projekcije, će se na ovom mjestu ispisati poruka na ekran koja upozorava da se uklone dodatne projekcije, a algoritam za praćenje će se privremeno zaustaviti.



Slika 37 Prikaz odnosa brzine kretanja unaprijed i pozicije laserske projekcije



Slika 38 Prikaz odnosa brzine kretanja unaprijed i pozicije laserske projekcije uz skretanje ulijevo ili udesno



Slika 39 Prikaz stanja i poruka prilikom ispravne pozicije projekcije, nepostojanja projekcije i pronalaženja više od jedne validne projekcije

Nakon pronalaženja x i y koordinata, kao i definisanja načina ponašanja robota u odnosu na lokaciju laserske prijekcije, je potrebno ove informacije prenijeti hardverskim komponentama projekta.

3.3. Značenje, otvaranje i komunikacija serijskog porta

Sve informacije potrebne da bi budući robot znao kuda i na koji način da se kreće su do ovog poglavlja uspješno kreirane. Potrebno je ove informacije prenijeti hardverskim komponentama. Hardverska komponenta odgovorna za fizičko kretanje robota je Arduino Mega. Međutim, za sada je bitno samo znanje o postojanju Arduina, koji će pomoću USB konekcije biti spojen na računar, kako će rad i funkcija Arduina biti znatno detaljnije objašnjena u poglavlju implementacije hardvera. Za sada je dovoljno saznanje da je Arduino priključen na mašinu i da će kod pisan pomoću OpenCV biblioteke unutar CLI Visual Studio projekta, komunicirati sa Arduinoom koristeći serijsku vezu.

U računarskoj tehnologiji, serijski port ili ulaz je fizički, serijski interfejs za komunikaciju, kroz koji se informacije prenose, bilo da ulaze ili izlaze, tačno jedan bit po bit [5]. Kroz historiju personalnih računara su serijski portovi konektovali računare sa drugim uređajima kao što su terminali ili razne periferije. Za mnoge računarske periferije je USB interfejs zamijenio serijski port, ali upravo interfejsi kao što su Ethernet, FireWire ili USB šalju i prenose podatke kao serijski tok podataka. Iako skoro svaki server posjeduje serijski port, mnoge moderne radne stanice, računari ili laptopi ga ne posjeduju nikako. Međutim, u svrhu ovog projekta će se serijska komunikacija sa Arduinoom ostvariti upravo pomoću USB interfejsa.

Unutar Visual Studio CLI projekta je moguće otvoriti, slati i primati podatke serijskom vezom. Generisanjem CLI projekta unutar Visual Studia se generišu sve pod biblioteke potrebne za otvaranje veze, kao i sva neophodna sintaksa i tipovi podataka sa novim .NET standardom. Pomenuti posebni tipovi podataka unutar ovog projekta predstavljaju CLR objekte, koji imaju svoju posebnu sintaksu i način rada. Prije svega, da bi se mogli koristiti CLR objekti, potrebno je na početku projekta uključiti neophodne namespace-ove.

```
using namespace System;  
using namespace System::IO::Ports;
```

Programski kod 44 Namespaces za potrebne CLR objekte

Karakteristika CLR objekata je postojanje posebnog tipa pokazivača nad njima (oznaka „^“ umjesto “*”). Pokazivač koji sadrži adresu CLR objekta se alocira koristeći komandu **gcnew** (umjesto samo **new**). Svaki CLR objekat alociran gcnew komandom je podležan sakupljačima smeća (engl. garbage collector), što znači da se memorija alocirana nad ovim tipom pokazivača ne mora ručno dealocirati, nego će biti automatski uništena kada svaki pokazivač prestane sadržavati adresu tog objekta. Prvi CLR objekat potreban za dalju implementaciju softvera je **SerialPort**. SerialPort objekat u svom konstruktoru prima dva parametra. Prvi parametar je string koji predstavlja naziv **COM porta** (engl. communication port).

COM ili komunikacijski port, je originalni, ali i danas često korišteni naziv za interfejs serijskog porta na personalnim računarima. Može se odnositi ne samo na fizičke portove, kao što je to slučaj u ovom projektu, nego i na emulirane portove stvorene Bluetooth ili USB-to-serial adapterima. String parametar koji očekuje COM port, se formira u zavisnosti od toga na koji USB port je priključen Arduino na računar. Tačan naziv porta se, u zavisnosti verzije operativnog sistema, može pronaći u **Device Manager-u**. U ovom kontekstu je specifični COM port mašine označen imenom COM10.

Drugi parametar konstruktora je integer broj koji predstavlja brzinu prenosa (engl. baud rate). Brzina prenosa označava brzinu kojom se informacije prenose u komunikacijskom kanalu. U kontekstu serijskog porta, „9600 baud“ znači da je serijski ulaz sposoban prenijeti maksimalno 9600 bita po sekundi. Nakon deklarisanja i inicijalizovanja serijskog porta u kodu, je potrebno pozvati funkciju članicu **open()**, koja finalno otvara komunikaciju sa arduinom na tom portu.

```
//open serial arduino connection
SerialPort^ port = gcnew SerialPort("COM10", 9600);
port->Open();
cout << "Serial open" << endl;
```

Programski kod 45 Otvaranje porta za serijsku komunikaciju sa Arduinoom

Nakon kreiranja port objekta na vrhu main funkcije, ga je potrebno unutar beskonačne petlje, proslijediti kao dodatni parametar u funkciju za praćenje objekta. Cilj je svaki put kada se ispravno odrede x i y koordinate objekta, te koordinate poslati preko serijskog porta prema Arduino, u string formatu. Dalje će Arduino očitavati pronađene x i y koordinate iz primljene string naredbe, te slati konkretne naredbe motoru robota. Kako će se ovo dešavati konstantno u beskonačnoj petlji, sve dok laserska projekcija nije u kvadrantu koji označava stajanje, nove koordinate će se serijskim portom slati Arduino, koji će pomjerati robota, te stvarati novu poziciju projekcije, sa novim koordinatama za obradu.

Potrebno je, nakon kalkulisanja x i y koordinata formirati string koji će se upisivati u port. Koordinate mogu biti u obliku od jednocifrenog do trocifrenog broja, kako nam definiše maksimalan visina i širina prozora. Prvo je neophodno brojeve konverovati u string, te ih formatirati u ispravan oblik. Ispravan oblik predstavlja dodavanje nula prije broja, ako je u pitanju jednocifren ili dvocifren broj. Nakon konvertovanja koordinate u string, se ispituje dužina broja pomoću funkcije **length()**, te se pomoću funkcije **append(int amount, char symbol)** dodaje jedna ili dvije nule na početak broja, ako je potrebno. Krajnje se formira string u obliku „**location-x-y-action**“, u kojem su x i y koordinate zamijenjene stvarnim vrijednostima, a akcija može zauzeti jednu od dva oblika: **move** ako je potrebno da se robot kreće i **stop** ako je potrebno da se robot zaustavi. Nakon uspješnog otkrivanja koordinata će se uvijek slati akcija **move**, dok će se akcija **stop** pozivati samo kada laserska projekcija nije pronađena. I akcijom **move** se robot neće kretati ako su x i y koordinate u donjem kvadrantu, jer akcija označava samo uspješan pronalazak koordinata za pokretanje robota, dok koordinate predstavljaju stvarni uslov za kretanje.

Nakon kreiranja stringa, potrebno ga je poslati, odnosno upisati u serijski port. Za to je potreban drugi tip CLR objekta koji će se koristiti u ovom projektu, odnosno objekat tipa **String**. CLR objekat **String** i klasični objekat **string** unutar programskog jezika **c++**, su različiti tipovi podataka, zbog čega je potrebno konvertovati **c++ string** u **System::String**. Konstruktor **String** objekta prima niz karaktera za konvertovanje i spašavanje u alociranoj memoriji. **C++ string** objekat posjeduje funkciju članicu **c_str()**, koja upravo pretvara **c++ string** objekat u niz karaktera. U kombinaciji je moguće konvertovati **c++ kreirani string** te ga sačuvati u objekat tipa **System::String**. Na kraju je potrebno upisati kreiranu komandu u serijski port, za šta se može koristiti ugrađena funkcija članica **SerialPort** objekta **WriteLine(String)**, koja će komandu upisati i proslijediti serijskim portom uređaju koji prima i može očitati naredbu.

```
void trackFilteredObject(int& x, int& y, Mat binary, Mat& cameraFeed,
SerialPort^ port) {
    ...
    if (objectFound) {
        drawObject(x, y, cameraFeed);

        string modifiedX = intToString(x);
        if (modifiedX.length() == 1) {
            modifiedX = string(2, '0').append(modifiedX);
        }
        else if (modifiedX.length() == 2) {
            modifiedX = string(1, '0').append(modifiedX);
        }

        string modifiedY = intToString(y);
        if (modifiedY.length() == 1) {
            modifiedY = string(2, '0').append(modifiedY);
        }
        else if (modifiedY.length() == 2) {
            modifiedY = string(1, '0').append(modifiedY);
        }

        string serialString = "location-" + modifiedX + "-" + modifiedY + "-
move";

        System::String^ serialCommand = gcnew
        System::String(serialString.c_str());
        port->WriteLine(serialCommand);
        ...
        ...
        else {
            port->WriteLine("location-000-000-stop");
            putText(cameraFeed, "No laser detected", Point(0, 350), 2, 1, Scalar(0,
0, 255), 1);
        }
    }
}
```

Programski kod 46 Kreiranje i slanje komandnog stringa serijskom portu

4. Implementacija hardvera

Do ovog poglavlja je uspješno napisan i implementiran softver za praćenje laserske projekcije. Međutim, kako bi projekat poprimio praktičnu formu, potrebno je implementirati sve hardverske komponente, kao i napisati prateći softver za njih. Kao što se u prethodnim poglavljima moglo kratko nagovijestiti, glavna hardverska komponenta potrebna za realizaciju projekta je mikrokontroler. Unutar mikrokontrolera će biti implementirana sva logika koja će upravljati ostalim komponentama budućeg robota.

Mikrokontroler je uređaj koji posjeduje sve neophodne komponente kako bi se nazivao kompjuterom. Minimalne neophodne komponente za bilo koji kompjuter su memorija, mikroprocesor te ulazne i izlazne komponente. U sebi sadrži integralna kola, koja su dalje sačinjena od seta elektronskih kola koji zajedno obavljaju određenu funkciju, izvršavajući zadane naredbe, najčešće u ugrađenim sistemima.

Kao što je već spomenuto, u ovom projektu će biti korištena Arduino mikrokontrolerska ploča [6]. Arduino je platforma otvorenog koda (engl. open source), te je zbog toga, u kombinaciji sa potpornom zajednicom koja podržava platformu, veoma pogodna za višenamjensku izradu prototipskih projekata. Arduino platforma, nastala 2005. godine i razvijena od strane „SmartProjects“ kompanije u Italiji, je višepatformski orijentirana (engl. cross platform), što znači da se može pokretati sa svih vodećih operativnih sistema današnjice, Mac-a, Linux-a i Windows-a.

Postoji više vrsta Arduino mikrokontrolerskih ploča, koje se međusobno razlikuju po specifikacijama i veličini. Neki od njih su Arduino Nano, Arduino UNO, Arduino Mega, Arduino Due, Arduino Leonardo itd.

Za implementaciju prve iteracije ovog rada se Arduino Mega 2560 nametnuo kao idealna opcija za realizaciju prototipa robota.

4.1. Arduino Mega 2560

Arduino Mega je mikrokontrolerska ploča bazirana na **Atmega2560** mikročipu. Posjeduje sve za podršku jednog mikrokontrolera, te se jednostavno povezuje sa računarom koristeći USB konekciju. Arduino Mega je mikrokontrolerska ploča dizajnirana za projekte koji zahtijevaju više memorije i više ulaznih i izlaznih linija. Bazirana je na mikročipu visokih performansi, koji se zasniva na RISC17¹⁰ arhitekturi.

Arduino MEGA sadrži 54 digitalnih input/output pinova, od kojih se 15 može koristiti kao PWM¹¹ output, 16 analognih inputa, 256KB flash memorije, od kojih je 8KB iskorišteno za bootloader, ugrađeni LED indikator na pinu 13, te operira sa 5V struje. Također posjeduje USB konekciju, priključak za napajanje, te fizičko restart dugme.

¹⁰ Reduced Instruction Set Computer – tip arhitekture mikročipa sa redukovanim setom instrukcija

¹¹ Pulse Width Modulation – široko-implusna modulacija za dobijanje analognih vrijednosti na osnovu digitalnih impulsa

Digitalni pinovi se mogu konfigurisati da operiraju kao ulazni ili izlazni. Ako se koriste kao ulazni, mikrokontroler očekuje određeno logičko stanje i naredbe od povezanog uređaja i obrnuto, ako se koriste kao izlazni, onda mikrokontroler kreira logičko stanje i naredbe na osnovu kojih će djelovati povezani uređaj.

Programski jezik komunikacije sa Arduinoom je C++, uz dodatne biblioteke platforme kao i posebno razvojno okruženje, o kojima će biti detaljnije govora u sljedećem poglavlju rada. Većina osnovnih, uvodnih funkcija Arduino platforme, kao parametar prima broj pina sa fizičke ploče, te naredbu u obliku zastave (engl. flag) koja diktira ponašanje tog pina.

Za rad sa digitalnim pinovima se koriste tri funkcije. Funkcija **pinMode(pin, flag)** prima dva parametra, broj pina na fizičkoj ploči, kao i zastavu koja može imati jednu od dvije vrijednosti: INPUT ili OUTPUT. Funkcija definiše da li će se željeni pin ponašati kao ulazni ili izlazni pin.

Funkcija **digitalWrite(pin, flag)** također prima dva parametra, od kojih je prvi broj fizičkog pina, a drugi zastava koja određuje da li će taj izlazni pin emitovati signal ili ne. Slično programerskoj logici boolean varijabli, koje mogu imati jednu i samo jednu od dvije vrijednosti, true ili false, tako i zastava za diktiranje digitalnog signala može imati jednu i samo jednu od dvije vrijednosti: **HIGH**, za prisustvo struje u pinu od 5V i **LOW** bez prisustva struje.

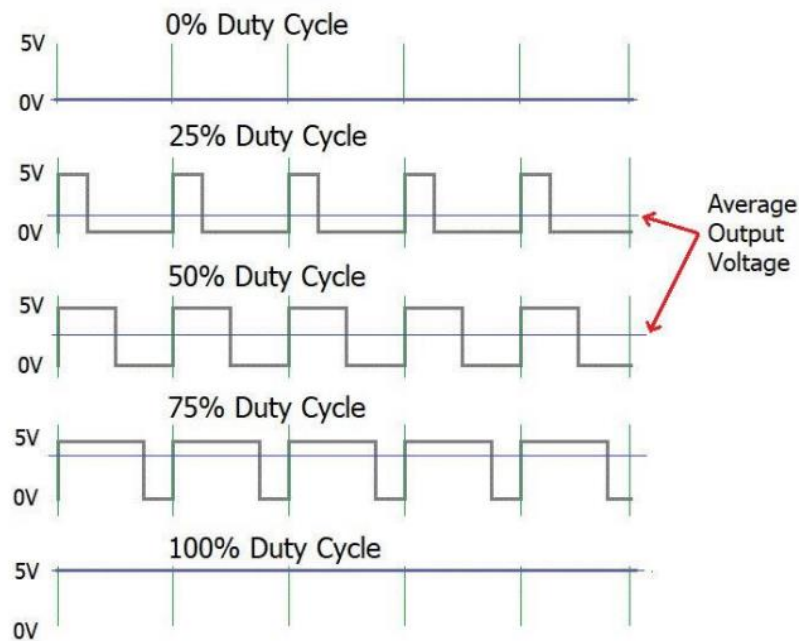
Treća funkcija, funkcija **digitalRead(pin)**, služi za digitalno čitanje podataka sa uređaja, sa parametrom koji predstavlja broj pina. Kao rezultat, funkcija vraća trenutno stanje u kojem se pin nalazi, HIGH ili LOW.

Analogni pinovi koji se nalaze na Arduino ploči su ulazni i koriste se za čitanje analognih senzora povezanih na Arduino mikrokontrolersku ploču. Pored pinMode(pin, flag) funkcije, koja ima istu definiciju kao za digitalne pinove, analogni pinovi koriste dvije dodatne funkcije. Prva funkcija je **analogWrite(pin, value)** koja služi za postavljanje širokopojasne modulacije. Prima dva parametra od kojih prvi predstavlja broj pina na fizičkoj ploči, a drugi cjelobrojnu vrijednost od 0 do 255 i proporcionalno određuje voltažu struje koja će se isporučiti od 0V do 5V. Druga funkcija, funkcija **analogRead(pin)**, služi za čitanje podataka sa uređaja, sa parametrom koji predstavlja broj pina na fizičkoj ploči. Kao rezultat vraća cjelobrojnu vrijednost, koja predstavlja proporcionalno određenu jačinu struje od 0V do 5V.

Kao što je već navedeno, 15 od 54 digitalnih input/output pinova se mogu koristiti kao PWM output, koji predstavlja kombinaciju očitavanja digitalnog signala i dobijanja analognih vrijednosti. Na Arduino Mega ploči, su PWM funkcionalnosti moguće na pinu od 2 do 13+.

PWM ili široko impulsna modulacija je tehnika dobijanja analognih vrijednosti na osnovu digitalnih impulsa. Signal koji PWM reproducira je digitalni sa stanjima „visoko“ (5V) i „nisko“ (0V). Na osnovu dužine stanja „visoko“ u jednom frekvencijskom ciklusu se određuje izlazna voltaža, te je tako moguće na izlazu dobiti voltažu u rangu od 0V do 5V

PMW tehnika i pinovi koji je podržavaju, će biti znatno iskorištena prilikom implementacije Arduino softvera i pokretanja samog robota.

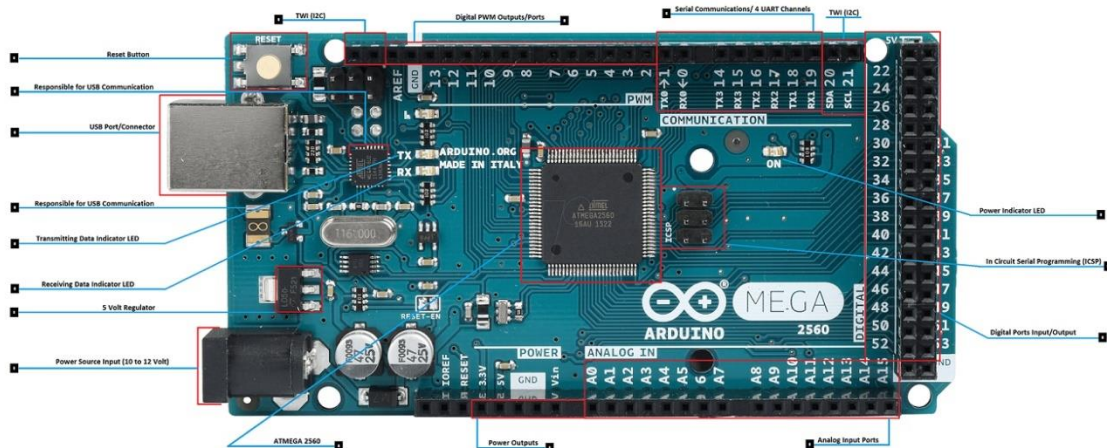


Slika 40 Prikaz široko impulske modulacije - PMW

Arduino Mega također sadrži **5 pinova za struju**. **VIN** pin predstavlja ulaznu struju za ploču, ukoliko se koristi eksterni izvor struje. Pinovi **3.3V** i **5V** obezbjeđuju izlaznu struju od 3.3V i 5V respektivno, te dva **GND** pina za uzemljenje.

USB port koji se nalazi na Arduino ploči ima dvojaku svrhu. Prva je da služi kao izvor napajanja od 5V, a druga da se preko njega vrši transfer programskog koda na mikročip. Arduino ploča će u svrhu izrade ovog rada biti priključena USB konekcijom na „COM10“ port, kako bi se naredba pozicije i kretanja robota, kalkulisana u dijelu softverkse implementacije, uspješno tranferovala na mikročip serijskom komunikacijom, za dalju obradu. Sam programski kod koji će se transferovati na ploču preko USB konekcije, prije pokretanja glavnog programa i koji će primiti komandu koja sadrži trenutne koordinate laserske projekcije i određivati dalje ponašanje robota, će biti prikazan u jednom od nardenih poglavlja.

Da se Arduino ploča ne bi morala konstantno isključivati sa napajanja, prilikom testiranja ili implementacije, **restart dugmetom** je moguće jednostavno prekinuti dovod struje i vratiti Arduino na početno stanje.



Slika 41 Fizički prikaz i shema Arduino Mega 2560 ploče

Bitno je naglasiti da do sada opisane komponente na Arduino ploči nisu jedine, nego osnovne ili komponente koje će biti korištene za realizaciju prve iteracije ovog projekta¹². Primjer jedne takve dodatne komponente je ICSP zaglavlje koje se koristi za vanjsko programiranje Atmega 2560 čipa. Obično se za te svrhe koristi bootloader program, koji se učitava u Arduino IDE23-u, ali u slučaju da je on oštećen ili nedostaje, onda se može koristiti zaglavlje za serijsko programiranje unutar kola.

4.2. Programski jezik i kompajliranje Arduino koda

Kao što je već ukratko spomenuto u prethodnim poglavljima, programski jezik u kojem se pišu datoteke koje sadrže Arduino izvorni kod je C++. Međutim, većina standardnih biblioteka je preuzeta iz programskog jezika C zbog male količine radne memorije kontrolera. Arduino datoteke, koje nose drugi naziv skice, je moguće prepoznati po njihov postfiksnoj ekstenziji .ino.

Nakon što se pokrene proces kompajliranja projekta, Arduino okruženje pravi promjene u kodu, kako bi se kod nakon toga mogao proslijediti gcc i g++ kompajleru, koji se koristi za kompajliranje c i c++ programskih jezika. U ovoj fazi se sve datoteke u direktoriju kombinuju u jednu i na njen početak se dodaje `#include<Arduino.h>` zaglavlje, te se kod povezuje sa standardnim Arduino bibliotekama i drugim bibliotekama uključenim u direktorij skice.

Kada je skica povezana i prevedena, transferuje se u Arduino memoriju preko USB konekcije, odakle će se dalje izvršavati.

Unutar Arduino platforme postoje određena pravila i prakse pisanja programskog koda. Osnova i najjednostavnije .ino datoteka se sastoji od dvije funkcije: `setup()` i `loop()`.

¹² <https://store.arduino.cc/arduino-mega-2560-rev3>

Funkcija `setup()` se izvršava samo jednom pri pokretanju mikrokontrolera i služi za inicijalizaciju komponenti i objekata, dok funkcija `loop()` predstavlja srž načina rada Arduino ploča. Programski kod koji se nalazi unutar `loop()` funkcije će se ciklično izvršavati sve dok je Arduino upaljen, te kod unutar ove funkcije predstavlja sekvencu ponašanja isprogramirane ploče u upotrebi.

Sva ostala pravila sintakse, logike i shema programiranja koje vrijede u programskom jeziku C++, vrijede i prilikom pisanja programskog koda za Arduino. Neke od njih su definisanje globalnih varijabli, dodatnih pomoćnih funkcija, te definisanje konstanti na samom vrhu projekta.

Dodatna funkcija koja će biti korištena za implementaciju ovog projekta je funkcija **`serialEvent()`**, koja osluškuje da li su dostupni ikakvi podaci unutar serijskog porta. Kako u implementaciji softvera šaljemo naredbu lokacije laserske projekcije u obliku stringa serijskim portom, će ova funkcija, sve dok je serijski port dostupan, očitavati taj string karakter po karakter, te ga nakon kompletnog očitavanja i formiranja slati u `loop()` funkciju na dalju obradu. Tačan način rada sve tri funkcije za ovaj projekat, će biti detaljno objašnjene u narednim poglavljima.

4.3. Arduino razvojna okruženja

Da bi se mogao pisati ikakav kod, kompajlirati i slati na Arduino ploču, potrebno je jedno od Arduino razvojnih okruženja. Razvojno okruženje nudi set funkcija, opcija, dodatnih prozora i funkcionalnosti koje su potrebne za pisanje, testiranje i implementiranje koda za bilo koju od Arduino ploča. Dva najpopularnija razvojna okruženja su oficijelno Arduino okruženje „Arduino IDE“, te Visual Studio Code okruženje „PlatformIO“. Naravno, ovo nisu jedina dva rješenja za razvojno okruženje, ali predstavljaju dva najčešće korištena.

4.3.1. Arduino IDE

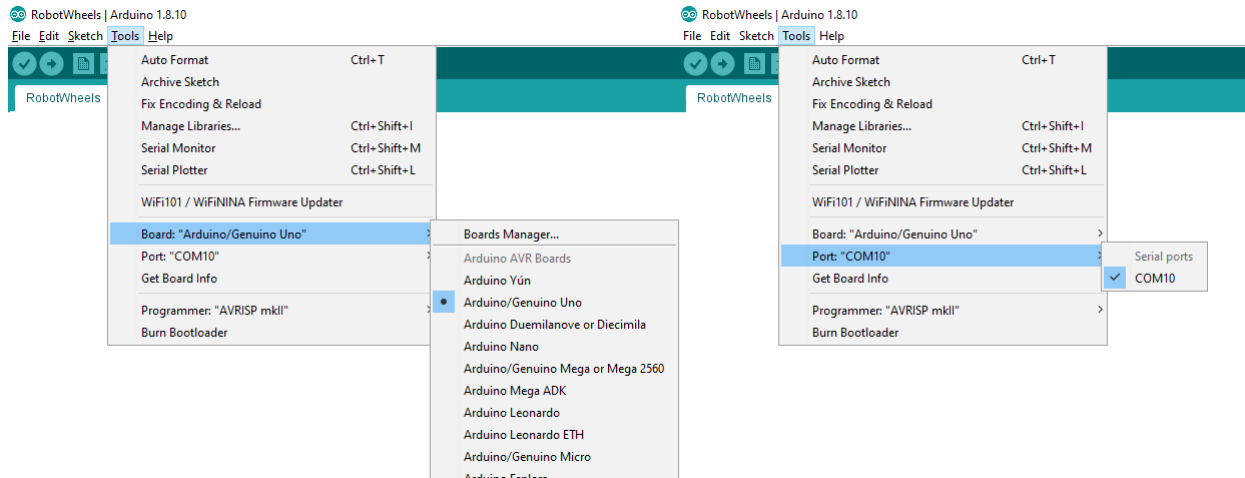
Arduino posjeduje svoje oficijelno razvojno okruženje pod nazivom „Arduino IDE“, koje se može koristiti u dvije verzije: online verzija dostupna na oficijelnoj Arduino stranici¹³, te offline verzija, koja je besplatna za download, također sa oficijelne Arduino stranice¹⁴. Za ovaj rad će biti korišteno offline okruženje, kako se uz njegovu instalaciju automatski instaliraju i svi Windows driveri potrebni za transfer programskog koda na Arduino ploču.

Nakon priključivanja Arduino ploče na računar, korsiteći USB konekciju, potrebno je razvojnom okruženju prenijeti informacije o tipu ploče, kao i na koji dostupan port je priključena Arduino ploča.

¹³ <https://create.arduino.cc/editor>

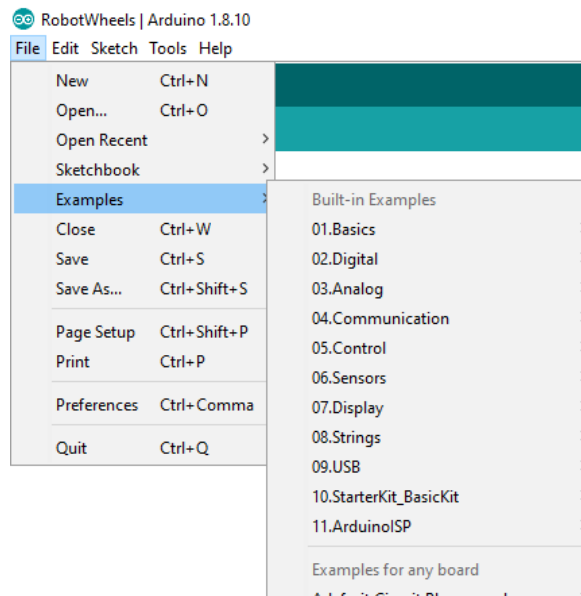
¹⁴ <https://www.arduino.cc/en/Main/Software>

To možemo postići klikom na tab **Tools** u gornjem lijevom ćošku prozora razvojnog okruženja, te daljim klikom na **Board** i/ili **Port**. Lista svih mogućih ploča kao i svih dostupnih portova, sa kojima je ostvarena konekcija, će biti ponuđena automatski. U slučaju da se određena Arduino ploča ne nalazi na spisku, potrebno je otvoriti **Boards Manager** tab te pronaći i instalirati željenu ploču.



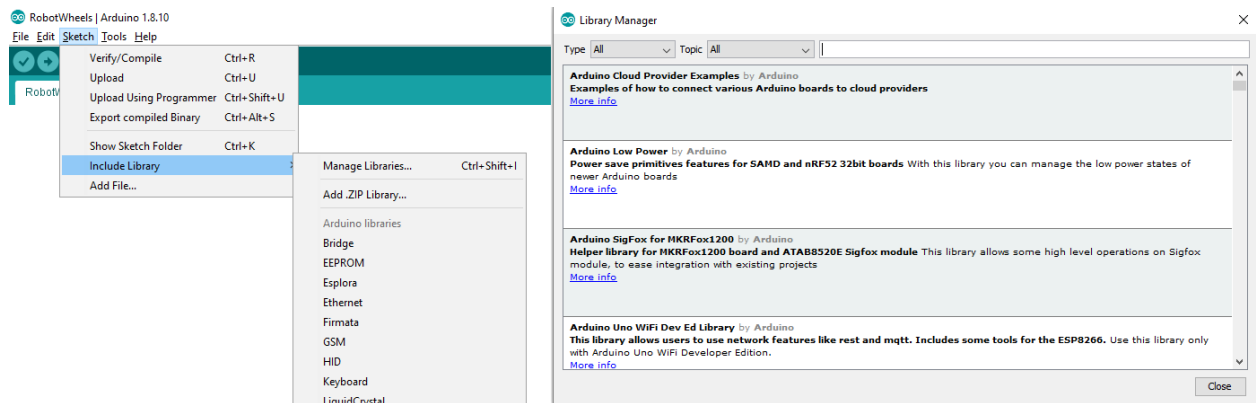
Slika 42 Lijevo: Odabir Arduino ploče; Desno: Odabir porta

Jedna od korisnih funkcionalnosti Arduino IDE razvojnog okruženja, je ugrađena lista svih osnovnih primjera programskog koda na jednom mjestu. Ova lista je korisna za „start-up“, početnog koda projekta, kao i za učenje Arduino logike uz ugrađenu dokumentaciju. Primjeri se mogu naći na tabu **File** u gornjem lijevom ćošku, te na opciji **Examples**.



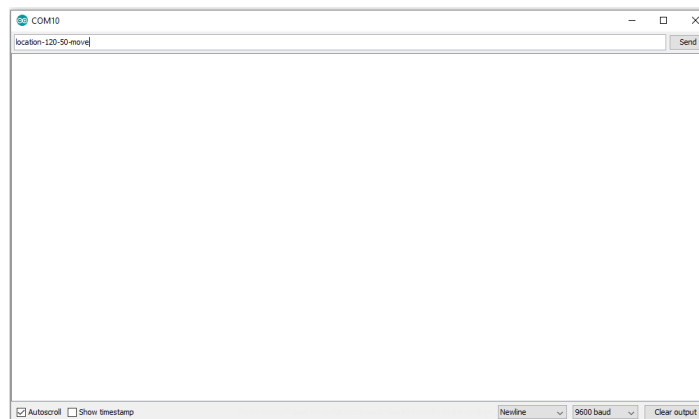
Slika 43 Odabir i kreiranje primjer programa za Arduino platformu

Prilikom pisanja bilo koje vrtse programskog koda je često potrebno uključivanje određenih biblioteka sa željenim ili potrebnim funkcionalnostima. Arduino IDE dolazi sa već predefinisanim repozitorijem biblioteka koje se nude za preuzimanje, instaliranje i uključivanje u projekat. Lista svih biblioteka kao i sam **Library Manager** za dodatne biblioteke se mogu pronaći na tabu **Sketch** i opciji **Include Library**. U slučaju da korisnik želi dodati eksternu biblioteku može to uraditi učitavanjem **.zip** datoteke ili dodavanjem putanje repozitorija u postavkama okruženja.



Slika 44 Lijevo: Odabir biblioteka za dodavanje; Desno: Library Manager

U svrhu testiranja i uspostavljanja komunikacije sa Arduino uređajem dok je pokrenut, se veoma često koristi **Serial Monitor** koji je ugrađen u Arduino IDE razvojno okruženje. Korisnik pomoću ovog alata može simulirati razmjenu tekstualnih poruka sa Arduino pločom pomoću serijske veze. Serijski monitor će ispisivati izlazne poruke po napisanom programskom kodu, u zavisnosti od primljene naredbe. Prilikom implementacije ovog rada se serijski monitor može koristiti za testiranje, slanjem string naredbe koja je formatirana ispravno i sadrži koordinate laserske projekcije. Dodatno, na dnu prozora se nalazi padajući meni čija odabrana stavka definiše brzinu komunikacije u bitovima po sekundi, naredba za čišćenje prozora kao i način prikazivanja svake sljedeće poruke. Serial Monitor prozor se može otvoriti na dva načina. Prvi je klikom na tab **Tools** pa na opciju Serial Monitor, ili klikom na ikonu lupe u gornjem desnom ćošku prozora razvojnog okruženja.



Slika 45 Serial Monitor

Nakon kreiranja nove Arduino skice za projekat će se datoteka sama inicijalizovati sa osnovne dvije funkcije koje većina Arduino projekata mora posjedovati, sa komentarima za upotrebu i način korištenja istih.

```
void setup() {  
  //put your setup code here, to run once:  
}  
  
void loop() {  
  //put your main code here, to run repeatedly  
}
```

Programski kod 47 Inicijalizacija prazne Arduino skice

Po završetku kucanja koda, je moguće provjeriti i potvrditi ispravnost koda klikom na ikonu tačnice u gornjem lijevom ćošku prozora, te izvršiti upload, odnosno transfer programskog koda na Arduino ploču, pod uslovom da je kod ispravno napisan, klikom na ikonu strelice, također u gornjem lijevom ćošku prozora. Nakon klika za upload će kod proći kroz već opisani proces kompajliranja i prenijeti se na ploču, odakle će, sve dok je ploča upaljena, izvršavati kod koji je prebačen na nju: u kontekstu ovog rada, osluškivati i primati komandu sa serijskog porta poslanu iz softvera, te davati robotu instrukcije koristeći PWM pinove.



Slika 46 Komande za provjeravanje i upload

4.3.2. PlatformIO

PlatformIO [7] je drugo najpopularnije okruženje za pisanje Arduino programskog koda. Svoju popularnost bazira na tome što se koristi i instalira u vidu nadogradnje za najpopularniji kod tekst editor Visual Studio Code [8], koji zajedno sa PlatformIO spada u kategoriju programa otvorenog tipa (engl. open source). Sve dodatne pogodne funkcionalnosti Visual Studio Code editora se direktno preslikavaju na doživljaj pisanja Arduino koda. Neke od tih pogodnosti su veoma napredan **IntelliSense**¹⁵ za dopunu i ispravljanje sinakse koda, kao i market ekstenzija za olakšavanje pisanja i razumijevanja kucanog koda.

Također, mnogi korisnici su upoznati sa interfejsom Visual Studio Code-a, koristeći ga predhodno ili poznavajući druge editore istog tipa i izgleda, te se mnogo lakše priviknu i nauče na sve novosti Arduino programiranja. Editor je fleksibilan, brz i nudi dodatne olakšice prilikom Arduino programiranja, tipa da se sam prepozna koji COM port se koristi, ugrađen Serial Monitor, debugging itd.

Jedna od uočljivijih razlika Arduino IDE i PlatformIO razvojnog okruženja je ta što glavna datoteka sa izvornim kodom u PlatformIO nije **.ino** već **main.cpp** datoteka, u kojoj se obavezno mora uključiti **Arduino.h** biblioteka na vrhu projekta.

¹⁵ <https://code.visualstudio.com/docs/editor/intellisense>

Dugmad za provjeravanje i upload koda, kao i dugme za otvaranje Serial Monitora, sa istim funkcionalnostima i mnogim drugim, se nalaze u donjem lijevom ćošku prozora.

4.4. Arduino programski kod za upravljanje robota

Nakon uspješne implementacije softvera u Visual Studio-u koristeći OpenCV biblioteku, te spajanja Arduino ploče preko USB konekcije sa računarom, je potrebno napisati programski kod koji će primati string naredbe i koordinate laserkse projekcije, te davati upute kroz Arduino mikrokontroler, koji će slati različite intervale struje u dva točka robota, koristeći PMW pinove. Po završetku pisanja koda, ga je potrebno prebaciti na Arduino ploču, te spojiti odgovarajuće pinove koji će emitovati struju sa motorima koji upravljaju točkove robota.

Pri otvaranju nove Arduino skice, prvo je potrebno definisati određene konstante i globalne varijable koje će se koristiti prilikom pisanja softvera. Sama logika programa je da šalje različite jačine struje, definisane cjelobrojnim vrijednostima, PMW pinovima koji će da emituju te različite intenzitete struje ka točkovima, u zavisnosti od koordinata laserske projekcije na ekranu. U zavisnosti u kojem kvadrantu se projekcija nalazi, će se slati različit intenzitet struje.

Već je navedeno da Arduino Mega ploča sadrži 15 PMW pinova (2-13+). Za implementaciju ovog projekta su potrebna dva PMW pina koja će predstavljati lijevi (pin 4) i desni (pin 3) točak. Također, kako bi se kalkulisala pozicija laserske projekcije, odnosno odredio kvadrant u kojem se projekcija nalazi, potrebno je ponovno definisati visinu i širinu prozora vrijednostima 640x480 piksela, kao i u implementaciji softvera.

Zatim je potrebno definisati tri različite brzine kretanja robota. Cjelobrojne vrijednosti odabrane za ove brzine su 200 za brzo kretanje, 100 za umjereno kretanje i 50 za sporo kretanje. Ove vrijednosti, kada se digitalno pošalju PMW pinovima, će se konvertovati u analgone signale različite jačine struje, u ovisnosti o veličini cjelobrojne vrijednosti. Ove vrijednosti se mogu definisati u opsegu od 0 do 255. Međutim, ostavljeno je prostora za ubrzavanje točkova, kako se oni moraju rotirati brže, odnosno primiti veću količinu struje, ako se robot mora kretati ulijevo ili udesno. Dodavanjem vrijednosti na lijevi ili desni točak (pin) ne smije preći maksimalnu granicu od 255, zbog čega je brzo kretanje definisano sa 200, kako bi se moglo nadodati 55 da bi se postigla maksimalna brzina od 255 za intenzivnije okretanje točka ulijevo ili udesno, prilikom generalnog najbržeg kretanja naprijed.

U implementaciji softvera je već bilo govora da se laserska projekcija može nalaziti 30 piksela ulijevo ili udesno od čistog centra, kako bi se i dalje smatrala u horizontalnom centru na x osi. Potrebno je dakle definisati sam prostor od 30 piksela, te lijevu i desnu granicu na x osi, izvan kojih se robot mora početi kretati lijevo ili desno.

Krajnje, potrebo je deklarirati globalnu varijablu tipa `System::String`, u kojoj će se čuvati primljena string naredba sa serijskog porta, poslana iz softvera, te kako se glavni kod Arduina beskonačno vrti unutar `loop()` funkcije, potrebno je deklarirati jednu boolean varijablu, koja će osigurati da je cijela komanda uspješno primljena i prebačena u input string, prije slanja ikakvih signala.

```
#define leftWheel 4
#define rightWheel 3
#define width 640
#define height 480
#define highSpeed 200
#define mediumSpeed 100
#define lowSpeed 50
#define speedUp 55
#define xFromCenter 30
#define xLeft (width/2)-xFromCenter
#define xRight (width/2)+xFromCenter

String inputString = "";
bool stringComplete = false;
```

Programski kod 48 Definisanje svih konstanti i globalnih varijabli

Funkcija **setup()** će započeti serijsku vezu koristeći funkciju članicu **begin(baud)** nad ugrađenim objektom **Serial**, koja kao parametar prima vrijednost maksimalnog prenosa bita po sekundi. Također, potrebno je za definisani input string rezervirati dovoljno prostora u memoriji, što se može postići funkcijom članicom **reserve(amount)**, koja kao parametar prima cjelobrojnu vrijednost količine memorije. Dodatno, kako se funkcija **setup()** poziva samo jednom i inicijalizuje varijable, će se vrijednosti struje za lijevi i desni točak ovdje postaviti na nulu odnosno LOW.

```
void setup() {
    Serial.begin(9600);
    inputString.reserve(200);
    analogWrite(leftWheel, LOW);
    analogWrite(rightWheel, LOW);
}
```

Programski kod 49 Funkcija setup()

Prije početka **loop()** ili druge osnovne funkcije za većinu Arduino projekata, potrebno je obraditi dodatnu funkciju **serialEvent()**. Ova funkcija prima sadržaj poslan serijskim portom, sve dok serijski port postoji. Jedna od osnovnih karakteristika String-a je ta da po završetku sadržaja svakog niza karaktera od kojih je građen String, se nalazi poseban karakter „\n“, koji označava kraj datog String-a.

Srž **serialEvent()** funkcije je da, ako postoji sadržaj serijskog porta, učitava karakter po karakter primljenog string sadržaja, te svaki karakter nadodaje na input string definisan na početku programa. U trenutku kada sljedeći karakter za nadodavanje bude posebni karakter „\n“, boolean varijabla **stringComplete** se postavlja na true, kako je čitav string uspješno učitao, jer se uspio pronaći terminirajući karakter koji označava kraj stringa.

Preuzimanje svakog karaktera redom se može postići funkcijom članicom **read()** nad Serial objektom, te konvertovanjem rezultata funkcije u objekat tipa **char**, a postojanje veze se može osigurati funkcijom članicom **available()** također nad Serial objektom, koja kao rezultat vraća true ili false u zavisnosti od toga da li je veza na raspolaganju.

```
void serialEvent() {  
    while (Serial.available()) {  
  
        char inChar = (char)Serial.read();  
  
        inputString += inChar;  
  
        if (inChar == '\n') {  
            stringComplete = true;  
        }  
    }  
}
```

Programski kod 50 Funkcija serialEvent()

Sadržaj loop() funkcije će se realizovati tek kada boolean varijabla bude postavljena na true, odnosno kada se uspješno učitava sva komanda. Nakon što se izvrši analiza komande, **stringComplete** varijable će se postaviti na false, da se funkcija ne bi ponovno pozvala na istoj komandi, nego čekala novu, čije će učitavanje ponovno pokrenuti loop() petlju, sa novom komandom u trenutku kada se ona sva uspješno učitava. Također se na kraju loop() petlje input string briše i postavlja na prazan string. Ovaj proces se izvršava u krug.

U tijelu loop() petlje se prvo vrši provjera da li je input string kompletan i u potpunosti učitavan. Nakon toga se iz komande moraju ekstrahovati x i y koordinate, kao i akcija na kraju komande. Dodatno će se radi testiranja, sva komanda prvo ispisati na Serial Monitor, koristeći funkciju članicu **println(string)** nad Serial objektom.

Ekstrahovanje koordinata i akcije se vrši pomoću funkcije članice **substring(int, int)**, nad input stringom. Ova funkcija prima dva parametra, početni i krajnji indeks, te kao rezultat vraća substring između ta dva indeksa. Kako su brojevi unutar prosljeđenog input stringa ispravno formatirani u softveru, poznato je da su brojevi uvijek trocifreni, te da se uvijek nalaze na istim indeksima. Isto vrijedi za akciju **move** ili **stop** na kraju komande.

Nakon ekstrahovanja se prvo provjerava da li je akcija jednaka akciji **stop**, kako se u tom slučaju robot mora odmah zaustaviti. U slučaju da jeste, na desni i lijevi točak, odnosno na PMW pin broj 3 i broj 6 se funkcijom **analogWrite()** šalje zastava **LOW**, odnosno struja kroz te pinove se spušta na 0V što dovodi robota u stanje mirovanja, kako točkovi ne primaju struju.

Ako akcija nije definisana kao **stop**, to znači da je sadržaj akcije jednak akciji **move**, te se nastavlja sa analizom koordinata jer se robot mora kretati. Prvo je potrebno koordinate konvertovati iz string u integer vrijednosti, što se može postići funkcijom članicom **toInt()**, nad string vrijednostima pojedinačnih koordinata. Opet, kako su svi brojevi formatirani ispravno u trocifrene brojeve, konverzija će i za jednocifrene i dvocifrene brojeve proći uspješno, kako se nule ispred cijelog broja ignorišu u konverziji.

Nakon toga je potrebno odrediti u kojem se kvadrantu laserska projekcija nalazi, na osnovu njenih x i y koordinata, koje svoj put od kamere preko softvera i hardvera završavaju na ovom mjestu. Ako je y koordinata manja od 120 piksela u pitanju je prvi, gornji kvadrant, te se na lijevi i desni točak šalje maksimalna vrijednost struje. Ako je između 120 i 220, šalje se umjerena brzina struje na oba točka, te ako je između 220 i 330, u pitanju je posljednji kvadrant za kretanje, te se šalje najmanji intenzitet struje. U slučaju da y koordinata ne ispunjava nijedan od ova 3 uslova, to znači da se laserska projekcija nalazi u donjem, četvrtom kvadrantu, te se i desnom i lijevom točku oduzima struja, kako robot u tom trenutku mora stati.

Prilikom određivanja kvadranta, osim u četvrtom kvadrantu kada robot mora stati, se provjerava i x koordinata. Ako je ona manja od definisane lijeve granice, u desni točak se šalje struja trenutne brzine i na nju se nadodaje još potrebno ubrzanje da bi se projekcija dovela na horizontalni centar. Obrnuto, ako je x koordinata veća od desne granice, u lijevi točak, odnosno u lijevi pin, se šalje dodatni intenzitet struje. Kako se dodatna brzina uvijek nadodaje na trenutnu brzinu kretanja naprijed, robot će se kretati glatko, bez naglih trzaja i poteškoća. Opcionalno, u zavisnosti od hardverske opreme i jačine motora, se može u potpunosti ugasiti suprotni točak za lakše skretanje. Na kraju svake pojedinačne analize naredbe, se input string prazni a boolean varijabla **stringComplete** postavlja na **false**.

Sa ovim smo obezbijedili da se robot uvijek kreće u određenom pravcu, dok ne dođe na ispravnu poziciju, gdje se zaustavlja. Ispravna pozicija se mjeri po tome da je laserska projekcija u četvrtom kvadrantu slike robota, što se kalkuliše pomoću koordinata. Nova naredba se učitava svaki frame u kojem je projekcija uspješno detektovana u softveru.

```
void loop() {  
  if (stringComplete) {  
    Serial.println(inputString);  
    String stringX=inputString.substring(9,12);  
    String stringY=inputString.substring(13,16);  
    String command=inputString.substring(17,21);  
  
    if (command=="stop") {  
      analogWrite(rightWheel, LOW);  
      analogWrite(leftWheel, LOW);  
    }  
  }  
}
```

Programski kod 51 Funkcija loop(): ekstraktovanje i zaustavljanje

```
else{
    int x=stringX.toInt();
    int y=stringY.toInt();

    if(y<120){
        analogWrite(rightWheel, highSpeed);
        analogWrite(leftWheel, highSpeed);
        if(x<xLeft){
            analogWrite(rightWheel, highSpeed+speedUp);
        }
        if(x>xRight){
            analogWrite(leftWheel, highSpeed+speedUp);
        }
    }
    else if(y>120 && y<220){
        analogWrite(rightWheel, mediumSpeed);
        analogWrite(leftWheel, mediumSpeed);
        if(x<xLeft){
            analogWrite(rightWheel, mediumSpeed+speedUp);
        }
        if(x>xRight){
            analogWrite(leftWheel, mediumSpeed+speedUp);
        }
    }
    else if(y>220 && y<320){
        analogWrite(rightWheel, lowSpeed);
        analogWrite(leftWheel, lowSpeed);
        if(x<xLeft){
            analogWrite(rightWheel, lowSpeed+speedUp);
        }
        if(x>xRight){
            analogWrite(leftWheel, lowSpeed+speedUp);
        }
    }
    else{
        analogWrite(rightWheel, LOW);
        analogWrite(leftWheel, LOW);
    }
}

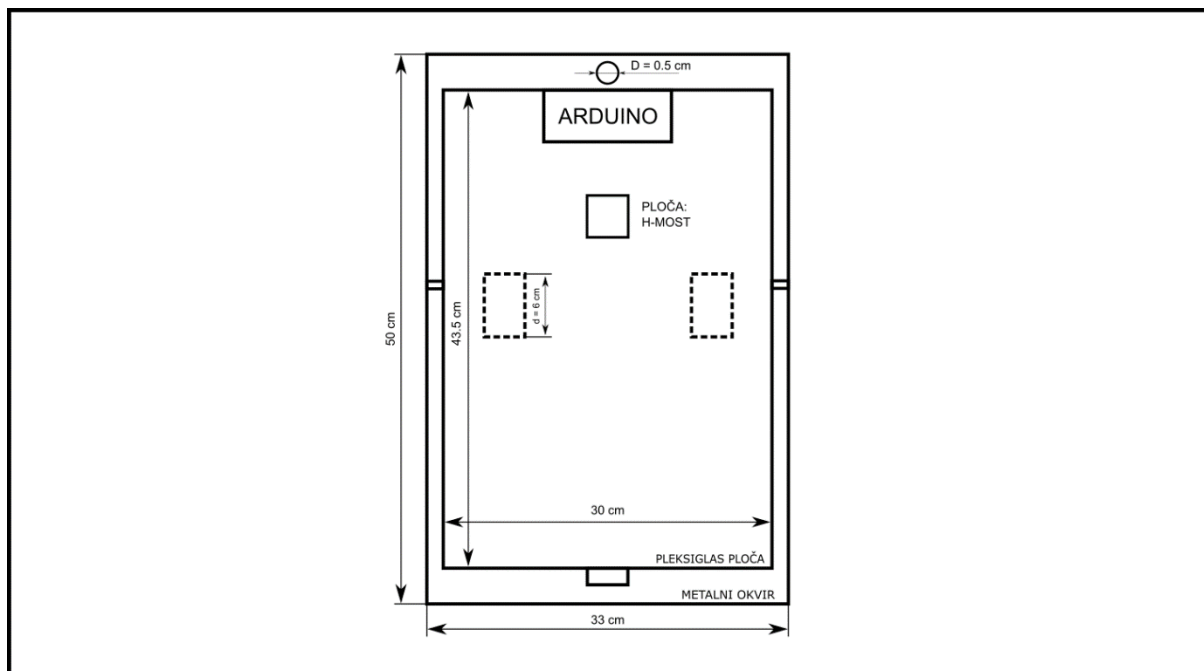
inputString = "";
stringComplete = false;
}
```

Programski kod 52 Funkcija loop(): analiza koordinata i slanje signala pinovima / točkovima

4.5. Konstrukcija robota i spajanje komponenti

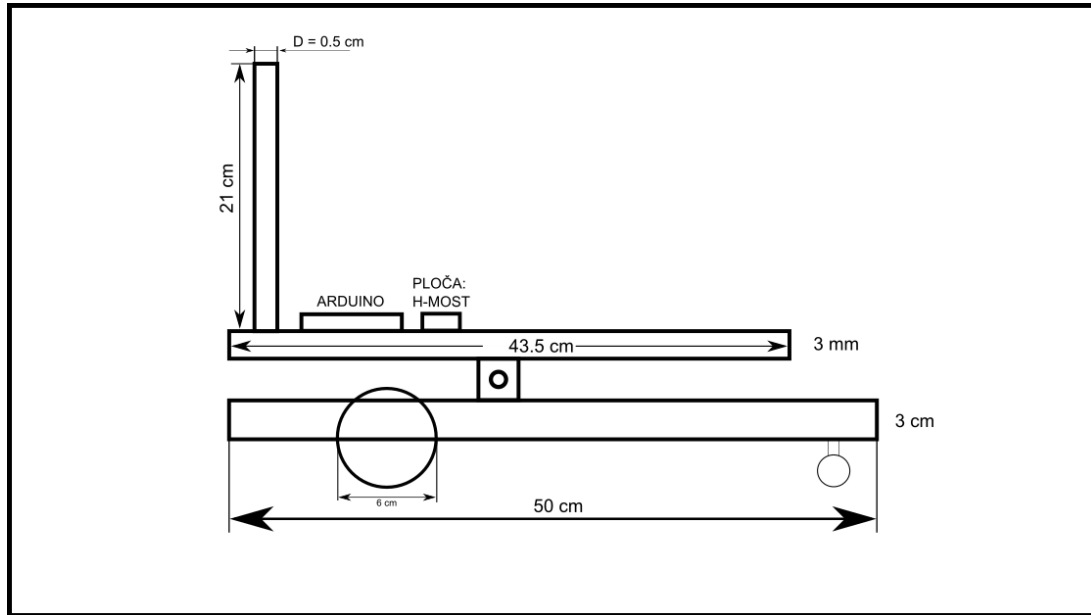
Nakon sve implementacije softvera, potrebno je izgraditi i osposobiti robota, koji će u prvoj iteraciji ovog projekta demonstrirati željeno ponašanje kretanja, uz sve ulaze i izlaze. U svrhu rada će biti konstruisana platforma za čuvanje laptopa, kamere, te svih ostalih hardverskih komponenti neophodnih za demonstraciju.

Platforma, izgrađena od jedne pleksiglas ploče sa gornje strane, položene na metalni okvir sa donje strane, sadrži ispod dva točka, na koja su spojeni DC motori¹⁶. Motori su odgovorni za okretanje točkova različitim brzinama. Sa gornje strane platforme je obezbijeđeno dovoljno prostora za laptop, Arduino ploču, dodatne kablove i opremu, kao i vertikalni most koji će čuvati kameru. Laptop se pozicionira na platformu, te se USB konekcijom i kablom spaja sa Arduino pločom, koja se pozicionira iza laptopa. Za napajanje se na Arduino ploču konektuje 2300 mAh lithium polymer baterija. Struju koju Arduino ploča emituje iz PWM pinova je potrebno spojiti sa lijevim i desnim motorom koji upravlja točkove respektivno. Kako bi se to realizovalo, potrebna je ploča uz H-most, koja će kontrolisati motore točkova. Izlaz iz PWM pinova mikrokontrolera šalje signale na H-most, koji će jačinu primljene struje konvertovati u brzinu okretanja točkova, te pokretati motore. Količina struje se kalkuliše softverom hosritanim na laptopu, na osnovu udaljenosti laserke projekcije. Kako bi platforma mogla vidjeti lasersku projekciju, sa prednje strane je monitran veritaknli most na čijem kraju se nalazi kamera, okrenuta prema podlozi po kojoj se robot kreće, spremna za praćenje laserske projekcije.

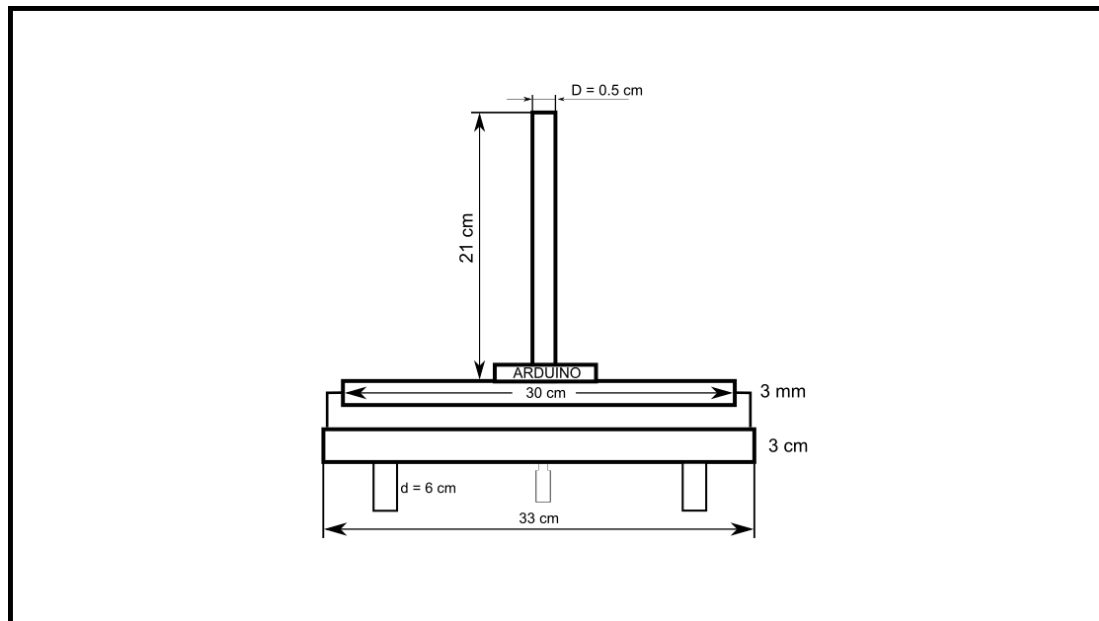


Slika 47 Shema: tlocrt platforme

¹⁶ Bilo koji od klasa rotacionih elektromotora koji pretvaraju direktnu električnu u mehaničku energiju.



Slika 48 Shema: bokocrt platforme



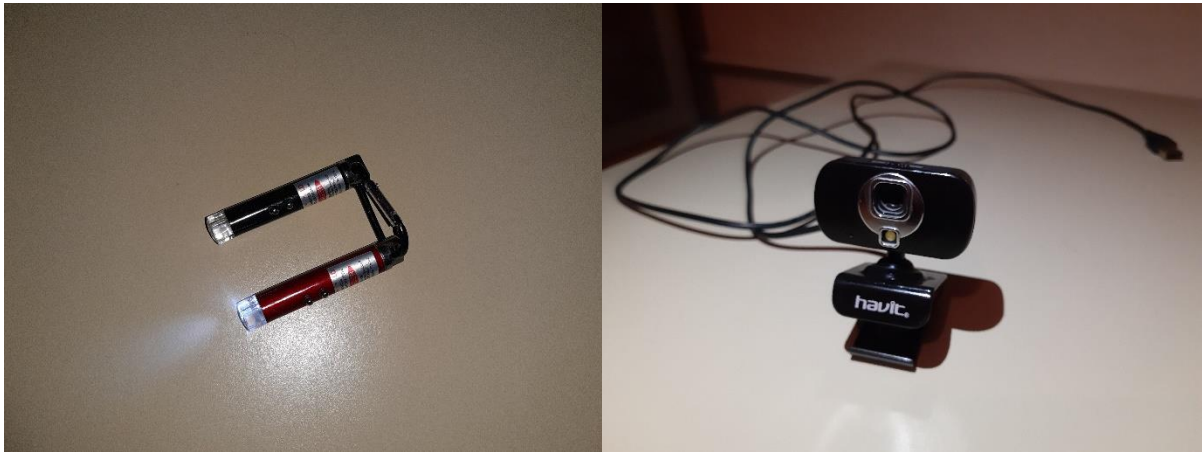
Slika 49 Shema: nacrt platforme

H-most na ploči korištenoj za implementaciju prve iteracije platforme, je kreiran sa integralnim L298N kolom¹⁷ i predstavlja glavni dio ploče kako se u njemu nalazi H-most, neophodan za realizaciju projekta [9]. Lociran je sa gornje strane platforme, blizu i spojen sa Arduino pločom.

¹⁷ <https://pdf1.alldatasheet.com/datasheet-pdf/view/22440/STMICROELECTRONICS/L298N.html>

Kamera korištena za realizaciju platforme je „Venus USB2.0 Camera“. Ova kamera je specifično odabrana iz više razloga, mada se projekat može realizovati sa bilo kojom drugom kamerom. Venus USB2.0 kamera sa gornje strane ima ugrađen rotor klizač, koji omogućuje izoštravanje input slike za različita okruženja i podloge na kojima se robot nalazi. Također, posjeduje ugrađenu LED lampicu sa prednje strane i fizičko dugme koje pali i gasi osvjetljenje uzrokovano tom lampicom. Ovi faktori i elementi kamere pomažu pri kreiranju približno perfektnog okruženja i podloge, kako izoštravaju sliku i uklanjaju bilo kakve sjene, koje bi mogle smetati prilikom detektovanja laserske projekcije. Na dnu kamere se nalazi držač iskorišten za montiranje i pričvršćivanje kamere za veritkalni most na platformi.

Laser korišten pri realizaciji rada se sastoji od laserske diode talasne dužine između 630 i 650 nm (nanometara). Maksimalni izlaz je manji od 5 mW (milliwatti), kako za svrhu testiranja, ali i korištenja sa pravim korisnicima, mora biti siguran do određene mjere, iako se i sa ovim količinama ne preporučuje u direktnom kontaktu sa očima. Također, prilikom stvaranja približno perfektnog okruženja i podloge, na vrhu lasera, kod diode, se nalazi dodatna LED lampica, koja pomaže pri osvjetljenju podloge i uklanjanju dodatnih sjena.



Slika 50 Lijevo: set lasera korišten za projekat; Desno: Venus USB2.0 Camera korištena za projekat

Dodatno, podloga koja će se koristiti ispod točkova platforme će biti teksturno uniformna, lahko zahvatljiva za točkove, dakle ne previše glatka, kao i jednobojna, kako bi se simuliralo što bolje okruženje za detektovanje laserske projekcije.

Spajanjem svih elemenata na platformi i pokretanjem softvera na laptopu se uspješno kreirao robot koji prati lasersku projekciju, korištenjem computer vision tehnologije.

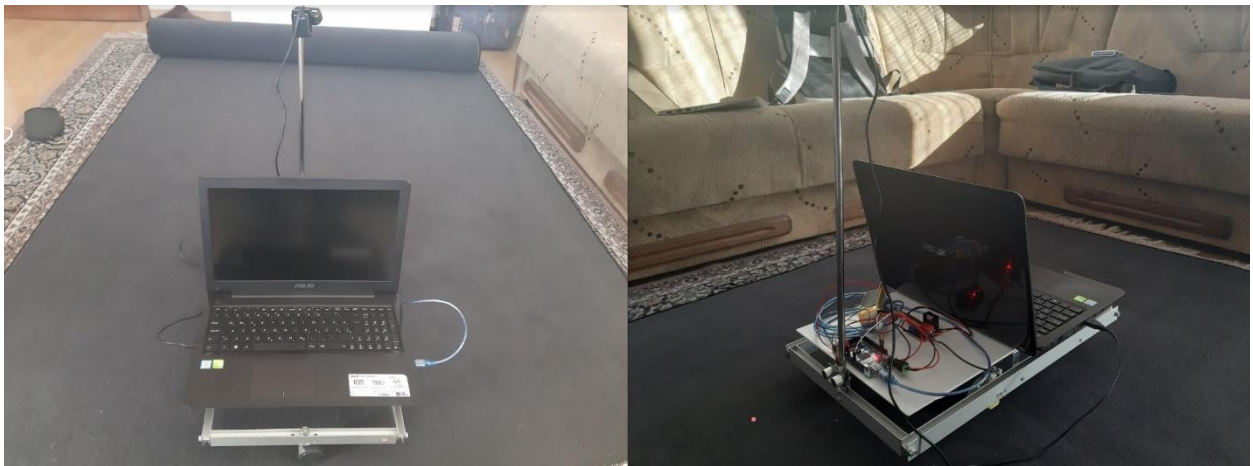
5. Testiranje robota

Nakon uspješnog pisanja softvera i sklapanja svih hardverskih komponenti u jednu platformu, je krajnje potrebno izvršiti osnovni test prve iteracije cijelog projekta. Platforma će se položiti na uniformnu i jednoboju podlogu. Kamera će se prikačiti na vertikalni most i orijentisati da posmatra podlogu ispred i ispod sebe. Laptop se pozicionira na platformu, iza vertikalnog mosta, te se jednom USB konekcijom spaja sa kamerom, a drugom sa Arduino Mega mikrokontrolerom, koji se nalazi ispred laptopa i u koji je priključena baterija.

Prije samog testiranja treba osigurati da je ispravan kod prebačen na Arduino ploču, da su ispravne USB konekcije priključene sa ostalim elementima robota, te da je baterija dovoljno napunjena.

Kada su svi drugi faktori spremni, na laptopu se, iz razvojnog okruženja Visual Studio, pokreće softver, koji pali kameru, te na ekran laptopa prikazuje rgb sliku sa smjernicama, te HSV kao i binarnu sliku. Laserskom projekcijom se nakon toga sija u vidokrug kamere, koja će, kada detektuje projekciju i njenu poziciju kroz softver na laptopu, slati komandu Arduino ploči, koja će dalje odlučiti koje signale da šalje H-mostu. H-most će po tome pokretati i zaustavljati motore točkova u zavisnosti od odluke koja se kroz sistem dovela do mosta, dok će se rad softvera moći posmatrati na ekranu laptopa: detektovanje i pozicija laserske projekcije.

Test se smatra uspješnim ako robot savlada sve osnovne funkcionalnosti, koje se sastoje od kretanja naprijed različitim brzinama, skretanja ulijevo ili udesno, te stajanja, naravno sve u skladu sa laserskom projekcijom i njenom pozicijom, koja se nalazi i pomjera, u svrhu testiranja, ispred kamere i samog robota.



Slika 51 Fizički prikaz i testiranje platforme

6. Budućnost projekta

Uspješnom konstrukcijom robota koji prati lasersku projekciju, se postavlja dublje pitanje o budućoj svrhi i upotrebi projekta. Prva iteracija i sam koncept rada je kao svrhu imao olakšati upravljanje većih i težih mašina i kolica, te ukloniti ljudski faktor pri istom, što smanjuje rizike grešaka i generalnih povreda. Upravljanje kolica sa namirnicima u supermarketu, ili navođenje kanti za smeće ili kontejnera prema komunalnom vozilu su bile neke od prvih ideja prilikom izrade projekta.

Međutim, dokazivanjem kocepta i izgradnjom prototipa se prikazuju razne ideje i koncepti za buduću upotrebu projekta. Jedna od njih je pomoć pri evakuaciji i otklanjanju vozila sa puta, bilo policijskim službenicima ili radnicima auto-moto klubova. Policijsko auto bi moglo laserom, koji se nalazi na zadnjoj strani vozila, upravljati i navoditi drugo vozilo, zbog prekršaja, pregleda, nemogućnosti vlasnika da moralno ili zakonski vozi itd. Vozila šlep službe bi mogla navoditi vozila koja su ostala oštećena na putu, ili razdvojiti vozila u sudara, dok bi pauk vozila mogla pomjerati i navoditi automobile bez da ih dižu sa zemlje i rizikuju povrede radnika ili oštećenja na vozilima.

Također, uvođenjem 3D prostora i treće koordinate, ili prostora dubine kao dodatnog parametra u ovaj projekat, se otvaraju nove mogućnosti praćenja laserske projekcije pomoću drona ili drugih zračnih vozila.

U poglavlju implementacije softvera su ostavljeni klizači, kojima se mogu mijenjati minimalne i maksimalne vrijednosti pojedinačnih HSV komponenti, u slučaju lasera drugačije boje.

Iako ovaj rad direktno ne koristi spomenute klizače, ostavljeni su u programskom kodu kao demonstracija ili dodatna mogućnost kalibracije boje lasera za input video. Međutim, tu su i da demonostriraju još jednu od budućih ideja i načina korištenja ovog projekta. Postoji mogućnost pravljenja mobilne aplikacije, koja se bluetooth modulom spaja na robote ugrađene u npr. kolica supermarketu. Time je moguće kalibrirati robota da prati različite, individualne tipove lasera, sa bojama koje korisnik ponese od kuće, uz aplikaciju kojom to korisnik sam može uraditi. Sa time se mogu napraviti razlike u boji lasera u zavisnosti od uloge korisnika, tipa da plavi laser znači kupac, crveni laser radnik itd., dok mobilna aplikacija ima modul za svaku grupu i ulogu korisnika koji kalibrira privatni laser, ili laser od konkretne firme.

Konstrunkcijom prototipa ovog projekta se otvaraju vrata raznim mogućnostima i mnogo većim i kompleksnijim projektima, koji ovaj rad mogu koristiti kao bazu i početak za budućnost.

7. Zaključak

Kombinovanjem softvera koji koristi programski jezik c++ i OpenCV biblioteku otvorenog koda, hardvera u obliku Arduino mikrokontrolera, njegovog programskog koda i načina rada, te drugih osnovnih hardverskih komponenti, se uspješno mogao realizovati projekat navođenja robota korištenjem computer vision tehnologija.

Ovaj multidisciplinarni projekat povezuje komponente softvera i hardvera i prikazuje mnogo više od samo programskog koda i hardverskih komponenti. Prikazuje vezu između ljudskog razmišljanja, procesa donošenja odluka i kretanja sa kompjuterskim programiranjem i hardverskim komponentama. Direktno simulira čulo vida i posjedovanje mozga i razuma sa video kamerom i Arduino pločom, te programskim kodom, kao vlastitim mislima robota, kojima sopstveno donosi odluke.

Projekat služi prvenstveno kako bi pomogao i olakšao svakodnevni ljudski život, smanjio rizike povreda, te olakšao i proširio mogućnosti svih, pa i onih koji možda nisu u stanju operirati svakodnevnim objektima i situacijama zbog povreda ili nedostataka, bilo genetikom ili životnim ozljedama ili iskustvima.

Svoje predhodno znanje sam ovim zanimljivim projektom znatno proširio, upoznavajući se sa sferama hardvera kao i pisanja softvera koristeći opširne i globalno korisne biblioteke, te uveliko izašao iz svoje komfort zone.

Rezultat je jedan potpun projekat, sa velikim potencijalom za dalje širenje i nadogradnju, kako bi bio što korisniji i u budućnosti vrijedan spomena.

8. Slike

Slika 1 Lijevo: Logička reprezentacija piksela, Desno: Fizička reprezentacija piksela.....	5
Slika 2 Standardne rezoucije ekrana	6
Slika 3 Vizuelni prikaz različitih PPI vrijendosti.....	6
Slika 4 RGB spektar boja	7
Slika 5 Mapiranje RGB boja po pikselu.....	7
Slika 6 HSV model	8
Slika 7 Prikaz binarne slike	9
Slika 8 Proces konverzije RGB slike u binarnu sliku za žutu boju: a) RGB input slika b) HSV slika Podešavanje HSV vrijednosti za žutu boju d) Blnarna slika	10
Slika 9 Prikaz šuma ili buke na slici.....	11
Slika 10 Pобоljšanje slike koja болuje od nedostatka piksela koristeći median filter	12
Slika 11 Pобоljšanje буčne slike koristeći Gausovo zamagljenje	13
Slika 12 Eroziја tamno plavog kvadrata u svijetlo plavi kvadrat koristeći predefinsane granic	14
Slika 13 Dilataciја tamno plavog kvadrata u svijetlo plavi kvadrat, popunjavanjem prostora u određenim granicama za zaobljenim oblikom	15
Slika 14 Prikaz različitih "frames per second" projekciја slike.....	16
Slika 15 Postavke Visual Studio instalera prije kreiranja projekta.....	17
Slika 16 Keiranje CLR konzolne aplikacije unutar Visual Studia.....	18
Slika 17 Oficijelni logo OpenCV biblioteke	19
Slika 18 Dodavanje globalne putanje za build folder OpenCV biblioteke	20
Slika 19 Dodavanje putanje za binarne datoteke.....	20
Slika 20 Configuration manager za x64 bitno okruženje.....	21
Slika 21 Dodavanje include datoteka u razvojno okruženje koristeći globalnu putanju do build foldera	21
Slika 22 Dodavanje "opencv_world310d.lib" datoteke iz lib foldera biblioteke	22
Slika 23 Dodavanje lib datoteka u razvojno okruženje koristeći globalnu putanju do build foldera	22
Slika 24 Prikaz slike kao matrice unutar Mat objekta	23
Slika 25 Spisak svih kamera mašine unutar "Device Manager"-a.....	25
Slika 26 Matrica slike određena dvodimenzionalnim koordinatnim sistemom	29
Slika 27 RGB vrijednosti za najčešće korištene boje	30
Slika 28 Prikaz prozora sa klizačima minimalnih i maksimalnih vrijendosti.....	35
Slika 29 Prikaz BGR slike u HSV spektru	41
Slika 30 Konverzija od BGR, preko HSV do binarne slike.....	42
Slika 31 Prikaz binarne slike prije i poslije morfološke erozije	43
Slika 32 Prikaz binarne slike iz stanja erozije lijevo, do stanja dilatacije desno.....	44
Slika 33 Formule za izračunavanje x i y koordinata koristeći momente objekta na slici, određenih kontura	46
Slika 34 Prikaz originalne slike sa nišanom i koordinatama, HSV slike, te binarne slike sa laserskom projekcijom	50
Slika 35 Detaljniji prikaz nišana i koordinata na originalnoj slici.....	50

Slika 36 Prikaz smjernica za definisanje pozicije projekcije	52
Slika 37 Prikaz odnosa brzine kretanja unaprijed i pozicije laserske projekcije	55
Slika 38 Prikaz odnosa brzine kretanja unaprijed i pozicije laserske projekcije uz skretanje uvlijevo ili udesno	55
Slika 39 Prikaz stanja i poruka prilikom ispravne pozicije projekcije, nepostojanja projekcije i pronalaženja više od jedne validne projekcije	55
Slika 40 Prikaz široko impulske modulacije - PMW	61
Slika 41 Fizički prikaz i shema Arduino Mega 2560 ploče	62
Slika 42 Lijevo: Odabir Arduino ploče; Desno: Odabir porta	64
Slika 43 Odabir i kreiranje primjer programa za Arduino platformu	64
Slika 44 Lijevo: Odabir biblioteka za dodavanje; Desno: Library Manager	65
Slika 45 Serial Monitor	65
Slika 46 Komande za provjeravanje i upload	66
Slika 47 Shema: tlocrt platforme	72
Slika 48 Shema: bokocrt platforme	73
Slika 49 Shema: nacrt platforme	73
Slika 50 Lijevo: set lasera korišten za projekat; Desno: Venus USB2.0 Camera korištena za projekat	74
Slika 51 Fizički prikaz i testiranje platforme	75

9. Programski kodovi

Programski kod 1 Pseudo kod median filtera	13
Programski kod 2 Spisak svih pojedinačnih biblioteka i namespace-a za c++ i OpenCV na vrhu projekta	22
Programski kod 3 Osnovne funkcionalnosti objekata tipa Mat	24
Programski kod 4 Deklaracija video capture objekta	25
Programski kod 5 Otvaranje stražnje kamere, druge na spisku	26
Programski kod 6 Provjera uspostavljanja konekcije sa kamerom	26
Programski kod 7 Definisanje visine i širine videa	26
Programski kod 8 Kompletni postupak prikazivanja videa na ekran pomoću OpenCV biblioteke	28
Programski kod 9 Druga opcija preuzimanja frame-a: preklopni opeator unosa	28
Programski kod 10 Terminiranje petlje koristeći unos korisnika i waitKey funkciju	28
Programski kod 11 Kreiranje objekta tipa Point	30
Programski kod 12 Primjer klase Scalar za različite boje	30
Programski kod 13 Parametri i primjer crtanja linije na ekran	31
Programski kod 14 Parametri i primjer crtanja kruga na ekran	31
Programski kod 15 Parametri i primjer crtanja teksta na ekran	32
Programski kod 16 Inicijalne minimalne i maksimalne vrijednosti za HSV filter u binarnu sliku	33
Programski kod 17 Funkcija za kreiranje klizača	34
Programski kod 18 Poziv funkcije za kreiranje klizača	34

Programski kod 19 Definisanje broja maksimalnih objekata na ekranu	36
Programski kod 20 Definisanje minimalne površine objekta	36
Programski kod 21 Dimenzije prikazne slike	36
Programski kod 22 Definisanje maksimalne površine objekta	37
Programski kod 23 Konstante imena svih prozora	38
Programski kod 24 Zsatave za korištenje morfologije i praćenja projekcije	38
Programski kod 25 Deklaracija svih Mat objekata za projekat	39
Programski kod 26 Deklarisanje i inicijalizacija varijabli za čuvanje vrijednosti koordinata laserske projekcije	39
Programski kod 27 Postupak deklarisanja i započinjanja video prenosa	40
Programski kod 28 Konvertovanje BGR u HSV sliku koristeći cvtColor() funkciju	40
Programski kod 29 Median filter nad HSV slikom	40
Programski kod 30 Konvertovanje HSV slike u binarnu sliku koristeći inRange() funkciju	42
Programski kod 31 Poziv funkcije za morfološke operacije nad binarnom slikom	42
Programski kod 32 Funkcija morfoloških operacija: kreiranje morfoloških objekata za eroziju i dilataciju	43
Programski kod 33 Funkcija morfoloških operacija: erozija binarne slike koristeći erode() funkciju	43
Programski kod 34 Funkcija morfoloških operacija: dilatacija binarne slike koristeći dilate() funkciju	44
Programski kod 35 Poziv funkcije za praćenje lasera nad modifikovanom binarnom slikom ..	45
Programski kod 36 Kopiranje binarne slike u pomoćni objekat	45
Programski kod 37 Deklarisanje potrebnih vektora, varijabli i pronalazak objekata i kontura objekata na binarnoj slici	47
Programski kod 38 Algoritam za pronalaženje i praćenje laserske projekcije	48
Programski kod 39 Funkcija za konvertovanje integer broja u string	49
Programski kod 40 Funkcija za crtanje nišana i ispis vrijednosti koordinata na sliku	49
Programski kod 41 Prikaz svih prozora	50
Programski kod 42 Crtanje smjernica za poziciju laserke projekcije	52
Programski kod 43 Određivanje i ispis brzine i načina kretanja robota	54
Programski kod 44 Namespaces za potrebne CLR objekte	56
Programski kod 45 Otvaranje porta za serijsku komunikaciju sa Arduinoom	57
Programski kod 46 Kreiranje i slanje komandnog stringa serijskom portu	58
Programski kod 47 Inicijalizacija prazne Arduino skice	66
Programski kod 48 Definisanje svih konstanti i globalnih varijabli	68
Programski kod 49 Funkcija setup()	68
Programski kod 50 Funkcija serialEvent()	69
Programski kod 51 Funkcija loop(): ekstraktovanje i zaustavljanje	70
Programski kod 52 Funkcija loop(): analiza koordinata i slanje signala pinovima / točkovima	71

10. Literatura

- [1] Margaret Rouse, „Pixel“, [Na mreži]. Dostupno: <https://whatis.techtarget.com/definition/pixel>, [Posljednji pristup 19.02.2020].
- [2] “Visual Studio product family documentation”, [Na mreži]. Dostupno: <https://docs.microsoft.com/en-us/visualstudio/?view=vs-2019>, [Posljednji pristup 19.02.2020].
- [3] „OpenCV“, [Na mreži]. Dostupno: <https://opencv.org/>, [Posljednji pristup 19.02.2020].
- [4] “OpenCV Documentation”, [Na mreži]. Dostupno: <https://docs.opencv.org/2.4/>, [Posljednji pristup 19.02.2020].
- [5] Jeff Tyson, „How Serial Ports Work“, [Na mreži]. Dostupno: <https://computer.howstuffworks.com/serial-port.htm>, [Posljednji pristup 19.02.2020].
- [6] „Arduino“, [Na mreži]. Dostupno: <https://www.arduino.cc/>, [Posljednji pristup 19.02.2020].
- [7] “PlatformIO”, [Na mreži]. Dostupno: <https://platformio.org/>, [Posljednji pristup 19.02.2020].
- [8] „Visual Studio Code“, [Na mreži]. Dostupno: <https://code.visualstudio.com/>, [Posljednji pristup 19.02.2020].
- [9] “AllDataSheet”, [Na mreži]. Dostupno: <https://www.alldatasheet.com/>, [Posljednji pristup 19.02.2020].
- [10] S. Habota „Laser_tracking_robot“, [Na mreži]. Dostupno: https://github.com/SamirHabota/Laser_tracking_robot, [Posljednji pristup 19.02.2020].