Primjena računalnog vida u mehatronici

Tropčić, Tomislav

Primjena računalnog vida u mehatronici

Tropčić, Tomislav

Veleučilište u Karlovcu

Strojarski odjel

Stručni studij mehatronike

Tomislav Tropčić

Primjena računalnog vida u mehatronici

ZAVRŠNI RAD

Karlovac, 2020. godina

Karlovac University of Applied Sciences

Mechanical Engineering Department

Professional undergraduate study of Mechatronics

Tomislav Tropčić

Application of computer vision in mechatronics

Final paper

Karlovac, 2020

Veleučilište u Karlovcu

Strojarski odjel

Stručni studij mehatronike

Tomislav Tropčić

Primjena računalnog vida u mehatronici

ZAVRŠNI RAD

Mentor: dr. sc. Adam Stančić, viši predavač

Karlovac, 2020. godina

PREDGOVOR

Izjavljujem da sam završni rad na temu „Primjena računalnog vida u mehatronici“ izradio samostalno koristeći znanje stečeno tijekom studija, navedenu literaturu te samostalnim istraživanjem.

Zahvaljujem se mentoru dr. sc. Adamu Stančiću na savjetima i pomoći pri izradi ovog rada.

SAŽETAK

Računalni vid i strojno učenje su među brzo rastućim i popularnim granama računarstva, ponajviše zbog razvoja računalnog hardware-a i većih brzina obrade podatka. Već i mikroračunala poput Raspberry Pi sustava su dovoljno brza za računalni vid i strojno učenje.

U radu je prikazana izrada prototipa sustava kugle na ploči, koji je popularan problem u teoriji upravljanja. Prezentirani sustav je dobra podloga za testiranje i drugih kontrolnih metoda.

Korišten je programski jezik Python jer je jednostavan za učenje i implementaciju, te nudi podršku za OpenCV biblioteku otvorenog koda koja je namijenjena za računalni vid i strojno učenje.

Tehnologija 3D tiskanja postaje sve jeftinija te nudi brzu izradu dijelova. Spojiti se može s CAD programima poput SolidWorks-a gdje se dijelovi mogu dizajnirati, složiti u cjelinu i analizirati, te se relativno jednostavno može doći do traženih rješenja.

Arduino platforma nudi jednostavna mikrokontrolerska rješenja te se s velikim brojem biblioteka može izbjeći dodatno pisanje koda.

Sve navedene aktivnosti i korištena programska podrška odnose se na cjelokupno područje mehatronike: strojarstvo, elektrotehniku i računarstvo.

Ključne riječi: Arduino, Kugla na ploči, OpenCV, Python, Računalni vid, 3D tisak

SUMMARY

Computer vision and machine learning are among fast growing and popular fields in computer science. That is mostly because of advances in computer hardware and increases in speed of data processing.

The topic of the thesis is building a prototype of a ball on plate system, which is a popular control problem. The same system is also good for testing other control strategies.

Python programming language was used because it's simple to learn and implement, and it has packages for OpenCV- an open source library for computer vision and machine learning.

3D printing technology is getting cheaper and parts can be constructed fast. Combined with computer aided design programs like Solidworks, parts can be designed, assembled and analyzed so that it's relatively simple to come up with a desired solutions.

Arduino platform offers simple microcontroller solutions, and combined with lots of libraries there is no need to write additional lines of code.

All the activities and software used are related to the whole field of mechatronics: mechanical engineering, electrical engineering and computer science.

Key words: Arduino, Ball on plate, Computer vision, OpenCV, Python, 3D printing

SADRŽAJ

POPIS SLIKA

1. UVOD

Tema završnog rada je primjena računalnog vida na prototipu sustava kugle na ploči. Jedna opcija je bila izrada istog sustava sa zaslonom osjetljivim na dodir kao senzorom za određivanje položaja kugle, ali kako je to zahtijevalo čekanje naručenih dijelova, i zbog veće mase dijelova komplicira izradu prototipa, autor se odlučio za primjenu računalnog vida.

S osnovama računalnog vida i OpenCV biblioteke autor se upoznao rješavanjem primjera s Internet stranice PyImageSearch [1]. Algoritam za lokalizaciju objekta temelji se na radu s iste stranice [2]. Također su korišteni različiti on-line izvori podataka i stručna literatura [3] [4] [5] [6] [7] [8].

Za izradu konstrukcijskih elemenata korištena je tehnika 3D tiskanja. Nakon prvih testiranja sustava, utvrđeno je da bi grafičko sučelja znatno olakšalo upravljanje sustavom pa je isto izrađeno korištenjem PyQt5 biblioteke. U nastavku je opisan pojam, povijest i područja primjene računalnog vida.

1.1. Računalni vid

Oko je najvažnije ljudsko osjetilo jer njime primamo 90% svih informacija iz okoline [9]. Od rođenja vizualnim putem primamo mnoštvo informacija te tako gradimo našu sliku o svijetu. Zahvaljujući tom iskustvu, za nas se dobivanje informacija vidom čini vrlo jednostavnim, ali za računala je to kompleksan zadatak. Računalo „vidi“ sliku tek kao matricu brojeva, iz koje raznim tehnikama i algoritmima dobiva informacije [10].

Računalni vid je interdisciplinarno područje koje se bavi računalnim razumijevanjem sadržaja vizualnih podataka. Da bi to bilo moguće, veže se na mnoga područja znanosti i inženjerstva. Za razumijevanje kako lomom svjetlosti nastaje slika u senzorima potrebna su znanja iz područja fizike, biologija i psihologija opisuju kako ljudski mozak vidi i obrađuje vizualne informacije, te računarstvo, matematika i druga područja inženjerstva koja grade računalne sustave i razvijaju algoritme za računalni vid [10].

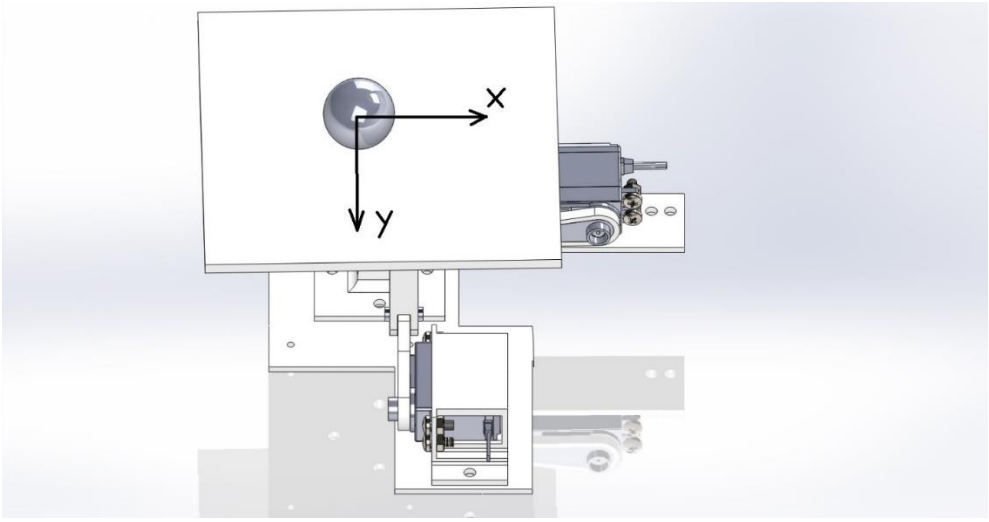
Počeci računalnog vida sežu u 60-te godine 20.stoljeća, s počecima razvoja umjetne inteligencije. 1966. godine na sveučilištu MIT započinje „The Summer Vision Project“. Cilj je bio da preko spojene kamere računalo opiše što „vidi“. Područje računalnog vida se razlikovalo od tada već zastupljenog područja digitalne obrade slike po tome što je cilj računalnog vida bio izdvajanje trodimenzionalnih struktura iz slika s ciljem potpunog razumijevanje scene. U 70-tim godinama postavljeni su temelji za mnoge algoritme računalnog vida koji se koriste i danas, uključujući raspoznavanje linija, prikazivanje objekata kao spoj manjih struktura, optički tok (engl. optical flow) i drugi. U 80-tim godinama je fokus bio na sofisticiranijim matematičkim metodama korištenim za kvantitativnu analizu slike i scene. Razvijeni su algoritmi za dobivanje trodimenzionalnog oblika iz slike površine (engl. shape from shading), detekciju rubova (engl. edge detection), rekonstrukciju površine (engl. regularization based surface reconstruction) i drugi. U 90-tim godinama dolazi do značajne promjene preklapanjem područja računalnog vida i računalne grafike. Razvijeni su algoritmi za raspoznavanje lica, praćenje lica, segmentaciju slike i drugi. Zadnjih godina je značajan napredak postignut spajanjem računalnog vida s područjem strojnog učenja. Velike količine podataka s Interneta omogućuju da računala raspoznaju objekte s velikom točnošću i bez nadzora ljudi [11].

Primjena u mehatronici i strojarstvu se može pronaći u području strojnog vida (engl. machine vision), gdje se koristi za npr. automatsku inspekciju proizvodne linije ili navigaciju robotske ruke.

Također se koristi u medicini (detekcija tumora, mjerenje dimenzija organa), vojnoj tehnologiji (detekcija neprijateljskih vojnika i vozila, navigacija projektila), za navigaciju autonomnih vozila i mobilnih robota, brojanje ljudi, itd. [10].

2. PROJEKTIRANJE SUSTAVA KUGLE NA PLOČI

Cilj sustava je kontrola položaja kugle na ravnoj ploči korištenjem sustava s dva stupnja slobode gibanja. Kugla se pod utjecajem gravitacije kotrlja te se nagibom ploče kontrolira njen položaj, brzina i ubrzanje. Sustav koristi senzor koji će snimiti položaj kugle, te se koristi USB kamera koja je u 3D tiskanom kućištu montirana na aluminijski okvir, tako da snima situaciju iznad ploče na kojoj se nalazi kugla. Snimka se šalje na računalo te se u Python skripti pomoću OpenCV biblioteke određuje položaj kugle. Sukladno tome PID (proporcionalno-integralno-derivacijski) kontroler računa kuteve ravne ploče te ih serijskom vezom šalje Arduino pločici. Dva servo motora su međusobno pod kutem od 90 stupnjeva i svaki upravlja nagibom ploče na jednoj osi. U sredini je oslonac koji omogućuje rotaciju ploče. Arduino pločica pulsno-širinskom modulacijom kontrolira kuteve servo motora, koji su preko 3D tiskanih poluga i magnetskih kugličnih zglobova povezani s ravnom pločom.



Slika 1: Predodžba kretanja kugle na ravnoj ploči

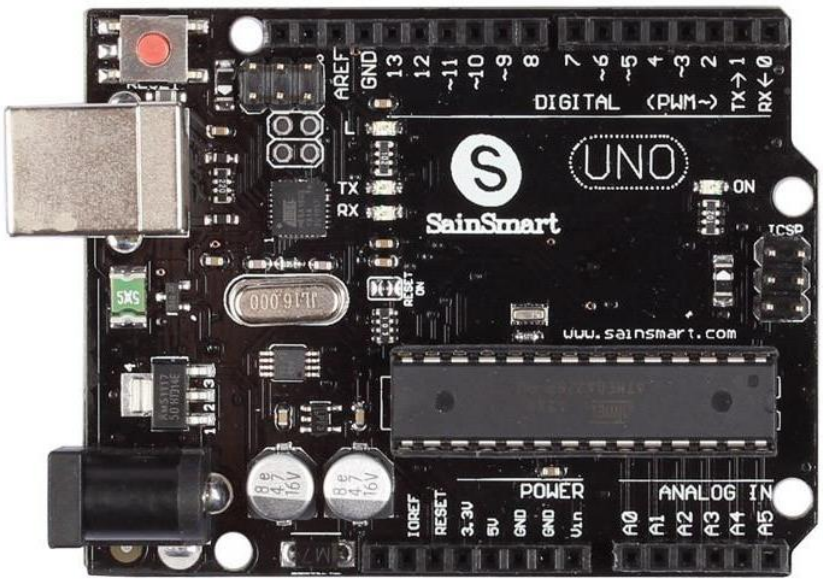
Na slici iznad je prikazan dio sustava i strelicama je prikazano kretanje kugle. Kugla se po ravnoj ploči giba u smjeru osi x i y te se pretpostavlja da zakretanje ploče na jednoj osi nema utjecaj na gibanje kugle po drugoj osi. Zato se sustav može promatrati kao dva odvojena jednoosna sustava.

2.1. Komponente sustava

Upravljačka jedinica sustava je računalo na kojemu se pokreće Python skripta. Arduino Uno pločica služi kao veza između računala i aktuatora. Za aktuatore su odabrani servo motori radi jednostavnosti i točnosti upravljanja. Korištena je USB kamera bez kućišta kako bi se isto moglo izraditi prema potrebama. U nastavku je dan kratak opis karakteristika i funkcionalnosti svake komponente.

2.1.1. Arduino Uno

Arduino Uno je open-source (engl. open-source označava sustav otvorenog koda) razvojna pločica temeljena na Microchip ATmega328P mikrokontroleru. U prezentiranom završnom radu korištena je SainSmart Aduino Uno pločica [12] , koja je prikazana na slici ispod:



Slika 2: SainSmart Arduino Uno [12]

Ima 14 digitalnih ulazno-izlaznih pinova, od kojih 6 mogu biti PWM (engl. pulse-width modulation) izlazi te 6 analognih ulaza. Keramički rezonator daje takt od 16Mhz. Napaja se preko USB kabela ili vanjskog napajanja od 7 do 20V, koje se preko regulatora napona spušta na 5V za rad mikrokontrolera ili napajanje drugih krugova. Pored navedenog, Arduino Uno ima 3.3V

regulator napona. Za pohranu programa Arduino Uno ima 32kB flash memorije, 2kB SRAM memorije za radnu memoriju te 1kB EEPROM memorije za čuvanje podataka nakon prekida napajanja. Maksimalna izlazna jakost struje pojedinog pina na naponu od 5V iznosi 20mA te 50mA na naponu od 3.3V. Arduino Uno ima I2C, SPI i UART module za komunikaciju.

Za serijsku komunikaciju mikrokontroler koristi UART TTL 5V na digitalnim pinovima 0 i 1. Za komunikaciju putem USB sučelja koristi dodatni ATmega16U2 mikrokontroler implementiran na pločici [13].

2.1.2. USB kamera

Za snimanje pozicije kugle na ravnoj ploči je odabrana ELP-USBFHD01M kamera [14].

Slika 3: USB kamera ELP-USBFHD01M [14]

Na slici iznad je dan prikaz odabrane USB kamere. Kamera pri maksimalnoj rezoluciji od 1920x1080 piksela snima 30 slika u sekundi, a pri rezoluciji 640x480 piksela snima 120 slika u

sekundi. Na računalo se spaja USB kabelom, preko kojega se napaja i šalje podatke. Male dimenzije i mala potrošnja čine je pogodnom za ugradnju u razne sustave.

2.1.3. Servo motor

Servo motor je rotacijski ili linearni aktuator koji služi za preciznu kontrolu položaja. Ima regulacijski krug koji mjeri položaj vratila i putem povratne veze radi korekcije položaja. Jednostavni servo motori poput ovog korištenog u radu imaju ugrađeni regulacijski krug i koriste potenciometar da bi mjerili položaj vratila. Vratilo je spojeno na reduktor koji smanjuje broj okretaja i povećava moment motora [15].

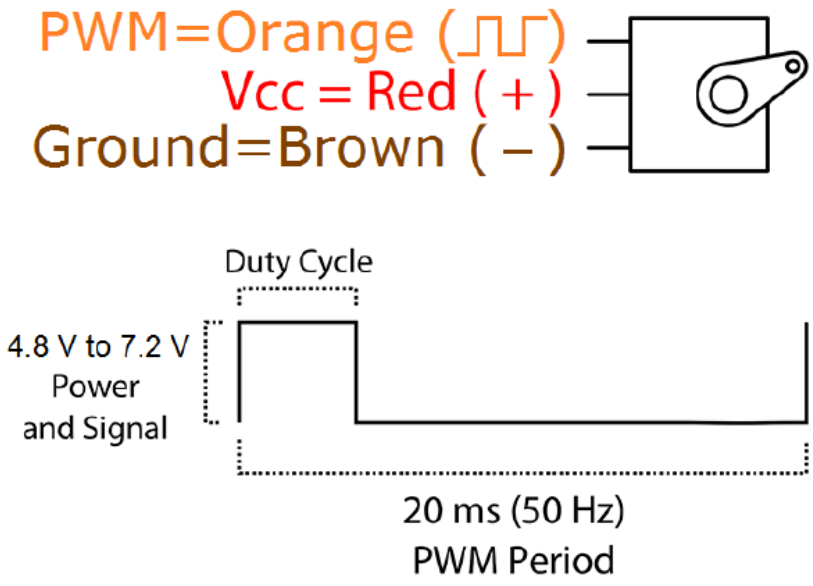
Radi niske cijene i dobrih specifikacija, odabran je rotacijski servo motor TowerPro MG995 [16]. Navedeni rotacijski servo motor prikazan je na slici ispod:

Slika 4: Servo motor TowerPro MG995 [17]

Specifikacije rotacijskog servo motora:

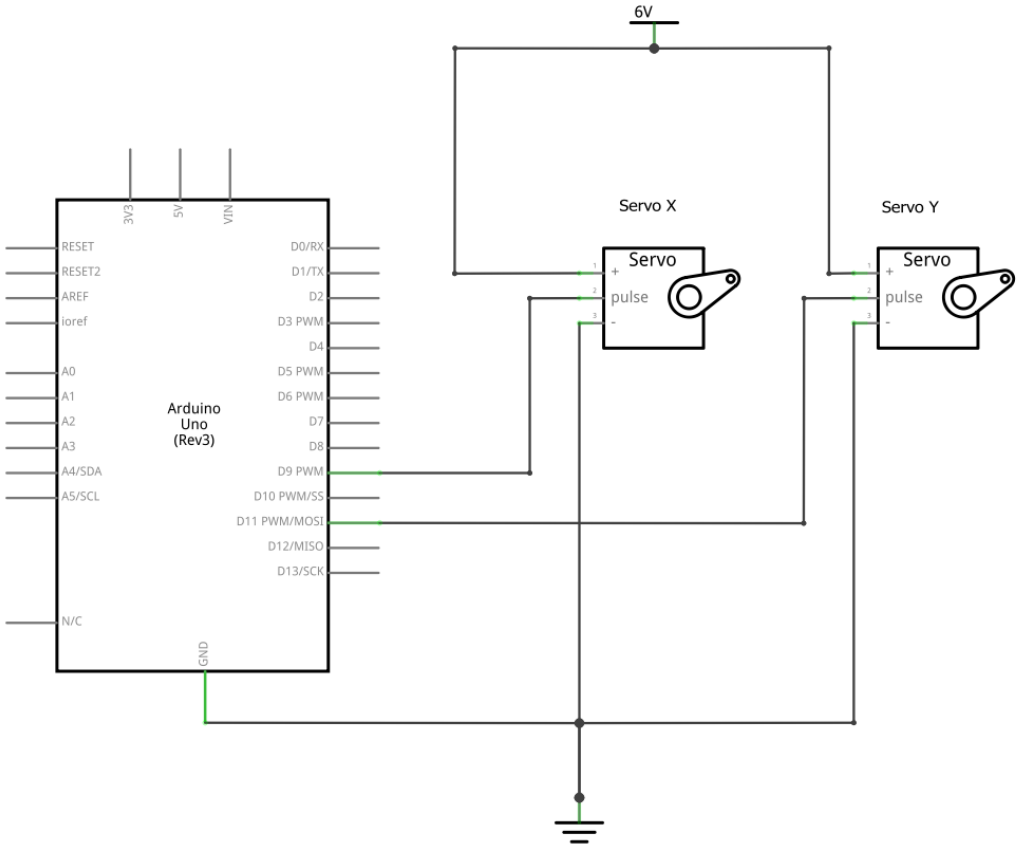
- Maksimalni kut zakreta: 120° (60° u svakom smjeru)
- Brzina zakreta: 0.20s/60° (4.8V), 0.16s/60° (6V)
- Radni napon: 4.8V – 7.2V
- Moment: 9.4kg/cm(4.8V) - 11kg/cm(6V)
- Dimenzije: 40mm x 20mm x 43mm
- Masa: 55g
- Radna struja bez opterećenja: 170mA
- Maksimalna radna struja: 1200mA

Servo motorom se upravlja pulsno-širinskom modulacijom. Motor ima tri izvoda, dva za napajanje te jedan signalni izvod. Signalnom žicom se periodički šalje puls. Period (engl. PWM period) ovisi o tipu servo motora, a za korišteni motor iznosi 20ms. Ovisno o dužini impulsa (engl. duty cycle), vratilo će zakrenuti za određeni kut.



Slika 5: Pulsno-širinska modulacija [16]

Kako Arduino pločica nema dovoljno snage za napajanje servo motora, izvedeno je vanjsko 6V napajanje preko DC konektora (engl. DC power jack) [18]. Shema spajanja Arduino pločice, servo motora i vanjskog napajanja dana je na slici ispod.



Slika 6: Shema spajanja Arduino pločice, servo motora i vanjskog napajanja

Signalni izvod servo motora koji kontrolira položajem kugle na ravnoj ploči u x smjeru je na Arduino pločici spojen na digitalni pin 9, a signalni izvod servo motora koji kontrolira položajem kugle u y smjeru je spojen na digitalni pin 11. Izvodi za napajanje servo motora spojeni su na vanjsko napajanje od 6V. Za izjednačavanje potencijala je masa na Arduino pločici (izvod GND) spojena s masom na vanjskom izvoru napajanja.

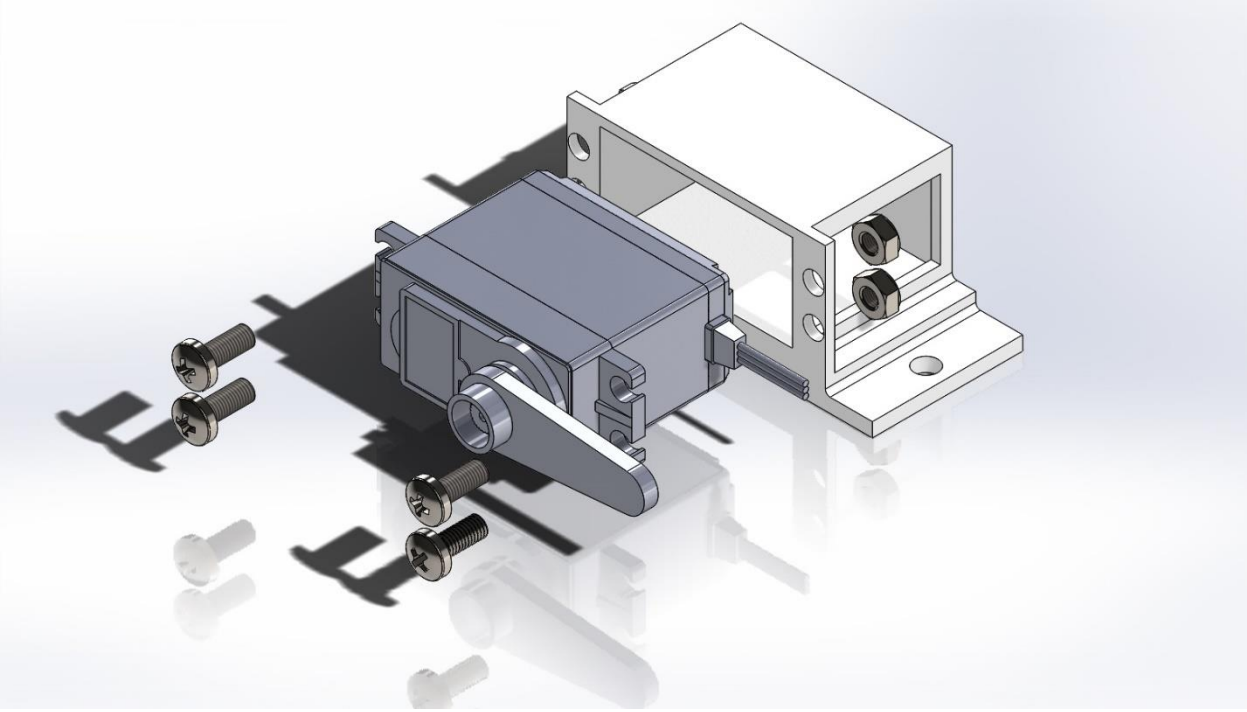
2.2. 3D tiskani dijelovi sustava

Radi bržeg razvoja prototipa odabrana je tehnika 3D tiskanja. Dijelovi su dizajnirani u DSS Solidworks 2017 radi vizualizacije, tiskani i testirani. Tako da se malo po malo došlo do rješenja.

Tiskana su kućišta servo motora, poluge, ploča, kućište kamere, spojnice okvira, postolja, oslonac, ploča za pozicioniranje i ploča za Arduino i DC priključak. U nastavku teksta bit će dan detaljniji opis dizajniranih dijelova.

2.2.1. Kućište servo motora

Kako bi se servo motori mogli učvrstiti za podlogu te da se spriječe pomaci i vibracije, dizajnirano je kućište za servo motor.



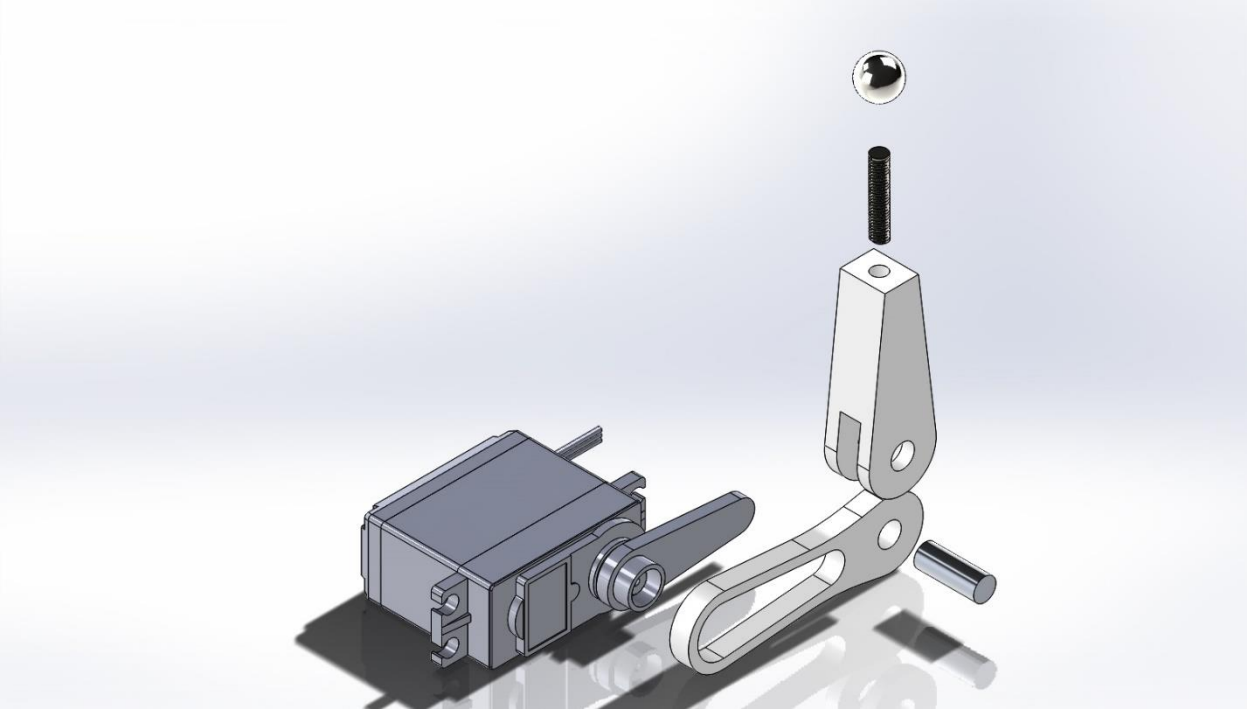
Slika 7: Predodžba spajanja servo motora i kućišta

Na slici iznad je u eksplodiranom prikazu dana predodžba spajanja servo motora i kućišta vijčanim spojem. Servo motori se montiraju unutar kućišta i učvrste s četiri M4x10mm vijka i maticama,

pri čemu se strana servo motora koja sadrži izvode (nisu prikazani na slici) poklapa sa stranom na kućištu koja ima urez. Sa svake strane je provrt kojim se kućište montira na podlogu.

2.2.2. Poluge

Za prijenos momenta sa servo motora na ploču dizajnirane su poluge koje se montiraju na servo ruke.



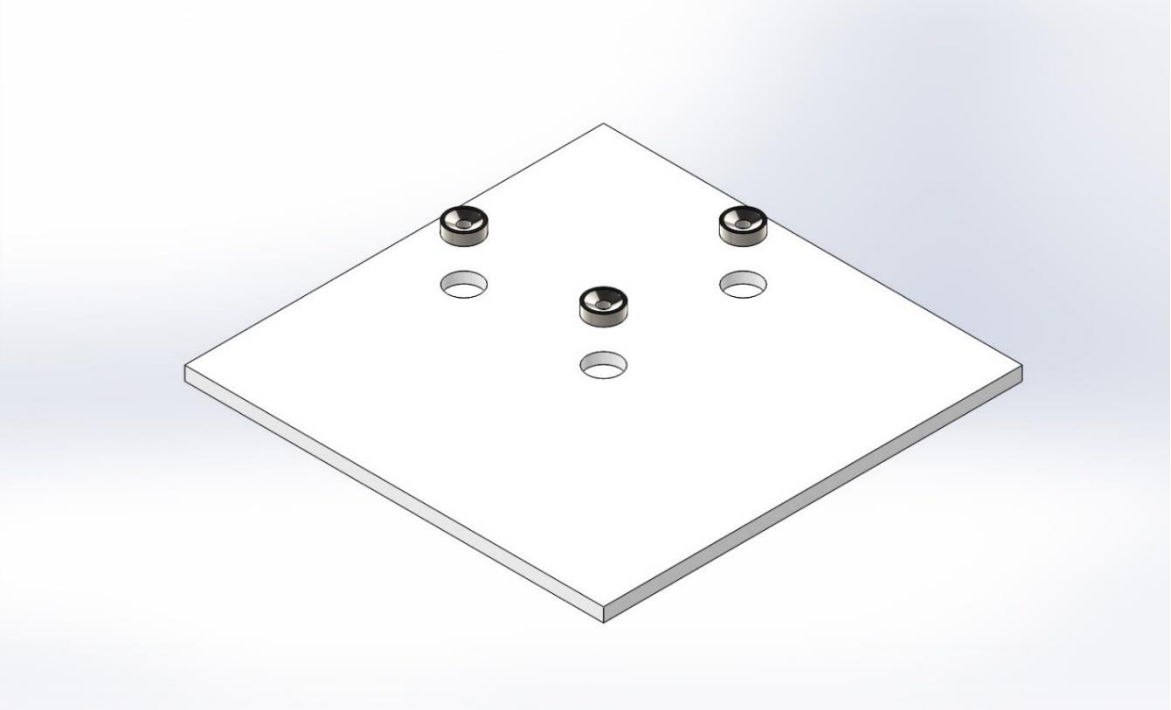
Slika 8: Predodžba spajanja servo ruke, poluga, vijka i kuglice

Na slici iznad je u eksplodiranom prikazu dana predodžba spajanja servo ruke, poluga, vijka i čelične kuglice. Granična mjera utora je malo veća od granične mjere servo ruke te namještanjem i uporabom sile servo ruka usjeda u utor u poluzi. Kako se ne prenose veliki momenti, spoj se pokazao dovoljno dobar za potrebe rada. Poluge su međusobno spojene okruglim aluminijskim profilom. Granične mjere provrta na polugama su malo veće od granične mjere aluminijskog

profila te se klizanjem poluga po profilu kružno gibanje vratila servo motora pretvara u vertikalno gibanje poluge. U uvrt na vrhu poluge se uvine vijak bez glave te na njega čelična kuglica.

2.2.3. Ploča

Ploča je dimenzije 150x150mm, debljine 5mm. Površina ploče je dovoljno velika za testiranje sustava, a opet dovoljno mala da sustav bude kompaktan.

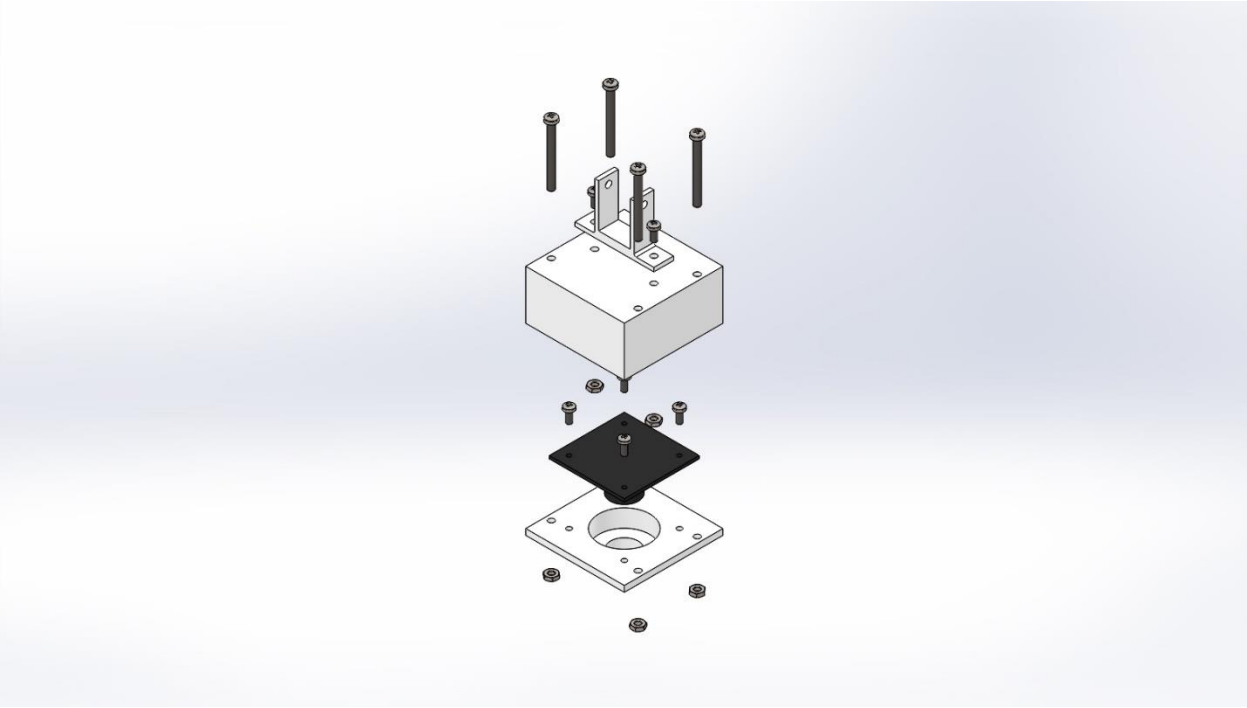


Slika 9: Predodžba spajanja ploče i magneta

Na slici iznad je u eksplodiranom prikazu dana predodžba spajanja ploče i magneta. Magnetski kuglični ležaj [19] dozvoljava rotaciju kuglice u svim smjerovima. Za magnete su na ploči dizajnirani utori čija je granična mjera ista kao granična mjera magneta. Magneti se uporabom sile utisnu i tako fiksiraju. Središnji magnet montira se na kuglicu na osloncu, a dva sa strane, međusobno pod kutem od 90 stupnjeva, na kuglice koje su montirane na poluge.

2.2.4. Kućište kamere

Da bi se kamera mogla montirati na aluminijski profil i zaštititi od utjecaja okoline, dizajnirano je kućište.

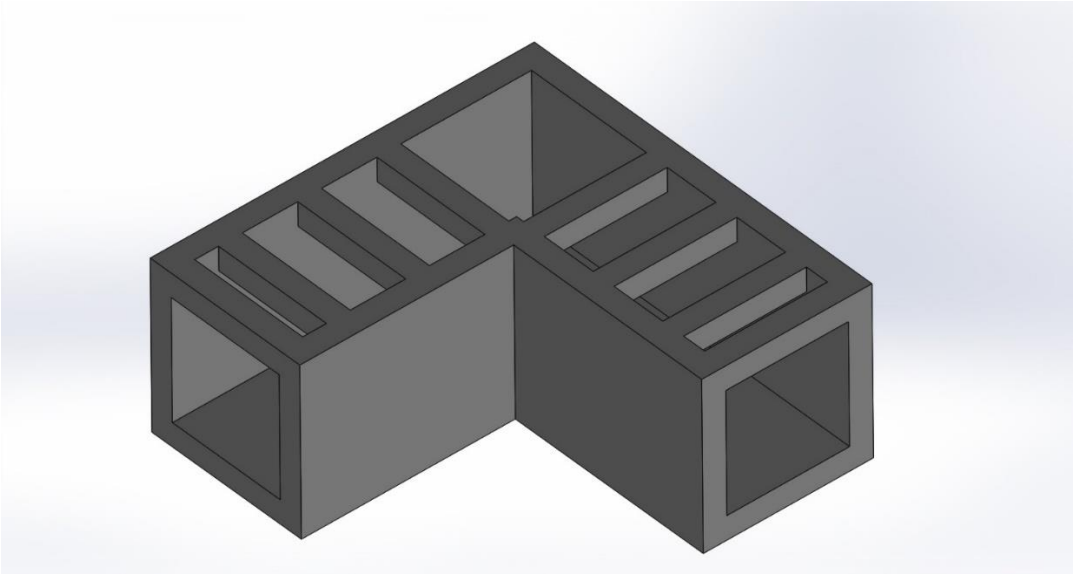


Slika 10: Predodžba spajanja kamere, kućišta i nosača

Na slici iznad je u eksplodiranom prikazu dana predodžba spajanja kamere, kućišta i nosača. Kućište kamere se sastoji od donjeg i gornjeg poklopca. Kamera se na donji poklopac pričvrsti s četiri M2.5x6mm vijka koji se uvinu u uvrte na donjem poklopcu. Nosač se s dva M3x10mm vijka i maticama pričvrsti na gornji poklopac. Zatim se gornji i donji poklopac spoje s četiri M3x30mm vijka i maticama.

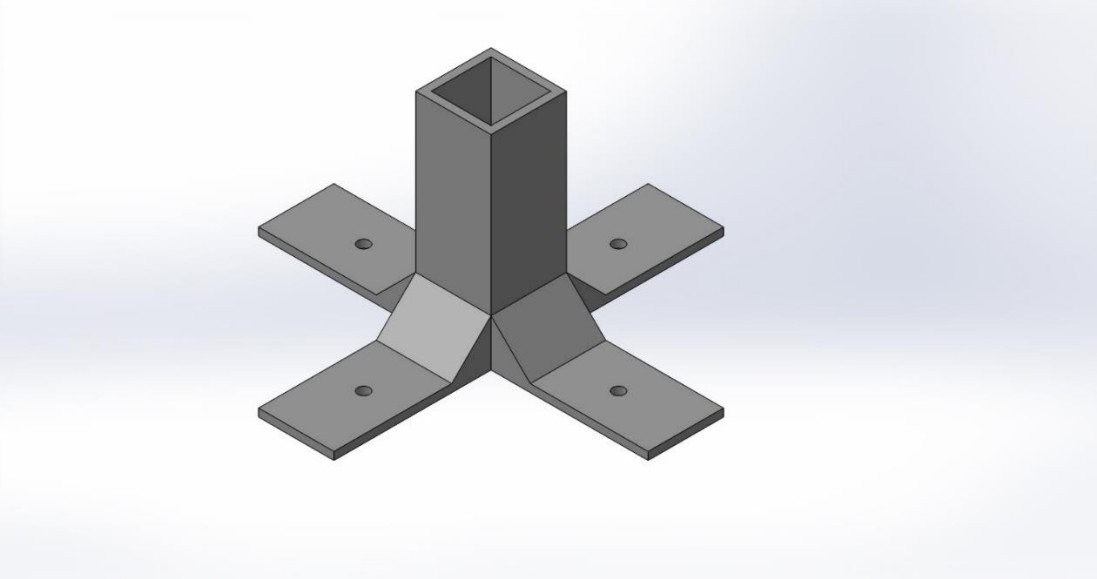
2.2.5. Ostali dijelovi

Od ostalih dijelova su dizajnirane spojnice okvira, postolja koja nose okvir, oslonac, ploča kojom se pozicioniraju servo motori i oslonac, te ploča za Arduino i DC priključak.



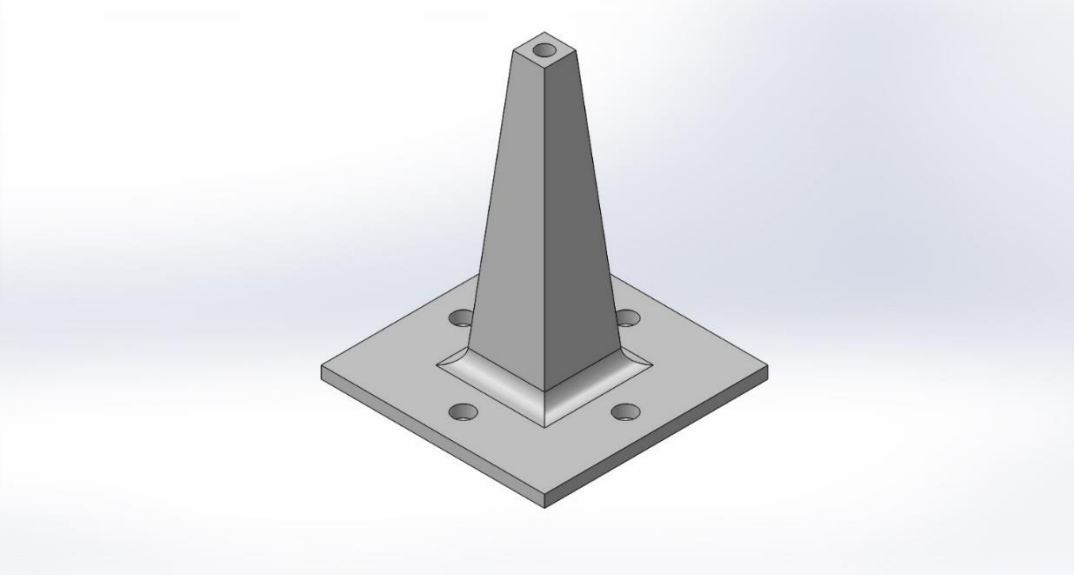
Slika 11: Spojnica okvira

Na slici iznad je dan prikaz spojnice okvira. Aluminijski kvadratni profili dimenzija 15x15mm se spajaju pod kutem od 90 stupnjeva te su u tu svrhu dizajnirane spojnice. Prorezi na gornjoj strani su dizajnirani kako ni se nakon 3D tiskanja mogle ukloniti potporne strukture.



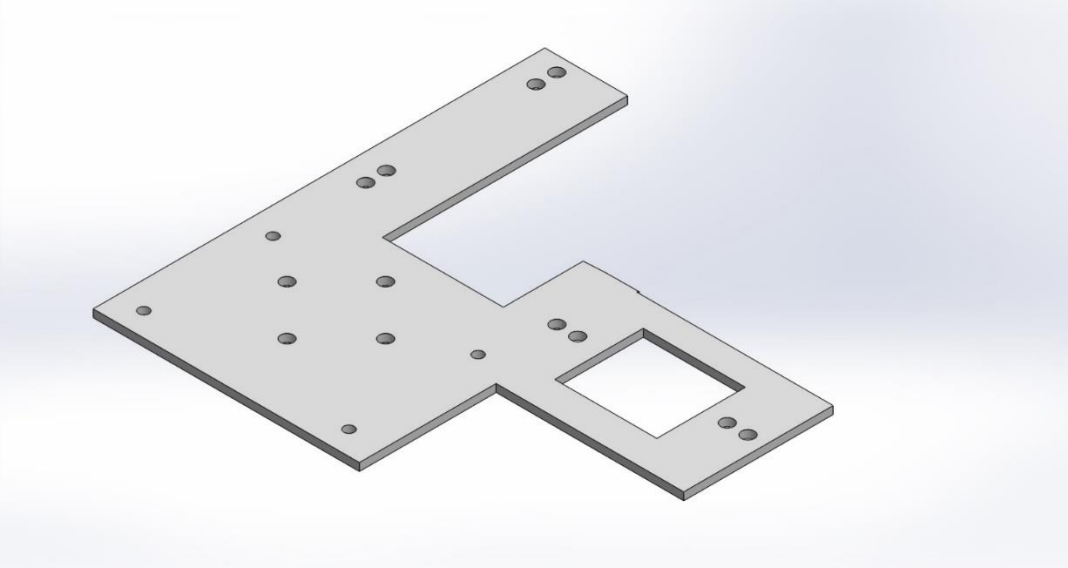
Slika 12: Postolje

Na slici iznad je dan prikaz postolja. Okvir, koji je spoj aluminijskih profila, se kroz provrte na postolju vijcima montira na podlogu sustava.



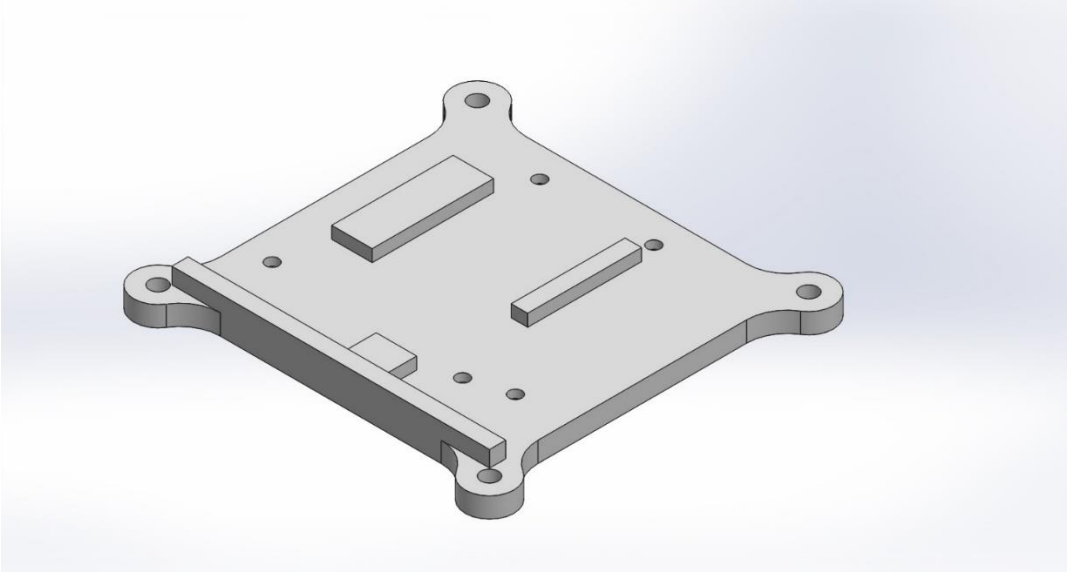
Slika 13: Oslonac

Na slici iznad je dan prikaz oslonca. Na vrhu se kao kod poluga vijčanim spojem montira čelična kuglica. Na bazi ima četiri provrta za montažu na podlogu sustava.



Slika 14: Ploča za pozicioniranje

Na slici iznad je dan prikaz ploče za pozicioniranje. Dizajnirana je tako da su servo motori međusobno pod kutem od 90 stupnjeva i da bi pri horizontalnom položaju servo ruke poluga bila u približno vertikalnom položaju. Također ima provrte kojima se oslonac pozicionira na željeni položaj i provrte kojima se montira na podlogu sustava. Od početnog pravokutnog oblika se došlo do konačnog kako bi se prilikom 3D tiskanja uštedjelo na materijalu.

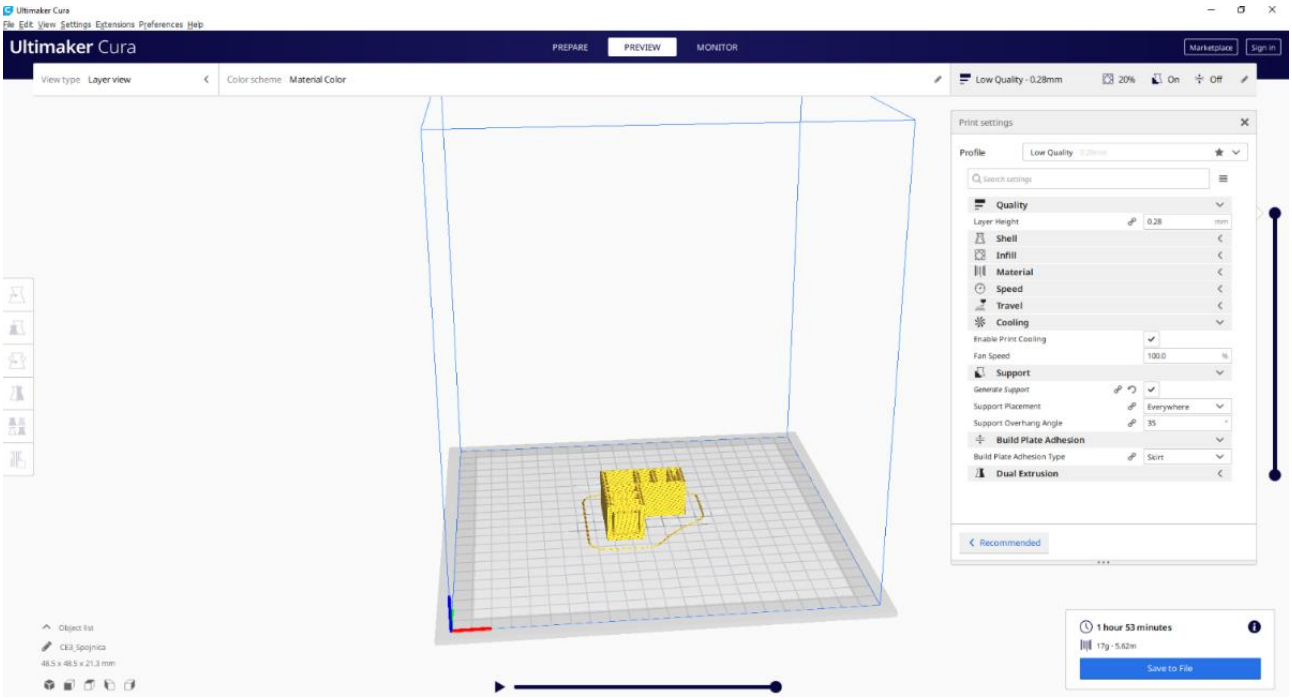


Slika 15: Ploča za Arduino i DC priključak

Na slici iznad je dan prikaz ploče za Arduino pločicu i DC priključak. Ploča je dizajnirana tako da DC priključak na Arduino pločici i DC priključak za vanjsko napajanje budu jedan pored drugoga. Vijcima i maticama se kroz provrte montiraju na ploču, a ploča se provrtima u uglovima montira na podlogu sustava.

2.3. Priprema za tisak

3D modeli se pohranjuju u STL formatu i otvaraju u programu Ultimaker Cura, koji pretvara 3D model u seriju tankih slojeva i stvara G-kod koji sadrži instrukcije za 3D printer. Unutar programa se model može pozicionirati, odabrati kvaliteta ispisa, ispuna modela (engl. infill), temperatura mlaznice i plohe, definirati potpora (engl. support), itd. Izgled prozora dan je na slici ispod:



Slika 16: Ultimaker Cura

Materijal ispisa je poli mliječna kiselina (engl. polylactic acid - PLA). Poli mliječna kiselina se proizvodi od obnovljivih organskih izvora (kukuruzni škrob ili šećerna trska) te je najčešće korišten materijal za 3D tisak. Odlikuje ju dobar omjer cijene i kvalitete, velik izbor boja, biorazgradivost i lakoća izrade modela [20]. U radu je korištena visina sloja (engl. layer height) 0.28mm, temperaturom mlaznice 200°C te temperaturom radne površine 60°C. Veća visina sloja znači slabiju kvalitetu ispisa, ali u ovome slučaju to nije bilo bitno, a ubrzalo je izradu modela. Za dijelove poput spojnica okvira koji imaju stranice koje vise, generirana je potporna struktura. Nakon izrade modela ta se struktura lagano ukloni, ako je model dobro dizajniran., kao npr. utori na spojnicama profila

2.4. 3D tisak

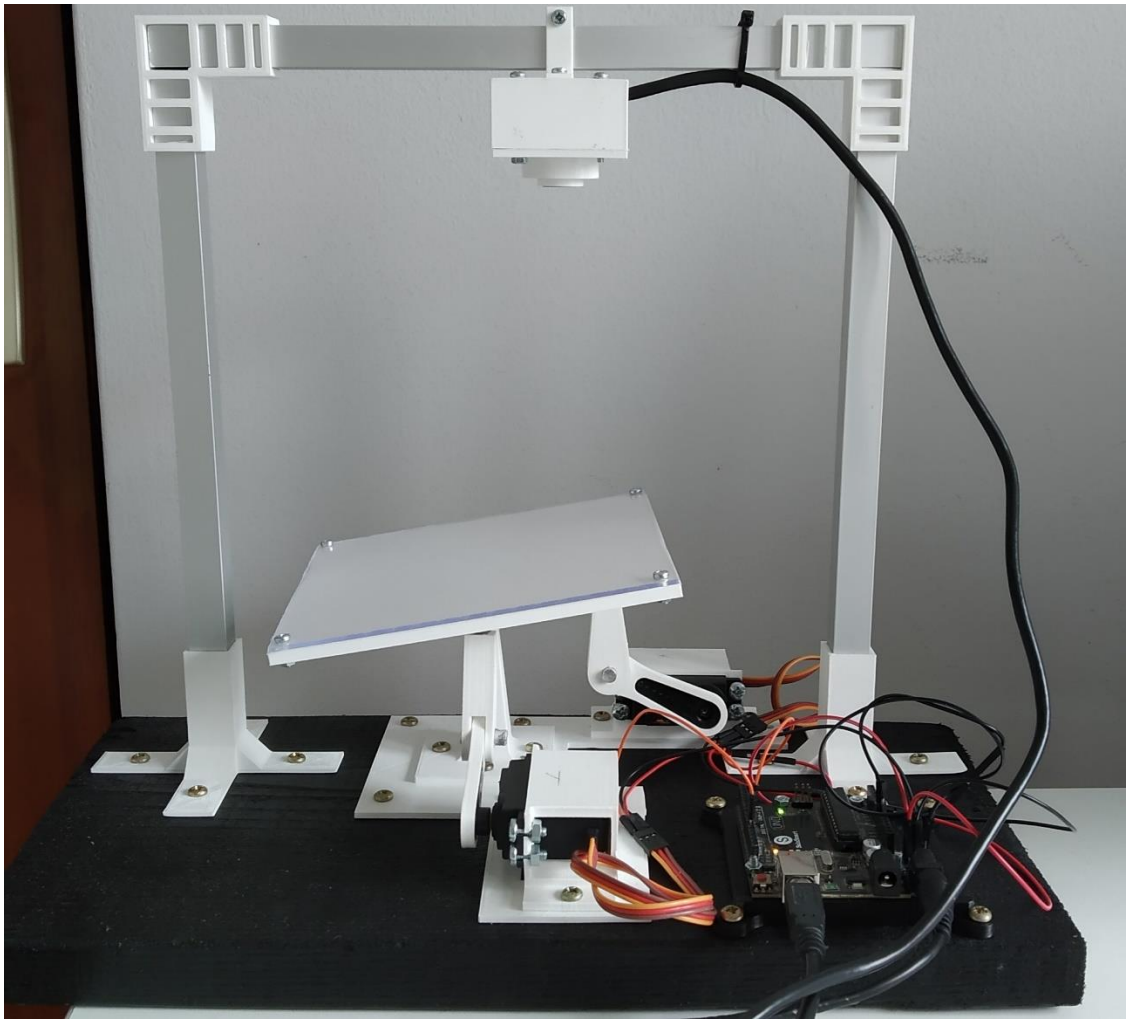
Dijelovi su tiskani na Creality Ender 3 Pro 3D printeru. Izgled 3D printera dan je na slici ispod:

Slika 17: Creality Ender 3 Pro [21]

Za ispis koristi FDM (engl. fused model depositing) tehnologiju. Dimenzije radnog područja su 220x220x250mm. Debljina sloja je od 0.1 do 0.4mm. Maksimalna temperatura mlaznice je 255°C, a podloge 110°C. Promjer filamenta je 1.75mm te može biti PLA, ABS, drvo, karbonska vlakna, i dr. Dolazi s magnetskom podlogom koja prijanja na radnu plohu te se nakon izrade model lagano odvoji [21].

2.5. Montirani sustav

Nakon tiskanja dijelovi su međusobno spojeni i vijcima za drvo montirani na dasku. Prikaz montiranog sustava dan je na slici ispod:



Slika 18: Prikaz montiranog sustava

Kamera je pozicionirana tako da je ploča otprilike u sredini slike i zategnut je vijčani spoj na nosaču kako bi se spriječilo njeno pomicanje. Testiranjem je utvrđeno da ploča nije dovoljno ravna i da neravnine utječu na rad sustava. Izbušene su četiri rupe u uglovima ploče te je vijčanim spojem montirano akrilno staklo debljine 2mm. Time se navedeni problem riješio.

2.6. Programski jezik Python

Python je programski jezik visoke razine korišten u mnogim područjima kao što su razvoj web stranica i računalnih igara, znanost, ugradbeni sustavi, umjetna inteligencija, strojno učenje, itd. [22]. U odnosu na kompajlerske jezike poput C-a i C++-a je dosta sporiji. Razlog sporosti leži u tome što je Python interpretirani jezik, odnosno napisane instrukcije se izvode direktno, liniju po liniju, bez da se prije prevode (kompajliraju) u strojni kod. Snaga Pythona leži u jednostavnosti pisanja koda i čitljivosti, tako da se korisnik može posvetiti konkretnom problemu, a ne pravilima jezika. Uz to postoji velik broj biblioteka i korisnika, što znatno olakšava razvoj. U radu je korištena verzija 3.7.7.

2.7. OpenCV biblioteka

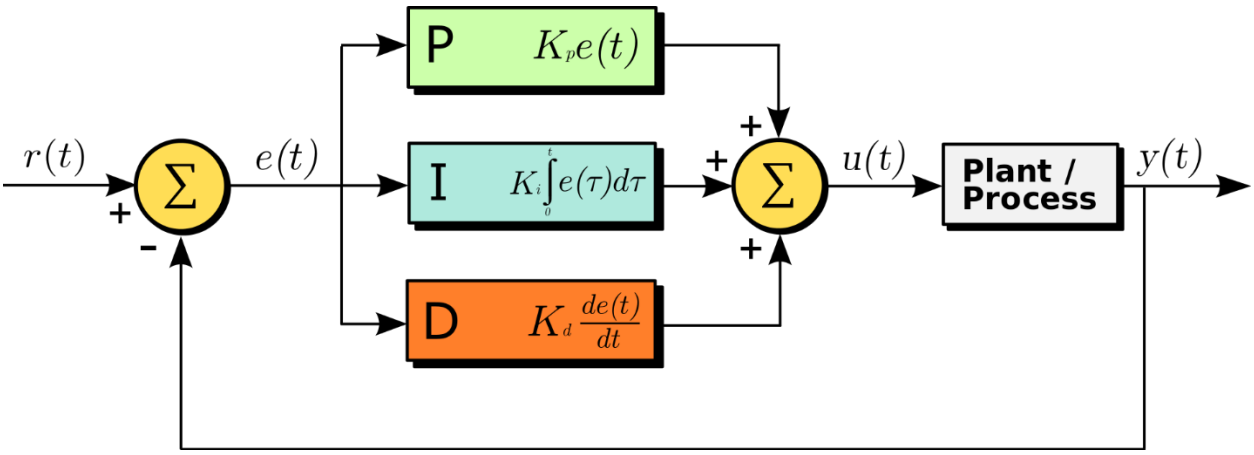
OpenCV je software biblioteka otvorenog koda namijenjena za računalni vid i strojno učenje (engl. machine learning). Stvorena je da pruži zajedničku infrastrukturu za aplikacije koje koriste računalni vid i da ubrza primjenu računalnog vida. Ima više od 2500 optimiziranih algoritama koji se mogu koristiti za raspoznavanje lica, identifikaciju objekata, klasifikaciju akcija, praćenje objekata, digitalnu obradu slike, itd. Vrlo je popularna te je s interneta preuzeta više od 18 milijuna puta [23].

Koriste je velike internacionalne kompanije poput Google-a, Microsofta, Intela, IBM-a, Honde, ali i mnoštvo manjih. Napisana je u C i C++ programskim jezicima, a može se još koristiti u Python, Java i Matlab programskim jezicima. Podržava Windows, Linux, Android i Mac OS operativne sustave [23]. U radu je korištena verzija 3.4.2.

2.8. PID kontroler

PID kontroler je često korišten kontroler u industrijskoj automatizaciji [24]. Jednostavan je za implementaciju i razumijevanje te se relativno jednostavno može postići zadovoljavajući odziv sustava i bez matematičkog modela. Za stabilizaciju sustava koristi sustav povratne veze.

Na izlazu sustava (engl. plant/process) mjeri se regulirana vrijednost y(t) (engl. process value), koja se uspoređuje sa zadanom vrijednosti (engl. Setpoint). Regulacijsko odstupanje e(t) (engl. error) je razlika između zadane vrijednosti i regulirane vrijednosti. Regulacijsko odstupanje se šalje u PID kontroler, gdje svaki član, ovisno o koeficijentima Kp, Ki i Kd, daje izlaznu vrijednost. Vrijednosti se zbrajaju te upravljačkom veličinom u(t) (engl. control variable) djeluju na sustav. Na slici ispod prikazan je blok dijagram PID kontrolera:



Slika 19: Blok dijagram PID kontrolera [24]

Proporcionalni član uzima u obzir trenutnu vrijednost regulacijskog odstupanja i množi je s koeficijentom Kp. Povećanje koeficijenta Kp ubrzava odziv sustava, ali može povećati prebačaj (engl. overshoot) te uz prevelik Kp sustav postaje nestabilan. Derivacijski član djeluje na brzinu promjene regulacijskog odstupanja, predviđa ponašanje sustava te poboljšava vrijeme smirivanja i stabilnost. Integralni član akumulira regulacijsko odstupanje. Akumulirana vrijednost se množi s koeficijentom kI za eliminaciju trajnog regulacijskog odstupanja. Kako uzima u obzir prošle vrijednosti da ispravi trenutnu vrijednost regulacijskog odstupanja, ubrzava odziv, ali povećava prebačaj i smanjuje stabilnost.

3. PROGRAMSKO RJEŠENJE

Programsko rješenje sastoji se od grafičkog sučelja, algoritma za lokalizaciju objekata , PID kontrolera i Arduino koda. Da bi se mogle mijenjati boje objekata, podešavati PID koeficijenti, mijenjati zadana vrijednost klikom tipke miša, te eventualno dodavati druge elemente u budućnosti, potrebno je grafičko sučelje.

Grafičko sučelje dizajnirano je u Qt Designeru. Elementi radnog sučelja se povlače iz liste raspoloživih grafičkih elemenata i pozicioniraju se klikom miša. Automatski se generira kod koji definira izgled sučelja i elemente. Generiranje nudi alternativu pisanju velikog broja linija koda u Python skripti te je lakše dodavati elemente i mijenjati izgled sučelja.

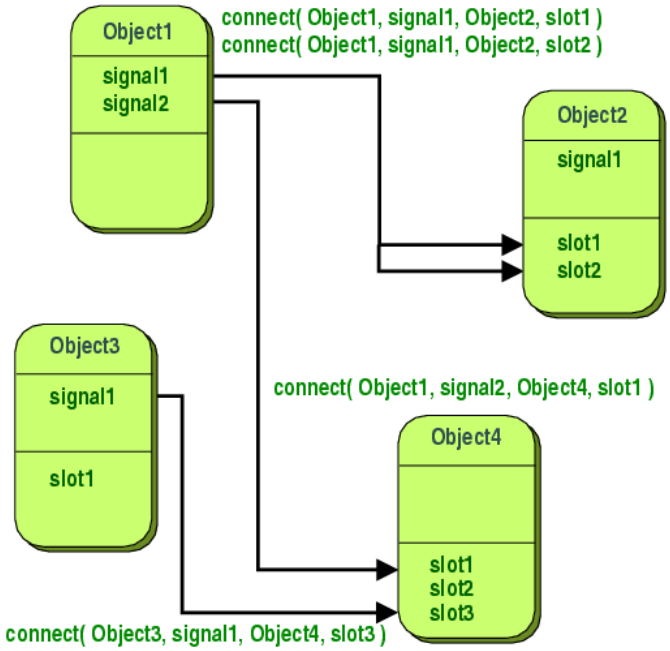
Algoritam za lokalizaciju objekata izveden je u funkciji *process_frame().* Pronalazi objekte čija je boja između minimalnih i maksimalnih vrijednosti u HSV (engl. hue, saturation, value) spektru boja. Ovisno o osvjetljenju i pozadini, pronaći će više objekata, a lokalizira onaj s najvećom površinom. Rad motora, odnosno platforme, može se uključiti i isključiti klikom tipke miša na kontrolu u grafičkom sučelju (gumb). Ako se uključi, koordinate objekta se daju funkciji *PID(),* koja kuteve šalje Arduino pločici.

3.1. Qt platforma

Qt je platforma (engl.framework) za razvoj desktop, mobilnih i aplikacija za ugradbene (engl. embedded) sustave. Bazirana je na C++ programskom jeziku te podržava Windows, Linux, OS X, iOS, Android i druge platforme. Koriste je velike kompanije poput AMD-a, Autodesk-a i Siemens-a. Ima preprocesor naziva MOC (Meta Object Compiler) koji u C++ jezik dodaje za Qt specifična svojstva kao što su signali i utori (engl. signals and slots) [25].

Signali i utori služe za komunikaciju između objekata. Svi objekti su u Qt-u definirani u QObject klasi. Dio QObject klase su npr. Qwidget objekti. Widgeti su dijelovi GUI-a koji omogućavaju interakciju s aplikacijom te pozivaju razne akcije. Neki od widgeta su gumbi, klizači, kutije za unos teksta, itd. Tako npr. odabirom kontrole (gumba) u grafičkom sučelju, želimo da se nešto određeno dogodi. U tome slučaju imamo gumb, koji je objekt, i klikom na njega emitiramo signal

koji onda poziva utor, koji je u biti funkcija predefinirana u Qt-u ili napisana od strane programera [26].

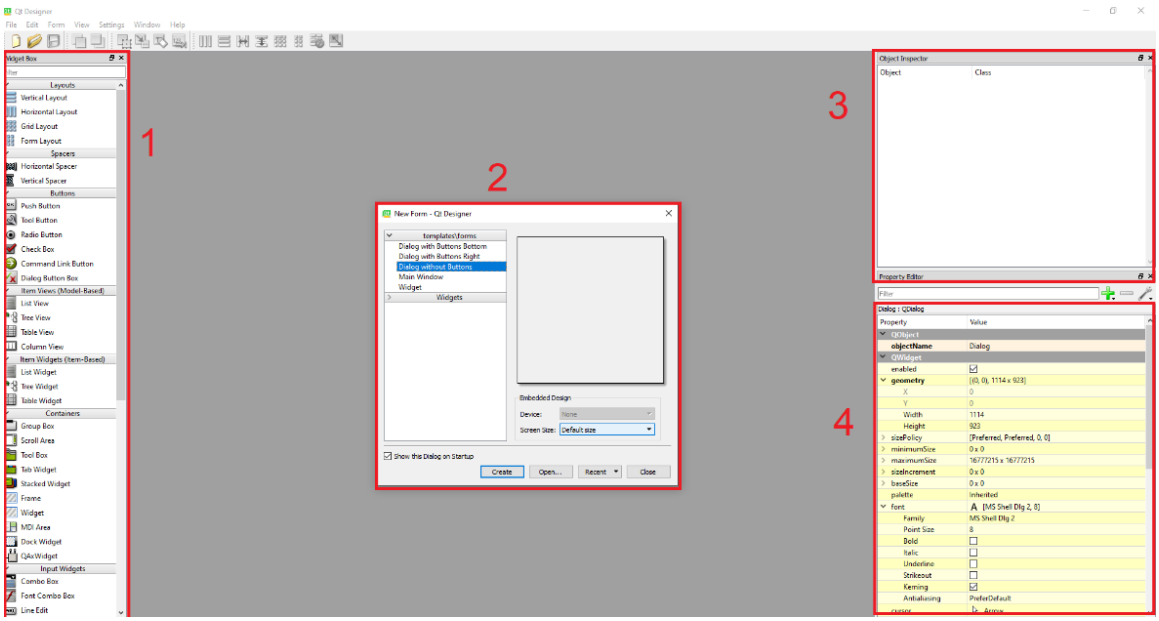


Slika 20: Qt signali i utori [26]

PyQt5 je biblioteka namijenjena za Python koja pruža alternativu C++ verziji. Tako se dodatno olakšava i ubrzava razvoj aplikacije. U radu je korištena verzija 5.15.0.

3.2. Izrada grafičkog sučelja aplikacije

Na slici ispod prikazan je prozor aplikacije Qt Designer nakon pokretanja:



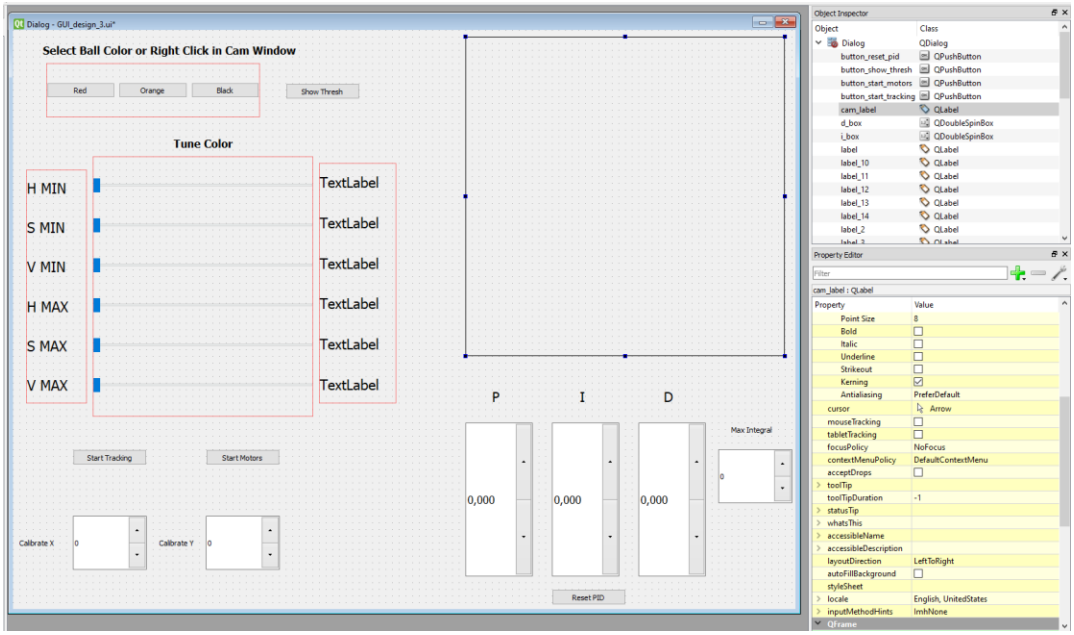
Slika 21: Qt Designer prozor

Dijelovi prozora označeni brojevima su:

1. Izbornik Widget Box - izbor objekata koji se mogu povlačiti u prozor. Podijeljen je na Layouts, Spacers, Buttons, Item Views, Item Widgets, Containers, Input Widgets i Display Widgets.
2. New form dijalog - pri pokretanju Qt Designera nudi se izbor između formi kao što su Dialog, Main Window i Widget.
3. Object inspector - sadrži popis objekata i ime klasa.
4. Property Editor - mijenjanje svojstva objekta kao npr. naziv, geometrija, font, iznos, itd.

Kao glavni prozor grafičkog sučelja odabrana je klasa QDialog. U nju su dodani gumbi klase QPushButton. Služe za odabir boje, mijenjanje prikaza slike, početak praćenja, pokretanje motora i resetiranje PID koeficijenata. Svakom gumbu je u prozoru Property Editor dan naziv kojim će biti referenciran u skripti. Gumbi za odabir boje su stavljeni u QHBoxLayout klasu, što služi kao

nevidljivi okvir u kojemu se gumbi pomiču i skaliraju kao cjelina. Klizači su stavljeni u QVBoxLayout klasu. Klizači za odabir boje su klase QSlider. Raspon vrijednosti je od 0 do 255 te je parametar singleStep postavljen na 1. Korisnici pomicanjem klizača HSV vrijednosti boja povećavaju ili smanjuju vrijednost klizača za 1. Za prikaz teksta služe QLabel objekti. Tekst koji se ne mijenja se može definirati odmah, a tekst koji služi za prikaz vrijednosti varijabli se mijenja referenciranjem u skripti. QLabel objekt također služi za prikaz slike u grafičkom sučelju, pri čemu se koriste funkcije koje će biti naknadno opisane. Za kalibraciju servo motora i ograničavanje vrijednosti integralnog člana služi klasa QSpinBox. Njenim odabirom se podešavaju cjelobrojne vrijednosti. Za PID koeficijente odabrani su objekti klase QDoubleSpinBox, koji su slični klasi QspinBox, ali koriste decimalne vrijednosti. Dizajn se pohranjuje u .ui formatu te se tim nazivom referencira u skripti. Izgled dizajniranog grafičkog sučelja prikazan je na slici ispod:



Slika 22: Dizajnirano grafičko sučelje aplikacije

3.3. Python kod aplikacije

Dio koda za lokalizaciju objekta je podijeljen slijedno u funkcije, tako da npr. prva dohvaća sliku, daje je kao argument drugoj koja je obrađuje, nakon toga daje obrađenu sliku kao argument trećoj koja lokalizira objekt, itd. Druge funkcije pozivaju se samo interakcijom s objektima sučelja. Detaljnije objašnjenje cijelog koda bit će prikazano u nastavku teksta.

3.3.1 Dohvaćanje modula i biblioteka

Na početku skripte naredbom *import* dohvaćaju se potrebni moduli. Interpreter koristi varijable i funkcije sadržane u modulu sys. cv2 je OpenCV biblioteka. Uvezeni su potrebni objekti iz PyQt5 biblioteke. Iz biblioteke imutils se koristi funkcija za kompatibilnost. Modul serial sadrži funkcije za serijsku vezu s Arduino pločicom, te modul time funkcije za računanje vremena.

Kod za dohvat modula i biblioteka:

```
import sys
import cv2
from PyQt5.QtCore import QTimer
from PyQt5.QtCore import Qt
from PyQt5 import QtGui
from PyQt5.QtGui import QImage, QPixmap
from PyQt5.QtWidgets import QApplication, QDialog, QLabel, QColorDialog, QWidget
from PyQt5.uic import loadUi
from PyQt5.QtCore import pyqtSlot
import imutils
import serial
import time
```

3.3.2. Kreiranje klase

Grafičko sučelje je definirano kao klasa GUI. Naknadno kreirana instanca klase se koristi za pokretanje grafičkog sučelja. Kako je u Qt Designeru klasa QDialog odabrana za glavni prozor, u skripti se definira klasa GUI koja nasljeđuje svojstva klase QDialog. Funkcija *__init__()* je konstruktor i automatski se poziva svaki puta kada se stvori nova instanca klase. Argument self

odnosi se na instancu klase koja se stvara. Naredba *loadUI()* učitava .ui datoteku u koju je prethodno pohranjen dizajn grafičkog sučelja te se tako mogu referencirati elementi.

Kod kreiranja klase grafičkog sučelja:

```
class GUI(QDialog):
    def __init__(self):
        super(GUI, self).__init__()
        loadUi('GUI_design_3.ui', self)
```

3.3.3. Povezivanje događaja s funkcijama

Kako bi se interakcijom s elementima sučelja pokrenule definirane akcije, treba povezati događaje (engl. events) s funkcijama. Svaki objekt ima definirane signale, tako se npr. klikom na gumb opcije start_tracking emitira signal *clicked*, koji je povezan s funkcijom *start_tracking()*. Mijenjanjem vrijednosti klizača emitira se signal *valueChanged,* koji je povezan s funkcijom *update_color_sliders()*. Mijenjanjem vrijednosti PID koeficijenata, kao i svih drugih objekata grafičkog sučelja koji su te klase, isto se emitira signal *valueChanged*, ali se povezuje s funkcijama *update_spinbox() i update_start_angle()*. Klikom na sliku, odnosno na objekt za prikaz slike, emitira se signal *mousePressEvent* koji se povezuje s funkcijom *mouse_event()*.

Kod povezivanja događaja s funkcijama:

```
self.button_red.clicked.connect(lambda: self.change_color("red"))
self.button_orange.clicked.connect(lambda: self.change_color("orange"))
self.button_black.clicked.connect(lambda: self.change_color("black"))

self.button_start_tracking.clicked.connect(self.start_tracking)
self.button_start_motors.clicked.connect(self.start_motors)
self.button_show_thresh.clicked.connect(self.show_thresh)
self.button_reset_pid.clicked.connect(self.reset_spinboxes)

self.h_min_slider.valueChanged.connect(self.update_color_sliders)
self.s_min_slider.valueChanged.connect(self.update_color_sliders)
self.v_min_slider.valueChanged.connect(self.update_color_sliders)
self.h_max_slider.valueChanged.connect(self.update_color_sliders)
self.s_max_slider.valueChanged.connect(self.update_color_sliders)
self.v_max_slider.valueChanged.connect(self.update_color_sliders)

self.p_box.valueChanged.connect(self.update_spinbox)
```

```
self.i_box.valueChanged.connect(self.update_spinbox)
self.d_box.valueChanged.connect(self.update_spinbox)
self.max_int_box.valueChanged.connect(self.update_spinbox)

self.x_box.valueChanged.connect(self.update_start_angle)
self.y_box.valueChanged.connect(self.update_start_angle)

self.cam_label.mousePressEvent = self.mouse_event
```

3.3.4. Definiranje varijabli i inicijalizacija

Nakon što su definirane varijable za postavke rezolucije kamere, PID koeficijente, računanje vremena, mijenjanje zadane vrijednosti, boje, računanje broja slika u sekundi (engl. frames per second, FPS) i zastavice (engl. flag), na kraju inicijalizacije pozvane su funkcije koje postavljaju početne vrijednosti elemenata sučelja i boju za lokalizaciju objekata. Funkcijom *start_cam()* pokreće se snimanje.

Kod definiranja varijabli i inicijalizacija:

```
self.cam_width = 640
self.cam_height = 480


self.default_P = 0.033
self.default_D = 0.023
self.default_I = 0
self.kP, self.kD, self.kI = self.default_P, self.default_D, self.default_I
self.last_time = 0
self.default_setpoint_x = self.cam_width / 2
self.default_setpoint_y = self.cam_height / 2
self.setPointX, self.setPointY = self.default_setpoint_x, self.default_setpoint_y
self.lastErrorX, self.lastErrorY = 0, 0
self.errorSumX, self.errorSumY = 0, 0
self.max_error_sum = 0
self.zero_x = 32
self.zero_y = 35

self.my_colors = {"orange": ([0, 77, 115], [61, 153, 255]),
                  "red": ([121, 157, 86], [243, 255, 255]),
                  "black": ([0, 0, 0], [25, 25, 25])}
```

```
self.hsv = None

self.last_FPS_time = 0
self.fps_counter = 0
self.fps = 0

self.flag_start_tracking = False
self.flag_show_thresh = False
self.flag_start_motors = False

self.init_spinboxes()
self.init_servo_boxes()

self.color = self.change_slider_value("orange")

self.start_cam()
```

3.3.5. Pokretanje kamere i brojača

Iz OpenCV biblioteke kreiran je objekt klase *VideoCapture()*. Argument 1 označava indeks kamere koja se aktivira. Ako npr. računalo ima ugrađenu kameru, ona ima indeks 0, a USB kamera ima indeks 1.

Funkcijom *set()* postavljaju se svojstva VideoCapture objekta. Svojstava ima više, a u prezentiranom slučaju su postavljena horizontalna i vertikalna rezolucija videa. Premda kamera može snimati u većoj rezoluciji, veća rezolucija znači više podataka i sporija obrada slike, tako da treba naći odgovarajući odnos rezolucije i brzine obrade.

Kreiran je objekt klase Qtimer, koji je u biti brojač. Signal *timeout* emitira se kada vrijednost brojača prijeđe vrijednost definiranog intervala, te je povezan s funkcijom *update_frame()*, koja dohvaća novu sliku. Interval (engl. timeout interval) je dan je u funkciji *start()* sa argumentom 0 (druge vrijednosti označavaju vrijeme u milisekundama). Argumentom 0 će se emitirati signal odmah nakon što se procesiraju drugi događaji u grafičkom sučelju. Funkcija *start()* pokreće brojač.

Kod pokretanja kamere i brojača:

```
def start_cam(self):
    self.capture = cv2.VideoCapture(1)
```

```
    self.capture.set(cv2.CAP_PROP_FRAME_WIDTH, self.cam_width)
    self.capture.set(cv2.CAP_PROP_FRAME_HEIGHT, self.cam_height)

    self.timer = QTimer(self)
    self.timer.timeout.connect(self.update_frame)
    self.timer.start(0)
```

3.3.6. Čitanje slike

Na početku funkcije *update_frame()* računa se prosječni broj slika u sekundi. Mjeri se vrijeme potrebno za dohvaćanje 20 slika. Za mjerenje vremena koristi se funkcija *time()*. Funkcijom *putText()* ispisuje se broj slika u sekundi. Prvi argument je slika na koju se ispisuje. Drugi argumenti određuju položaj teksta na slici, font, skaliranje i boju. Funkcija *read()* vraća logičku vrijednost True ili False, ovisno o tome da li je slika dohvaćena i sliku. Kako bi horizontalna i vertikalna rezolucija bile identične, provodi se uklanjanje piksela slike. Zatim se slika daje kao argument funkciji *process_frame()*.

Kod čitanja slike:

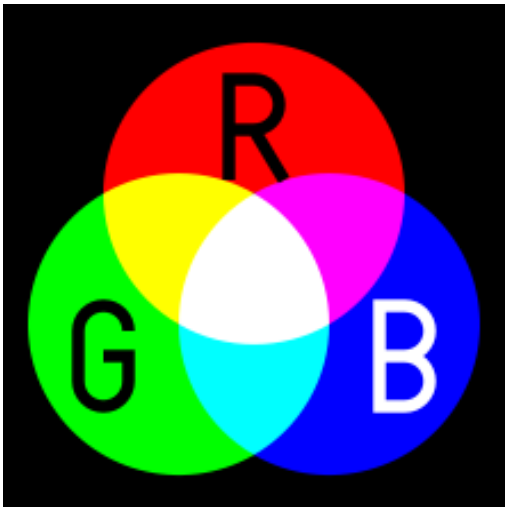
```
def update_frame(self):
    if self.fps_counter == 0:
        self.last_FPS_time = time.time()
    self.fps_counter += 1
    if self.fps_counter >= 20:
        self.fps = int(self.fps_counter / (time.time() - self.last_FPS_time))
        self.fps_counter = 0
    ret, self.frame = self.capture.read()

    self.frame = self.frame[0:480, 0:480]

    cv2.putText(self.frame, str(self.fps), (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255,
255), 2)
    self.process_frame(self.frame)
```

3.3.7. Obrada slike

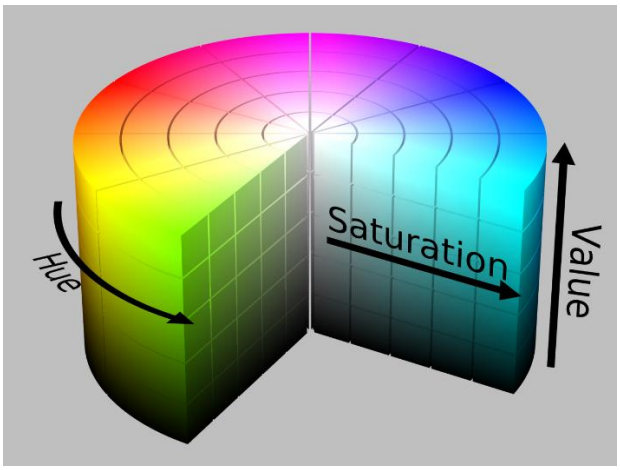
OpenCV pohranjuje slike iz kamere u BGR (engl. blue, green, red – plava, zelena, crvena) spektru boja. To je isto kao RGB, samo su zamijenjeni kanali boja. Na slici ispod dan je prikaz RGB spektra boja:



Slika 23: RGB spektar boja [27]

RGB je aditivni model boja u kojemu se miješaju crvena, zelena i plava boja da bi se dobio široki spektar drugih boja. Većina televizija, računalnih ekrana i projektora koristi taj model za prikaz boje [27]. Podešavanje boje u ovome spektru je neintuitivno i ne poklapa se s ljudskim razumijevanjem miješanja boja [28].

HSV(engl. hue, saturation, value – ton, zasićenost, svjetlina) spektar je alternativni prikaz RGB modela koji je sličniji načinu na koji ljudi vide boje. Na slici ispod prikazan je HSV spektar boja:



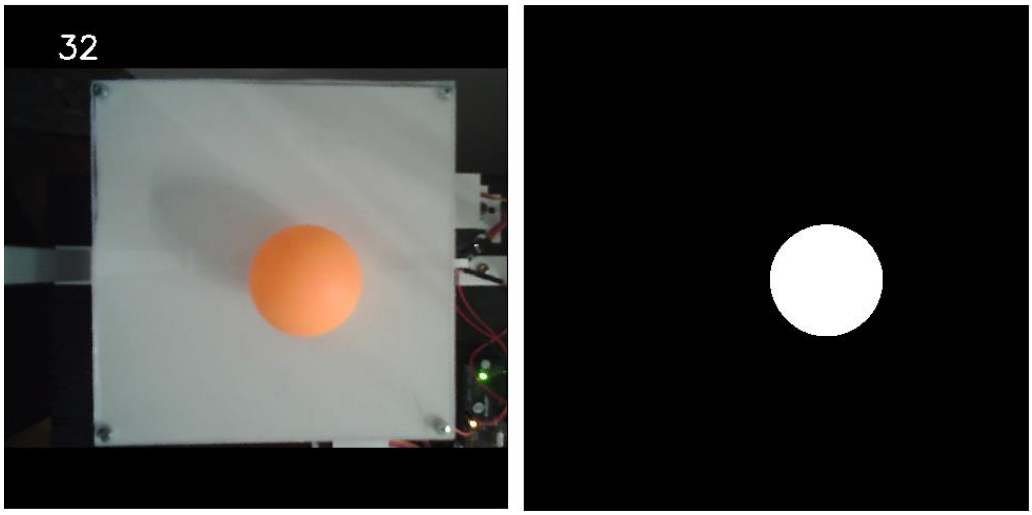
Slika 24: HSV spektar boja [28]

Kombinacija dvije osnovne boje daje novu boju. Na HSV cilindru je vidljivo da vrijednost Hue odgovara toj percepciji boje. Vrijednost Saturation opisuje zasićenost boje dok vrijednost Value opisuje zastupljenost bijele, odnosno crne boje [28].

Funkcija *cvtColor()* pretvara sliku iz BGR spektra u HSV spektar. Kao prvi argument dana je dohvaćena slika, a drugi argument je kod koji označava pretvorbu. Mogu se također koristiti kodovi za pretvorbe između drugih modela boja. Za uklanjanje visokofrekventnog šuma slika se zamućuje funkcijom *GaussianBlur().* Argument *(11,11),* odnosno *Kernel size* određuje veličinu matrice koja služi za filtriranje [29]. Na slici ispod dan je prikaz slike prije i nakon primjene funkcije *GaussianBlur():*

Slika 25: Slika prije i nakon zamućivanja [29]

Funkcijom *inRange()* stvara se binarna slika. Pikseli koji imaju vrijednosti boje između *color_low* i *color_high* primaju vrijednost 1 te su u binarnoj slici bijele boje i označavaju konturu objekta, ostali primaju vrijednost 0 i crne su boje. Na slici ispod dan je prikaz slike prije i nakon primjene funkcije inRange():



Slika 26: Stvaranje binarne slike

Funkcija *erode()*, odnosno erozija, uklanja male objekte koji su na granici konture i predstavljaju šum. Argument iterations označava broj iteracija uklanjanja. Erozija uklanja šum, ali i smanjuje veličinu konture. Zato se nakon nje primjeni funkcija *dilate()* da spoji dijelove i poveća površinu konture [30]. Na slici ispod dan je primjer originalne slike, slike nakon primjene funkcije *erode()* i slike nakon primjene funkcije *dilate()*:

Slika 27: Primjer funkcija erode() i dilate() [30]

Kod obrade slike:

```
def process_frame(self, frame):
    self.hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    blurred = cv2.GaussianBlur(self.hsv, (11,11), 0)

    mask = cv2.inRange(blurred, self.color_low, self.color_high)
    mask = cv2.erode(mask, None, iterations=2)
    mask = cv2.dilate(mask, None, iterations=2)
```

3.3.8. Lokalizacija objekta

Nakon što je formirana binarna slika, funkcija *findContours()* pronalazi konture i vraća ih kao Python listu (engl. list). Funkcija *grab_contours()* iz modula *imutils* služi za kompatibilnost s drugim verzijama OpenCV-a. Izrazom *if len(cnts) > 0* provjerava se da li je pronađena ijedna kontura. Ako nije, preskače ostatak koda. Ako je, funkcija *max()* vraća onu s najvećom površinom. Funkcija *minEnclosingCircle()* računa najmanji krug koji čini konturu. Vraća koordinate centra i polumjer kruga.

Petlja se nastavlja samo ako je polumjer konture veći od 10. Odabirom opcije za početak praćenja mijenja se logička vrijednost zastavice i radi vizualne predodžbe se oko pronađene konture funkcijom *circle()* prikazuje opisana kružnica i njen centar. Odabirom opcije *start_motors* mijenja se logički iznos zastavice i prikazuje se krug koji predstavlja koordinate nazivne veličine i kao argumenti funkciji *PID()* daju se koordinate nazivne veličine i koordinate centra pronađene konture. Boja se kalibrira odabirom opcije *Show Thresh*, čime se prikazuje binarna slika. Zastavica *flag_show_thresh* poprima vrijednost *True* i binarna slika se pretvara u BGR spektar boja. Slika se daje kao argument funkciji *display_image()*.

Kod lokalizacije objekta:

```
cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
                    cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
if len(cnts) > 0:
    c = max(cnts, key=cv2.contourArea)
    ((x, y), radius) = cv2.minEnclosingCircle(c)
    if radius > 10:
        if self.flag_start_tracking:
            cv2.circle(frame, (int(x), int(y)), int(radius), (255, 255, 255), 2)
            cv2.circle(frame, (int(x), int(y)), 5, (255, 255, 255), -1)
        if self.flag_start_motors
            cv2.circle(frame, (int(self.setPointX), int(self.setPointY)), 5, (0, 0, 255), -1)
            self.PID(self.setPointX, self.setPointY, x, y)
if self.flag_show_thresh == True:
    self.frame = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)
self.display_image(self.frame)
```

3.3.9. Prikaz slike u grafičkom sučelju

Slika se pretvara iz BGR formata nazad u RGB format. Za prikaz u grafičkom sučelju, treba biti u formatu Qimage. Na ekran, odnosno objekt *cam_label*, se prikazuje s funkcijom *setPixmap()*.

Kod prikaza slike u grafičkom sučelju:

```
def display_image(self, img):
    rgb_image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    h, w, ch = rgb_image.shape
    bytesPerLine = ch * w
```

```
    img_qt = QImage(rgb_image.data, w, h, bytesPerLine, QImage.Format_RGB888)
    self.cam_label.setPixmap(QPixmap.fromImage(img_qt))
```

3.3.10. PID funkcija

PID() funkcija se poziva aperiodski te se računa interval, odnosno razlika u vremenu između poziva. Računa se regulacijsko odstupanje, integralna komponenta i derivacijska komponenta. Maksimalna vrijednost integralne komponente se kontrolira iznosom varijable *max_error_sum*, a ona se namješta u grafičkom sučelju. Iznosi *zero_x* i *zero_y* su kutevi servo motora kod kojih je ploča u horizontalnom položaju. Zbrajanjem s proporcionalnom, integralnom i derivacijskom komponentom dobiva se konačni kut. Kutevi se ograničavaju na ±15° od vrijednosti u kojima je ploča u horizontalnom položaju. Funkcijom *write()* se kutevi serijskom vezom šalju Arduino pločici kao znakovni niz (engl. string). Kutevi se odvajaju zarezom i na kraj se stavlja znak za novi red.

Kod PID funkcije:

```
def PID(self, setPointX, setPointY, inputX, inputY):
    now = time.time()
    dT = now - self.last_time
    self.last_time = now
    errorX = self.setPointX - inputX
    errorY = self.setPointY - inputY

    self.errorSumX = errorX * dT
    self.errorSumY = errorY * dT

    if self.errorSumX > self.max_error_sum:
        self.errorSumX = self.max_error_sum
    elif self.errorSumX < - self.max_error_sum:
        self.errorSumX = - self.max_error_sum

    if self.errorSumY > self.max_error_sum:
        self.errorSumY = self.max_error_sum
    elif self.errorSumY < - self.max_error_sum:
        self.errorSumY = - self.max_error_sum

    dErrorX = (errorX - self.lastErrorX) / dT
    dErrorY = (errorY - self.lastErrorY) / dT

    angleX = self.zero_x + (errorX * self.kP + dErrorX * self.kD + self.kI * self.errorSumX)
```

```
    angleY = self.zero_y + (errorY * self.kP + dErrorY * self.kD + self.kI * self.errorSumY)

    if angleX > self.zero_x + 15:
        angleX = self.zero_x + 15
    elif angleX < self.zero_x - 15:
        angleX = self.zero_x - 15
    if angleY > self.zero_y + 15:
        angleY = self.zero_y + 15
    elif angleY < self.zero_y - 15:
        angleY = self.zero_y - 15

    angleX = round(angleX, 1)
    angleY = round(angleY, 1)

    self.lastErrorX = errorX
    self.lastErrorY = errorY

    arduino.write((str(angleX) + "," + str(angleY) + "\n").encode())
```

3.3.11. Uključivanje i isključivanje funkcija

Promjenom logičkih vrijednosti zastavica se kontrolira izvođenje programa. Za mijenjanje vrijednosti zastavica i teksta gumba služe funkcije *start_tracking()*, *start_motors()* i *show thresh()*. Odabirom opcije (gumba) dolazi do promjene logičke vrijednosti zastavice, što utječe na funkcionalnost drugih dijelova programa. Funkcijama *setText()* se mijenja tekst, kako bi se lakše pratilo koje su funkcionalnosti uključene.

Kod uključivanja i isključivanja funkcija:

```
def start_tracking(self):
    if self.flag_start_tracking == False:
        self.flag_start_tracking = True
        self.button_start_tracking.setText("Stop Tracking")

def start_motors(self):
    if self.flag_start_motors == False:
        self.flag_start_motors = True
        self.button_start_motors.setText("Stop Motors")
    else:
        self.flag_start_motors = False
        self.button_start_motors.setText("Start Motors")
        arduino.write((str(self.zero_x) + "," + str(self.zero_y) + "\n").encode())
```

```
def show_thresh(self):
    if self.flag_show_thresh == False:
        self.flag_show_thresh = True
        self.button_show_thresh.setText("Normal View")
    else:
        self.flag_show_thresh = False
        self.button_show_thresh.setText("Show Thresh")
```

3.3.12. Ažuriranje klizača

Boje su definirane u rječniku *my_colors* i mogu se odabrati klikom na gumb kontrole u grafičkom sučelju. Da bi se imala vizualna predodžba, vrijednosti klizača se moraju mijenjati kako se mijenjaju i boje te se odabirom boje poziva funkcija *change_slider_value()*. Funkcija *setValue()* postavlja određeni klizač na vrijednost definiranu u rječniku *my_colors*. Tekst s desne strane, koji označava trenutnu vrijednost pojedinog klizača, mijenja se s funkcijom *setText()*. Nakon ažuriranja vrijednosti klizača, emitira se signal *valueChanged()* te se automatski poziva funkcija *update_color()*.

Kod ažuriranja klizača:

```
def change_slider_value(self, color):
    loc_color = self.my_colors[color]

    self.h_min_slider.setValue(loc_color[0][0])
    self.s_min_slider.setValue(loc_color[0][1])
    self.v_min_slider.setValue(loc_color[0][2])
    self.h_max_slider.setValue(loc_color[1][0])
    self.s_max_slider.setValue(loc_color[1][1])
    self.v_max_slider.setValue(loc_color[1][2])
    self.label_h_min.setText(str(loc_color[0][0]))
    self.label_s_min.setText(str(loc_color[0][1]))
    self.label_v_min.setText(str(loc_color[0][2]))
    self.label_h_max.setText(str(loc_color[1][0]))
    self.label_s_max.setText(str(loc_color[1][1]))
    self.label_v_max.setText(str(loc_color[1][2]))
```

3.3.13. Postavke boje

Pomicanjem klizača se postavljaju minimalne i maksimalne HSV vrijednosti boje. Funkcije *value()* vraćaju vrijednosti klizača i spremaju ih u varijable *color_low* i *color_high*. S funkcijom *setText()* se mijenja prikaz teksta koji označava trenutnu vrijednost klizača.

Kod postavki boje:

```
def update_color(self):
    self.color_low = (
        self.h_min_slider.value(), self.s_min_slider.value(),
        self.v_min_slider.value())
    self.color_high = (
        self.h_max_slider.value(), self.s_max_slider.value(),
        self.v_max_slider.value())

    self.label_h_min.setText(str(self.color_low[0]))
    self.label_s_min.setText(str(self.color_low[1]))
    self.label_v_min.setText(str(self.color_low[2]))
    self.label_h_max.setText(str(self.color_high[0]))
    self.label_s_max.setText(str(self.color_high[1]))
    self.label_v_max.setText(str(self.color_high[2]))
```

3.3.14. Promjena zadane vrijednosti i boje klikom tipke miša

Zadana vrijednost se mijenja klikom na lijevu tipku miša, a klikom na desnu se odabire boja. Klikom na sliku emitira se signal *MousePressEvent* i poziva se funkcija *mouse_event()*. U varijable *x* i *y* se spremaju koordinate događaja. Ovisno o tome koja je tipka miša kliknuta, mijenja se zadana vrijednost ili boja za lokalizaciju objekta.

Kod promjene zadane vrijednosti i boje klikom tipke miša:

```
def mouse_event(self, event):
    x = event.x()
    y = event.y()
    if event.buttons() == Qt.LeftButton:
        self.setPointX = x
        self.setPointY = y
        print(x,y)
    elif event.buttons() == Qt.RightButton:
        hsv = self.hsv
```

```
        self.h_min_slider.setValue(hsv[x, y][0] - 15)
        self.s_min_slider.setValue(hsv[x, y][1] - 60)
        self.v_min_slider.setValue(hsv[x, y][2] - 60)
        self.h_max_slider.setValue(hsv[x, y][0] + 15)
        self.s_max_slider.setValue(hsv[x, y][1] + 60)
        self.v_max_slider.setValue(hsv[x, y][2] + 60)
```

3.3.15. Ažuriranje ostalih elemenata grafičkog sučelja

Funkcije *update_spinbox()*, *init_spinbox()*, *init_servo_boxes()*, *reset_spinboxes()* i *update_start_angle()* rade na istom principu kao već opisane funkcije. Služe za inicijalizaciju ili mijenjanje vrijednosti objekata.

Kod ažuriranja ostalih elemenata grafičkog sučelja:

```
def update_spinbox(self):
    self.kP = self.p_box.value()
    self.kI = self.i_box.value()
    self.kD = self.d_box.value()
    self.max_error_sum = self.max_int_box.value()

def init_spinboxes(self):
    self.p_box.setValue(self.default_P)
    self.i_box.setValue(self.default_I)
    self.d_box.setValue(self.default_D)
def init_servo_boxes(self):
    self.x_box.setValue(self.zero_y)
    self.y_box.setValue(self.zero_x)

def reset_spinboxes(self):
    self.init_spinboxes()


def update_start_angle(self):
    self.zero_x = self.x_box.value()
    self.zero_y = self.y_box.value()
    arduino.write((str(self.zero_x) + "," + str(self.zero_y) + "\n").encode())
```

3.3.16. Pokretanje skripte i grafičkog sučelja

Za komunikaciju se otvara serijsko sučelje (engl. serial port) i postavlja se brzina prijenosa podataka (engl. baud rate) od 19200 bitova po sekundi. Kreira se instanca klase GUI i postavlja naziv prozora na „Ball Tracking“. Funkcijom show() se prikazuje grafičko sučelje.

Kod pokretanja skripte i grafičkog sučelja:

```
if __name__ == "__main__":
    arduino = serial.Serial('COM3', baudrate=19200, timeout=1)
    app = QApplication(sys.argv)
    window = GUI()
    window.setWindowTitle("Ball Tracking")
    window.show()
    sys.exit(app.exec_())
```

3.4. Arduino kod

Za upravljanje servo motorima korištena je Arduino Servo biblioteka [31]. Definiraju se servo objekti, *servoX* i *servoY*. Definiraju se varijable za kuteve svakog motora i postavljaju se na početne vrijednosti kuteva kod kojih je ploča u otprilike horizontalnom položaju.

Kod za definiranje servo objekata i početnih vrijednosti kuteva:

```
#include <Servo.h>
Servo servoX;
Servo servoY;
float angleX = 34;
float angleY = 34;
```

U funkciji *setup()* se serijsko sučelje postavlja na istu brzinu prijenosa podataka kao i u Python skripti. Funkcija *attach()* definira servo pinove te minimalnu i maksimalnu duljinu pulsa. Puls duljine 550µs postavlja vratilo servo motora na kut od 0°, a puls duljine 2350µs na kut od 180°. Minimalna vrijednost je pronađena eksperimentalno korištenjem funkcije *writeMicroseconds(),* koja postavlja vratilo servo motora na kut koji odgovara duljini pulsa te čitanjem kuta vratila funkcijom *read()*, koja vraća kut vratila u stupnjevima. Funkcija *write()* pulsno-širinskom modulacijom zakreće servo motore na početni položaj.

Kod funkcije setup():

```
void setup() {
  Serial.begin(19200);
  servoX.attach(9, 550, 2350);
  servoY.attach(11, 550, 2350);
  servoX.write(angleX);
  servoY.write(angleY); }
```

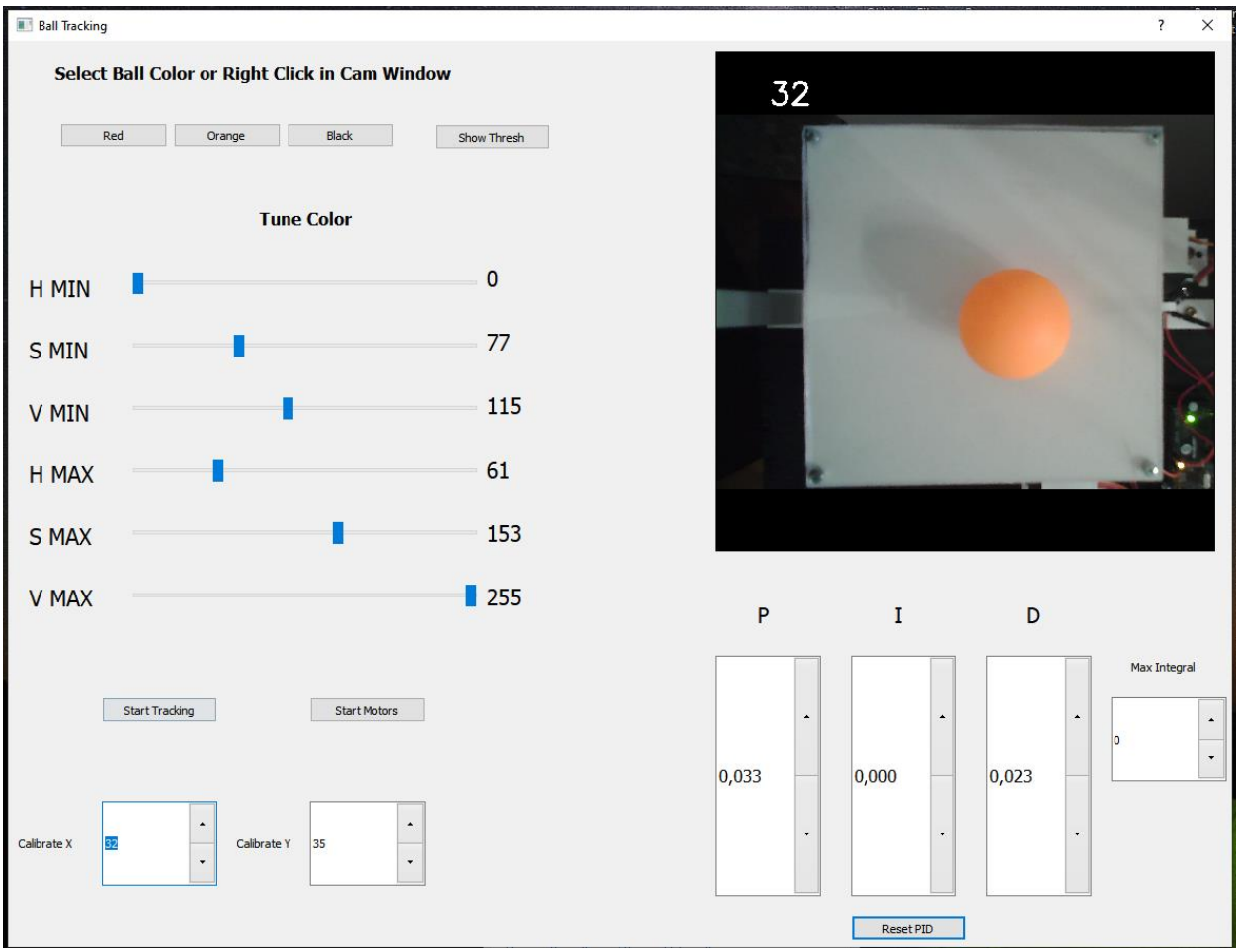
U glavnoj petlji provjerava se prisutnost podataka u spremniku (engl. serial buffer), odnosno da li je što poslano iz Python skripte. Kako su kutevi odvojeni zarezom, string do zareza predstavlja kut u x smjeru, a od zareza do znaka za novi red je kut u y smjeru. Stringovi se transformiraju u tip podataka float i funkcijom *write()* se pulsno-širinskom modulacijom zakreću servo motori na željeni položaj.

Kod glavne petlje:

```
void loop() {
  if(Serial.available() > 0) {
    angleX = Serial.readStringUntil(',').toFloat();
    angleY = Serial.readStringUntil('\n').toFloat();
    servoX.write(angleX);
    servoY.write(angleY);
  }}
```

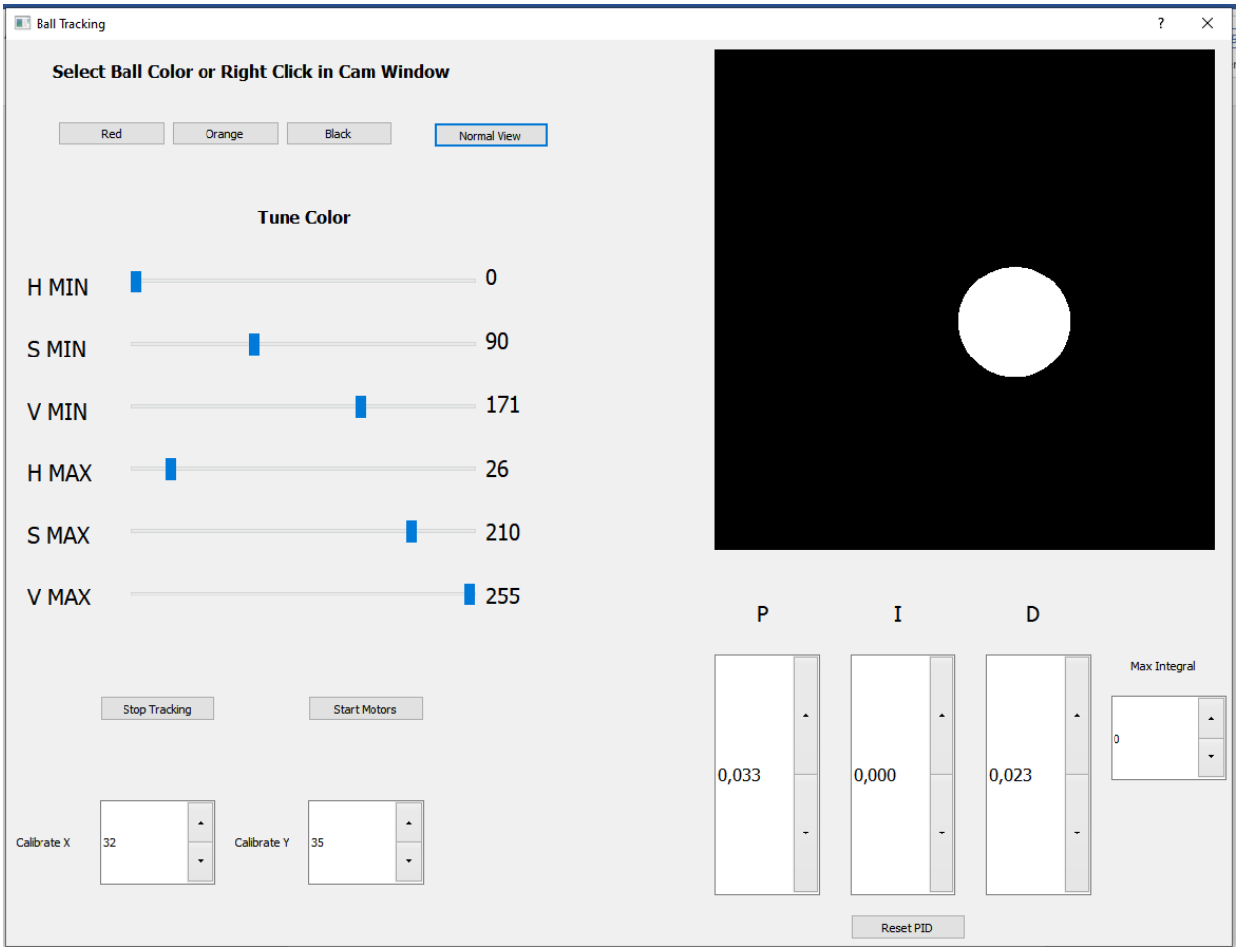
4. REZULTAT RADA

Na sljedećim slikama će biti prikazan izgled sučelja te uključivanje nekih funkcionalnosti. Pokretanjem skripte pokreće se radno sučelje aplikacije prikazano na slici ispod:



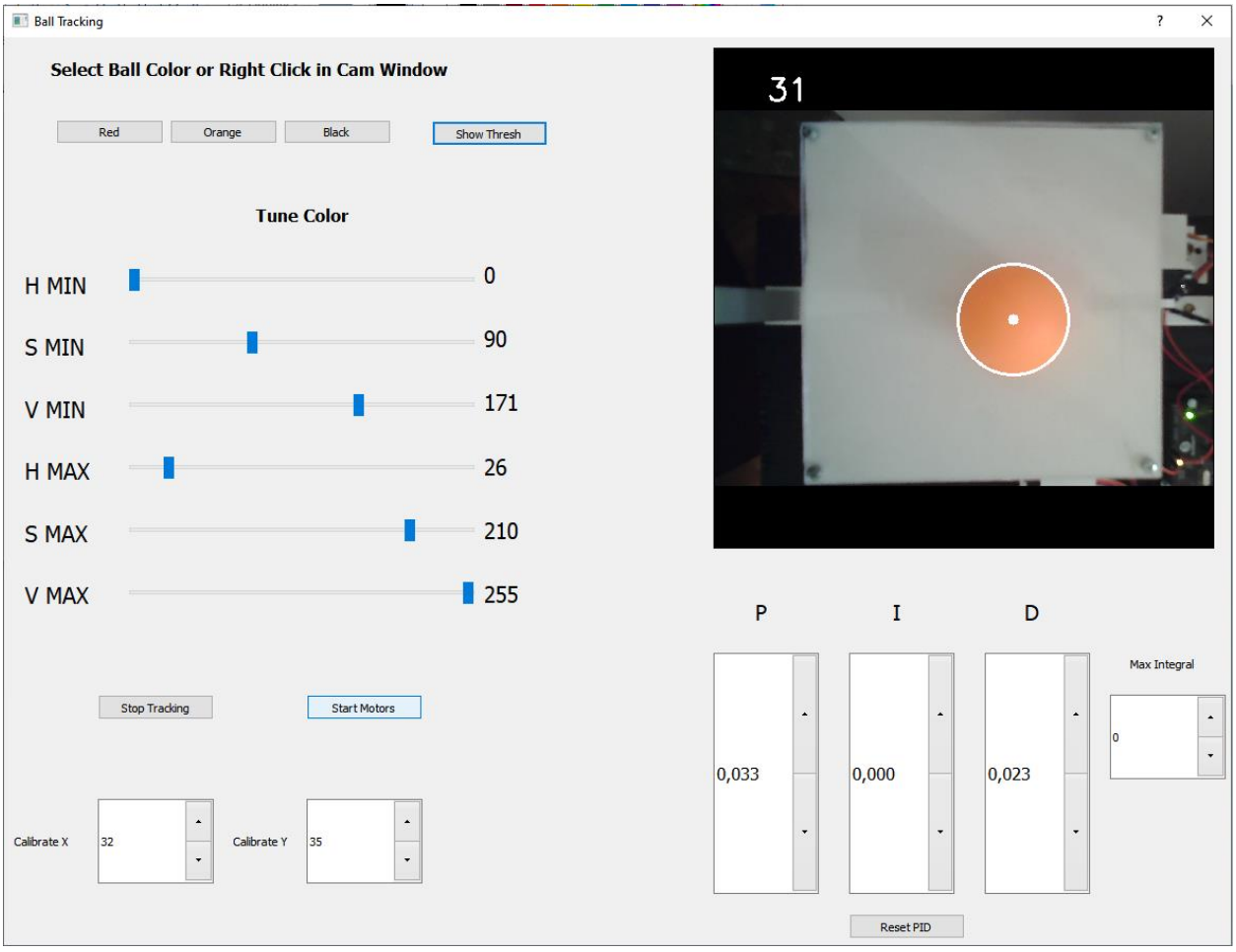
Slika 28: Izgled grafičkog sučelja nakon pokretanja skripte

Odabirom opcije Show Thresh prikazuje se binarna slika, koja nakon podešavanja boje izgleda kao na slici ispod:



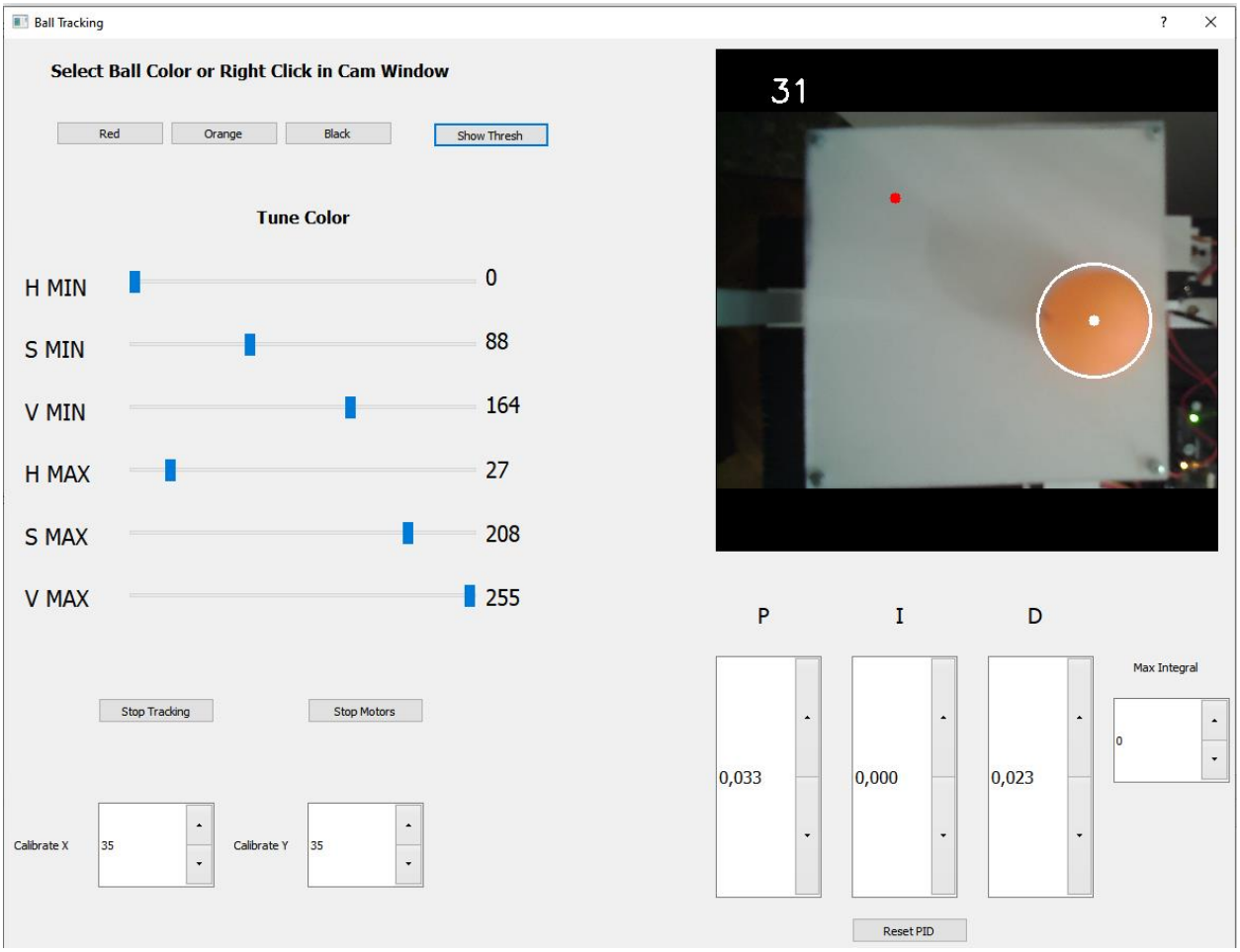
Slika 29: Binarna slika

Odabirom opcije start tracking prikazuju se kružnica oko loptice i njen centar radi lakšeg praćenja kao što je prikazano na slici ispod:



Slika 30: Lokalizirani objekt

Odabirom opcije Start Motors prikazuje se zadana vrijednost i pokreće rad platforme kao što je prikazano na slici ispod:



Slika 31: Pokretanje platforme i ispis zadane vrijednosti

Grafičko sučelje radi kako je zamišljeno te bi se moglo primijeniti i kod sličnih sustava. Sami algoritam za lokalizaciju objekta je dosta robustan te locira i djelomično zaklonjen objekt. Nedostatak je što objekt mora imati znatnu razliku u vrijednostima boja od okoline. Tako npr. za rad prezentiranog sustava loptica ne može biti bijele ili crne boje jer su te boje u pozadini i imaju veću površinu pa algoritam odabire njih kao objekte za praćenje. Točnost regulacije najviše ovisi o točnosti lokalizacije loptice. Pod slabijim osvjetljenjem lokalizacija radi lošije, pogotovo u uglovima ploče. Sustav snima sa 30-tak slika u sekundi, što je zadovoljavajuće. Početne vrijednosti

proporcionalnog i derivacijskog koeficijenta su pronađene tijekom rada na sustavu i ne predstavljaju optimalne vrijednosti, već početne kod kojih će sustav biti u stanju pozicionirati lopticu u otprilike željeni položaj. Daljnjom kalibracijom i uključivanjem integralnog člana se regulacijsko odstupanje može smanjiti, ali ne i ukloniti. Svejedno, jasno se vidi kako svaki član utječe na regulaciju i promatranjem platforme se dobije bolje razumijevanje sustava regulacije i nedostataka prototipa.

5. ZAKLJUČAK

Zadatak završnog rada je bio primijeniti računalni vid na mehatroničkom sustavu kugle na ploči. Kroz izradu prototipa autor se upoznao s osnovama računalnog vida i izrade grafičkog sučelja, te poboljšao znanja iz 3D dizajna, 3D tiskanja, automatske regulacije i Python programskog jezika. Spajanjem tim tehnologija sustav se dizajnirao dio po dio, modificirao, te se došlo do prototipa koji daje zadovoljavajuće rezultate i može poslužiti kao referenca za izradu boljeg sustava u budućnosti.

Dodatno poboljšanje može biti nabavka ili izrada boljih poluga kako bi se eliminirala zračnost u spojevima. Također, korišteni servo motori mogu biti precizniji pa bi se nabavkom boljih postigla i bolja regulacija. Brži rad programa, i samim time kraće vrijeme između poziva PID funkcije, bi se mogao postići korištenjem programskog jezika Cython, koji kombinira jednostavnost Pythona i brzinu programskog jezika C.

6. POPIS LITERATURE

[1] PyImageSearch, URL: https://www.pyimagesearch.com, datum pristupa: 29.8.2020.

[2] PyImageSearch: Ball tracking with OpenCV, URL: https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv, datum pristupa: 29.8.2020.

[3] LearnPyQt, URL: https://www.learnpyqt.com/, datum pristupa: 1.9.2020.

[4] OpenCV-Python Tutorials, URL: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html, datum pristupa: 1.9.2020.

[5] OpenCV Tutorials, URL: https://docs.opencv.org/master/d9/df8/tutorial_root.html, datum pristupa: 1.9.2020.

[6] Stack Overflow: PyQt showing video stream from opencv, URL: https://stackoverflow.com/questions/44404349/pyqt-showing-video-stream-from-opencv, datum pristupa: 1.9.2020.

[7] pySerial, URL: https://pyserial.readthedocs.io/en/latest/index.html, datum pristupa: 1.9.2020.

[8] Sumega, M; Struharnansky, L; Gorel, L; Pacha, M: „Simulation and experimental study of ball position control at biaxial platform using state space approach“

[9] Wikipedia: Ljudsko oko, URL: https://hr.wikipedia.org/wiki/Ljudsko_oko, datum pristupa: 29.8.2020.

[10] Wikipedia: Computer Vision, URL: https://en.wikipedia.org/wiki/Computer_vision, datum pristupa: 30.8.2020.

[11] Szeliski, R: „Computer Vision: Algorithms and Applications“

[12] Sainsmart: Arduino Uno, URL: https://www.sainsmart.com/products/uno-r3-atmega328p-pu-16u2-arduino-compatible, datum pristupa: 30.8.2020.

[13] Wikipedia: Arduino Uno, URL: https://en.wikipedia.org/wiki/Arduino_Uno, datum pristupa 30.8.2020.

[14] Aliexpress: ELP-USBFHD01M USB kamera, URL: https://www.aliexpress.com/item/32736010155.html?spm=a2g0s.9042311.0.0.27426c370wkYTE, datum pristupa: 30.8.2020.

[15] Wikipedia: Servomotor, URL: https://en.wikipedia.org/wiki/Servomotor, datum pristupa: 30.8.2020.

[16] e-radionica: Servo motor Tower Pro 995, URL: https://e-radionica.com/productdata/MG995.pdf, datum pristupa: 30.8.2020.

[17] rcnhobby: TowerPro MG995 Metal Servo, https://www.rcnhobby.com/towerpro-mg995-metal-servo.html, datum pristupa: 30.8.2020.

[18] e-radionica: DC connector breakout, URL: https://e-radionica.com/en/dc-connector-breakout.html, datum pristupa: 30.8.2020.

[19] Aliexpress: Magnetski kuglični ležajevi, URL: https://www.aliexpress.com/item/32818135577.html?spm=a2g0s.9042311.0.0.27426c370wkYTE, datum pristupa: 30.8.2020.

[20] all3dp: 2020 PLA Filament Buyer's Guide, URL: https://all3dp.com/1/pla-filament-3d-printing/, datum pristupa: 30.8.2020.

[21] Creality3dshop: Ender 3 Pro, URL: https://www.creality3dshop.eu/collections/3d-printers/products/creality3d-ender-3-pro-high-precision-3d-printer?gclid=CjwKCAjwnK36BRBVEiwAsMT8WLiBspz5OY-q4rty_bgNwWDmCPcYfYOQ9aohnkX3sZBdI4bCwwcNrBoCOe8QAvD_BwE, datum pristupa: 30.8.2020.

[22] Wikipedia: Python programming language, URL: https://en.wikipedia.org/wiki/Python_(programming_language), datum pristupa: 30.8.2020.

[23] OpenCV: About, URL: https://opencv.org/about/, datum pristupa: 29.8.2020.

[24] Wikipedia: PID controller, URL: https://en.wikipedia.org/wiki/PID_controller, datum pristupa: 29.8.2020.

[25] Qt Wiki: About Qt, URL: https://wiki.qt.io/About_Qt, datum pristupa: 30.8.2020.

[26] Qt Documentation: Signals & Slots, URL: https://doc.qt.io/qt-5/signalsandslots.html, datum pristupa: 30.8.2020.

[27] Wikipedia: RGB color model, URL: https://en.wikipedia.org/wiki/RGB_color_model, datum pristupa: 31.8.2020.

[28] Wikipedia: HSL and HSV, URL: https://en.wikipedia.org/wiki/HSL_and_HSV, datum pristupa: 31.8.2020.

[29] OpenCV: Smoothing images, URL: https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html, datum pristupa: 30.8.2020.

[30] OpenCV: Morphological Transformations, URL: https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html, datum pristupa: 30.8.2020.

[31] Arduino reference: Servo, URL: https://www.arduino.cc/reference/en/libraries/servo/, datum pristupa: 30.8.2020.