Navigiranje Sphero robota kroz labirint uz pomoć bespilotne letjelice

Tehnička dokumentacija

Verzija 1.0

Studentski tim: Leonardo Max Golušin
Denis Đurašinović
Dominik Erić
Filip Karaj
Filip Kirn
Josip Ante Kozulić
Marin Maletić
Filip Pušnik
Marija Rozić
Filip Škoro
Nikola Špoljarec
Ivan Vuksan

Nastavnik: prof.dr.sc Stjepan Bogdan

Tehnička dokumentacija

1. Uvod

Ideja projekta je upoznavanje s radom zasebnih uređaja kao autonomni sustav. Za izvedbu navedenog projekta, od uređaja su nam na raspolaganju programibilni robot Sphero SPRK+ te bespilotna letjelica Parrot Bebop 2. Kao što ime projekta nalaže, korištenjem Bebop-a, potrebno je Sphero robotu pridati autonomnu komponentu koju sam po sebi ne sadrži, tj. omogućiti Sphero robotu da se uz upute Bebop-a kreće kroz labirint.

Letjelica lebdi iznad labirinta te se pomoću referentnih točaka optimalno pozicionira iznad labirinta. Kamera letjelice konstantno snima labirint i Sphero-a te prati njegovu lokaciju u odnosu na referentne točke. Pozadinski kontroler sadrži algoritam koji pronalazi optimalni put kroz labirint te potom šalje upute Sphero-u. Pomoću slike koju kontroler dobiva s letjelice prati put koji je Sphero prešao. Nakon što je Sphero prešao dio puta, kontroler mu šalje set instrukcija: u kojem smjeru i kojom brzinom će se kretati dok ih Sphero ne izvrši. Potom će kontroler poslati novi set instrukcija dok Sphero ne prođe kroz cijeli labirint.

Unutar tima podijelili smo projekt na teme koje uključuju:

- Izrada matematičkih modela
- Parametriziranje regulatora u Matlabu
- Implementiranje regulatora u ROS-u
- Obrada slike/rješavanje labirinta
- Izrada labirinta

2. Matematički model, parametrizacija i diskretizacija

2.1 Parrot Bebop 2

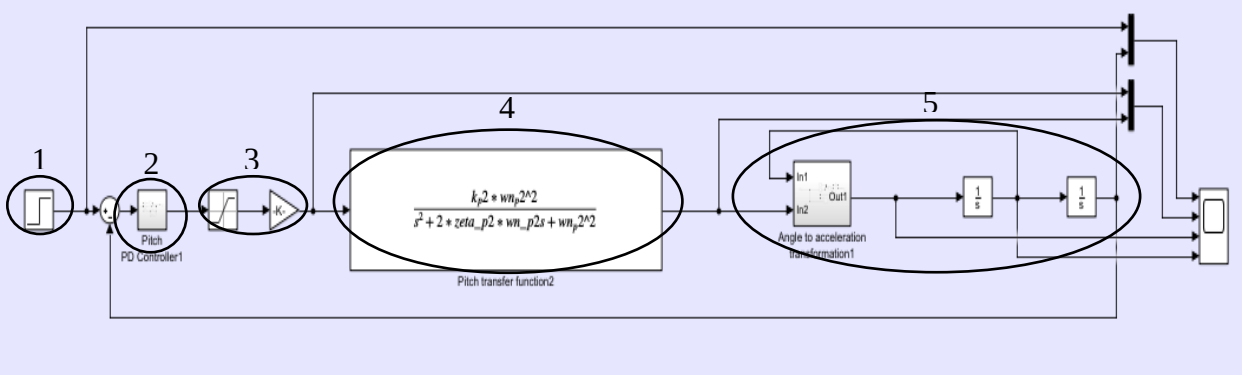
2.1.1 Matematički model

Za realizaciju projekta bilo je potrebno napraviti pripadajući matematički model letjelice koji opisuje njeno gibanje podijeljeno u četiri grupe, gore/dolje, lijevo/desno, naprijed/nazad i rotacija oko z-osi. Koristeći prijenosne funkcije (dobili smo ih od mentora) za svako navedeno gibanje napravljen je model u kojem se preko PID regulatora regulira izvedba zadane naredbe.

Model svake grupe gibanja se sastoji od nekoliko komponenti (prikazano na slici 1.):

1. Ulazna naredba
2. Regulator
3. Blok zasićenja i koeficijent (za ograničavanje izlaza regulatora)
4. Prijenosna funkcija
5. Pretvorba izlaza prijenosne funkcije na isti tip kao i regulirana veličina

Slika 1.; Primjer modela



Nakon zadavanja ulazne naredbe regulator na temelju usporedbe referentnog i reguliranog signala dalje na ulaz prijenosne funkcije daje naredbu kako bi se izvelo zadano gibanje. Ovisno o vrsti gibanja imamo različite ulazne naredbe te ulaze u prijenosnu funkciju:

1. Gore/dolje – naredba je željena visina, a ulaz prijenosne funkcije je brzina
2. Rotacija oko z-osi – naredba je željeni kut rotacija, a ulaz prijenosne funkcije kutna brzina
3. Naprijed/nazad i lijevo/desno – naredbe su željeni prijeđeni put, a ulaz prijenosne funkcije kut nagiba oko x-osi za naprijed/nazad odnosno y-osi za lijevo/desno

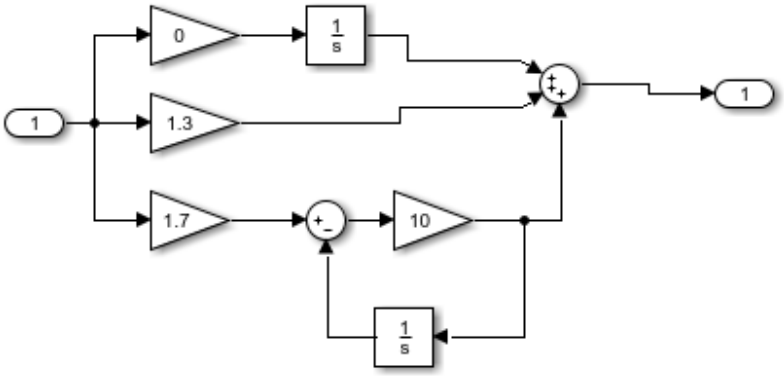
Model za regulator je napravljen tako da se sastoji od sve tri komponente (slika 2.):

1. P – proporcionalna
2. I – integracijska
3. D – derivacijska

te se zatim ovisno o preferencijama mogu koristiti željene komponente, a ostale se postave na nula.

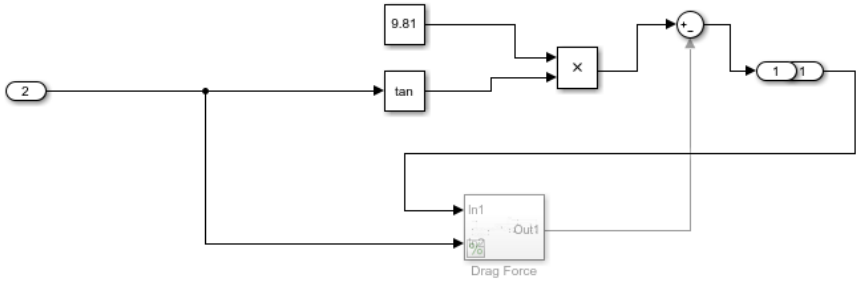
2.1.2 Parametrizacija

Slika 2.; PID-regulator

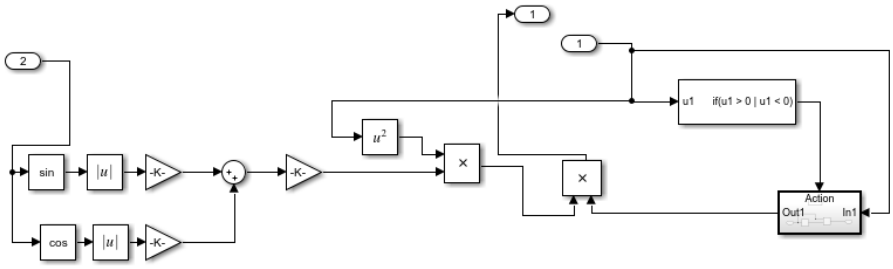


Model za gibanje naprijed/nazad i lijevo/desno je napravljen pod pretpostavkom malog kuta rotacije te se stoga može zanemariti otpor zraka, ali je za usporedbu dodana opcija kojom se otpor može uključiti (slika 3.).

Slika 3.; Pretvorba kuta u akceleraciju



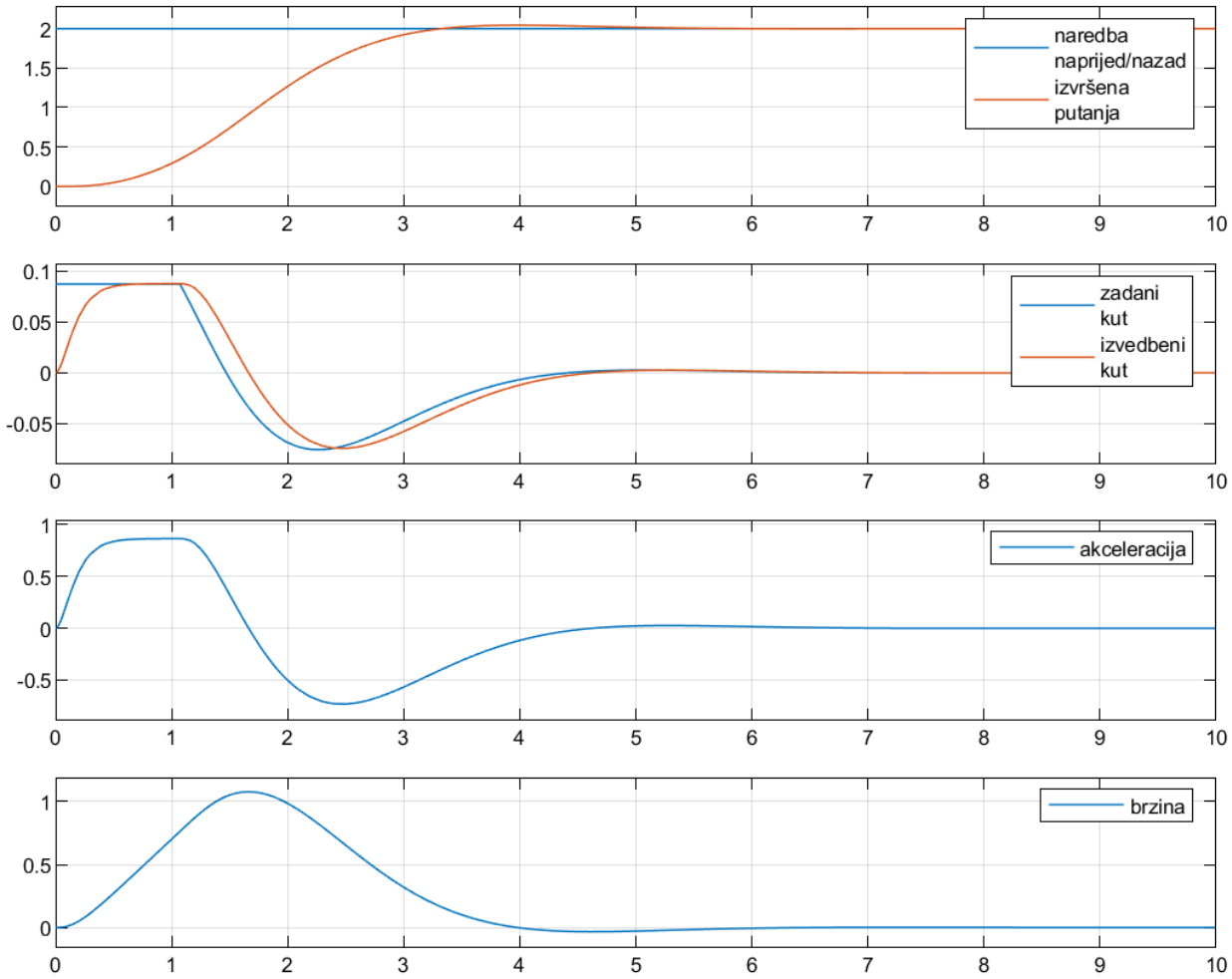
Slika 4.; Opcija uključivanja otpora zraka



Kako bi se provjerila ispravnost modela dodan je element 'scope' preko kojeg možemo u obliku grafa prikazati željene parametre. Slika 5 prikazuje nekoliko elemenata za analizu u primjeru gibanja naprijed/nazad:

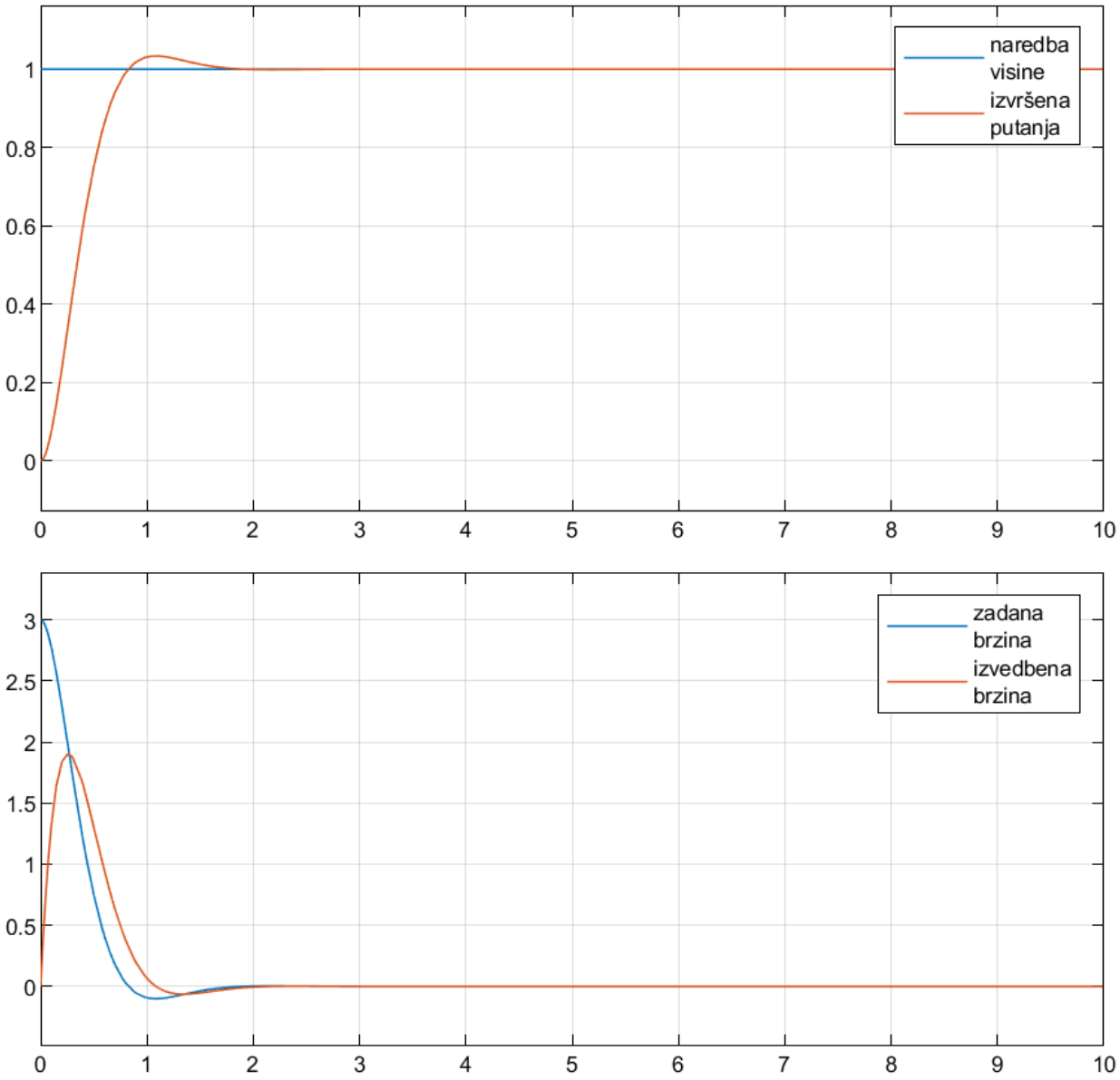
1. Referentna i regulirana putanja
2. Zadani i izvedbeni kut na ulazu/izlazu prijenosne funkcije
3. Akceleracija letjelice
4. Brzina letjelice

Slika 5.; Naprijed/nazad

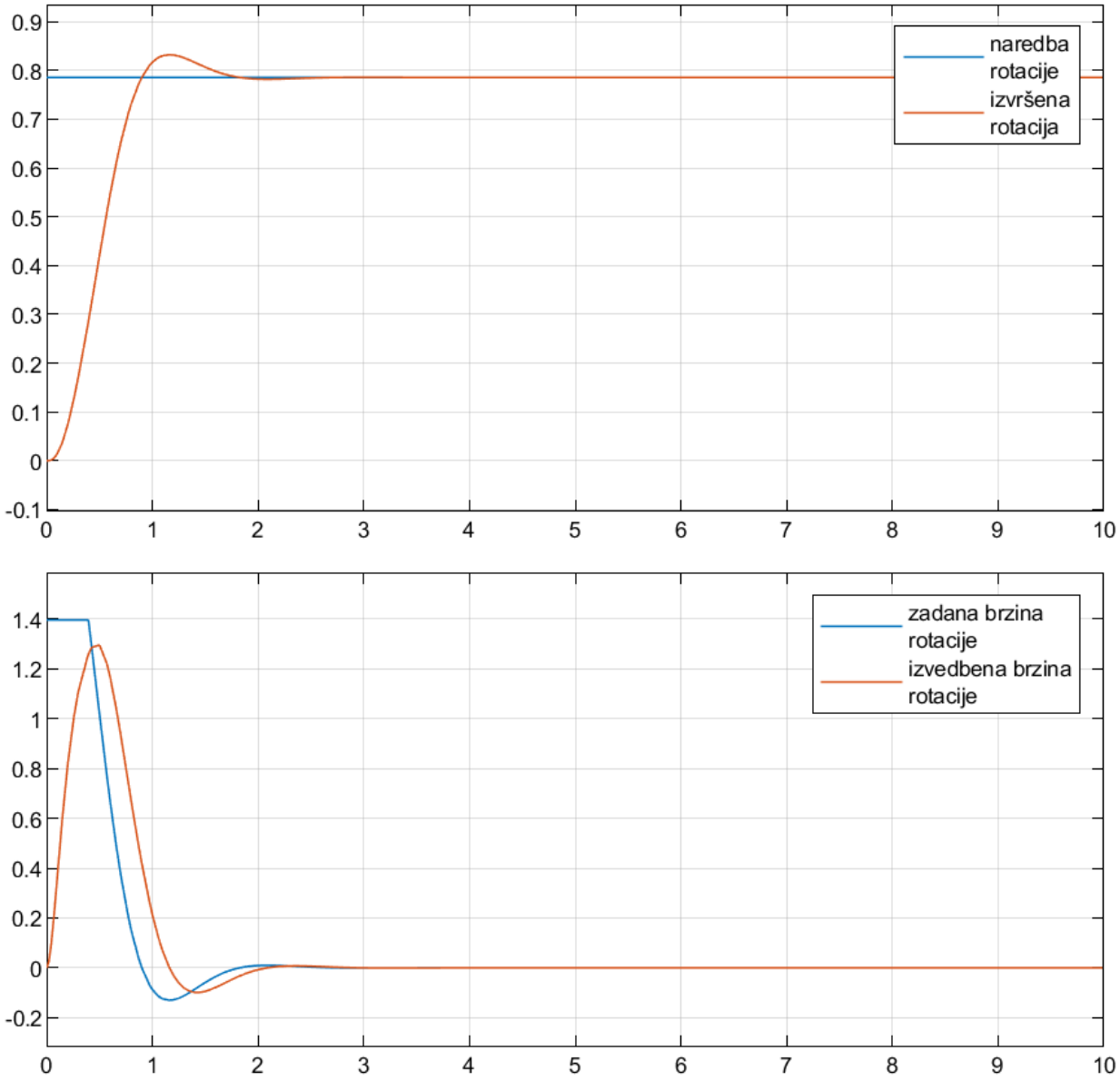


Model gibanja lijevo/desno je izveden na isti način kao i gibanje naprijed/nazad stoga su njegovi odzivi poput onih na slici 5. Na slikama 6. i 7. još se mogu pogledati i odzivi pri gibanju gore/dolje i rotacije oko z-osi.

Slika 6.; Gore/dolje



Slika 7.; Rotacija oko z-osi



2.1.3 Diskretizacija

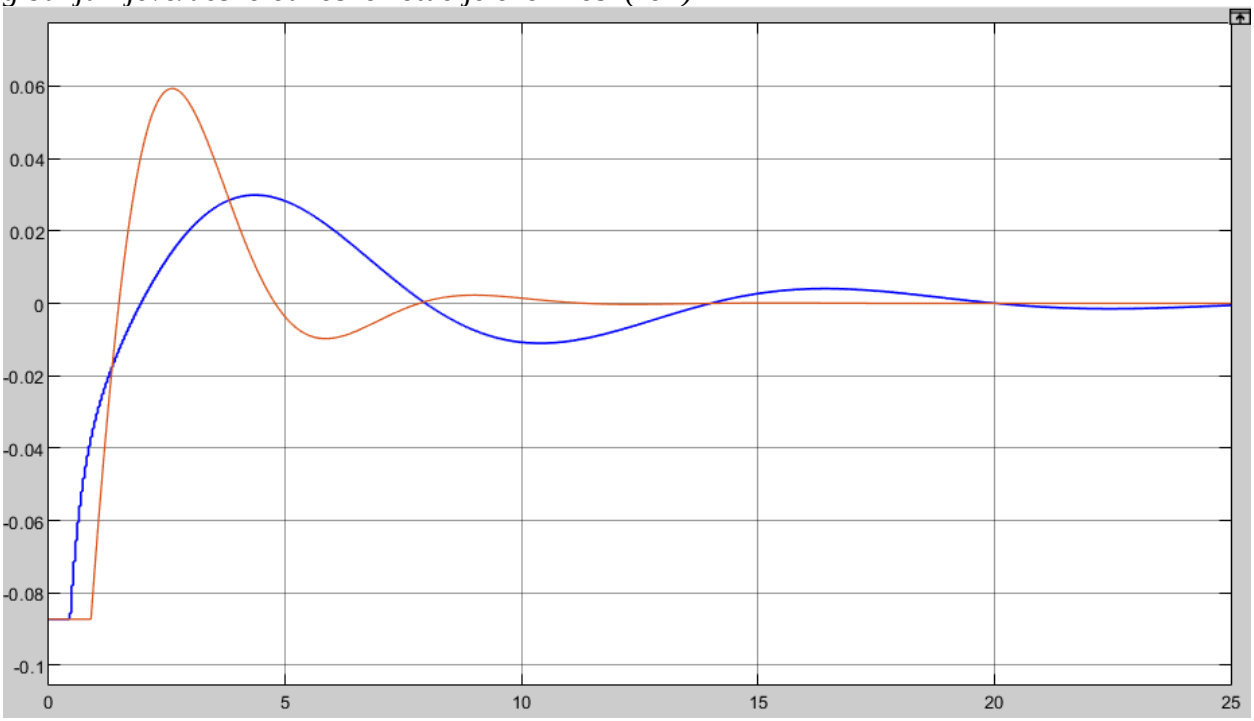
Nakon izrade matematičkog modela kretanja letjelice trebalo je diskretizirati regulatore za sva četiri načina gibanja. Prije diskretizacije regulatori su parametrizirani na način da se nasumičnim odabirom vrijednosti koeficijenata nastojalo postići željeno gibanje. Nakon što su odabrani koeficijenti dali željeni rezultat odnosno gibanje mogli smo započeti sa proces diskretizacije regulatora.

U matematičkom modelu letjelice vidimo da rotacije oko Y-osi (pitch) te X-osi (roll) koriste proporcionalnu i derivacijsku komponentu što znači da u tom slučaju imamo PD regulator dok je rotacija oko Z-osi (yaw) te gibanje po Z-osi ostvareno korištenjem isključivo proporcionalne komponente što znači da u tom slučaju imamo P regulator. Diskretizacija je provedena korištenjem postupka unazadne Eulerove diferencije što je vidljivo i na odzivima koji će biti prikazani u nastavku.

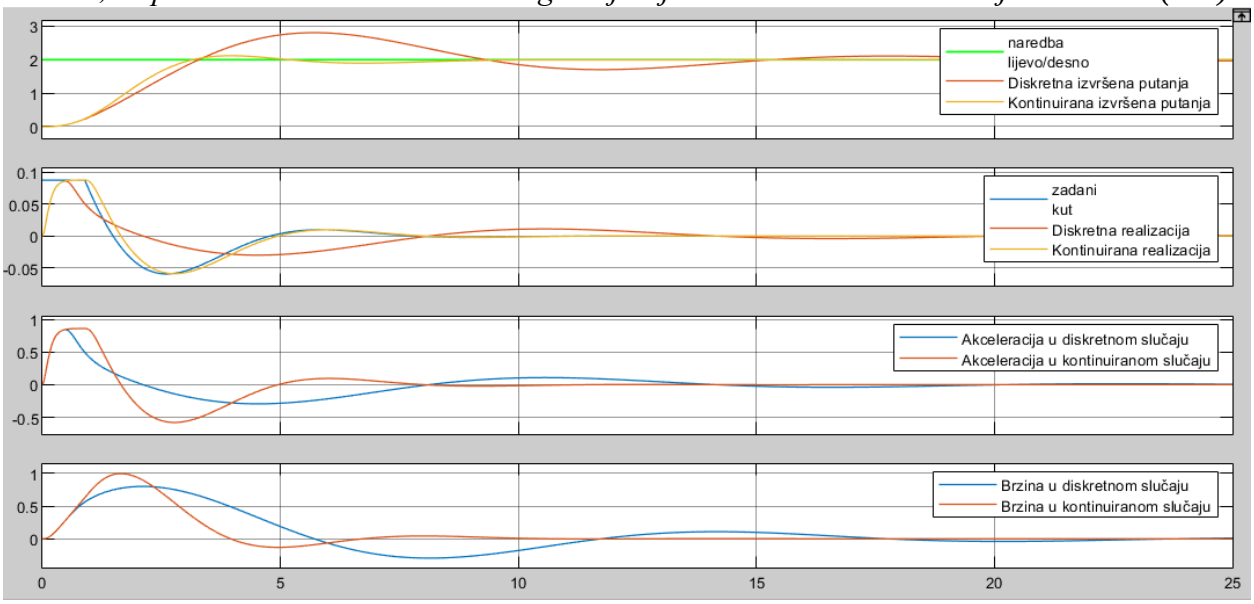
Sama svrha diskretizacije jest zamijeniti realni kontinuirani regulator odgovarajućom diskretnom prijenosnom funkcijom koja će davati isti rezultat i ponašanje, ali u diskretnoj domeni. Diskretizaciju regulatora je nužno provesti iz više razloga od čega je jedan od osnovnih onaj da računalo ne može pratiti zbivanja kontinuirano i primati podatke tijekom cijelog gibanja već treba odrediti određeni period kojim će se podatci slati i primati. Za potrebe ovog projekta korišten je period od 0.04 sekundi. To vrijeme odgovara periodu kojim letjelica šalje fotografije odnosno snimke labirinta računalu pomoću čega se ostvaruje pravilna navigacija Sphero robota.

U nastavku slijede grafovi usporedbe odziva pri korištenju kontinuiranog regulatora i diskretizirane varijante te usporedbe konačnih odziva.

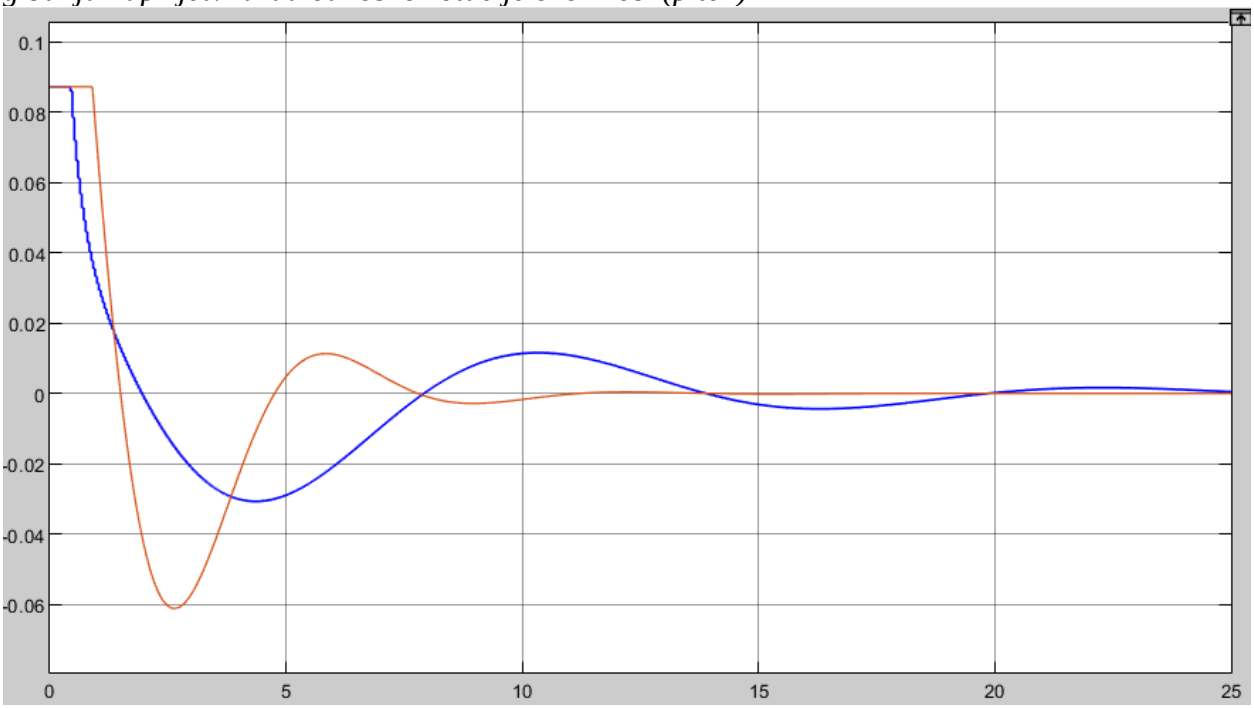
Slika 8.; Usporedba odziva kontinuiranog PD regulatora i njegove diskretizirane varijante kod gibanja lijevo/desno odnosno rotacije oko X-osi (roll)



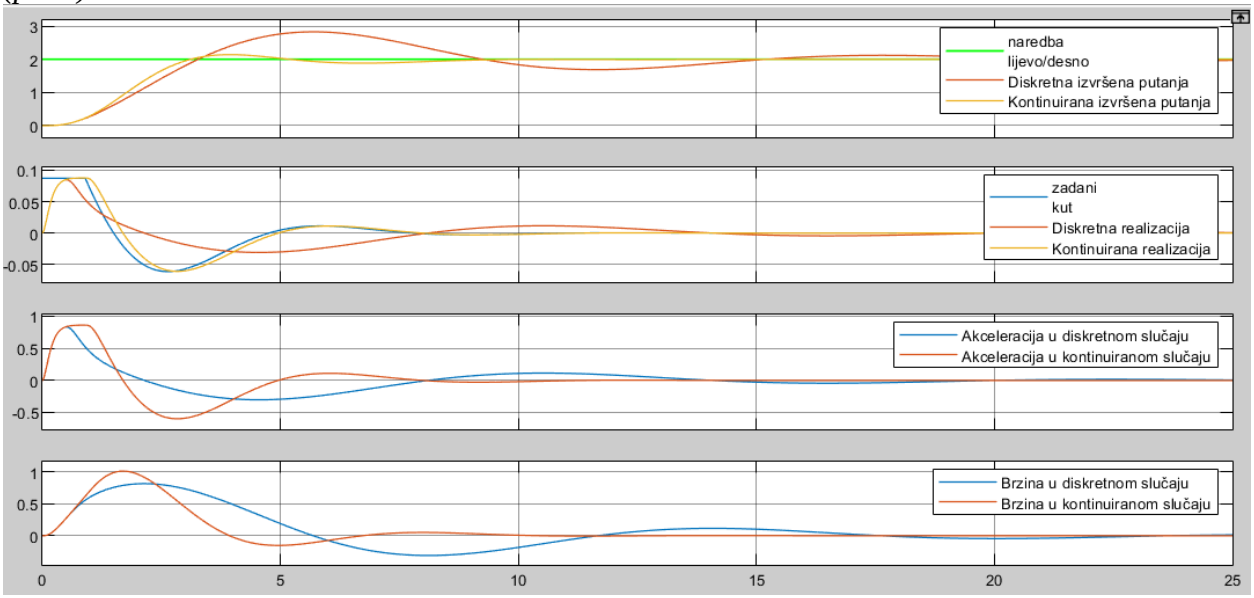
Slika 9.; Usporedba konačnih odziva kod gibanja lijevo/desno odnosno rotacije oko X-osi (roll)



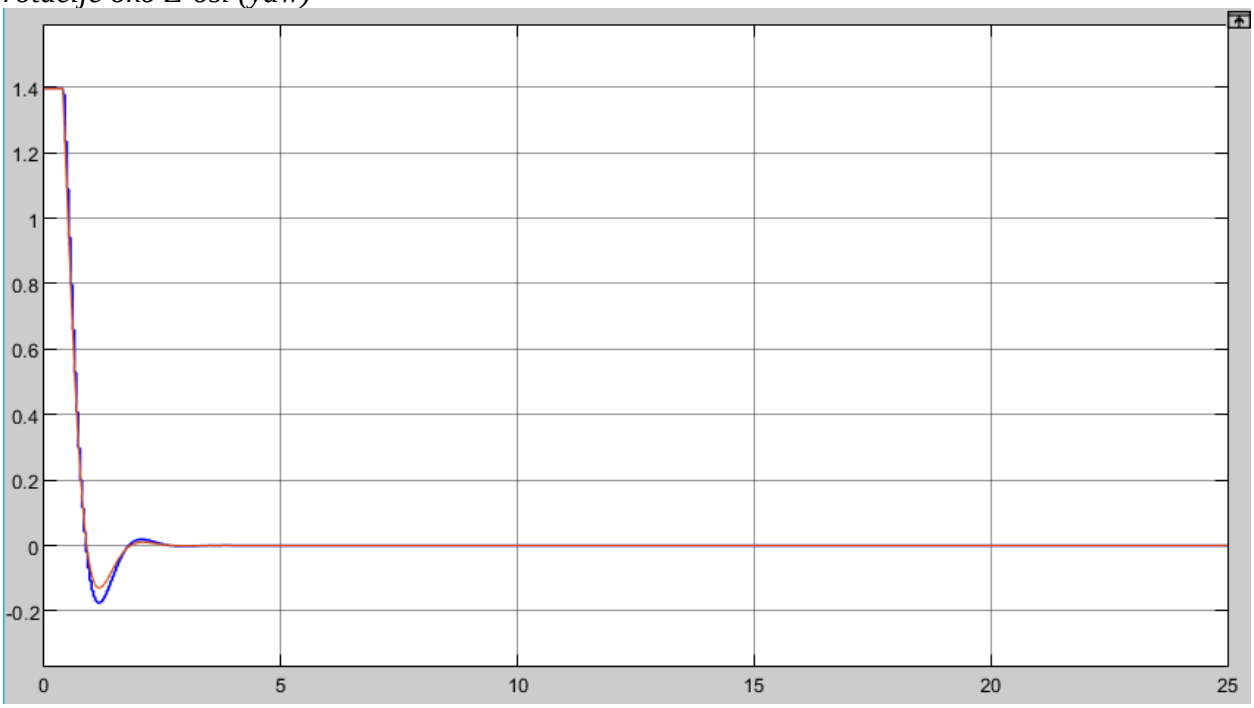
Slika 10.; Usporedba odziva kontinuiranog PD regulatora i njegove diskretizirane varijante kod gibanja naprijed/nazad odnosno rotacije oko Y-osi (pitch)



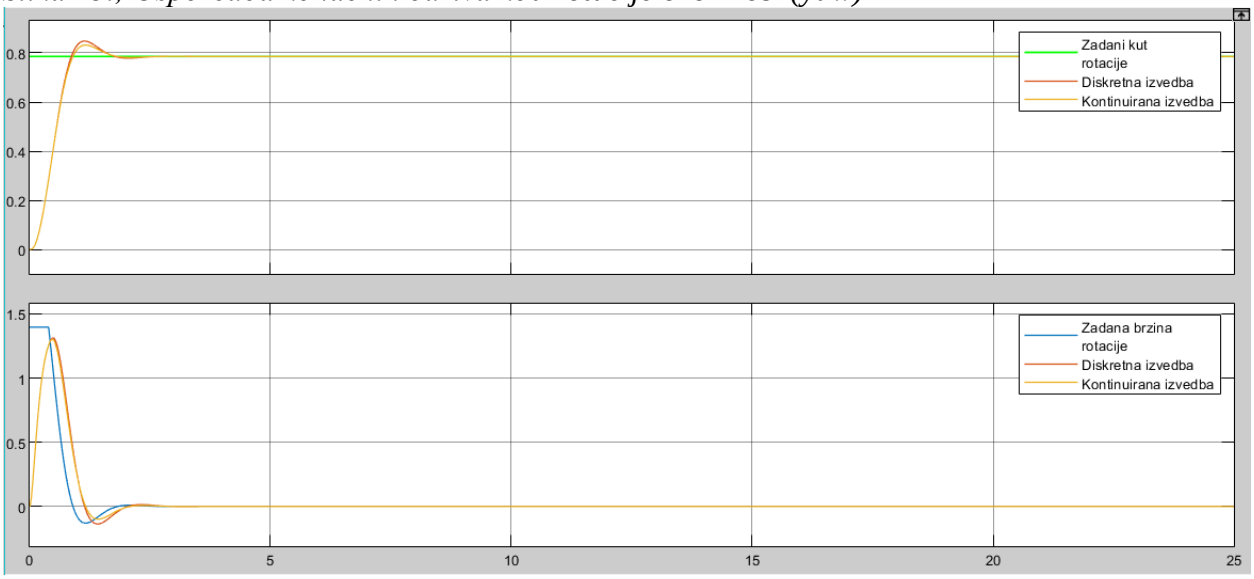
Slika 11.; Usporedba konačnih odziva kod gibanja naprijed/nazad odnosno rotacije oko Y-osi (pitch)



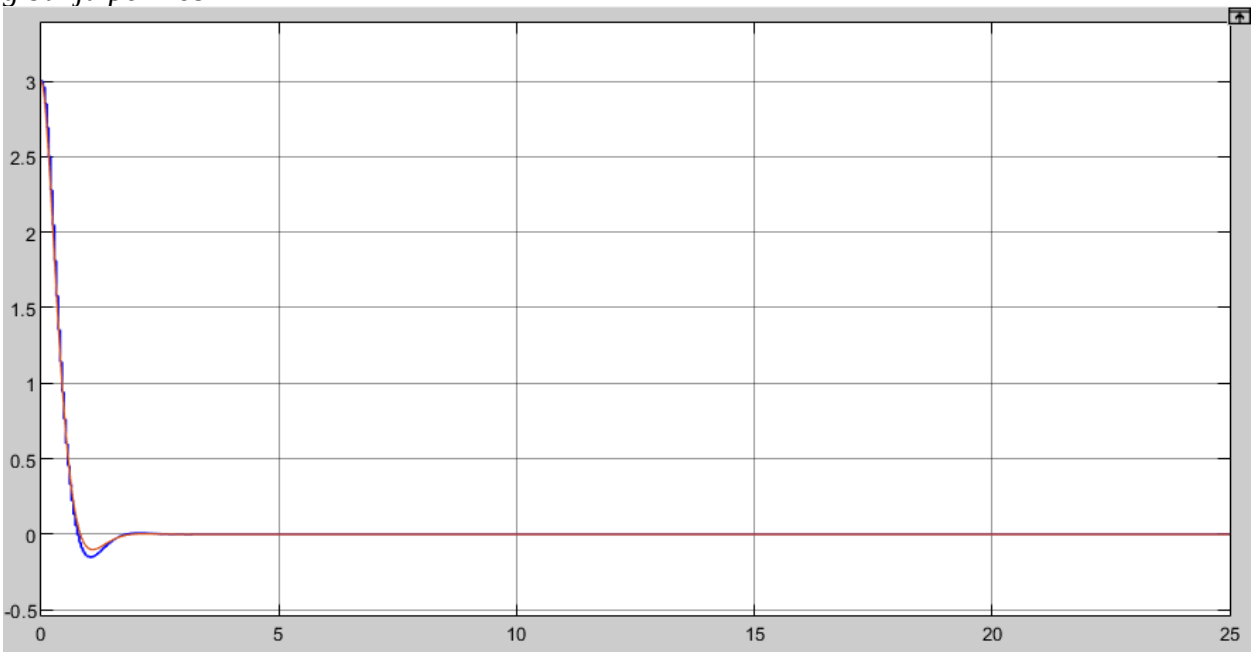
Slika 12.; Usporedba odziva kontinuiranog P regulatora i njegove diskretizirane varijante kod rotacije oko Z-osi (yaw)



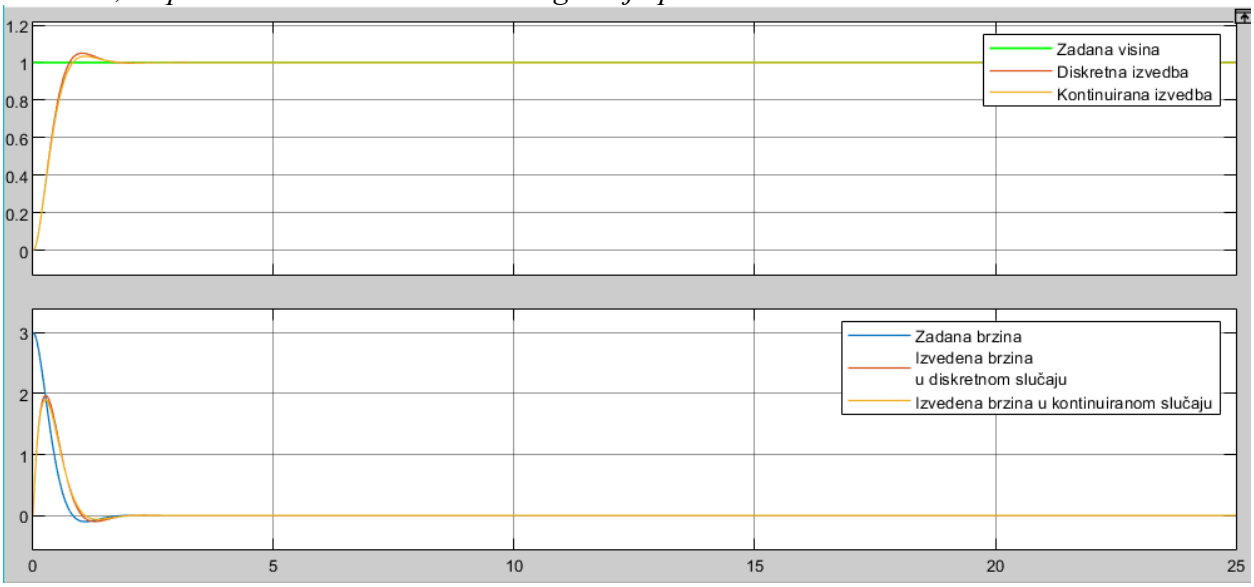
Slika 13.; Usporedba konačnih odziva kod rotacije oko Z-osi (yaw)



Slika 14.; Usporedba odziva kontinuiranog P regulatora i njegove diskretizirane varijante kod gibanja po Z-osi



Slika 15.; Usporedba konačnih odziva kod gibanja po Z-osi



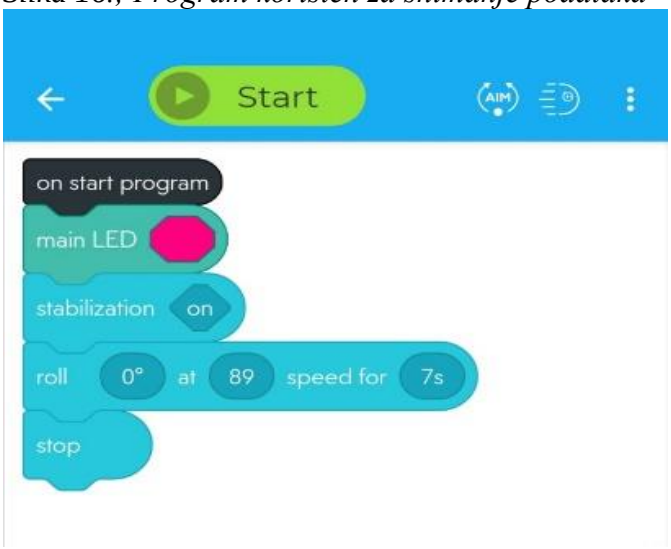
2.2 Sphero SPRK+

2.2.1 Matematički model i parametrizacija

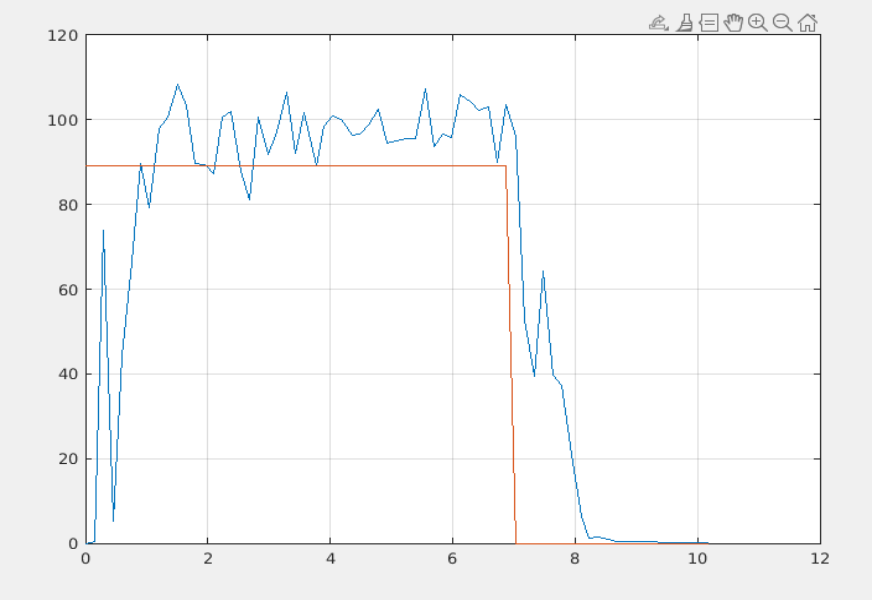
Za izradu Sphero-vog regulatora bilo je potrebno napravit nešto više koraka. Prvi korak kod modeliranja modela bio nam je odrediti prijenosnu funkciju. Za to smo koristili Matlabov paket S*ystem Identification Toolbox*. Podatke dobivene snimanjem kretanja Sphero-a pomoću aplikacije Sphero Edu (Slika 16.) - podatci ovisnosti brzine o vremenu (Slika 17.) smo unijeli u Matlab i zatim smo u *iddata* blok (Slika 18.) stavili pobudu i odziv sustava. U toolbox-ov GUI (Slika 19.) smo unijeli podatke te smo dobili gotovu prijenosnu funkciju koja je bila PT1 član, odnosno imala je 0 nula i 1 pol. Prijenosna funkcija glasi:

$$\frac{2.0019}{s + 1.8212}$$

Slika 16.; Program korišten za snimanje podataka



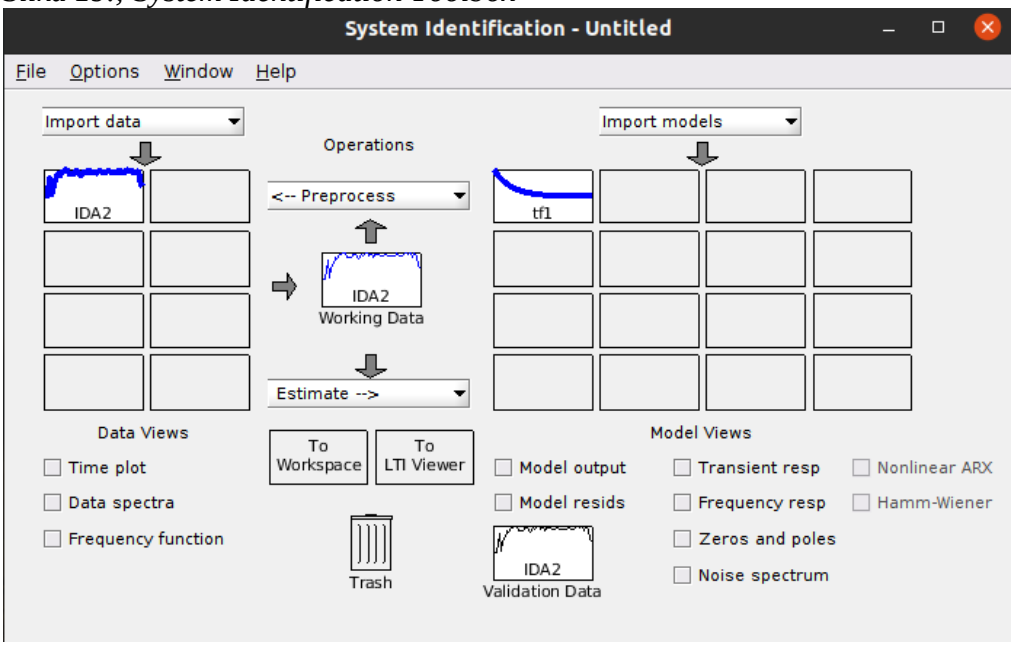
Slika 17.; v/t graf dobiven programom Sphero Edu (crveno – pobuda, plavo -odziv)



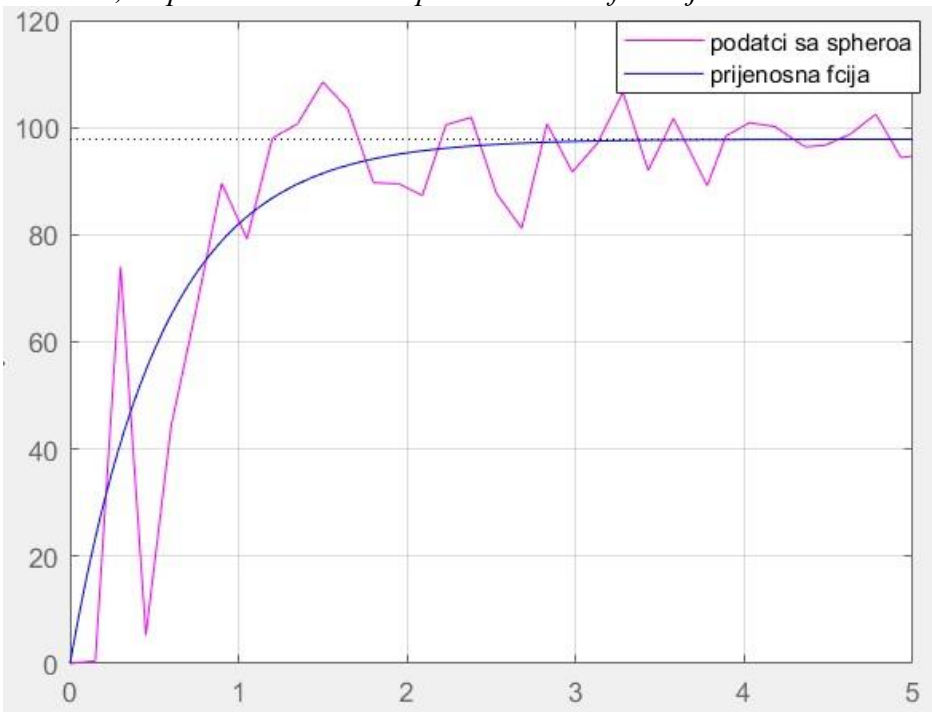
Slika 18.; Iddata blok

```
IDA2 = iddata(Velocity_Y, Input, 0.15);
plot(IDA2)
```

Slika 19.; System Identification Toolbox

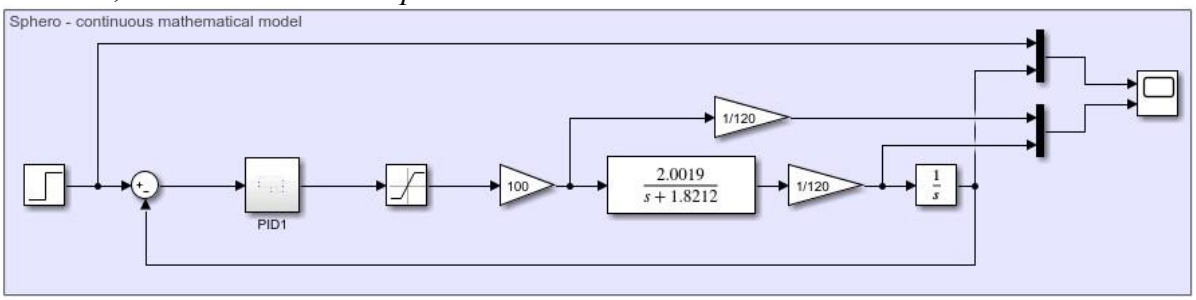


Slika 20.; Usporedba dobivenih podatak i naše funkcije

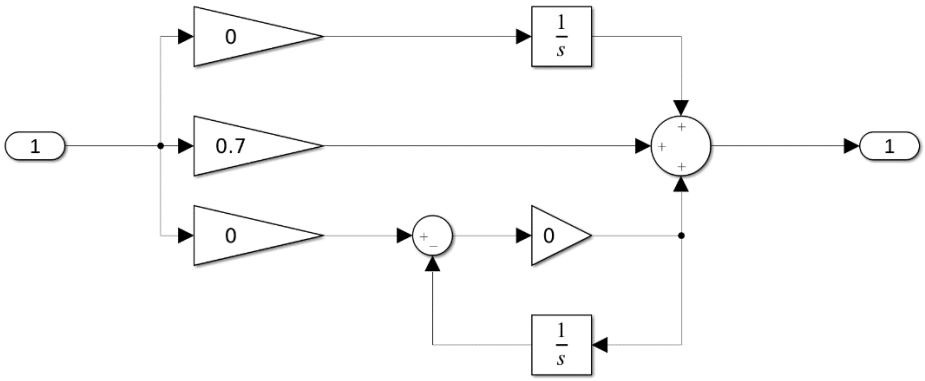


Pomoću dobivene prijenosne funkcije dolazimo do matematičkog modela koji reguliramo P regulatorom (izvedba je moguća i s PD regulatorom koji daje brži odziv, ali kako bismo izbjegli nadvišenja u poziciji koja rezultiraju zabijanjem u zid labirinta odlučili smo se za P regulator). Dijelovi korišteni u modelu su slični dijelovima korištenima za Bebop-a (Sličnosti se odnose na slike 1. i 21.). Razlika je što smo koeficijent postavili na 100 kao ograničenje za brzinu gibanja Sphero-a iako je maksimalna brzina 255. Također izlaz iz prijenosne funkcije smo podijelili s 120, zato da bismo dobili brzinu u m/s kako bi se u povratnoj vezi dobili vrijednost u metrima.

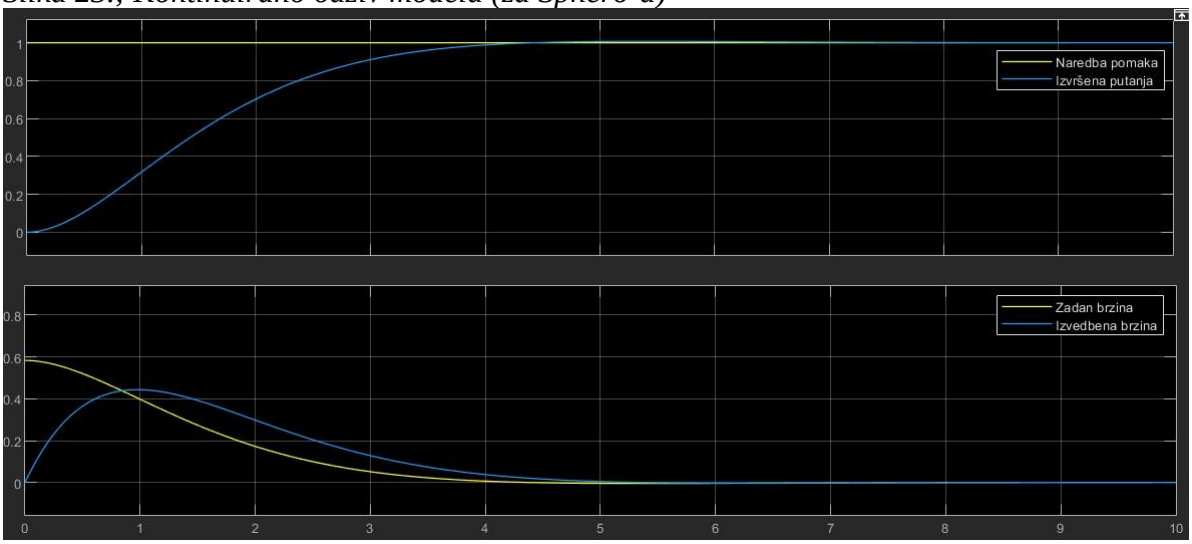
Slika 21.; Matematički model Sphero-a



Slika 22.; PID regulator Sphero-a



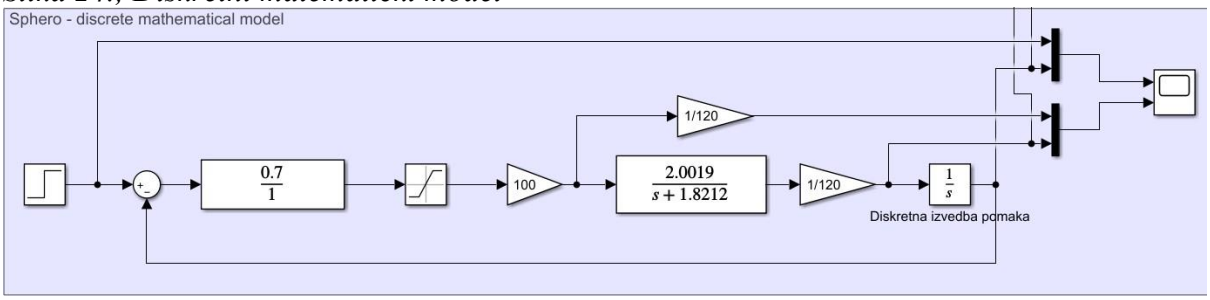
Slika 23.; Kontinuirano odziv modela (za Sphero-a)



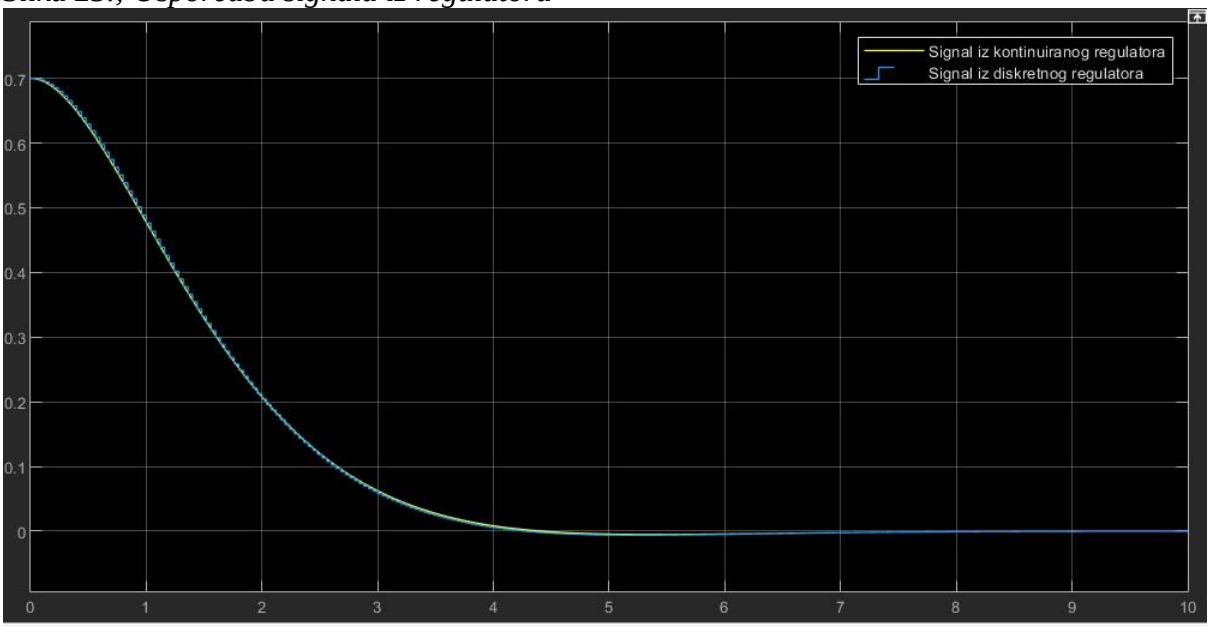
2.2.2 Diskretizacija

Diskretizacija i njena svrha korištenja je opisana u diskretizaciji modela za letjelicu.

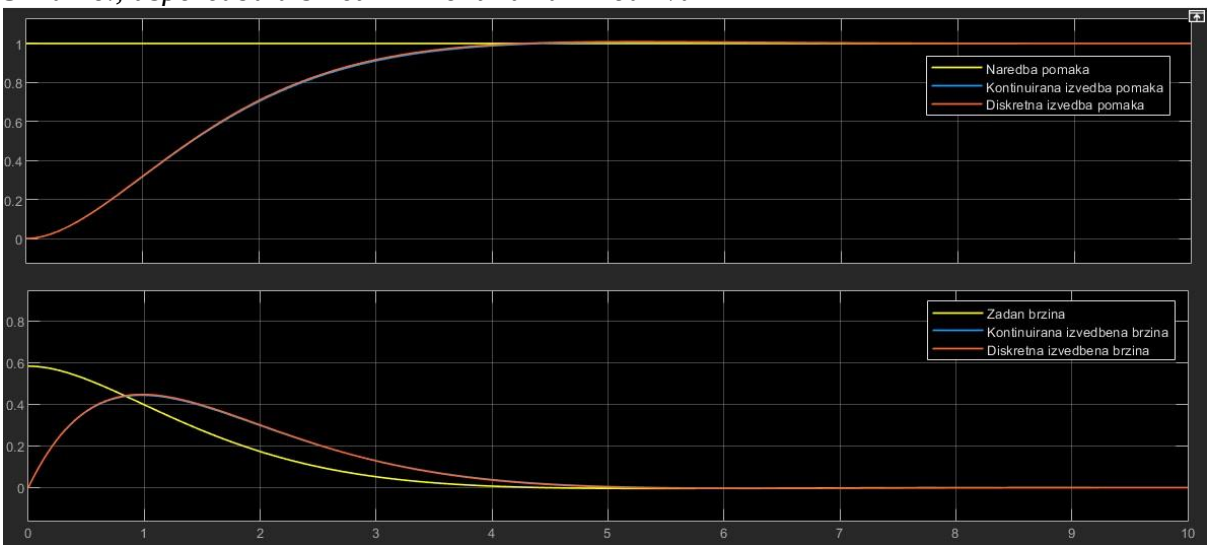
Slika 24.; Diskretni matematički model



Slika 25.; Usporedba signala iz regulatora



Slika 26.; usporedba diskretnih i kontinuiranih odziva



3. Implementacija regulatora u ROS-u

3.1 Uvod

Sav programski kod koji se koristio se nalazi na Github-a (9. točka u literaturi) te se u nadolazećim paragrafima svaka funkcija, skripta, klasa itd. nalazi na tom Github repozitoriju.

PID regulator implementiran je u Python skripti kao modularni dio koda koji koriste dron i Sphero. Regulator se parametrizira i upravlja pomoću drugih skripti u kojima je on instanciran. U implementaciji imamo tri skripte koje kontroliraju različite regulatore:

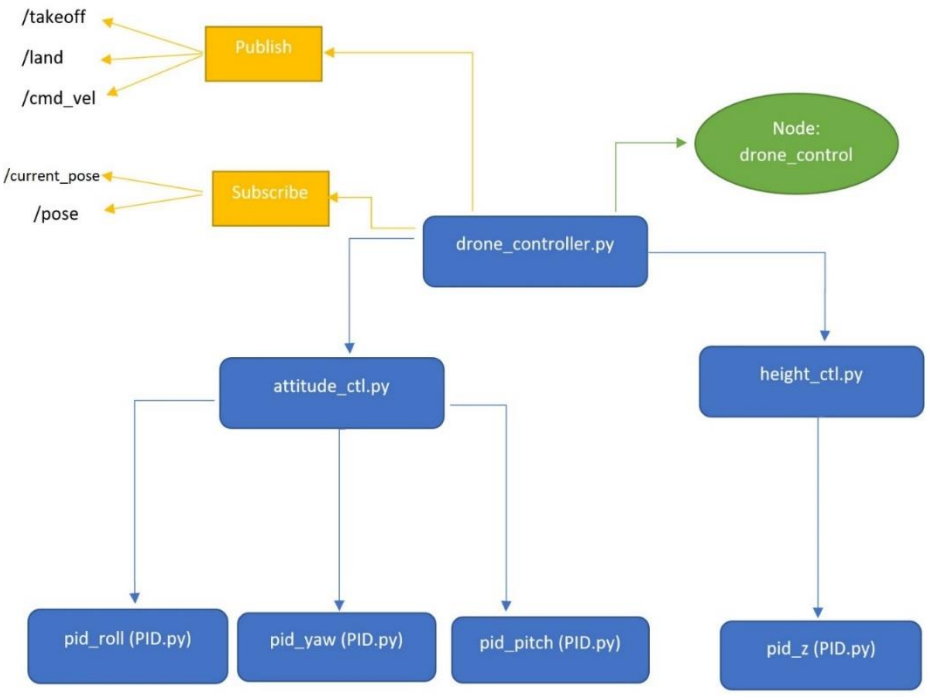
1. attitude_ctl.py – za rotacije drona
2. height_ctl.py – za vertikalni pomak drona
3. sphero_ctl.py – za kontrolu Sphero-a

3.2 Implementacija regulatora za Bebop-a

Kontroleri za regulatore drona su implementirani tako da komuniciraju sa dronovim driverom na raznim temama („topics“). Glavni kontroler (drone_controller.py) se pretplati („publish“) i objavljuje („subscribe“) na teme kao što je prikazano na slici 27.:

/takeoff i /land - služe za slanje signala drona za automatsko uzlijetanje ili spuštanje.
/cmd_vel - služi za slanje brzina dronu
/current_pose - primanje informacija za trenutnu poziciju drona
/pose - primanje informacija za referentnu točku regulatora (gdje dron želi doći)

Slika 27.; Dijagram kontrolera za Bebop-a



Glavni kontroler zatim prosljeđuje dobivene informacije pod-kontrolerima koji direktno komuniciraju sa PID regulatorima. Attitude_ctl.py kontrolira orijentacije, a s time i brzine drona, dok height_ctl.py kontrolira samo visinu drona. Sami regulatori koriste kontinuirane vrijednosti izračunate u 2.1.2 poglavlju, što znači da su regulatori orijentacije PD tipa, a regulator visine P tipa. Skripta pid.py implementira potpuni PID regulator, pa je za pretvorbu u navedene regulatore potrebno postaviti $k_i = 0$, odns. $k_i, k_d = 0$.

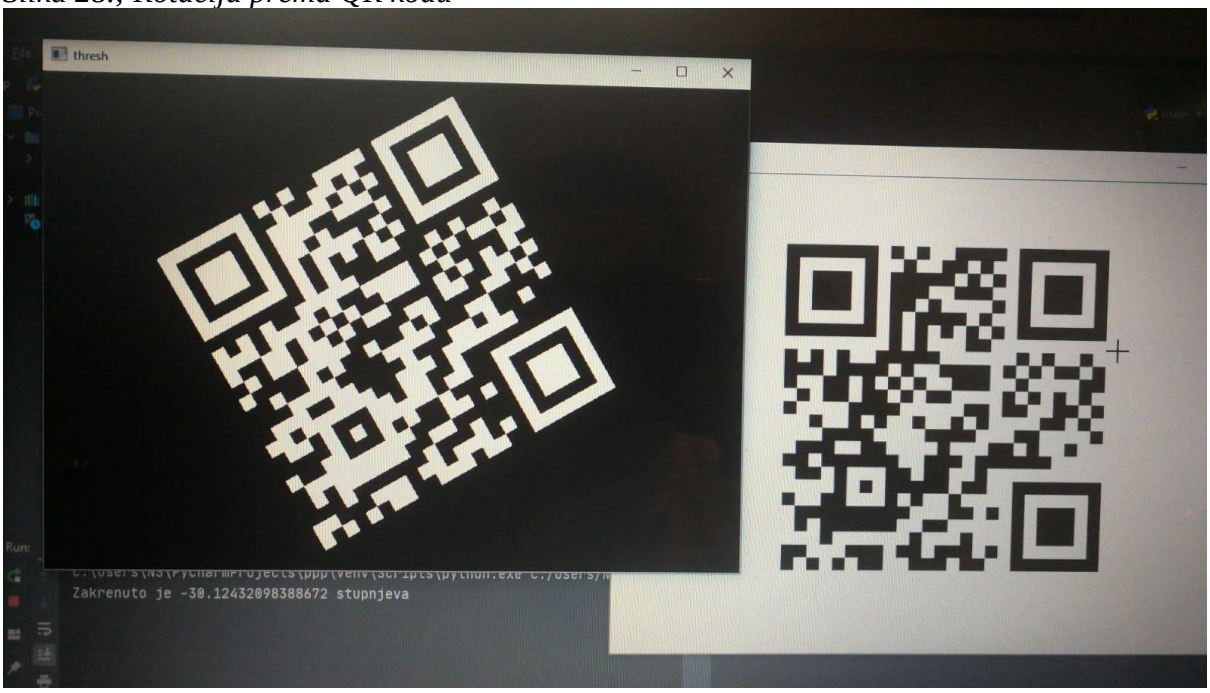
Izračunate vrijednosti iz regulatora se propagiraju kroz pod-kontrolere natrag do glavnog kontrolera, koji zatim spaja informacije iz obje skripte i objavljuje ih na /cmd_vel.

3.2.1 Pozicioniranje Bebop-a

Glavna zadaća triju funkcija *orijentirajSe()*, *pozicionirajSe()* i *izracunaj_koordinate_vrhova()* unutar skripte *position.py* služe kako i njihov sami naziv kaže orijentaciji Bebop drona i njegovom pozicioniranju iznad labirinta te dobivanju koordinata vrhova samog labirinta.

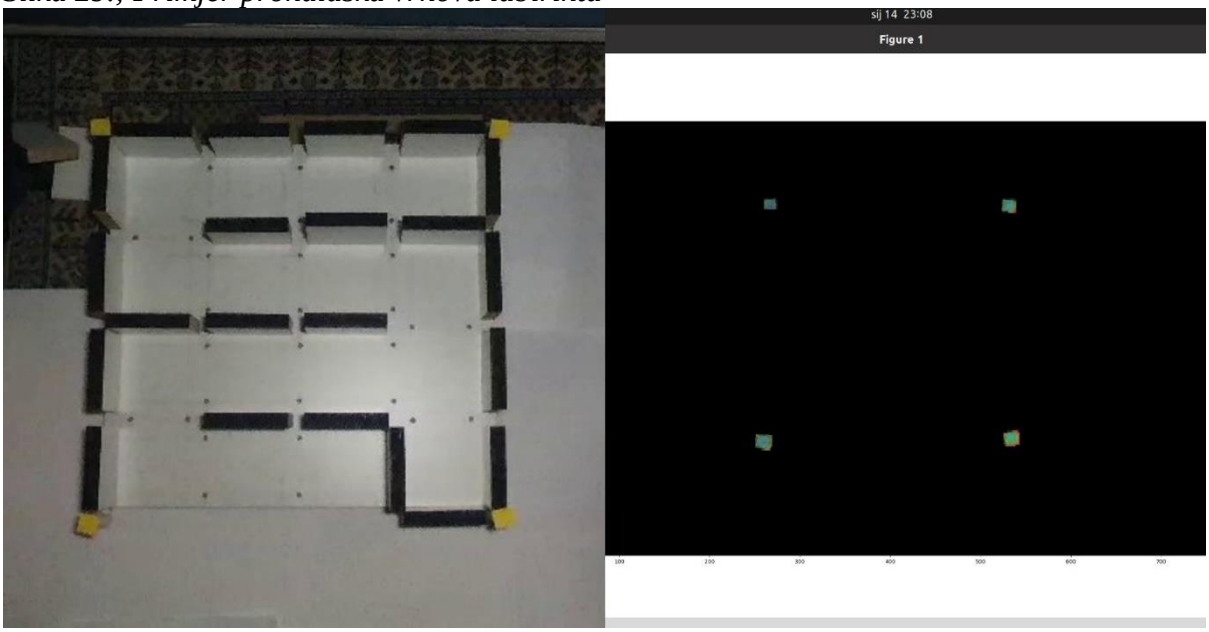
Orijentacija drona vrši se pomoću QR koda čiju sliku funkcija *orijentirajSe()* dobiva kao ulazni argument te pomoću poziva funkcije *minAreaRect()* „crta“ oko QR koda pravokutnik minimalne površine te se tako može dobiti kut pod kojim je Bebop zakrenut u odnosu na QR kod. Zatim se na temelju vrijednosti tog kuta (može biti pozitivna ili negativna) dron može zakrenuti u određenom smjeru kako bi bio ispravno orijentiran za daljnje kretanje prema labirintu. Na slici 28. je primjer kako se prepoznaje QR kod.

Slika 28.; Rotacija prema QR kodu



U funkciji *pozicionirajSe()* konstantno se pomoću funkcije *Canny()* te pomoću filtra za žutu boju određuju „rubovi“ objekata sa slike te je za dodatno uglađivanje robova korištena funkcija *GaussianBlur()*. Zatim se pomoću funkcije *findContours()* dobije lista kontura sa slike te se tako uspješno pronalaze vrhovi labirinta. Prilikom polijetanja, dron vidi samo jedan vrh labirinta, zatim se kreće ulijevo te prepozna i drugi vrh stoga se započinje kretati unaprijed sve dok ne prepozna sva četiri vrha labirinta te se zaustavi s kretanjem i prelazi u autopilot način rada. Na slici 29. vidi se kako Bebop prepoznaje vrhove labirinta

Slika 29.; Primjer pronalaska vrhova labirinta



3.3 Implementacija regulatora za Sphero-a

Zbog mehaničkih ograničenja Sphero zadajemo odjednom gibanje samo po jednoj osi, bilo to x ili y-os. Prvo se unutar sphero_ctl.py upisuje vrijednost P regulatora (izračunatog u matematičkom modelu) te se pomoću pid.py računa i vraća potrebna vrijednost koja se objavljuje kao /cmd_vel za Sphero-a. Nakon dobivanja potrebne vrijednost od pid.py, gleda se, kreće li se Sphero po x ili y-osi te se pomoću *run()* objavljuju /cmd_vel vrijednosti za x ili y-os.

3.3.1 Manualno upravljanje Sphero-a

Jedan od zadataka bio je napraviti skriptu pomoću koje bi se Sphero robot mogao upravljat putem tipkovnice, ili joystick-a. Mi smo odlučili napraviti skriptu za upravljanje pomoću tipkovnice. Korištenjem postojećeg ROS-ovog paketa „teleop“ napravila se skripta „sphero_keyboard_control.py“ koja koristeći inpute preko tipkovnice objavljuje /cmd_vel za Sphero-a.

Preko skripte se Sphero upravlja po svojem referentnom koordinatnom sustavu po x i y osi pomoći tipki „a“ („lijevo“, -x), „d“ („desno“, x), „w“ („naprijed“, y), „s“ („nazad“, -y). Također pomoću tipki „r“ i „f“ može se povećavati ili smanjiti brzina koja je objavljuje. Pomoću tipki „m“ i „M“ (Shift + m) kontrolira se /manual_calibration tema koji Sphero-a postavlja u mod manualne kalibracije (upali se lampica na Sphero-u koju nam pokazuje Sphero-ov referentni koordinatni sustav u -y smjeru. Tako možemo postaviti Sphero-a da odgovara našem koordinatnom sustavu) ili normalan način rada.

4. Praćenje Sphero-a

4.1 Prepoznavanje Sphero-a po bojama

Obzirom da naši Sphero-i nemaju funkcionalni povrat pozicije (a i on nije precizan), upute za kretanje dobivaju na temelju obrađenog video prijenosa s bespilotne letjelice Bebop. Prvi zadatak bio je na slici prepoznati 3 različita Sphero-a. Jedan pronalazi put kroz labirint izbjegavajući ostala dva.

Sliku obrađujemo u Python-u koristeći OpenCV biblioteku. OpenCV radi sa slikama te kako bismo imali konstantan video obrađujemo 25 slika u sekundi (računajući da letjelica snima 25 fps). Budući da filtre radimo sa HSV vrijednostima, a ne RGB, naredbom *cv2.cvtColor(imageFrame,cv2.COLOR_BGR2HSV)* transformiramo sliku s navedenim vrijednostima.

Na dobivenoj slici primjenjujemo filtere propuštajući piksele samo određenog raspona boja definiranih za tu boju Sphero-a. Sphero-i će biti osnovnih boja: crveni, zeleni i plavi pa radimo ″masku″ propuštajući samo piksele za tu boju Sphero-a.

Na toj filtriranoj slici pronalazimo konture objekata i definiranjem kruga minimalnog polumjera možemo dobiti njihovu poziciju unutar slike u x i y koordinatama te polumjer objekta. Koordinate su iskazane u udaljenosti središta objekta od ishodišta u pikselima (ishodište je gornji lijevi kut slike, a desno i dolje su pozitivni x i y smjerovi respektivno). Koordinate koje dobivamo koristimo za regulaciju brzine Sphero-a koji se navigira kroz labirint. Također pomoću koordinata koje dobijemo od ostalih Sphero-a (neprijatelja), glavnom Sphero-u objavljujemo naredbe ako se treba zaustaviti ili nastavit nakon što je neprijateljski Sphero prošao.

Ovaj proces stavljen je u jednu klasu, pa za svakog Sphero-a moramo inicijalizirati posebnu klasu, te u tu klasu proslijediti željenu boju. Na ovaj način smo pojednostavili proces dodavanja nove boje Sphero-a, pošto se za njega samo treba definirati novi spektar boja za masku, te inicijalizirati nova klasa.

4.2 Izbjegavanje „neprijateljskih“ Sphero-a

Poznavajući koordinate npr. crvenog i zelenog Sphero-a, tzv. ″neprijatelji″ (iz koda za prepoznavanje boja) i orijentacije našeg Sphero-a koji rješava labirint, možemo te parametre koristiti za izbjegavanje istih. U funkciji možemo provjeravati nalazi li se ″neprijatelj″ na planiranoj putanji našeg Sphero-a te ako je put zapriječen funkcija vraća „False“, odnosno „True“ ako je osiguran slobodan prolaz. Ako se naš Sphero kreće po x-osi provjeravamo x koordinate neprijatelja te ako se kreće po y-osi, provjeravamo im y koordinate. Ako se x ili y koordinate našeg Sphero-a i koordinate neprijatelja podudaraju treba to dojaviti glavnom programu pomoću „boolean“ vrijednosti.

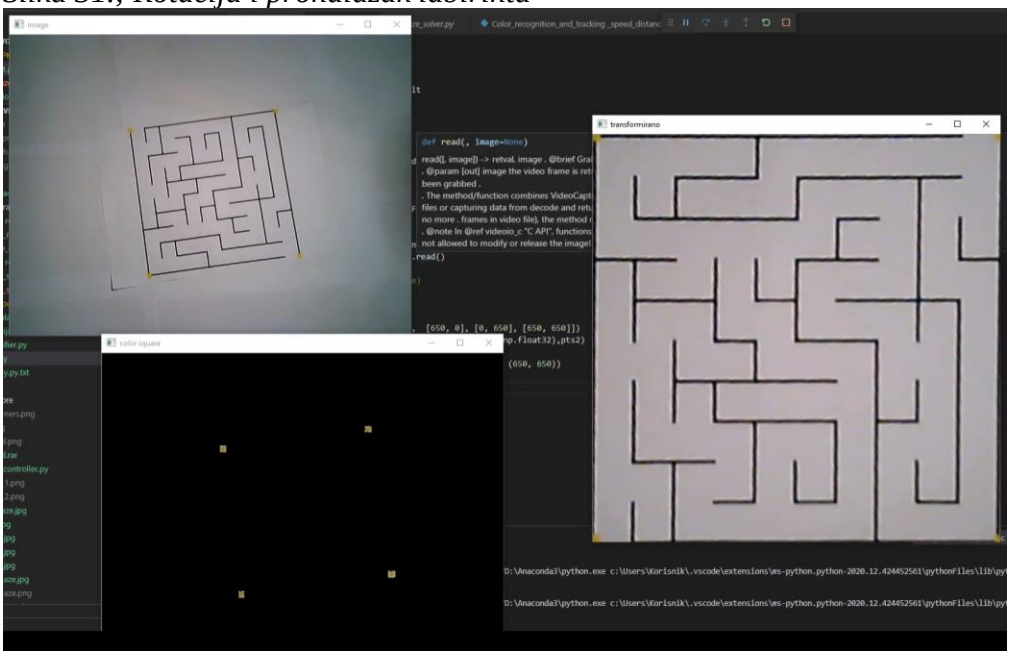
5. Rješavanje labirinta

Za rješavanje labirinta koristi se transformirana slika veličine 650x650. Ona se filtrira tako da dobijemo crno-bijelu sliku takvu da je sve osim zidova bijeli piksel koja se prosljeđuje funkciji *find_path()*. Rješavanje labirinta koristeći piksele traje predugo te se zato labirint prvo pojednostavljuje na labirint manjih višestruko manjih dimenzija koji sadrži znakove '#' za zid i '.' za prolaz. Pojednostavljivanje radi funkcijom *simplify()* koja kao parametar prima sliku te duljinu zidova i prolaza u pikselima. Zato što je kompleksno automatski tražiti tu duljinu, nju smo postavili na vrijednosti prikladne za našu uporabu. Dimenzije labirinta prema broju kvadrata se također moraju namjestiti prije pokretanja, kod nas je dimenzija labirinta n=8 (na slici 30. vidi se primjer labirinta veličine n=8).

Slika 30.; Labirint veličine n = 8

Za uspješno rješavanje labirinta potrebne su i koordinate sva četiri vrha istoga te se pomoću funkcije *izracunaj_koordinate_vrhova()*, koja funkcionira na sličan način kao i funkcija *pozicionirajSe()* korištenjem *Canny()* i *findContours()* funkcija te funkcije *moments()*, dobije sortirana lista koordinata vrhova labirinta u kojoj su vrhovi redom: gornji lijevi, donji lijevi, gornji desni i donji desni. Na slici 31. vidi se primjer rotacije i pronalaska labirinta.

Slika 31.; Rotacija i pronalazak labirinta

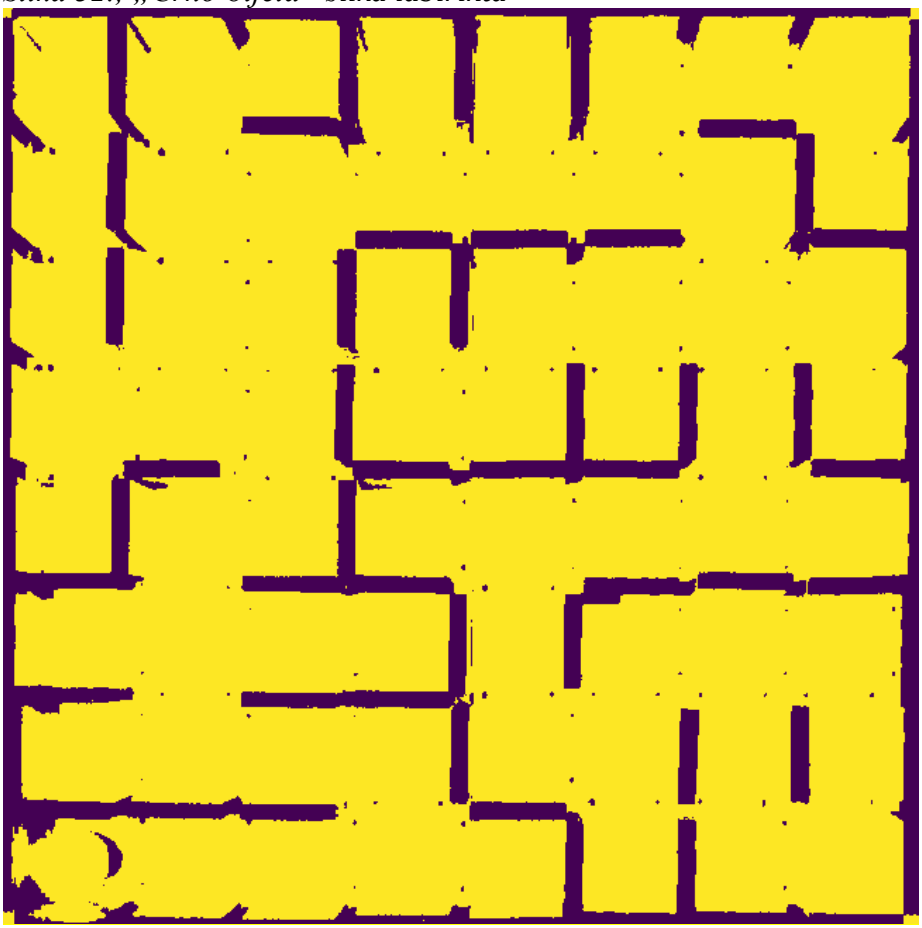


Nakon što je labirint pronađen i rotiran slijedi pojednostavljivanje. Pojednostavljivanje se odvija tako da najprije u liste zabilježimo vrijednosti pixela svakog reda i svakog stupca. Za prvi red se pozicioniramo na *start_row = black_width+white_width/2* kako bismo bili na sredini prolaza. Zatim se pomičemo vertikalno za duljinu zida i prolaza dok ne dođemo do kraja.

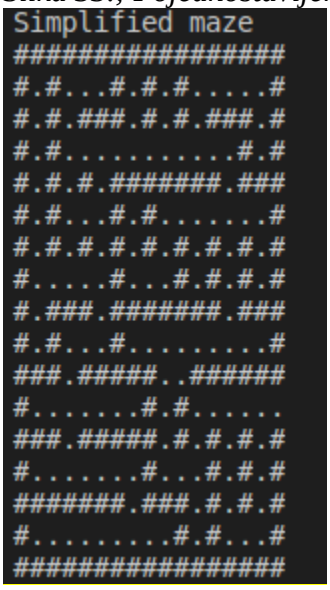
Kada smo našli sve vrijednosti pixela, za svaki red prvo provjeravamo ukoliko je slika nesavršena te možda ima bijele pixele na rubovima umjesto zidova te njih ignoriramo. Ukoliko ih ima više od 20 (što se prilikom testiranja pokazalo kao dobra procjena za grešku) onda znači da na tom mjestu ni ne treba biti zid te ga zabilježimo i nastavimo dalje. Svaki put kada dođemo do crnog piksela, provjerimo koliko smo bijelih prešli te dodamo odgovarajući broj prolaza. Nakon što prođemo zid odnosno bijele piksele, ili dođemo do kraja, dodajemo u novu listu '#' za zid.

Novi labirint se konstruira tako da uzmemo u obzir neke pretpostavke. Prva je ta da na svakom križanju zidova mora biti zid te da će svaki „veliki“ prolaz uvijek biti prolaz te ne može biti zid. Nakon što smo analizirali mjesta gdje se nalaze zidovi i prolazi u redovima i stupcima i uzevši u obzir gornje pretpostavke, konstruiramo labirint. Kako bi jedan labirint onda izgledao vidi se na slikama 32. i 33. gdje na slici 32. se nalazi „Crno-bijela“ slika labirinta a na slici 33. pojednostavljeni labirint sa slike 32.

Slika 32.; „Crno-bijela“ slika labirinta



Slika 33.; Pojednostavljeni labirint



Labirint se rješava jednostavnom implementacijom BFS (breadth first search) algoritma pomoću funkcije *simple_bfs()*. Nakon što se nađe put kroz labirint, on se transformira u jednostavne naredbe (ex. 'down' 3, 'right' 4 …) te potom u koordinate u koordinatnom sustavu transformirane slike uzimajući u obzir širinu zidova i prolaza u pikselima.

6. Izrada labirint

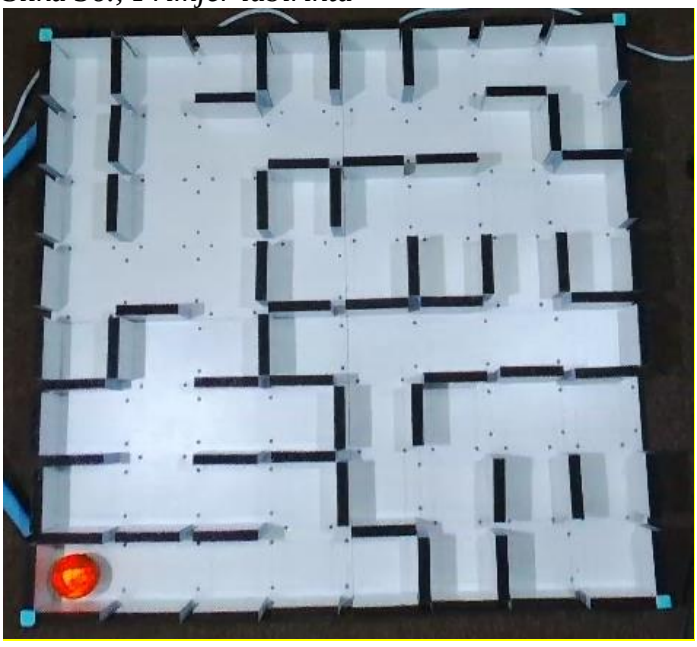
Dizajn labirinta je ključan za testiranje. Potrebno je napraviti modularni labirint kako bi eksperimentiranje bilo efikasnije. Odlučili smo se na dizajn u kojem su zidovi labirinta jednakih dimenzija tj. skup manjih pločica koje bi bile dimenzija 104mm×75mm×19mm te su od iverala. Ploče koje se koriste za dno labirinta su također od iverala te su jedne od 4 dimenzija:
500mm×500mm×19mm,
500mm×519mm×19mm,
519mm×500mm×19mm,
519mm×519mm×19mm.
Na slici 35. se nalazi originalni dizajn labirinta, a na slici 34. dizajn zidova labirinta. Obje slike su tehnički crteži generirani pomoću softvera Fusion 360, u kojem dijelovi bili dizajnirani.

Na slici 35. nalaze se četiri ploče sa puno rupa. U te rupe je namijenjeno postaviti zidove labirinta (na slici 34. se vidi kako zidovi imaju dvije kružne izbočine). Sa slike se također može primijetiti ponavljajuća šablona od 4 rupe. Te četiri rupe predstavljaju kut ili križanje labirinta. Kako bi labirint bio modularan svaki zid labirinta bi se moramo moći postaviti na bilo koje mjesta, stoga postoji razmak između zidova od 21mm nužni. S time se osiguralo da svaki zid ima dovoljno prostora te da se može postaviti bilo gdje drugdje na podložnim pločama.

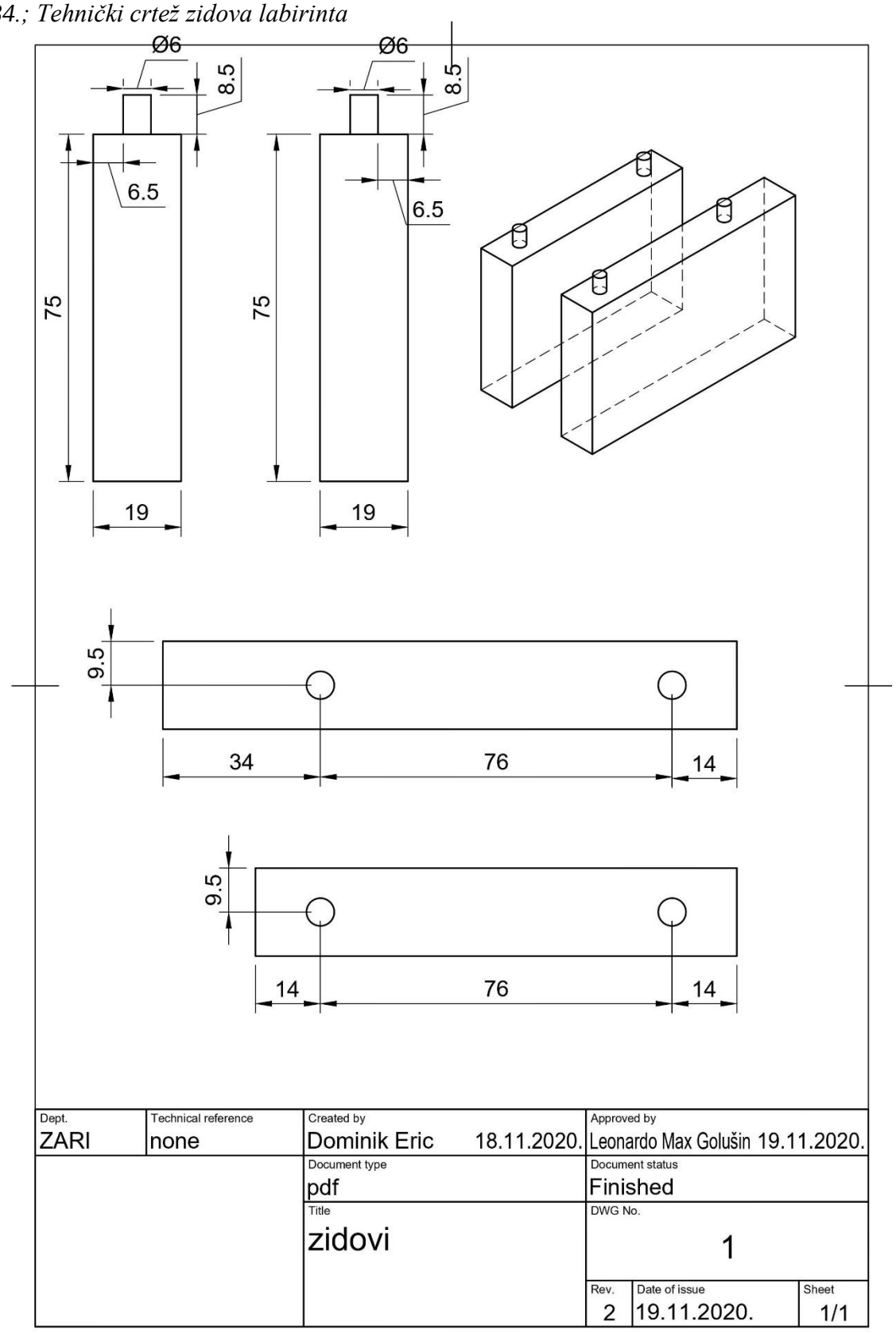
Na slici 34. se može primijetiti da su nacrtane dvije daščice, jedna dulja od druge. Radi se o rubnim daščicama koje su dulje za 20mm. Sveukupno je potrebno 4 dulje daščice. Dulje daščice su napravljene sa idejom da produžetak bude druge boje (u odnosu na boju zida labirinta) te predstavlja kut labirinta po kojem se Bebop orijentira.

Na slici 36. nalazi se primjer jednog labirinta kojeg smo koristili tokom testiranja.

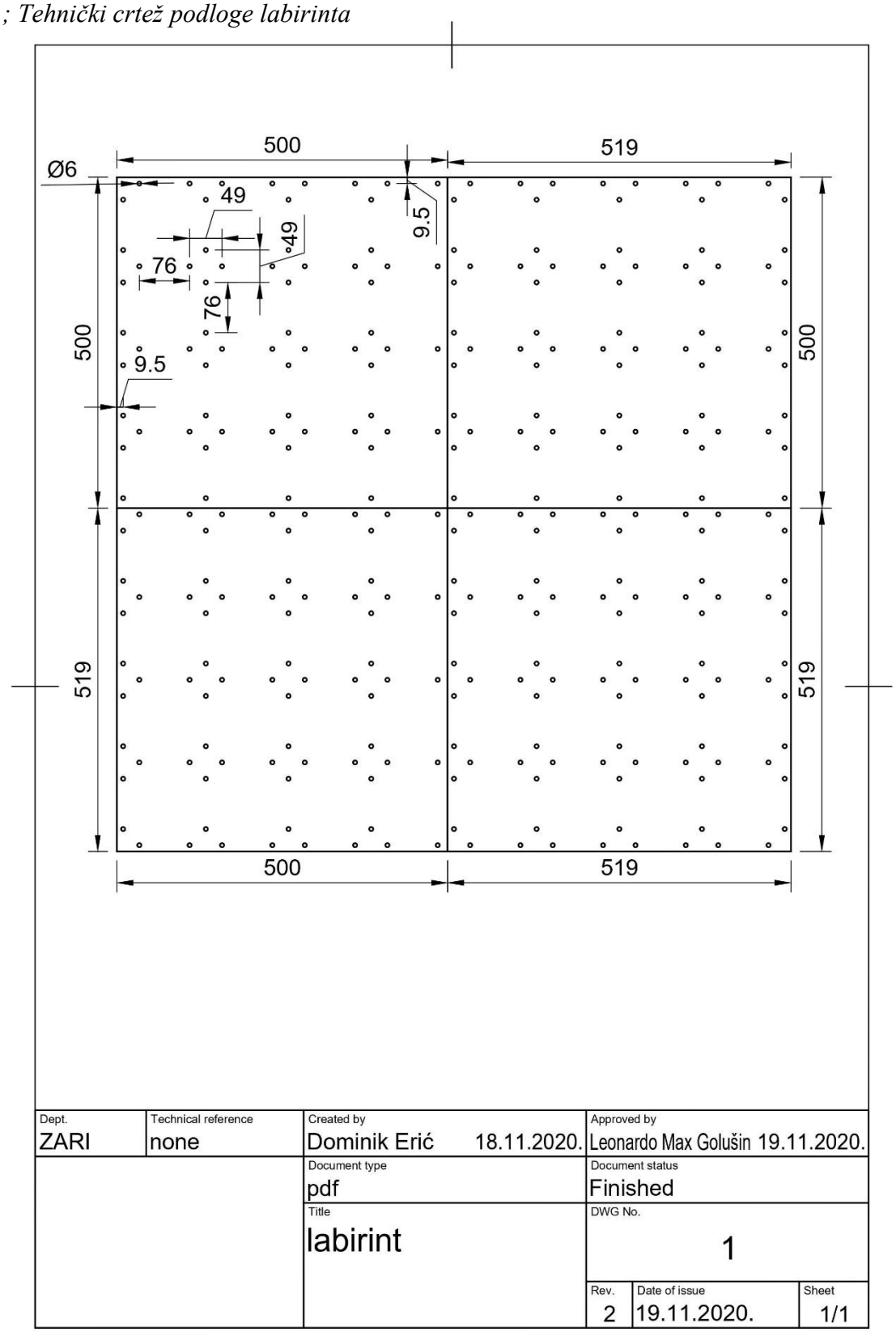
Slika 36.; Primjer labirinta



Slika 34.; Tehnički crtež zidova labirinta



Slika 35.; Tehnički crtež podloge labirinta



7. Literatura

1. Baotić, M., Mišković, N., Lešić, V. and Novoselnik, B., 2020. *Automatsko Upravljanje*.

2. Cormen, T., Leiserson, C., Rivest, R. and Stein, C., 2009. *Introduction To Algorithms.* 3rd ed. MIT Press, pp.594-601.

3. OpenCV. (2015). *Open Source Computer Vision Library*.

4. Stanford Artificial Intelligence Laboratory et al. (2018). *Robotic Operating System*. Retrieved from https://www.ros.org

5. MATLAB. (2010). *version 9.9 (R2020b)*. Natick, Massachusetts: The MathWorks Inc.

6. The MathWorks, I. (2019). S*ystem Identification Toolbox*. Natick, Massachusetts, United State. Retrieved from System Identification Toolbox - MATLAB (mathworks.com)

7. The MathWorks, I. (2019). *Control System Toolbox*. Natick, Massachusetts, United State. Retrieved from Control System Toolbox - MATLAB (mathworks.com)

8. The MathWorks, I. (2019). *Signal Processing Toolbox*. Natick, Massachusetts, United State. Retrieved from Signal Processing Toolbox - MATLAB (mathworks.com)

9. Golušin, L., M., *maze-project*, (2020)., Github repository, GitHub - zeroth-dev/maze-project: Making a maze solving algorithm for a system with drone and sphero robot