Kinematika četveronožnog hodača

Paleka, Ivan

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Ivan Paleka

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

Dr. sc. Mladen Crneković, dipl. ing. Ivan Paleka

Zagreb, 2024.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se svojim roditeljima što su mi omogućili studiranje.

Ivan Paleka

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 24 – 06 / 01**	
Ur.broj: **15 – 24 –**	

ZAVRŠNI ZADATAK

Student: **Ivan Paleka** JMBAG: **0035230730**

Naslov rada na hrvatskom jeziku: **Kinematika četveronožnog hodača**

Naslov rada na engleskom jeziku: **Kinematics of a quadrupedal walker**

Opis zadatka:

Za kretanje po neuređenim čvrstim okolinama hodanje pomoću nogu nema alternativu. I dok dvonožni hod zahtijeva rješenje složenog problema stabilnosti, četveronožni hod je puno jednostavniji, a zahtijeva rješenje koordinacije ekstremiteta, tj. rješiv je kinematikom.

Za četveronožnog hodača potrebno je definirati kinematiku gibanja, te izraditi i testirati model.

U radu je potrebno:

- projektirati četveronožnog hodača za hod u ravnini
- definirati kinematiku hoda svake noge tako da se osigura stabilnost i funkcija
- izraditi i testirati model

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
30. 11. 2023.	**1. rok:** 22. i 23. 2. 2024. **2. rok (izvanredni):** 11. 7. 2024. **3. rok:** 19. i 20. 9. 2024.	**1. rok:** 26. 2. – 1. 3. 2024. **2. rok (izvanredni):** 15. 7. 2024. **3. rok:** 23. 9. – 27. 9. 2024.

Zadatak zadao:
Prof. dr. sc. Mladen Crneković

Predsjednik Povjerenstva:
Prof. dr. sc. Damir Godec

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
$M_{\max\_motora}$	Nm	Maksimalni moment servo motora
m_{max}	kg	Maksimalna masa robota
g	m/s^2	Sila teže
L	mm	Duljina članka noge
L_1	mm	Udaljenost težišta robota od kuka
X_1	mm	Koordinatna udaljenost u smjeru osi X
Y_1	mm	Koordinatna udaljenost u smjeru osi Y
R	mm	radijus
X_2	mm	Koordinatna udaljenost u smjeru osi X
Y_2	mm	Koordinatna udaljenost u smjeru osi Y
α_{yaw}	°	Kut između radijusa *R* i osi X
β_{yaw}	°	Kut između radijusa *R* i osi X
yaw	°	Željeni zakret oko osi Z
i_{yaw}	mm	Pomak koji proizlazi iz rotacije oko Z osi
j_{yaw}	mm	Pomak koji proizlazi iz rotacije oko Z osi
h_p	mm	Visina težišta
h_r	mm	Visina prednje ili stražnje strane robota
j_{pitch}	mm	Pomak koji je proiziša iz rotacije oko X osi
$pitch$	°	Željeni zakret oko osi X
$roll$	°	Željeni zakret oko osi Y
h_2	mm	Visina lijeve ili desne strane robota
i_{roll}	mm	Pomak koji proizlazi iz rotacije oko Y osi
i	mm	Željeni pomak po osi X
I	mm	Ukupni pomak po osi X
A	mm	Kateta trokuta
C	mm	Hipotenuza trokuta
γ_{xz}	°	Kut u XZ ravnini
δ_{xz}	°	Kut u XZ ravnini
φ_{kuk}	°	Kut koji se šalje na servo motor
J	mm	Ukupni pomak po osi Y
j	mm	Željeni pomak po osi Y
R_1	mm	Hipotenuza trokuta
α_{yz}	°	Kut u YZ ravnini
β_{yz}	°	Kut u YZ ravnini
γ_{yz}	°	Kut u YZ ravnini
φ_{rame}	°	Kut koji se šalje na servo motor
φ_{lakat}	°	Kut koji se šalje na servo motor

$\boldsymbol{S}$	°	vektor stanja
$\boldsymbol{F}$	/	matrica prijelaza stanja
$\boldsymbol{G}$	/	kontrolna matrica
$\boldsymbol{U}$	°	matrica ulaza
$\boldsymbol{P}$	°	vektor nesigurnosti predviđanja
Q	°	nesigurnost procesa
$\boldsymbol{L}$	°	među matrica
R	°	nesigurnost mjerenja
$\boldsymbol{H}$	/	matrica
K	/	Kalmanovo pojačanje
$\boldsymbol{M}$	°	vektor mjerenja
T_{kr}	s	Kritični period
K_{Pkr}	/	Kritično pojačanje P djelovanja regulatora
K_P	/	Pojačanje djelovanja regulatora
T_d	s	Vremenska konstanta derivacijskog člana regulatora
K_D	s	Dreivacijsko pojačanje djelovanja regulatora

SAŽETAK

Ovaj završni rad prikazuje izradu moblinog robota četveronožne kinematike te kinematski model potreban za pokretanje samog robota. Prvobitno je prikazan odabir komponenti i postupak izrade konstrukcije robota. Zatim se u radu prolazi kroz kinematički model potrebam za definiranje kinematike hoda svake noge kako bi se osigurala stabilnost samog robota. Na kraju je pokazana izrada PD regulatora pomoću Ziegler-Nichols metode kako bi se omogućio stabilan i dinamički hod.
Tokom izrade rada korišteni su programski paketi SolidWorks, MATLAB i Arduino IDE.

Ključne riječi: kinematika, mobilni robot, hod, PD regulator

SUMMARY

This final thesis presents the development of a mobile robot with quadruped kinematics and the kinematic model necessary for the robot's movement. Initially, the selection of components and the process of constructing the robot's structure are shown. Subsequently, the thesis covers the kinematic model required to define the gait kinematics of each leg to ensure the stability of the robot itself. Finally, the implementation of a PD controller using the Ziegler-Nichols method is demonstrated to enable stable and dynamic locomotion. Throughout the thesis, the software packages SolidWorks, MATLAB and Arduino IDE were used.

Keywords: kinematics, mobile robot, gait, PD controller

1. UVOD

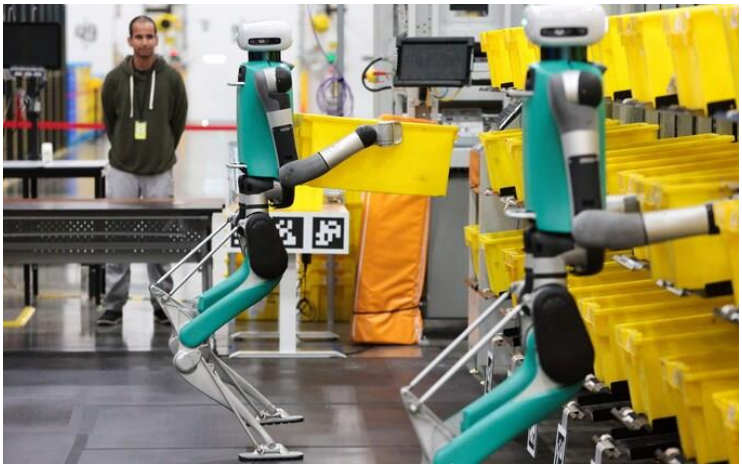
Mobilni roboti su strojevi koji nisu ograničeni svojim ishodišnim kordinatnim sustavom na samo jednu poziciju u prostoru već imaju slobodu kretanje u radnom prostoru ili bilo kojoj korisnički željenoj okolikni. Tu kretnju ostvaruju kotačima, hodom ili čak elisama. U ovome radu bazirati ćemo se na izradi i projektiranju hodajućeg robota jer njihove prednosti u odnosu na one s kotačima su veoma vidljive kada takve robote stavimo u vanjski svijet koji je nepredvidiv i pun prepreka koje robot mora rješiti.

1.1. Primjeri mobilnih robota

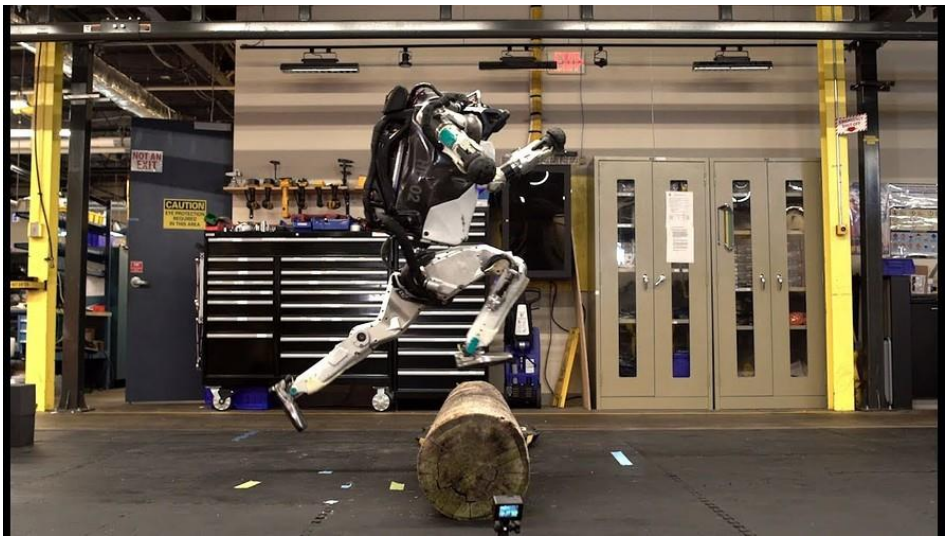
Primjere mobilnih robota danas možemo naći gdje god da se okrenemo. Tako postoje mobilni roboti za čišćenje kućanskog poda poput Roombe [Slika 1] ili roboti za dostavu [Slika 2] koje koristi tvrtka Amazon Prime. Svoju svrhu mobilni roboti su pronašli i u skladišnoj industriji, a primjer toga je ponovno tvrtka Amazon koja testira mobilne robote naziva Digit [Slika 3] kako bi poboljšali produktivnost skladišta i dostave. Robot Digit je jedan od primjera humanoidnih robota koji se trenutno razvijaju. Još jedan primjer takvih robota je i robot Atlas [Slika 4] iz tvrtke Boston Dynamics. Hodajući humanoidni roboti po naravi su nestabilni sustavi te ih nije lako projektirati kako bi zadržali balans. Zbog tog razloga se u mobilnoj robotici razvijaju stabilniji roboti koji hodaju pomoću tri, četri ili više nogu.

Slika 1. Usisavač Roomba

Slika 2. Dostavni robot tvrtke Amazon Prime



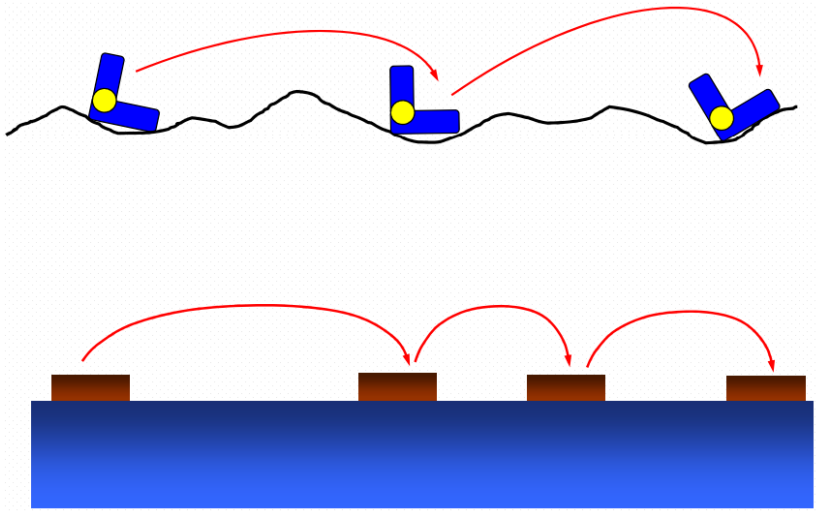
Slika 3. Robot Digit



Slika 4. Robot Atlas

1.2. Prednosti hoda u odnosu na kotrljanje

Prilikom kretanja mobilnog robota po neravnom terenu dolazi na vidjelo da mobilni roboti pokretani kotačim moraju dodirivati površinu kako bi prenjeli rotacijsko gibanje i omogućili si kretnju. S time dolazi do toga da se sve neravninie i udarci koji dolaze zbog njih prenose na robota tokom vožnje te on zbog tih neravnina gubi svoju efikasnost. Za razliku od toga roboti koji mogu hodati nisu primorani konstantno dodirivati tlo već njihova struktura i način pokretanja im omogućuje da preskaču nepoželjna udubljenja i neravnine na svome putu te time zadržavaju svoju efikasnot.



Slika 5. Prednost hodanja

Slika 6. Mobilni robot Spot

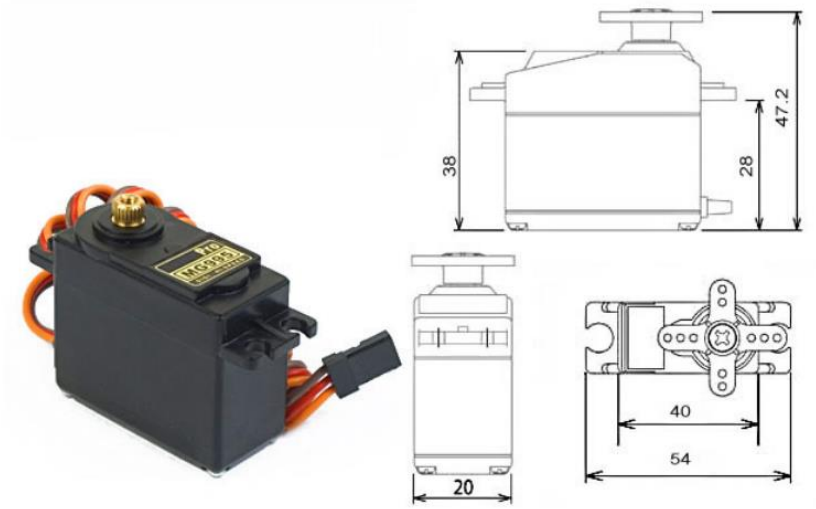
2. Projektiranje i izrada robota

2.1. Odabir komponenti

Kako cilj ovog završnog nije bio izrada mobilnog robota vrhunskih perfomansi, već model kojim će se pokazati i implementirati kinematika četveronožnog tipa hodanja, time je i bio usmjerne odabir komponenti. To će se kasnije pokazati na nešto lošijoj dinamici samog robota tokom hoda.

2.1.1. MG995

MG995 je tip servo motora koji omogućuje korisniku pozicioniranje vratila za otklon od 0° do 180° te se upravlja PWM signalom. Motor ne šalje nazad nikakve povratne informacije jer u sebi sadrži potenciometar i mali sklop koji PWM signal pretvara u željenu poziciju te ga zapravo gledamo kao crnu kutiju prilikom izrade algoritma.



Slika 7. Servo motor MG995

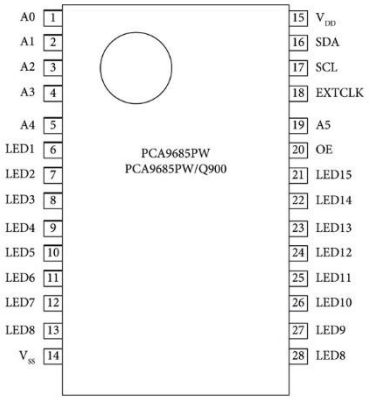
Tablica 1. Specifikacije MG995

Težina	55g
Dimenzije	40.7 x 19.7 x 42.9mm
Moment potezanja	8.5 kg·cm (4.8 V), 11 kg·cm (6 V)
Radna brzina	0.2 s/60º (4.8 V), 0.16 s/60º (6 V)
Radni napon	4.8 V a 7.2 V
Maksimalna struja	1.2A
Mrtvo vrijeme	5 µs
Stabilan dizajn dvosturkog kugličnog ležaja otporan na udarce	/

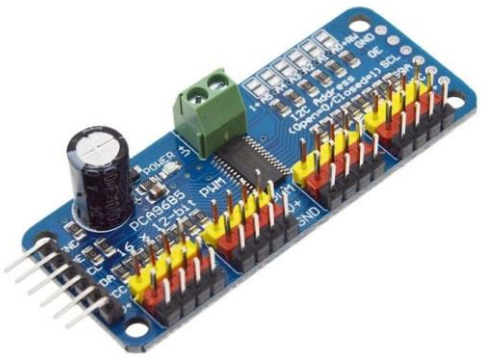
Raspon radne temperature	0 ºC – 55 ºC

2.1.2. PCA9685

PCA9685 [Slika 8] je čip koji služi primarno za upravljanje led trakama putem PWM signala, ali za ovaj projekt će poslužiti kao među upravljački uređaj između glavnog uređaja (mikrokontrolera ESP32) i motora jer nam daje više izlaznih linija sa PWM signalom nego što to imamo na glavnom upravljačkom mikroprocesoru. Sa tih izlaza ćemo upravljati servo motorima tako što ćemo na svaki izlaz spojiti samo jedan motor i time olakšati konekciju servo motora sa glavnim uređajem. Ovaj čip komunicira sa glavnim mikrokontorlerom putem I2C komunikacije što nam olakšava njegovo spajanje.



Slika 8. PCA9685 čip



Slika 9. PCA9685 breakout pločica

2.1.3. Napajanje

Kako bi smo mogli imati moblinog robota koji nije vezan nekim strujnim kabelom na vanjsko napajanje potrebno je odabrati prikladne baterije koje će napajati ovog mobilnog robota. U ovom slučaju su odabrane LiPo baterije koje su karakterizirane s mogućnošću visokog izboja struje iako su same po sebi manjih kapaciteta i mase što ih čini idealnim za ovaj projekt. S time na umu za ovaj projekt su odabrane slijedeće baterije tipa 3s1p

(nominalni napon 11.1V i 2.2Ah) [Slika 10] za napajanje motora i 2s1p (nominalni napon 7.4V i 1Ah) [Slika 11] za napjanje elektronike. Razlog za dvije baterije je taj da osiguramo odvojenost strujnih krugova kako nebi došlo do oštećenja osjetljivih komponenti.

Slika 10. 3S1P 2.2Ah LiPo baterija

Slika 11. 2S1P 1.0Ah LiPo baterija

2.1.4. Step-down modul s LM2596S

Kako bi smo mogli napjati uređaje moramo spustiti napon sa baterija jer ove baterije imaju značajno veće napone u odnosu na one koje naši uređaji mogu podnijeti stoga koristimo ovaj modul koji ulazni napon snižava na točno određeni izlazni napon kojega podešavamo putem trimera na pločici. Jedino ograničenje na koje sam naišao prilikom korištenja ovoga modula je taj da može maskimalno davati 3A zbog čega se moraja koristiti dva ovakva modula kako bi se napajali servo motori.

Slika 12. Step-down modul s LM2596S

Tablica 2. Step-down modul s LM2596S specifikacije

Ulazni napon	3V – 40V
Izlazni napon	1.5V – 35V
Maksimalna struja na izlazu	3A

2.1.5. HC-SR04 senzor udaljenosti

Ovaj jednostavn senzor udaljenosti je dodan kako bi omogućio robotu primitivno preopznavnje preprka koje se mogu naići na putu. HC-SR04 je ultrazvučni senzor udaljenosti koji omogućuje mjerenje udaljenosti od 0,02m do 4 m te mu je precizost 1cm.

Tablica 3. HC-SR04 specifikacije

Domet	2cm – 4000cm
Preciznost	1cm
Kut mjerenja	15°
Napon	5V
Struja	<2mA

Slika 13. HC-SR04 ultrazvučni senzor udaljenosti

2.1.6. MPU6050

Kako bi omogućili korištenje regulatora da stabiliziramo robota moramo imati nekavu povratnu vezu, a to ćemo u ovom projektu izvesti pomoću čipa MPU6050 koji je zapravo troosni žiroskopi i akcelerometra. Ovaj mali uređaj će nam dati sve potrebne informacije o nagibu robota koje kasnije možemo prosljediti u regulator kako bismo osigurali stabilan hod. Problem koji dolazi s ovom jeftinom komponentom je ogroman šum kojega treba kompenzirati. To je ovdje izvedeno primjenom Kalmanovog filtriranja.

Slika 14. MPU6050

2.1.7. ATmega328P

ATmega328P je jednočipni 8-bitni mikrokontorler sa 16MHz radnom brzinom te 32 KB flash memorije. U ovom radu su korištena dva ovakva proceora jedan za procesiranje signala sa senzora udaljenosti, a drugi kao glavna upravljača jedinica daljinskog upravljača putem radio veze.

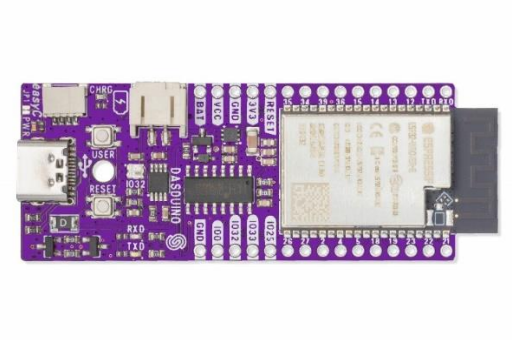
Slika 15. Atmega328P na Dasduino Core pločici

Tablica 4. Atmega328P specifikacije

Vrsta proscesora	8-bitni AVR
Maksimalna brzina procesora	16MHz
Brza memorija	32KB
SPRAM	2KB
EEPROM	1KB
Radni napon	5V
I/O pinovi	22
Komunikacija	UART,SPI,I2C

2.1.8. ESP32-WROVER-E modul

ESP32-WROVER-E je 32-bitni mikročip sa 240MHz maksimalne brzine procesuiranja podataka. Ovaj čip omogućuje Wi-Fi protokole i Bluetooth komunikaciju sa vanjskim uređajima, ali za ovaj projekt to nije korišteno. U ovom slučaju je ovaj mikročip korišten kao glavna upravljačka jedinica mobilnog robota na kojem se vrše sve kalkulacije potrebne za izvođenje kinematike, inverzne kinematike, regulacije te za komunikaciju sa radio daljinskim.



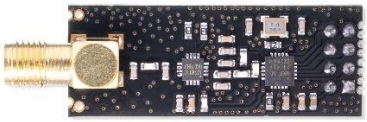
Slika 16. ESP32 mikročip na Dasduino Connectplus pločici

Tablica 5. ESP32-WROVER-E specifikacije

Radni napon	3,3V
Minimalna potreban struja	0.5A
Broj pinova	30
Komunikacija	UART,SPI,I2C
Flash memorija	4MB
PSRAM	8MB

2.1.9. NRF24L01+

NRF24L01+ je radio modul koji omogućuje bežičnu komunikaciju između dva mikrokontrolera te radi na frekveniciji od 2.4GHz što omogućuje velike brzine prijenosa podataka koji se šalju u obliku paketa. Doseg ovoga modula je moguć i preko 800m na otvorenome jer dolazi u sklopu sa pojčalom te antenom. Sama komunikacija između mikroprocesora i NRF24L01 čipa se vrši putem SPI komunikacije.



Slika 17. NRF24L01+

Tablica 6. NRF24L01+ specifikacije

Radni napon	3,3V - 5V
Struja	150mA
Frekvencija	2,4GHz

2.1.10. Ostale komponente

Od ostalih komponenata uzeti su prekidači, konektori, kablovi, izolacijski materijali te potrebni pinovi i headeri kako bi se mogle napraviti odgovarajući spojevi među komponentama.

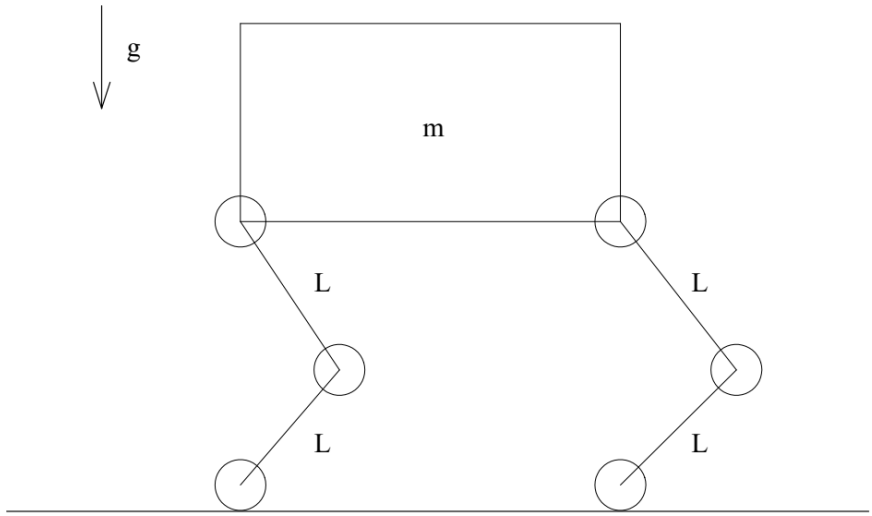
2.2. Izrada modela

Izrada modela sastojala se od dva dijela. Dizajniranja potrebnih djelova unutar softverskog paketa SOLIDWORKS te njihove izrade aditivniom tehnologijom 3D printanja. Primjenom ovog programa ušlo se u izradu i optimalno pozicioniranje djelova kako bi se poboljšala dinamika robota i njegova sveobuhvatna stabilnost te kako bi bio što kompaktniji i lakši radi jeftinih servo motora. Drugi dio izrade se sastojao od izrade radio daljinskog i potrebnih pločica za lakšu konekciju komponenti.

2.2.1. Proces izrade djelova u programu SOLIDWORKS i 3D printanje istih

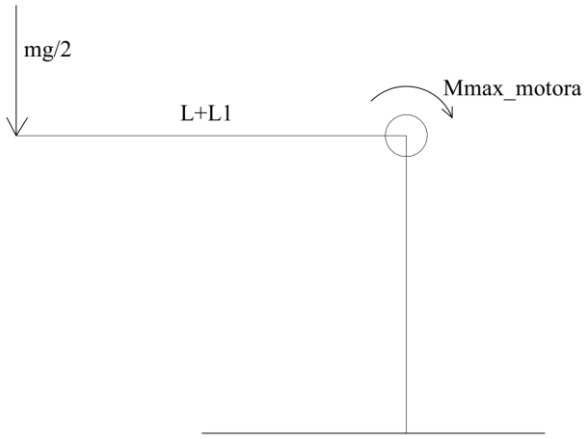
2.2.1.1. Proračun maksimalne mase robota

Kako bi smo konstruirali robota moramo na temelju momenta što odabrani motori daju prvobitno izračunati maksimalnu masu koju oni mogu pokretati. To ćemo napraviti na sljedeći nači, a to je da ćemo pretpostaviti da su u svakom trenutku barem dvije noge u dodiru sa tlom. Uz tu pretpostavku proračun ćemo raditi i pomoću pojednostavljenog modela za proračun [Slika 18].



Slika 18. Pojednostavljeni model za proračun mase

Kritični slučaj koji ćemo razmatrati, a do kojega u ovom dizajnu dolazi samo na stražnjim nogama, je taj da se cijela polovica mase koju treba nositi servo motor u laktu nađe točno kako je prikazano na sljedećoj slici.



Slika 19. Kritični slučaj za određivanje maksimalne mase

Iz ovako postavljenog problema proizlaze sljedeća rješenja:

$$M_{\max\_motora} = \frac{m_{max} \cdot g}{2} \cdot (L + L1)\ , \tag{2.1}$$

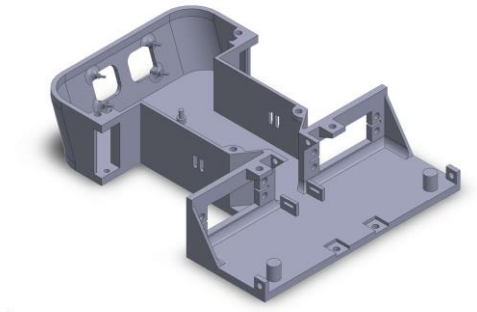
$$m_{max} = M_{\max\_motora} \cdot \frac{2}{g \cdot (L+L1)} = 1.079Nm \cdot \frac{2}{9.81\frac{m}{s^2} \cdot (0.1+0.1131)m}\ , \tag{2.2}$$

$$m_{max} = 1.032kg. \tag{2.3}$$

Sada kada imamo maksimalnu masu robota možemo krenuti s izradom stvarnog robota.

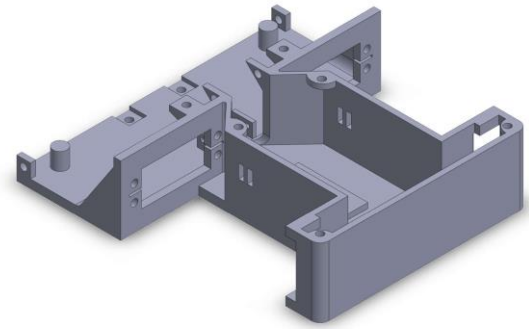
2.2.1.2. Proces modeliranje i izrade

Prva faza izrade je krenula sa izradom trupa robota kako bi se u njega stavila sva elektronika i dio motora koji upravljaju zakretima kukova. Sam trup je izrađen od dva dijela jer cijeli trup nebi stao na postolje 3D printera te je zbog toga bilo potrebno smisliti konstrukcijska rješenja kako bi se ta dva dijela spojila. Rješenje tog problema izvedeno je preko bočnih pokrovnih poloha, središnje pokrovne ploče, malih unutarnjih greda koje dodatno osiguravaju konstrukciju te samog dijela koji pridržava PCA9685 pločicu.

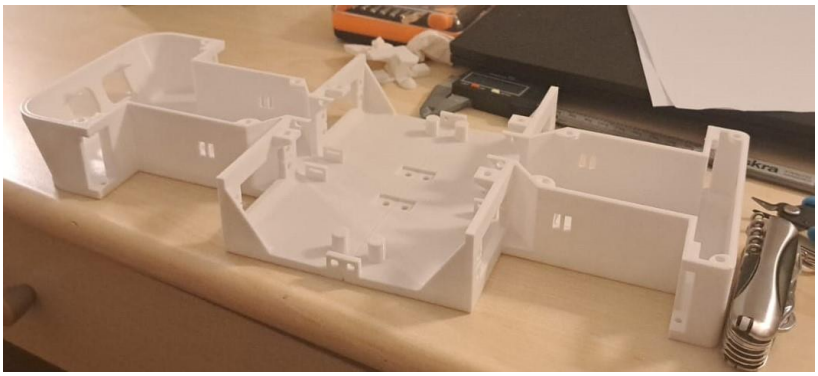


Slika 20. Prednji dio trupa 3D model

Slika 21. Prednji dio trupa proces izrade



Slika 22. Stražnji dio trupa 3D model

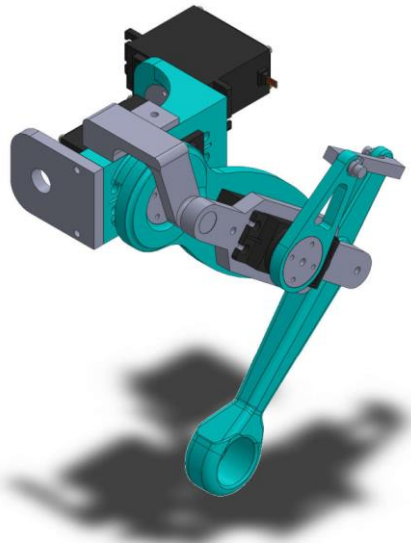


Slika 23. Trup robota

Treba napomenuti da su ova dva dijela trupa ujedno najteža dva izrađena dijela, ali njihovu masu smo mogli vrlo lako mijenjati bez promjene CAD modela promjenom postavki ispune u programu za 3D printanje.

Sljedeća faza izrade bazirala se na dizajniranju samih nogu kako bi se također ostvarila što veća kompaktnost i dinamika nogu, ali u ovome trenutačnom dizajnu išlo se

nauštrb dinamike te je radi veće kompaktnosti motor lakta postavljen nešto niže, bliže samome laktu robota [Slika 24].

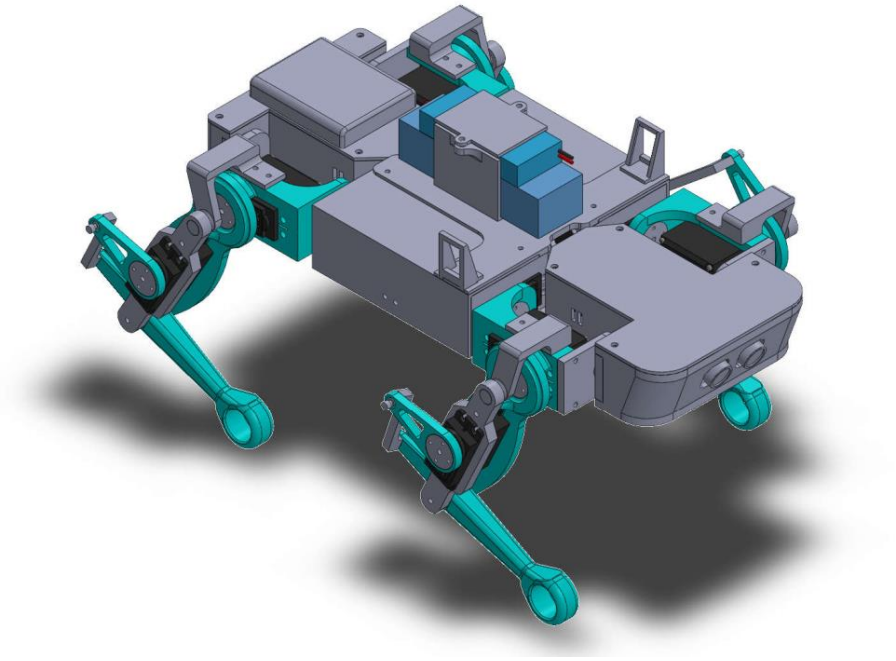


Slika 24. 3D model sklopa noge

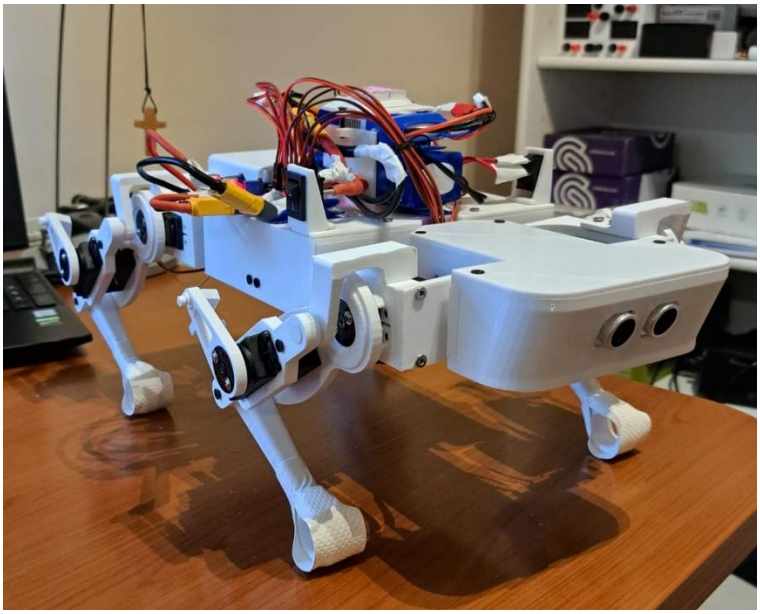
Kako je za potrebe izrade korištena aditivna tehnologija 3D-e prinatanja došlo je do potrebe za učestalim testiranjima tolerancija i iteracijama pojedinih djelova jer sam printer nije savršene preciznosti. Na slici koja sljedi prikazan je sklopljen prototip noge od kojeg se odustalo zbog prevelike pojave trenja usljed dodira plastičnih djelova. Naime da bi imali što veću i bolju dinamiku moramo smanjiti trenje koje se javlja u sustavu na što je to manje moguće.

Slika 25. Prototip noge s prevelikim trenjem u području ramena

Zadnja faza modeliranja sastojala se od izrade komponetni za prihvat energetskog sustava, pokrovnih elemenata i samog skapljanja robota kako u programskom paketu [Slika 26] tako i u stvarnosti [Slika 27].



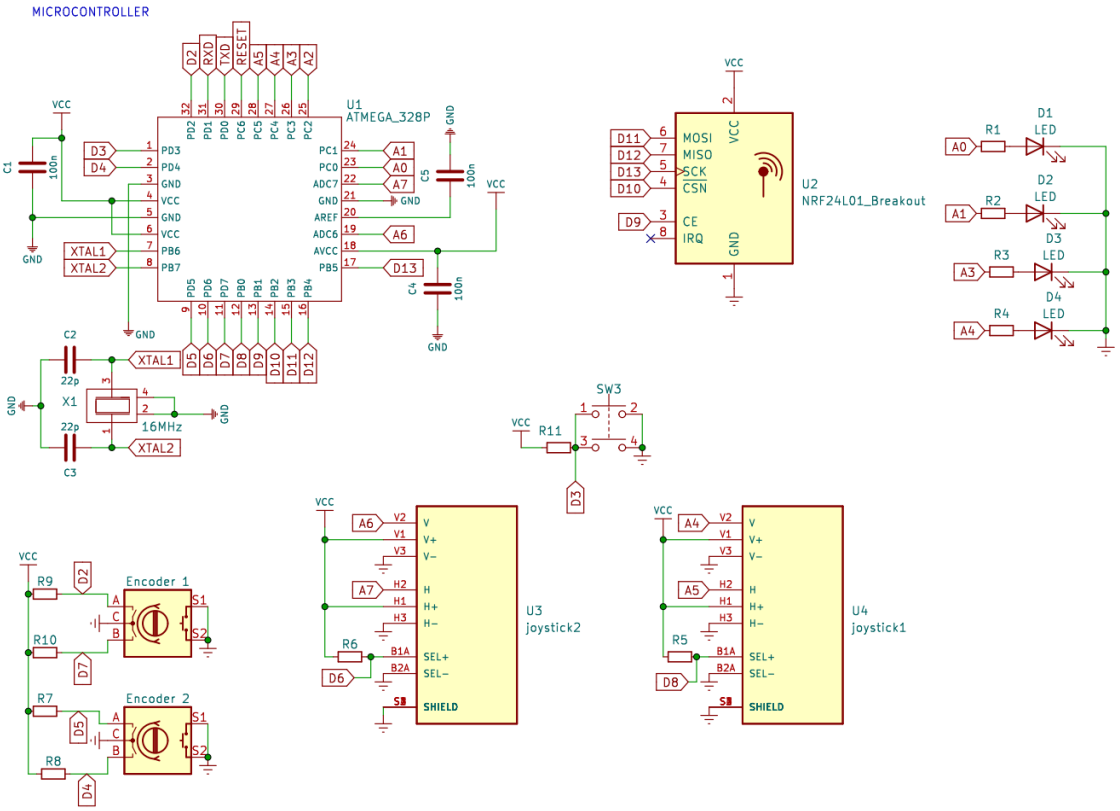
Slika 26. Sklopljeni robot u CAD programu



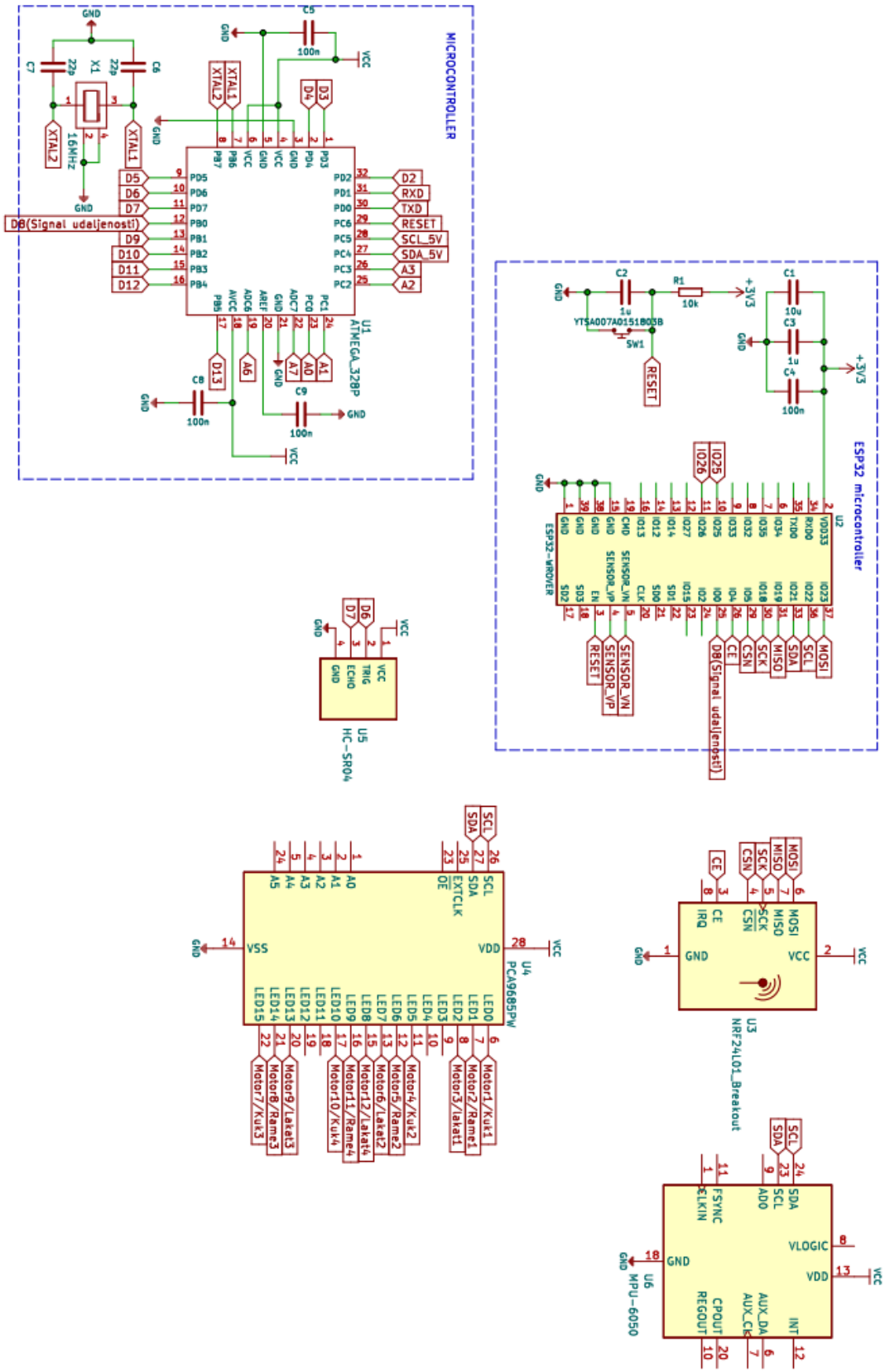
Slika 27. Sklopljeni robot

2.2.2. Izrada pločica i spajanje električnih komponenti

Ova faza se sastojala od izrade pojednostavljene električne sheme radi lakšeg spajanja pločica, izrade samih pločica za lakše spajanje te izrade radio daljinskog. Sheme spajanja su prikazane na sljedećim slikama.



Slika 28. Pojednostavljeni shematski prikaz radio daljinskog



Slika 29. Pojednostavljeni shematski prikaz konekcija na robotu

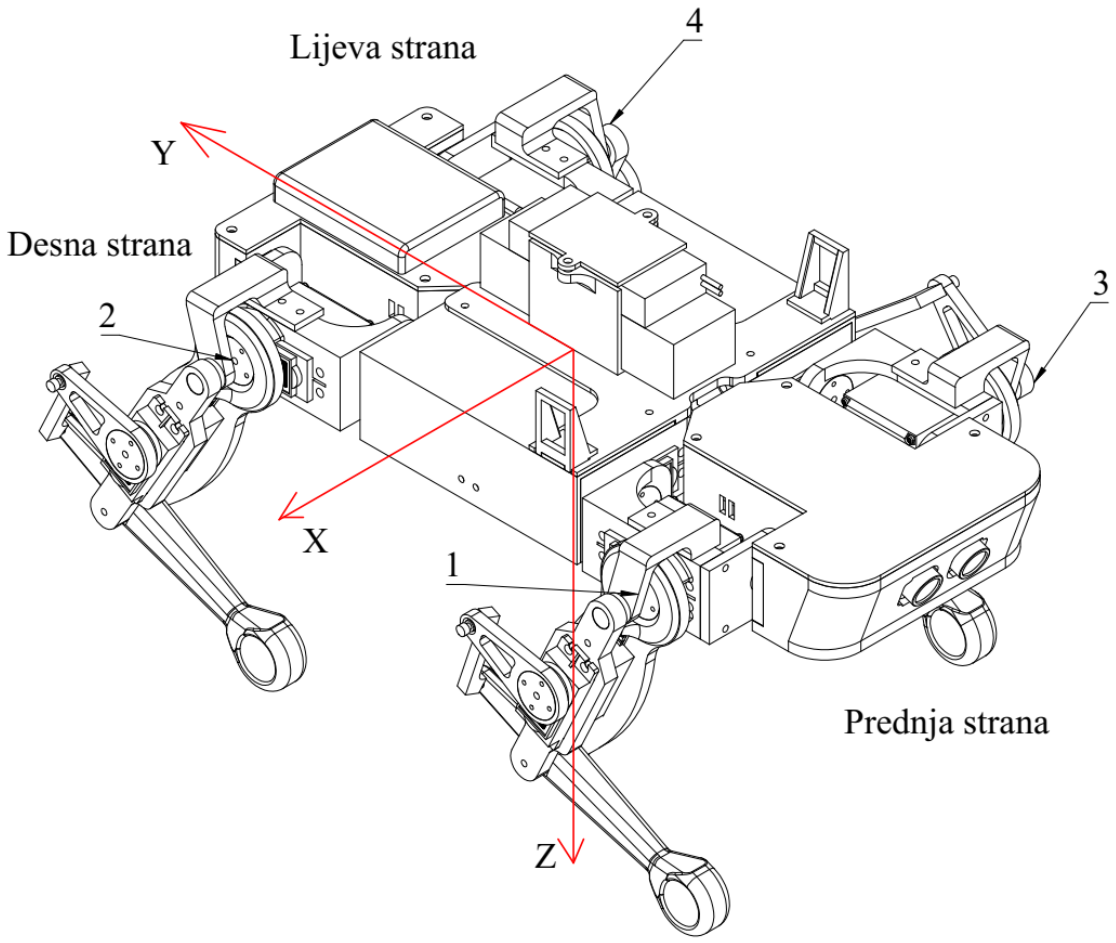
3. Inverzni kinematički model

Inverzni kinematički model je matematički model koji nam pomaže u upravljanju bilo kojim robotom bilo da je on industrijski ili mobilni tip robota. Inverzna kinematika se temelji na pretvaranju vanjskih koordinata u unutranje koordinate, to jest, preko nje računamo zakrete ili pomake naših aktuatora koji svojim pomacima dovode robota u željenu poziciju. U ovom slučaju za ovaj tip četveronožnog robota se to svodi na kutove zakreta svakog pojedinačnog servo motora.

Svako slobodno tijelo u prostoru ima 6 stupnjeva slobode gibanja kojima se može gibati pa tako isto i centar mase ovog mobilnog robota. Kako bi si olakšao izračun inverzne kinematike robota sa 6 stupnjeva slobode gibanja, matematika inverzne kinematike je svedena na plohe omeđene Kartezijevim kooridinatnim sustavom te se time 3D problem sveo na više jednostavnijih 2D problema koji se rješavaju putem trigonometrije.

3.1. Definiranje koordinatnog sustava, strana i numeriranje nogu robota

Kako bi se što lakše snalazili u provedbi izračuna inverzne kinematike u ovom je potpoglavlju definiran položaj ishodišnog Kartezijevog koordinatnog sustava, strane robota te je još napravljena numeracija nogu radi kasnije lakšeg programiranja.

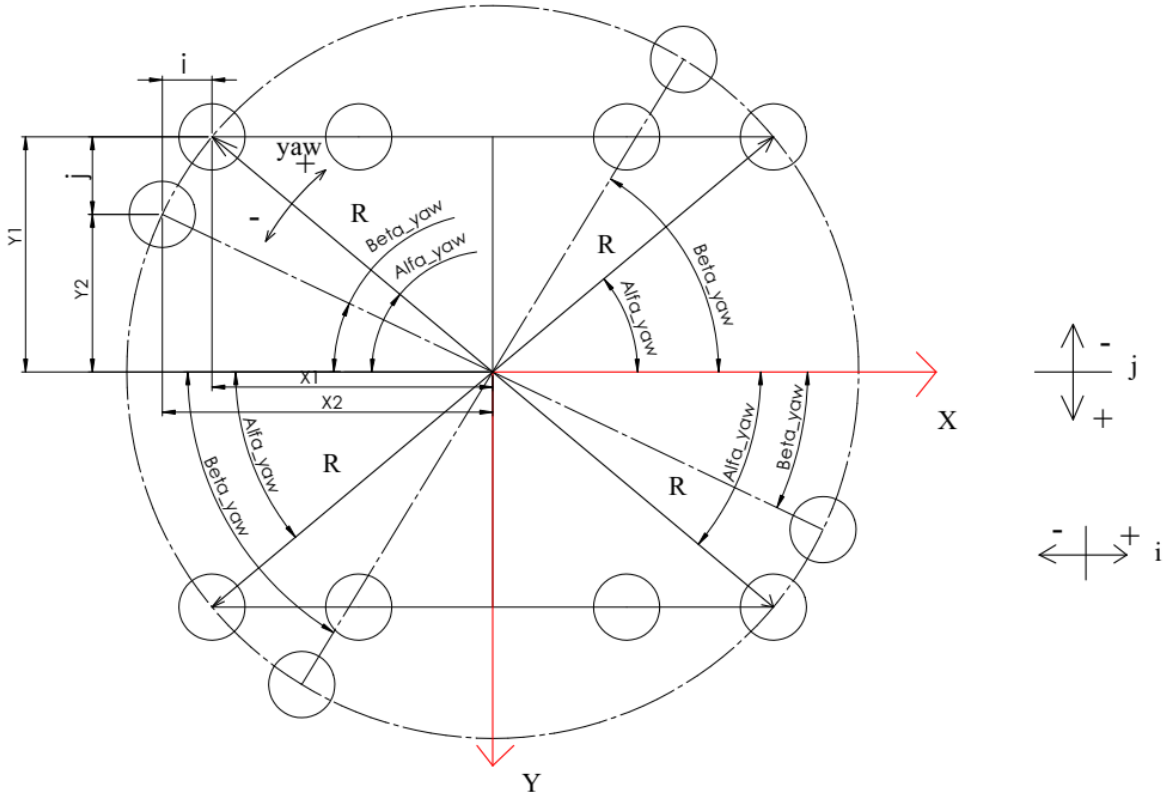


Slika 30. Definiranje koordinatnog sustava, strana i numeriranje nogu robota

3.2. Rotacija oko Z osi

Treba imati na umu da su sljedeći izvodi jednadžbi i formula vezani uz predznake jediničnih pomaka te da će se glavna dva pomaka *i* i *j* prikazati samo na ovoj slici te se neće ponovo prikazivati na idućim slikama. Što se tiče zakreta oko osi njihovi predznaci su definirani proizvoljno te će biti ucrtani radi lakšeg razumijevanja jednadžbi.

Prilikom definiranja sljedećih jednadžbi za rotaciju oko osi Z uzeto je da je kut *yaw* negativnog iznosa.



Slika 31. Rotacija oko Z osi

Kako bi još pojednostavnili i olakšali kasnije programiranje ovaj problem se razmatra za svaku nogu pojedinačno te je i tako raspisan.

Noga 1:

$$X_1 = 102mm, \quad (3.1)$$
$$Y_1 = -113{,}1mm, \quad (3.2)$$
$$R = \sqrt{X_1^2 + Y_1^2}, \quad (3.3)$$
$$\alpha_{yaw} = tan^{-1}(-\frac{Y_1}{X_1}), \quad (3.4)$$
$$\beta_{yaw} = \alpha_{yaw} - yaw, \quad (3.5)$$
$$X_2 = R \cdot \cos(\beta_{yaw}), \quad (3.6)$$
$$Y_2 = -R \cdot \sin(\beta_{yaw}), \quad (3.7)$$
$$j_{yaw} = Y_2 - Y_1, \quad (3.8)$$
$$i_{yaw} = X_2 - X_1. \quad (3.9)$$

Noga 2:

$$X_1 = 102mm, \quad (3.10)$$
$$Y_1 = 113{,}1mm, \quad (3.11)$$
$$R = \sqrt{X_1^2 + Y_1^2}, \quad (3.12)$$
$$\alpha_{yaw} = tan^{-1}(\tfrac{Y_1}{X_1}), \quad (3.13)$$
$$\beta_{yaw} = \alpha_{yaw} + yaw, \quad (3.14)$$
$$X_2 = R \cdot \cos(\beta_{yaw}), \quad (3.15)$$
$$Y_2 = R \cdot \sin(\beta_{yaw}), \quad (3.16)$$
$$j_{yaw} = Y_2 - Y_1, \quad (3.17)$$
$$i_{yaw} = X_2 - X_1. \quad (3.18)$$

Noga 3:

$$X_1 = -102mm, \quad (3.19)$$
$$Y_1 = -113{,}1mm, \quad (3.20)$$
$$R = \sqrt{X_1^2 + Y_1^2}, \quad (3.21)$$
$$\alpha_{yaw} = tan^{-1}(\tfrac{Y_1}{X_1}), \quad (3.22)$$
$$\beta_{yaw} = \alpha_{yaw} + yaw, \quad (3.23)$$
$$X_2 = -R \cdot \cos(\beta_{yaw}), \quad (3.24)$$
$$Y_2 = -R \cdot \sin(\beta_{yaw}), \quad (3.25)$$
$$j_{yaw} = Y_2 - Y_1, \quad (3.26)$$
$$i_{yaw} = X_2 - X_1. \quad (3.27)$$

Noga 4:

$$X_1 = -102mm, \quad (3.28)$$
$$Y_1 = 113{,}1mm, \quad (3.29)$$
$$R = \sqrt{X_1^2 + Y_1^2}, \quad (3.30)$$
$$\alpha_{yaw} = tan^{-1}(-\tfrac{Y_1}{X_1}), \quad (3.31)$$
$$\beta_{yaw} = \alpha_{yaw} - yaw, \quad (3.32)$$
$$X_2 = -R \cdot \cos(\beta_{yaw}), \quad (3.33)$$
$$Y_2 = R \cdot \sin(\beta_{yaw}), \quad (3.34)$$
$$j_{yaw} = Y_2 - Y_1, \quad (3.35)$$
$$i_{yaw} = X_2 - X_1. \quad (3.36)$$

3.3. Rotacija oko X osi

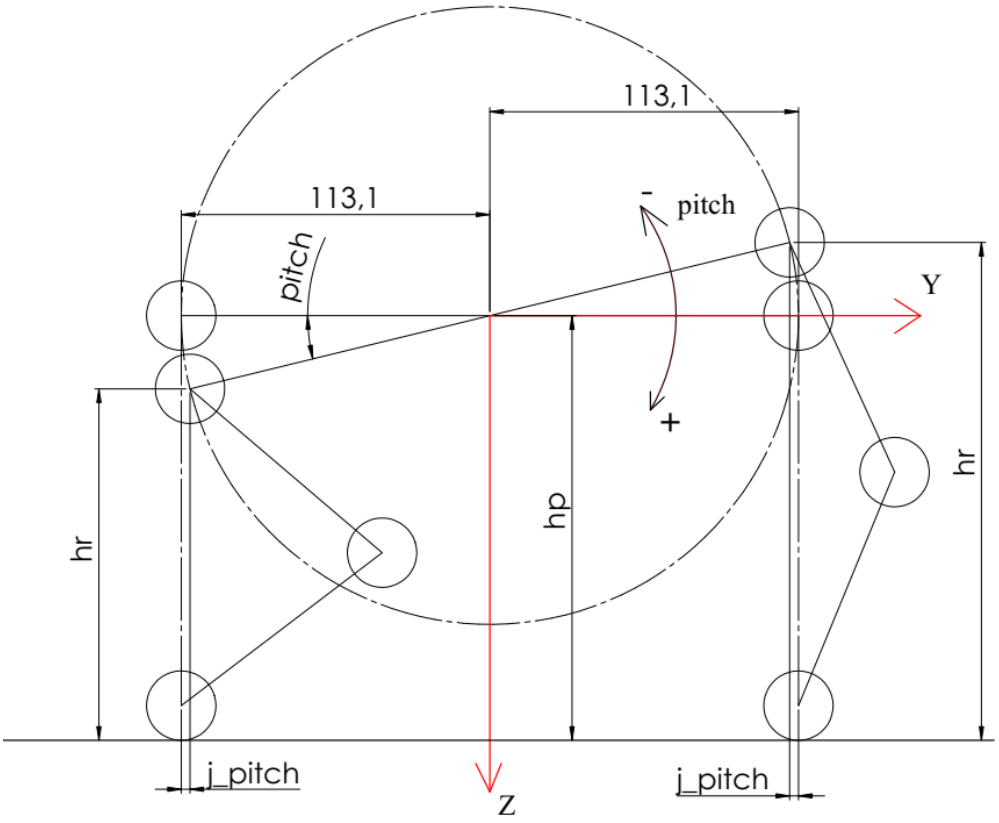
Za razliku od prijašnjeg slučaja nećemo raspisivati jednadžbe za svaku nogu posebno već ćemo u ovom slučaju samo gledati jesu li noge koje nas zanimaju na prednjoj ili stražnjoj strani. Formule su ponovno izvedene s pretpostavkom negativno iznosa zakreta.

Prednja strana:

$$h_r = h_p + 113.1 \cdot \sin(pitch), \quad (3.37)$$
$$j_{pitch} = -113.1 \cdot (1 - \cos(pitch)). \quad (3.38)$$

Stražnja strana:

$$h_r = h_p - 113.1 \cdot \sin(pitch), \quad (3.39)$$
$$j_{pitch} = 113.1 \cdot (1 - \cos(pitch)). \quad (3.40)$$



Slika 32. Rotacija oko X osi

3.4. Rotacija oko Y osi

Ovdje ćemo kao i u slučaju prije samo razmatrati strane robota te ćemo se opredijeliti na lijevu i desnu stranu robota. U ovome će slučaju za razliku od prijašnjih veličina zakreta biti pozitivnog iznosa.

Lijeva strana:

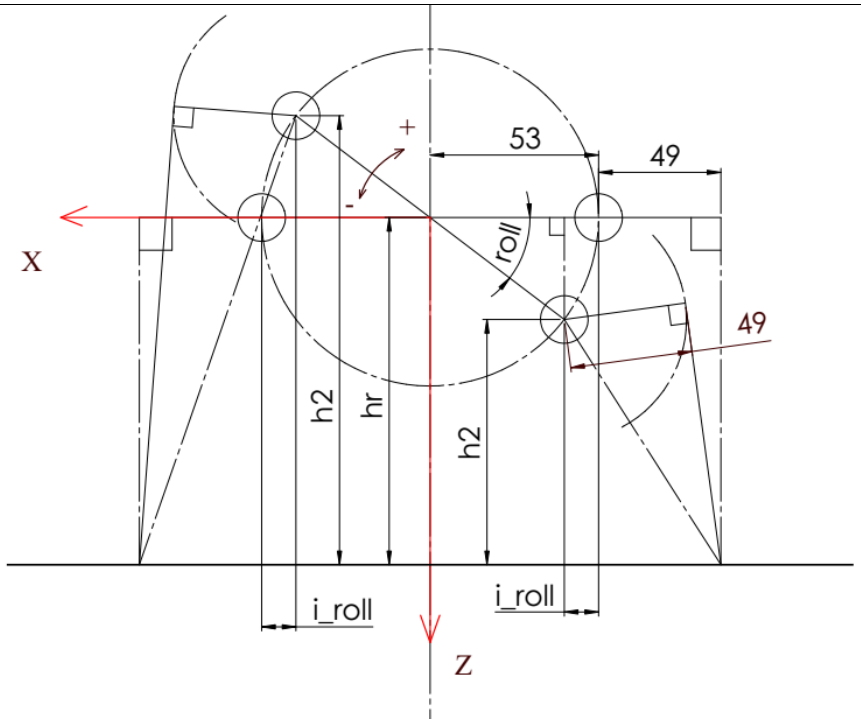
$$h_2 = h_r - 53\sin(roll), \tag{3.41}$$

$$i_{roll} = 53 \cdot (1 - \cos(roll)). \tag{3.42}$$

Desna strana:

$$h_2 = h_r + 53\sin(roll), \tag{3.43}$$

$$i_{roll} = -53 \cdot (1 - \cos(roll)). \tag{3.44}$$



Slika 33. Rotacija oko Y osi

3.5. Translacija u XZ ravnini

Translacija u XZ ravnini se direktno nadovezuje na rotaciju oko osi Y te ovdje također gledamo samo strane robota kako bi smo dobili iznose kutova koje moramo prenjeti na kukove robota, to jest, servo motore. Kako smo pojednostavnili izračun inverzne kinematike ukupni pomak po X osi je podjeljen na tri komponente, nuspordukte zakreta oko osi Z i Y (i_{roll}, i_{yaw}) te na željeni pomak po osi X iznosa i. Sada kad imamo sve komponente koje pomiču robota po X osi treba ih zbrojiti kako bi smo dobili ukupan iznos pomaka:

$$I = i + i_{yaw} + i_{roll}. \tag{3.45}$$

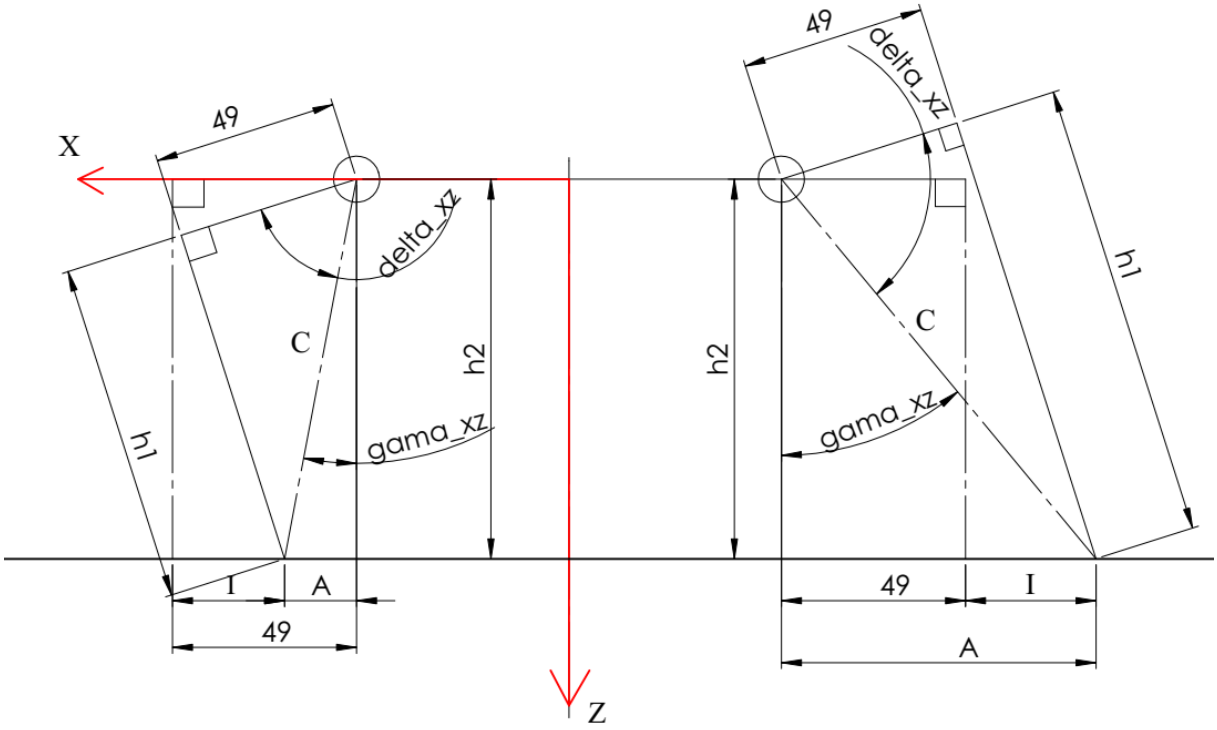
S tim pomakom ulazimo u daljne jednadžbe te dobivamo sljedeće:

- Lijeva strana:

$$A = 49 - I, \tag{3.46}$$
$$C = \sqrt{h_2^2 + A^2}, \tag{3.47}$$
$$h_1 = \sqrt{C^2 - 49^2}, \tag{3.48}$$
$$\gamma_{xz} = tan^{-1}(\frac{h_1}{49}), \tag{3.49}$$
$$\delta_{xz} = tan^{-1}(\frac{A}{h_2}). \tag{3.50}$$

- Desna strana:

$$A = 49 + I, \tag{3.51}$$
$$C = \sqrt{h_2^2 + A^2}, \tag{3.52}$$
$$h_1 = \sqrt{C^2 - 49^2}, \tag{3.53}$$
$$\gamma_{xz} = tan^{-1}(\frac{h_1}{49}), \tag{3.54}$$
$$\delta_{xz} = tan^{-1}(\frac{A}{h_2}). \tag{3.55}$$



Slika 34. Translacija u XZ ravnini

Konačno kut koji ćemo stavljati kao referentni kut servo motora biti će iznosa:

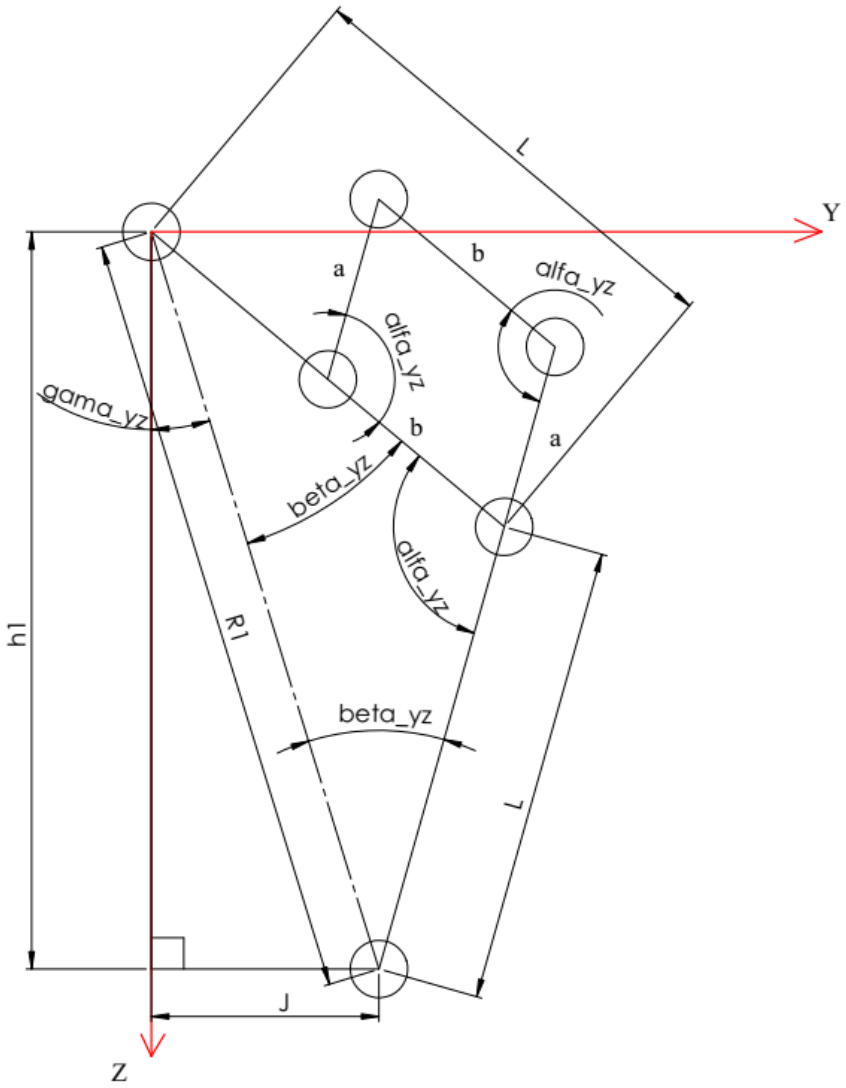
$$\varphi_{kuk} = \gamma_{xz} + \delta_{xz}. \tag{3.56}$$

3.6. Translacija u YZ ravnini

Posljednji set jednadžbi odnosi se na translaciju u YZ ravnini gdje isto imamo da nam je ukupni pomak po osi Y definiran sa tri druga pomaka po osi. Kao i prije to su nusprodukti rotacije oko osi Z (j_{yaw}) te rotacije oko osi X (j_{pitch}) te korisnički željeni pomak po osi Y iznosa j. U ovoj ravnini upravljati ćemo sa dva servo motora, a to su servo motor zadužen za rotaciju ramena i servo motor zadužen za rotaciju lakta. Ovdje za razliku od svih prijašnjih slučajeva nećemo gledati strane robota ili numeraciju nogu jer su njihovi pomaci isti te se izračunate virjednosti kuteva modu direktno unositi u motore.

Kako bi krenuli sa postavljanjem jednadžbi prvobitno ćemo izraziti ukupan pomak po osi Y, a on glasi:

$$J = j + j_{yaw} + j_{pitch}. \tag{3.57}$$



Slika 35. Translacija u YZ ravnini

Sljedeće jednadžbe su izvedene iz trigonometrijskih poučaka o torkutu te glase:

$$R_1 = \sqrt{h_1^2 + J^2}, \tag{3.58}$$

$$\gamma_{yz} = tan^{-1}\left(\frac{J}{h_1}\right), \tag{3.59}$$

$$\alpha_{yz} = \cos^{-1}(1 - 0.5 \cdot \left(\frac{R_1}{L}\right)^2), \tag{3.60}$$

$$\beta_{yz} = 90° - \frac{1}{2}\alpha_{yz}. \tag{3.61}$$

Kutovi koje unosimo kao reference za servo motore glase:

$$\varphi_{rame} = \gamma_{yz} + \beta_{yz}, \tag{3.62}$$

$$\varphi_{lakat} = \alpha_{yz}. \tag{3.63}$$

4. Programiranje i testiranje

Kako bi provjerili ispravnost jednadžbi trebalo ih je prevesti u programski kod koji će se onda prenjeti na mikrokontroler. Stoga je za tu primjenu izabran program Arduino IDE koji je otvorenog tipa što znači da za njega nije potreban nikakva dozvola te da ga se može proširivati sa library datotekama koje mogu biti vlastoručno napisane ili skinute sa interneta.

4.1. Implementacija inverznog kinematičkog modela

Implementaciju je bilo veoma lako provesti zbog pojednostavljenja na 2D slučajeve, numeriranja nogu te podjela na strane robota. Inverzna kinematika je napravljena kao zasebna funkcija koja zahtjeva sljedeće unose: koja je noga u pitanju, pomak po X osi, pomak po Y osi, visinu težišta robota, tj., pomak po osi Z te rotacije oko istih osi.

Kod koji je korišten u cjelini biti će priložen u prilogu, a ovdje ću prikazati samo neke od važnijih komponenti koda.

Prvo kod od korisnika zahtjeva unos podataka te se to izvodi na način:

```
void kinematika(int noga,float i, float j, float z, float roll, float
pitch, float yaw){.......} .
```

Sljedeće što program radi je to da prolazi kroz sve one jednadžbe te vrši proračun kako bi dobili željene rezultate. U nastavku su prikazane samo neke:

1. Rotacija oko Z osi

```
if(noga==1){
  X1=102;
  Y1=-113.1;
  R=sqrt(X1*X1+Y1*Y1);
  alfa_yaw=atan(abs(Y1/X1));
  beta_yaw=alfa_yaw-yaw;
  X2=R*cos(beta_yaw);
  Y2=-R*sin(beta_yaw);
  j_yaw=Y2-Y1;
  i_yaw=X2-X1;
}
```

2. Translacija nogu u XZ ravnini

```
I=i+i_yaw+i_roll;
//lijeva strana
if(noga==3 || noga==4){
  A=49-I;
  C=sqrt(h2*h2+A*A);
  h1=sqrt(C*C-49*49);
  delta_xz=atan(A/h2);
  gama_xz=atan(h1/49);
}
```

3. Trnslacija nogu u YZ ravnini

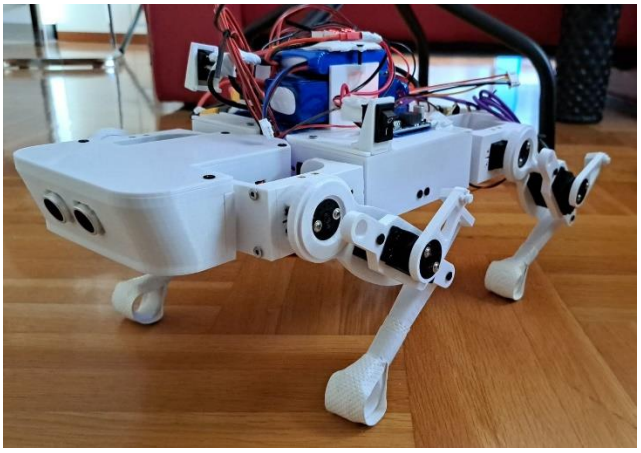
```
J=j+j_yaw+j_pitch;
R1=sqrt(h1*h1+J*J);
gama_yz=atan(J/h1);
alfa_yz=acos(1-(0.5*(R1*R1)/(L*L)));
beta_yz=(PI-alfa_yz)/2;
```

4. Zapis u kutove servomotora

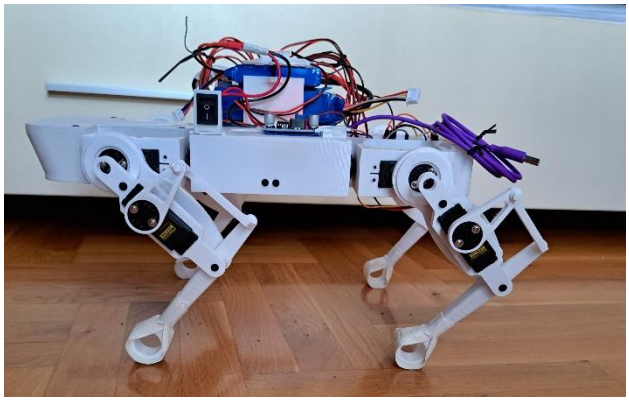
```
fi_kuk=(gama_xz+delta_xz)/PI*180;//kut u stupnjevima
fi_ram=(beta_yz+gama_yz)/PI*180;
fi_lak=alfa_yz/PI*180;
```

4.1.1. Rezultati implementacije inverzne kinematike

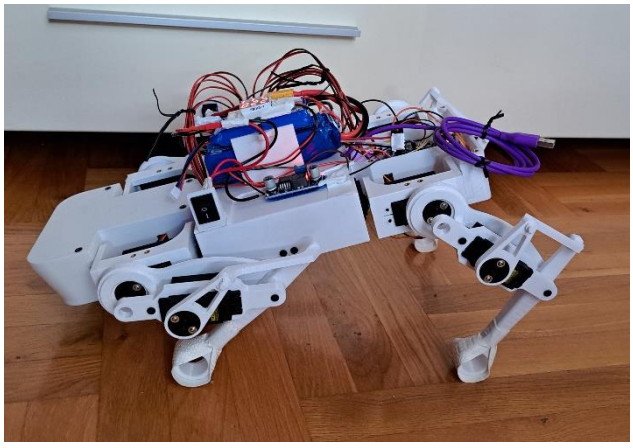
U ovom potpoglavlju su slikama prikazani rezultati inverzne kinematike dirketno na poziciji i orjentaciji robota.



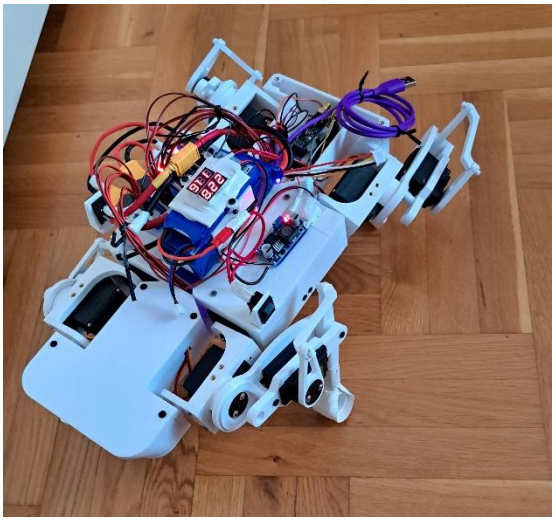
Slika 36. Početni položaj robota



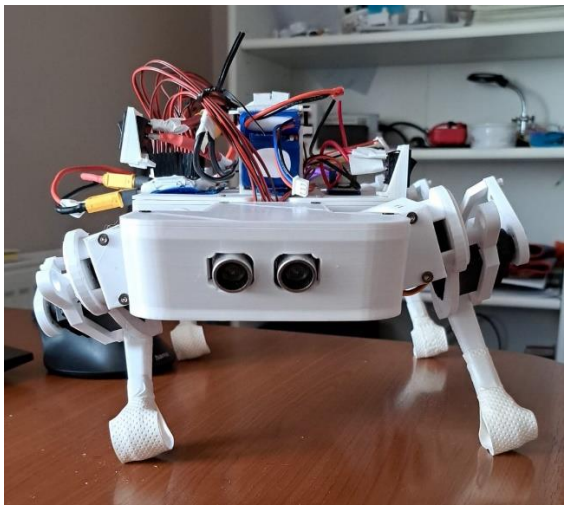
Slika 37. Maksimalan pomak robota po Z osi



Slika 38. Rotacija robota oko X osi



Slika 39. Rotacija robota oko Z osi



Slika 40. Translacija robota u XZ ravnini

Ovim slikama su prikazani samo neki od rezultata jer implementacijom u kodu postignuta je mogućnost upravljanja u svim željenim smjerovima prema željenim iznosima te stoga postoji bezbroj mogućih kombinacija pozicioniranja u prostoru.

4.2. Algoritam hodanja

Sada kada imamo dobro definiranu kinematika i dobru implementaciju iste izrada algoritma hoda se svodi na upravljanje pomakom svake noge zasebno u prostoru. Algoritam razrađen u ovome radu sastoji se od nekoliko djelova , a ovdje ću samo navesti glavne koji su redom:

1. Definiranje krajnjih lokalnih točaka u prostoru svake noge
2. Diskretizacija putanje
3. Prikaz putanje u jednoj ravnini

4.2.1. Definiranje krajnjih lokalnih točaka

Kako bi razradili ciklus hoda, prvo je bilo potrebno definirati krajnje točke u koje noga smije doći. To provodimo kako se nebi narušila stabilnost robota jer prevelika translacija u jednom smjeru može dovesti do gubljenja ravnoteže i pada. Stoga su u ovom algoritmu definiramo maksimalne translacije ovako:

```
void Algoritam_setanja(){
  float kut_zakreta=5;
  float duljina_koraka_i;
  float duljina_koraka_j;

  duljina_koraka_j=30;
  duljina_koraka_i=30;
  ...
```

U ove duljine su uzete u obzir i dodatne translacije koje nastaju usljed zakreta oko pojedinih osi.
Da bi definiranje točaka bilo gotov treba još definirati parametar visine po kojem će se kretati noge. To je napravljeno na vrlo jednostavan način, a to je da se zada visina hoda koja predstavlja visinu težišta te da prilikom dizanja noge krajnja visina dođe do 70% početne visine. To je zapravo virtualna visina na koju bi se robot inače supustio da mu ju zadamo u sve četiri noge istovremeno.

4.2.2. Diskretizacija putanje

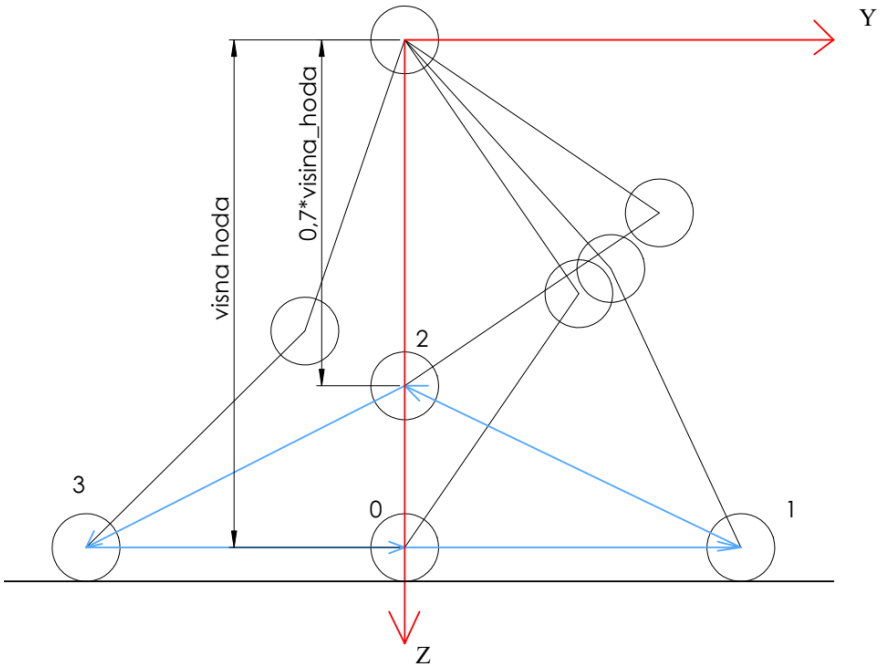
Kako ove servo motore nije moguće upravljati na nižim razinama regulacije već ih upravljamo samo zadavanjem krajnje pozicije oni zbog toga pokušavaju što je brže moguće doći do krajnje pozicije i to čine sa STEP odazivom. Time u sustav uvode neželjenje vibracije i trzaje koji ubrzano troše unutrašnje komponente motora, ali i ostale komponente samog robota. Kako bi se to spriječilo potrebno je bilo uvesti ili inetrpolaciju između točaka putanje ili diskretizirati putanju na puno manjih djelova s čime bi se trzaji motora smanjili. Za ovaj algoritam odabrana je diskretizacija putanje na principu da se zna virtualna početna, trenutna i krajnja pozicija (jer nema povratne veze s motora) te da se zna željeni parametar diskretizacije koji je u ovom algoritmu nazvan *Time*. Nakon toga se računa za koliko se noga mora pomaknuti u idućem prolasku progrma kroz petlje. Opisani dio algoritma u kodu izgleda ovako:

```
if(step_seq==2){//spustamo nogu 1 i 4
      z_targ1=visina_hoda;
      j_targ1=duljina_koraka_j*Direction_j;
      j_targ2=-duljina_koraka_j*Direction_j;

      i_targ1=duljina_koraka_i*Direction_i;
      i_targ2=-duljina_koraka_i*Direction_i;

      brzina_z1=(z_targ1-z_pocetni1)/Time;
      brzina_j1=(j_targ1-j_pocetni1)/Time;
      brzina_j2=(j_targ2-j_pocetni2)/Time;

      brzina_i1=(i_targ1-i_pocetni1)/Time;
      brzina_i2=(i_targ2-i_pocetni2)/Time;
    }

    if(step_seq==2 && z_tren1<z_targ1){// ovdje ovako jer spustamo nogu
      z_tren1=z_tren4=z_tren1+brzina_z1;
```

```
      j_tren1=j_tren4=j_tren1+brzina_j1;
      i_tren1=i_tren4=i_tren1+brzina_i1;

      j_tren2=j_tren3=j_tren2+brzina_j2;
      i_tren2=i_tren3=i_tren2+brzina_i2;

      kinematika(2,i_tren2,j_tren2,z_tren2,ROLL_PID,PITCH_PID,yaw_tren2);
      kinematika(3,i_tren3,j_tren3,z_tren3,ROLL_PID,PITCH_PID,yaw_tren3);

      kinematika(1,i_tren1,j_tren1,z_tren1,ROLL_PID,PITCH_PID,yaw_tren1);
      kinematika(4,i_tren4,j_tren4,z_tren4,ROLL_PID,PITCH_PID,yaw_tren4);
    }
    else if(step_seq==2 && z_tren1>=z_targ1){
      step_seq=3;
      z_pocetni1=z_pocetni4=z_tren1;
      j_pocetni1=j_pocetni4=j_tren1;
      i_pocetni1=i_pocetni4=i_tren1;

      z_pocetni2=z_pocetni3=z_tren2;
      j_pocetni2=j_pocetni3=j_tren2;
      i_pocetni2=i_pocetni3=i_tren2;
    }
```

4.2.3. Prikaz putanje u jednoj ravnini

Sama putanja nije kompleksnog oblika kada se gleda u ravninama Kartezijevog koordinatnog sustava. Napravljena je da bude jednostavna i laka za implementaciju te je zbog toga trokutnog oblika kako je prikazano na sljedećoj slici [Slika].



Slika 41. Primjer putanje noge u YZ ravnini

4.3. PD regulator

Iako su četveronožni roboti stabilni roboti u stacionarnom stanju, njihov hod nemora biti. Naime postoji više vrsta hoda koje je moguće ostvariti sa ovakvim robotima, a neke od njih su redom:

1. Korak – podiže, pomiče i spušta se jedna po jedna noga te se ostvaruje stabilan sustav naginjanjem u stranu suprotnu od podignute noge,
2. Kas – noge se ovdje izmjenjuju dijagonalno, ali su u svakom trenu barem dvije konstantno na tulu,
3. Trčanje – dolazi do dovoljno snažnog i brzog kretanja robota da neko vrijeme ni jedna noga ne dodiruje pod.

U ovom završnom radu pokušano je izvođenje kasa uz pomoć PD regulatora i akcelerometra.

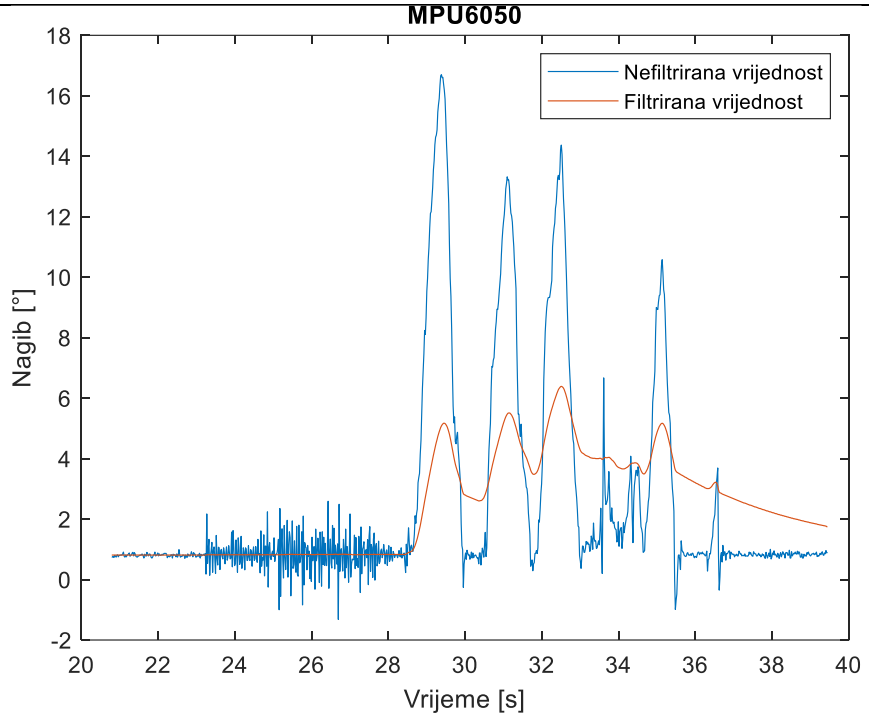
4.3.1. Kalibracija MPU6050

MPU6050 je jeftini troosini akcelerometar i žiroskop koji je vrlo osjetljiv na vibracije te na svom izlazu daje puno šuma. Taj šum je neželjena pojava te ga je poterbno filtrirati. Metoda filtracije se u ovom radu provela primjenom Kalmanovog filtra. Filter radi na sljedeći način:

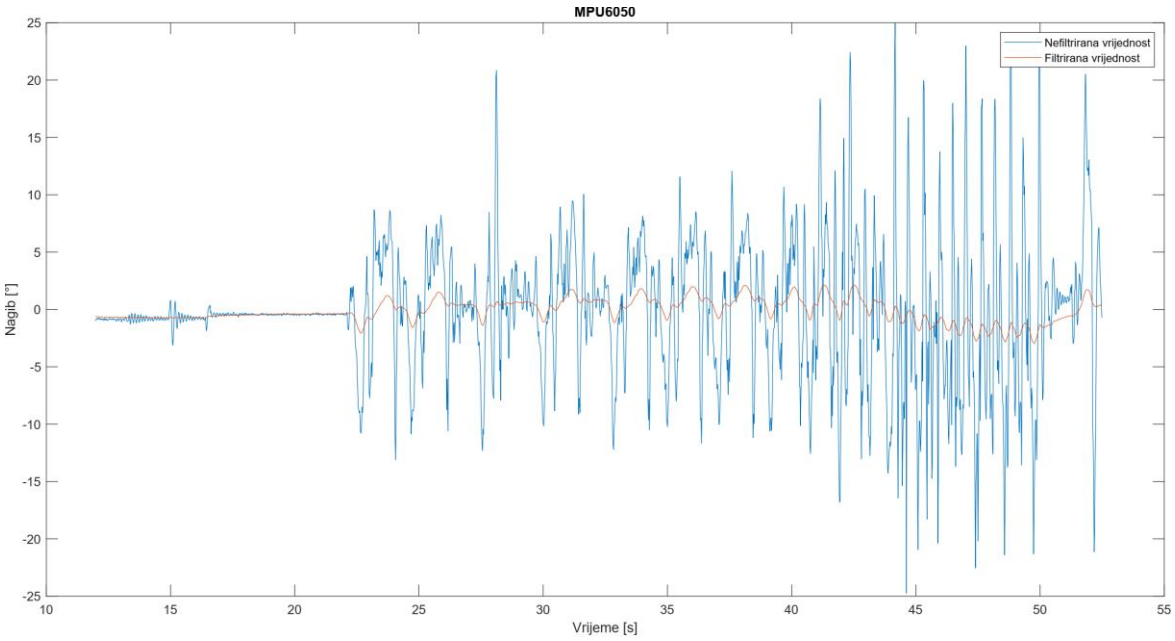
1. Predviđanje trenutačnog stanja sustava:
$$\boldsymbol{S}(k) = \boldsymbol{F} \cdot \boldsymbol{S}(k-1) + \boldsymbol{G} \cdot \boldsymbol{U}(k)$$
2. Izračun nesigurnosti predviđanja:
$$\boldsymbol{P}(k) = \boldsymbol{F} \cdot \boldsymbol{P}(k-1) \cdot \boldsymbol{F}^T + Q$$
3. Izračun Kalmanovog pojačanja iz nesigurnosti predviđanja i mjerenja:
$$\boldsymbol{L}(k) = \boldsymbol{H} \cdot \boldsymbol{P}(k) \cdot \boldsymbol{H}^T + R$$
$$K = \boldsymbol{P}(k) \cdot \frac{\boldsymbol{H}^T}{\boldsymbol{L}(k)}$$
4. Ažuriranje predviđenog stanja sustava s mjerenjem stanja kroz kalmanovo pojačanje:
$$\boldsymbol{S}(k) = \boldsymbol{S}(k) + K \cdot (\boldsymbol{M}(k) - \boldsymbol{H} \cdot \boldsymbol{S}(k))$$
5. Ažuriranje nesigurnosti predviđenog stanja:
$$\boldsymbol{P}(k) = (\boldsymbol{I} - K \cdot \boldsymbol{F}) \cdot \boldsymbol{P}(k)$$

U kodu to izgleda ovako:

```
void kalman_1d(float KalmanState,float KalmanUncertainty,float
KalmanInput,float KalmanMeasurement){
  KalmanState = KalmanState+0.004*KalmanInput;
  KalmanUncertainty = KalmanUncertainty+0.004*0.004*4*4;
  float KalmanGain = KalmanUncertainty*1/(1*KalmanUncertainty+3*3);
  KalmanState = KalmanState+KalmanGain*(KalmanMeasurement-KalmanState);
  KalmanUncertainty = (1-KalmanGain)*KalmanUncertainty;
  //Kalman filter output
  Kalman1DOutput[0]=KalmanState;
Kalman1DOutput[1]=KalmanUncertainty;}
```



Slika 42. Odziv na vibracije podloge i ručno pomicanje (plavo nefiltrirano, crveno filtrirano)



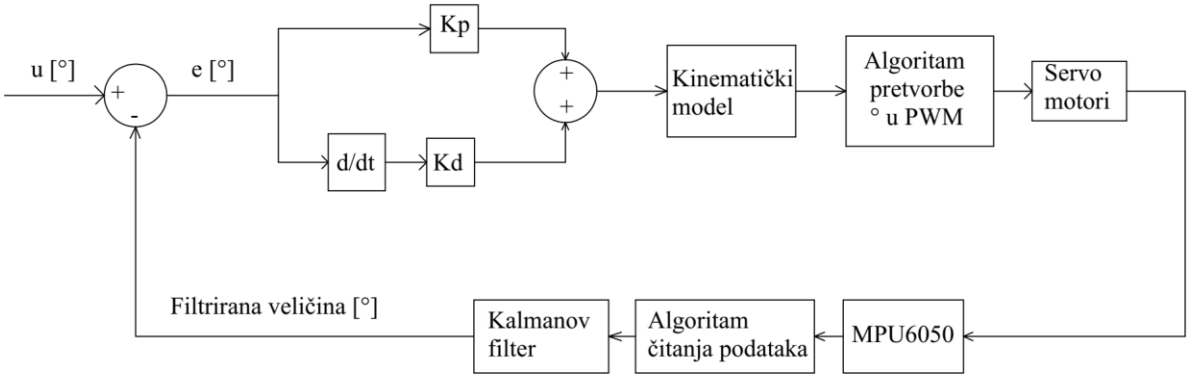
Slika 43. Odziv na kretnje robota (plavo nefiltrirano, crveno filtrirano)

4.3.2. Ziegler-Nichols metoda

Sada kada imamo filtrirani odziv akcelerometra možemo dizajnirati regulator koji će regulirati samo kutove nagiba *pitch* i *roll*. Kako u sklopu ovog rada nije napravljen matematički model, vrijednosti pojačanja PD regulatora morat ćemo odrediti primjenom eksperimentalne metode Ziegler-Nichols. Cilj je ovom metodom povečavati čisto *P* djelovanje regulatora sve dok ne dobijemo granično stabilan odziv te onda iz takvog

snimljenog odziva očitata kritični period. Iza toga se slijede proračuni za preporučene virjednosti pojačanja regulatora. Ovaj eksperiment ćemo odraditi samo za nagib *pitch* jer je prilikom prvobitnog korištenja algoritma hoda zamjećeno veće posrtanje robota prema naprijed.

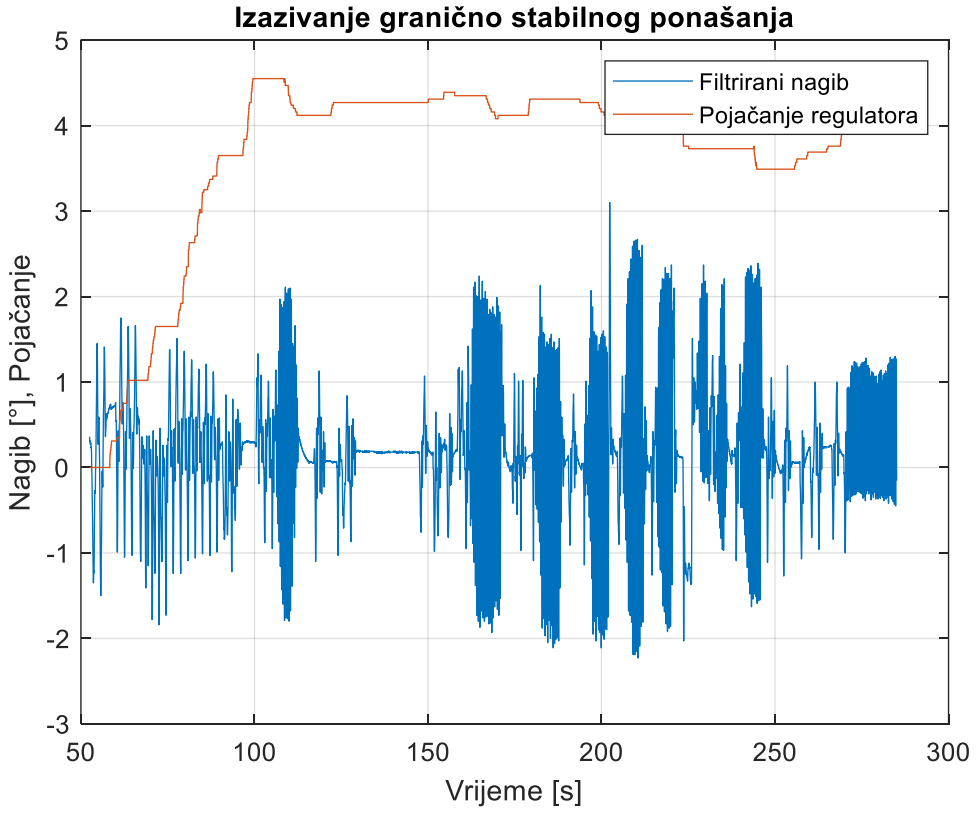
Odziva će se snimati tokom hoda robota kako bi dobili što bolji regulator, ali zbog diskretizacije ciklusa hoda algoritma nećemo imati savršenu snimku nagiba i pojaviti će se neželjeni vrhovi očitanja koji su uglavnom posljedica promjene smjera gibanja.



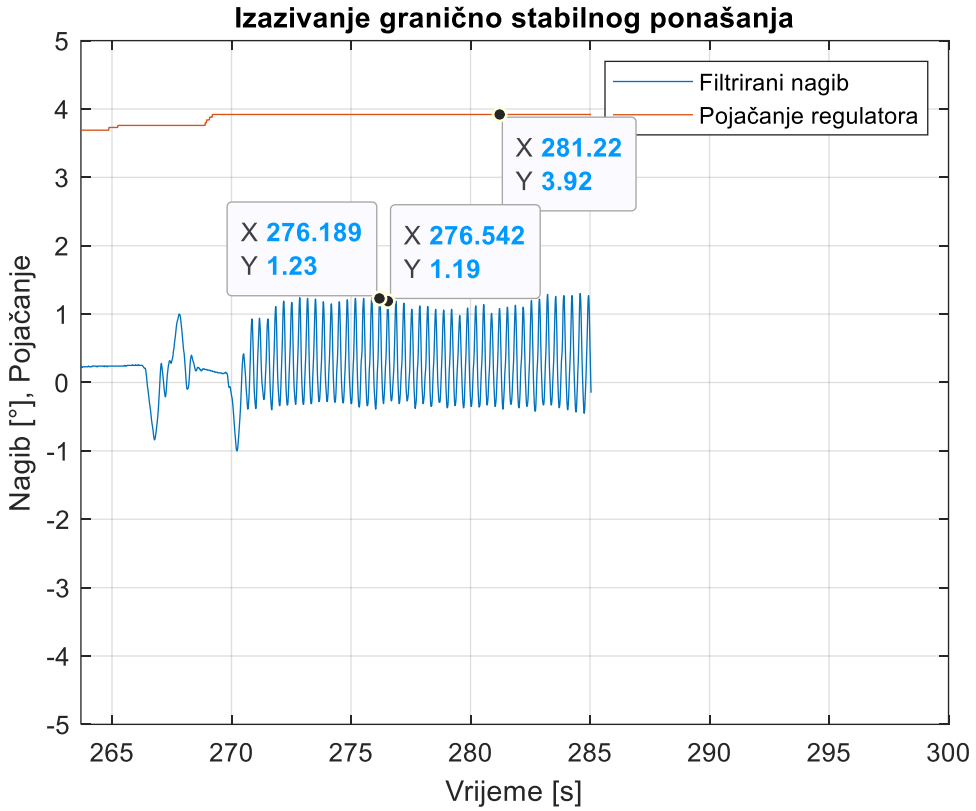
Slika 44. Shema regulacijskog kruga

4.3.2.1. Namještanje regulatora za nagib pitch

Povečavanjem pojačanja P do iznosa 4.31 ušli smo u granično stablino područje te ćemo za to pojačanje odrediti kritičan period tako što ćemo prebaciti očitane vrijednosti iz Serial Monitor (sučelju programa Arduino IDE) u MATLAB te tamo isčitati sve potrebne veličine.



Slika 45. Snimljen odziv senzora (plavo) i pojačanje (crveno)



Slika 46. Uvećana slika područja od interesa

Sada iz uvećane slike [Slika 46] možemo približno očitati da kritični period ponavljanja iznosi:

$$T_{kr} = 276.542 - 276.189 = 0.353s. \quad (4.1)$$

Sada kad znamo kritični period i pojačanje ulazimo u postupak određivanja pojačanja PD regulatora:

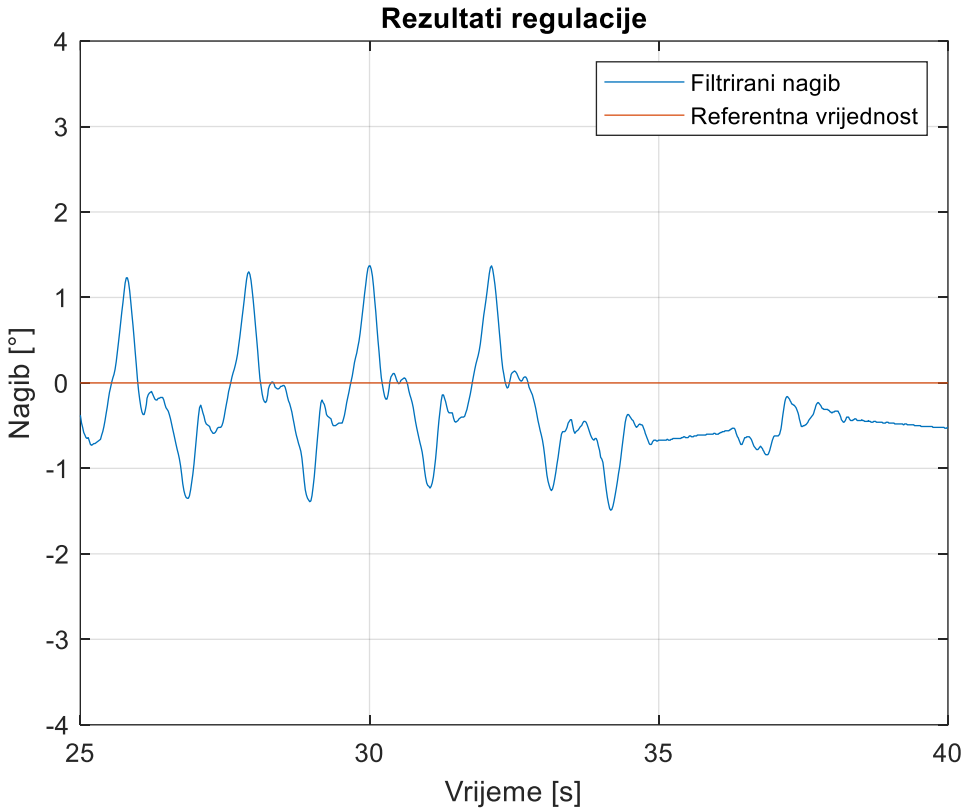
$$K_{Pkr} = 3.92, \quad (4.2)$$

$$K_P = 0.8 * K_{Pkr} = 0.8 * 3.92 = 3.136, \quad (4.3)$$

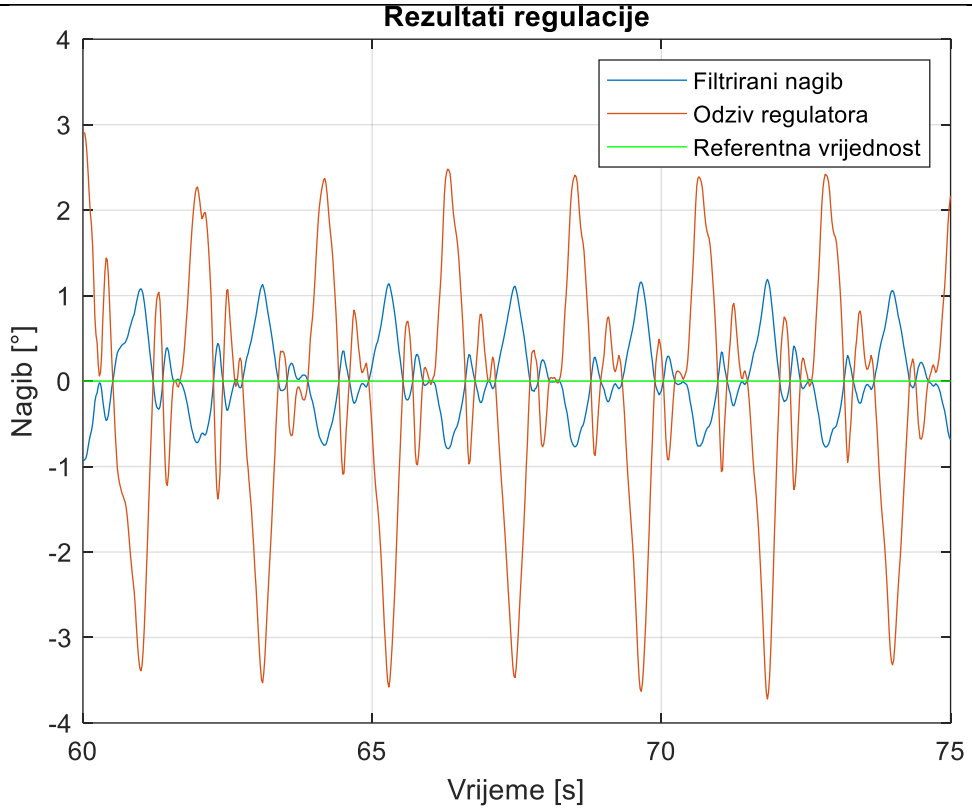
$$T_d = 0.125 * T_{kr} = 0.125 * 0.353s = 0.044s, \quad (4.4)$$

$$K_D = K_P * T_d = 3.92 * 0.044 = 0.1725s. \quad (4.5)$$

4.3.3. Rezultati regulacije



Slika 47. Prikaz nagiba (plavo) i referentna vrijednost (crveno)



Slika 48. Prikaz nagiba (plavo), izlaz regulatora (crveno), referentna vrijednost (zeleno)

U prikazanim slikama [Slika 47] i [Slika 48] vidljivo je da regulator djeluje kako je namjenjen jer se jasno vide razlike najviše u negativnim iznosima nagiba *pitch* prilikom hoda. Iako je dobiveno neko manje smanjenje osciliranja nagiba *pitch*, ono je i dalje prisutno radi ostalih nedostataka samog robota kao što su trenje među djelovima, velika zračnost u motorima koja dovodi do nepreciznog pozicioniranja i dodatnih trzaja usljed inercije te loše dinamike motora koja ne može pratiti željene zahtjeve jer je konačna masa robota nadišla prvobitno planiranu graničnu masu.

5. Budući planovi

Idućui korak kod ovakvog projekta prvobitno bi bio odabir snažnijih i dnamičnijih motora koji nebi ograničavali dinamiku. Poželjno bi bilo uzeti motore kojima bi mogli upravljati putem struje da ne bi dolazilo do trzaja zato što motor želi doći u krajnju poziciju. Usto bilo bi učinkovito imati povratnu vezu sa tlom da znamo koja se noga nalazi na podu, a koja ne te da s tom funkcijom ujedno provjeravamo postoji li nekakava prepreka dubinskog karaktera (rub balkona, rupa u podu, stepenica, itd.) i da reguliramo lakše različite načine hoda. Bilo bi poželjno i ažurirati upravljačke jedinice te umjesto postojećih staviti neke koje bi mogle vršiti puno kompleksniju matematiku matrica i putanja te ujedno koristiti neke od vizijskih sustava kamera kako bi smo mogli omogućiti upravljanje robotom na većim udaljenostima.

6. Zaključak

Projektiranje i izrada stabilnih i dovoljno dinamičkih četveronožnih robota je zahtjevan zadatak koji podrazumjeva znanje iz mnogo različitih područija strojarstva, matematike, informatike i elektrotehnike. Svo programiranje i kasnije testiranje nije korisno ako kinematika robota nije dobro definirana, ako nisu odabrane valjane komponente koje će omogućiti dinamiku te dobro optimiziran dizajn. U ovom završnom radu je na pojednostavljen način prikazan proces izrade jednog takovog robota. Taj se proces sveo najviše na dobro definiranje kinematskog modela kako bi imali što bolju kontrolu nad mobilnim robotom.

Tokom izrade ovog završnog bilo je potrebno steći znanje izrade 3D modela aditivnim tehnologijama poput 3D printanja, programiranja različitih mikrokontrolera, ručnog spajanja i lemljenja električnih komponenti, filtriranja podataka te znanje potrebno za odabir valjanih regulatora i izradu algoritama. Svo ovo znanje vrlo će dobro doći u daljnjem školovanju i u kasnijem radu u industriji.

LITERATURA

[1] Mladen Crneković - Industrijski i mobilni roboti, predavanja, Fakultet strojarstva i brodogradnje, Zagreb 2024.

[2] https://soldered.com/productdata/2017/01/Soldered_MG995_datasheet.pdf

[3] https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf

[4] https://soldered.com/productdata/2015/02/Soldered_LM2596_datasheet.pdf

[5] https://soldered.com/productdata/2015/02/Soldered_HCSR04_datasheet.pdf

[6] https://pdf1.alldatasheet.com/datasheet-pdf/download/1132807/TDK/MPU-6050.html

[7] https://soldered.com/productdata/2021/04/Soldered_atmega328p_datasheet.pdf

[8] https://soldered.com/productdata/2022/03/Soldered_ESP32-WROVER_datasheet.pdf

[9] https://soldered.com/productdata/2015/02/Soldered_nRF24L01Plus_datasheet.pdf

[10] https://konj.forumcroatian.com/t13-hodovi-konja

[11] https://www.youtube.com/watch?v=GZevJyabMdI

[12] https://github.com/CarbonAeronautics/Manual-Quadcopter-Drone/blob/main/Carbon_Aeronautics_Quadcopter_Manual.pdf

[13] Željko Šitum – Računalne simulacije, predavanja, Fakultet strojarstva i brodogradnje, Zgreb 2023.

[14] https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/

[15] https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library

PRILOZI

I. CD-R disc

II. Kod radio daljinskog

III. Kod robota

IV. Sklopni crtež robota

Kod radio daljinskog:

```
#include "SPI.h"
#include "RF24.h"
#include "nRF24L01.h"
#define CE_PIN 9
#define CSN_PIN 10
#define INTERVAL_MS_TRANSMISSION 250
RF24 radio(CE_PIN, CSN_PIN);
const byte address[][6] = {"00001","00002"};
//NRF24L01 buffer limit is 32 bytes (max struct size)
struct Datoteka {
  //postavimo inicijalne vrijednosti
   byte pos_x1=127;
   byte pos_x2=127;
   byte pos_y1=127;
   byte pos_y2=127;
   byte mod=0;
   uint8_t counter_1=33;
   uint8_t counter_2=128;

};
Datoteka dat;

//JOYSTICK////////////////////////////////////////////////////////
#define VRx_1 A4
#define VRy_1 A5
#define sw_1 8
int prev_stanje_sw_1;

#define VRx_2 A6
#define VRy_2 A7
#define sw_2 6
int prev_stanje_sw_2;

int cur_state_sw_1;
int cur_state_sw_2;

int prevCounterMode=0;// zato jer krecemo s modom nula
byte prev_counter1=33;
byte prev_counter2=127;
//VRSTE
MODOVA//////////////////////////////////////////////////////////
#define sw_3 3
#define led_0 A0
#define led_1 A1
#define led_2 A2
#define led_3 A3
int cur_state_sw_3;
int prev_stanje_sw_3;
int posljedni_sw;
unsigned long curMillis_sw1;
unsigned long prevMillis_sw1=0;
unsigned long curMillis_sw2;
unsigned long prevMillis_sw2=0;
unsigned long curMillis_sw3;
unsigned long prevMillis_sw3=0;

//enkoderi
//prvi ovaj je na livoj strani
#define CLK_1 5
```

```
#define DT_1 4
int currentStateCLK_1;
int lastStateCLK_1;
unsigned long prevMillis_enc_1=0;
unsigned long currentMillis_enc_1;

//drugi enkoder
#define CLK_2 2
#define DT_2 7
int currentStateCLK_2;
int lastStateCLK_2;
unsigned long prevMillis_enc_2=0;
unsigned long currentMillis_enc_2;

//
unsigned long previousMillis = 0;
unsigned long curentMillis;

void setup()
{
   Serial.begin(115200);

   //RADIO KOMUNIKACIJA
   radio.begin();
   radio.setAutoAck(false);
   radio.openWritingPipe(address[0]); // adresa slanja podataka 00001
   radio.openReadingPipe(1,address[1]); // adresa primanja 00002
   radio.setDataRate(RF24_250KBPS); //(RF24_250KBPS|RF24_1MBPS|RF24_2MBPS)
   //Greater level = more consumption = longer distance
   radio.setPALevel(RF24_PA_HIGH);
//(RF24_PA_MIN|RF24_PA_LOW|RF24_PA_HIGH|RF24_PA_MAX)
   radio.stopListening();//postavljam ga kao transmiter

   //PINOVI:
   //JOYSTICK
   pinMode(VRx_1,INPUT_PULLUP);
   pinMode(VRy_1,INPUT_PULLUP);
   pinMode(sw_1,INPUT_PULLUP);
   pinMode(VRx_2,INPUT_PULLUP);
   pinMode(VRy_2,INPUT_PULLUP);
   pinMode(sw_2,INPUT_PULLUP);

   //ENKODERI
   pinMode(CLK_1,INPUT);
   pinMode(DT_1,INPUT);
   pinMode(CLK_2,INPUT);
   pinMode(DT_2,INPUT);
   lastStateCLK_1=digitalRead(CLK_1);
   lastStateCLK_2=digitalRead(CLK_2);

   //TIPKALO ZA MODOVE SW_3
   pinMode(sw_3,INPUT);
   prev_stanje_sw_1=digitalRead(sw_1);
   prev_stanje_sw_2=digitalRead(sw_2);
   prev_stanje_sw_3=digitalRead(sw_3);
   digitalWrite(led_0,HIGH);
   digitalWrite(led_1,LOW);
   digitalWrite(led_2,LOW);
   digitalWrite(led_3,LOW);
```

```
    //pošalji prvu vrijednost podataka
    radio.write(&dat,sizeof(Datoteka));


}
void loop() {
  currentStateCLK_1=digitalRead(CLK_1);
  currentStateCLK_2=digitalRead(CLK_2);
  cur_state_sw_1=digitalRead(sw_1);
  cur_state_sw_2=digitalRead(sw_2);
  cur_state_sw_3=digitalRead(sw_3);

  PROVJERA_MODOVA();
  PROVJERA_ENKODERA();









  curentMillis = millis();

  if(curentMillis-previousMillis > 10){
    previousMillis=curentMillis;
    dat.pos_x1=map(analogRead(VRy_1),0,1023,0,255);
    dat.pos_y1=map(analogRead(VRx_1),0,1023,0,255);
    dat.pos_x2=map(analogRead(VRy_2),0,1023,0,255);
    dat.pos_y2=map(analogRead(VRx_2),0,1023,0,255);//mora prominiti jer je
krivo orjentiran joystick
//    Serial.println("posx1");Serial.println(dat.pos_x1);
//    Serial.println("posy1");Serial.println(dat.pos_y1);
//    Serial.println("posx2");Serial.println(dat.pos_x2);
//    Serial.println("posy2");Serial.println(dat.pos_y2);
    radio.write(&dat,sizeof(Datoteka));
//    Serial.print(prev_counter2);Serial.print(",");Serial.print(prev_count
er1);Serial.print(",");
//    Serial.print(dat.counter_2);Serial.print(",");Serial.println(dat.coun
ter_1);

    }//kraj if(curentMillis-previousMillis > 10


}//kraj loopa

void PROVJERA_ENKODERA(){
  //provjera prvog enkodera
  if(currentStateCLK_1 != lastStateCLK_1 && currentStateCLK_1==1){
    currentMillis_enc_1=millis();
    if(currentMillis_enc_1 - prevMillis_enc_1 > 6.5){
      if(digitalRead(DT_1) != currentStateCLK_1){
        dat.counter_1 --;
        dat.counter_1=constrain(dat.counter_1,1,64);
        if(dat.mod==0){
          prev_counter1 --;
          prev_counter1=constrain(prev_counter1,1,64);
        }
      }else{
```

```
        dat.counter_1 ++;
        dat.counter_1=constrain(dat.counter_1,1,64);
        if(dat.mod==0){
          prev_counter1 ++;
          prev_counter1=constrain(prev_counter1,1,64);
        }
      }

//      Serial.print(" | Counter_1: ");
//      Serial.println(dat.counter_1);
      prevMillis_enc_1=currentMillis_enc_1;
    }//end of timed function of Millis_enc_1

  }//end of state checking for first encoder
  lastStateCLK_1=currentStateCLK_1;

  //provjera drugog enkodera
  if(currentStateCLK_2 != lastStateCLK_2 && currentStateCLK_2==1){
    currentMillis_enc_2=millis();
    if(currentMillis_enc_2 - prevMillis_enc_2 > 6.5){
      if(digitalRead(DT_2) != currentStateCLK_2){
        dat.counter_2 --;
        dat.counter_2=constrain(dat.counter_2,1,255);
        if(dat.mod==0){
          prev_counter2 --;
          prev_counter2=constrain(prev_counter2,1,255);
        }
      }else{
        dat.counter_2 ++;
        if(dat.counter_2>=255){
          dat.counter_2 --;
          dat.counter_2=constrain(dat.counter_2,1,255);
        }

        if(dat.mod==0){
          prev_counter2 ++;
          if(prev_counter2>=255){
            prev_counter2 --;
            prev_counter2=constrain(prev_counter2,1,255);
          }
        }
      }
      prevMillis_enc_2=currentMillis_enc_2;
//      Serial.print(" | Counter_2: ");
//      Serial.println(dat.counter_2);
    }//end of timed function of Millis_enc_2

  }//end of state checking for second encoder
  lastStateCLK_2=currentStateCLK_2;


}//end of void PROVJERA_ENDKODERA()


void PROVJERA_MODOVA(){
  if(dat.mod==1&& prevCounterMode==0 || dat.mod==1&& prevCounterMode==2){
    Serial.println("USA");
    prevCounterMode=1;
    dat.counter_1=1;
    dat.counter_2=1;//resetiramo vrijednosti da moremo namistiti PD
regulator
```

```
  }
  else if(dat.mod==0 && prevCounterMode==1 || dat.mod==0 &&
prevCounterMode==2){
    dat.counter_1=prev_counter1;
    dat.counter_2=prev_counter2;
    prevCounterMode=0;
  }
  else if(dat.mod==2 && prevCounterMode==0 || dat.mod==2 &&
prevCounterMode==1){
    prevCounterMode=2;
    dat.counter_1=33;// stavljamo na pola jer želimo krenuti sa srednjom
brzinom
  }

  if(cur_state_sw_3 != prev_stanje_sw_3 && cur_state_sw_3==0){
    curMillis_sw3=millis();
    if(curMillis_sw3 - prevMillis_sw3>5){
      prevMillis_sw3=curMillis_sw3;
      dat.mod=0;
      digitalWrite(led_0,HIGH);
      digitalWrite(led_1,LOW);
      digitalWrite(led_2,LOW);
      digitalWrite(led_3,LOW);
    }//end of timed event for mode 0

  }//end of if clause for mode 0
  prev_stanje_sw_3=cur_state_sw_3;

  if(cur_state_sw_1 != prev_stanje_sw_1 && cur_state_sw_1==0){
    curMillis_sw1=millis();
    if(curMillis_sw1 - prevMillis_sw1>5){
      prevMillis_sw1=curMillis_sw1;
      dat.mod=1;
      digitalWrite(led_0,LOW);
      digitalWrite(led_1,HIGH);
      digitalWrite(led_2,LOW);
      digitalWrite(led_3,LOW);
    }//end of timed event for mode 1

  }//end of if clause for mode 1
  prev_stanje_sw_1=cur_state_sw_1;

  if(cur_state_sw_2 != prev_stanje_sw_2 && cur_state_sw_2==0){
    curMillis_sw2=millis();
    if(curMillis_sw2-prevMillis_sw2>5){
      prevMillis_sw2=curMillis_sw2;
      dat.mod=2;
      digitalWrite(led_0,LOW);
      digitalWrite(led_1,LOW);
      digitalWrite(led_2,HIGH);
      digitalWrite(led_3,LOW);
    }//end of timed event for mode 2

  }//end of if clause for mode 2
  prev_stanje_sw_2=cur_state_sw_2;

}// end of PROVJERA_MODOVA
```

Kod robota:

```
#include <Wire.h>
//================================================================================
=
//===                      PCA9685                                             ===
//================================================================================
=
#include <Adafruit_PWMServoDriver.h>
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
#define SERVOMIN  100 // This is the 'minimum' pulse length count (out of
4096)
#define SERVOMAX  515 // This is the 'maximum' pulse length count (out of
4096)
#define SERVO_FREQ 50 // Analog servos run at ~50 Hz updates-PROVJERNO OVO
NE MINJAJ

//================================================================================
========
//===                      RADIO
KOM                                                 ===
//================================================================================
========
#include "SPI.h"
#include "RF24.h"
#include "nRF24L01.h"
#define CE_PIN 4
#define CSN_PIN 5
#define INTERVAL_MS_TRANSMISSION 250
RF24 radio(CE_PIN, CSN_PIN);
const byte address[][6] = {"00001","00002"};
struct Datoteka {
   byte pos_x1;
   byte pos_x2;
   byte pos_y1;
   byte pos_y2;
   byte mod;
   uint8_t counter_1;//moram ovako jer esp32 je 32-bitan, a atmega328p je
8-bitan
   uint8_t counter_2;

};
Datoteka dat;


//================================================================
//                      MPU6050
//================================================================
//define gyroscope and accelerometer variables
float RateRoll,RatePitch,RateYaw;
float RateCalibrationRoll,RateCalibrationPitch,RateCalibrationYaw;

int RateCalibrationNumber;
float AccX,AccY,AccZ;
float AngleRoll,AnglePitch;
uint32_t LoopTimer;


float KalmanAngleRoll=0,KalmanUncertaintyAngleRoll=2*2;//define the
predicted angles and the uncertainties
float KalmanAnglePitch=0,KalmanUncertaintyAnglePitch=2*2;
```

```
float Kalman1DOutput[]={0,0};//initialize the output of the filter

//create the function that calculates the predicted angle and uncertainty
using the Kalman equations
void kalman_1d(float KalmanState,float KalmanUncertainty,float
KalmanInput,float KalmanMeasurement){
  KalmanState = KalmanState+0.004*KalmanInput;
  KalmanUncertainty = KalmanUncertainty+0.004*0.004*4*4;
  float KalmanGain = KalmanUncertainty*1/(1*KalmanUncertainty+3*3);
  KalmanState = KalmanState+KalmanGain*(KalmanMeasurement-KalmanState);
  KalmanUncertainty = (1-KalmanGain)*KalmanUncertainty;
  //Kalman filter output
  Kalman1DOutput[0]=KalmanState;
  Kalman1DOutput[1]=KalmanUncertainty;
}

void gyro_signals(void){
  Wire.beginTransmission(0x68); //Switch on the low-pass filter
  Wire.write(0x1A);
  Wire.write(0x05);
  Wire.endTransmission();
  Wire.beginTransmission(0x68);//configure the accelerometer
  Wire.write(0x1c);
  Wire.write(0x10);//this coresponds to something look in datasheet
  Wire.endTransmission();
  //Pull acc measurements
  Wire.beginTransmission(0x68);
  Wire.write(0x3B);
  Wire.endTransmission();
  Wire.requestFrom(0x68,6);
  int16_t AccXLSB = Wire.read()<<8|Wire.read();
  int16_t AccYLSB = Wire.read()<<8|Wire.read();
  int16_t AccZLSB = Wire.read()<<8|Wire.read();
  //Configure the gyroscope otuput and full rotation rate measuements from
the sensor
  Wire.beginTransmission(0x68);
  Wire.write(0x1B);
  Wire.write(0x8);
  Wire.endTransmission();
  Wire.beginTransmission(0x68);
  Wire.write(0x43);
  Wire.endTransmission();
  Wire.requestFrom(0x68,6);
  int16_t GyroX=Wire.read()<<8 | Wire.read();
  int16_t GyroY=Wire.read()<<8 | Wire.read();
  int16_t GyroZ=Wire.read()<<8 | Wire.read();
  RateRoll=(float)GyroX/65.5;
  RatePitch=(float)GyroY/65.5;
  RateYaw=(float)GyroZ/65.5;
  //Converte the measurements to physical values
  AccX=(float)AccXLSB/4096-0.08;//look in datashete to see wich value to
subtract do gain more or less sensitivity
  AccY=(float)AccYLSB/4096-0.02;
  AccZ=(float)AccZLSB/4096-0.16;
  //Calculate the absolute angles
  AngleRoll=atan(AccY/sqrt(AccX*AccX+AccZ*AccZ))*1/(3.142/180);
  AnglePitch=-atan(AccX/sqrt(AccY*AccY+AccZ*AccZ))*1/(3.142/180);//this is
retured in degrees
}
//=============================================================
```

```
//============================================================
//      PID regulator
//============================================================
float K[2]={0,0};
float KD[2]={0.0,0.0};
float KI[2]={0,0};
float YMAX=20,YMIN=-20;
float u;
float y;
float yD;
float yP;
float yI[2]={0,0};
float u_1[2]={0,0};
float ROLL_PID;
float PITCH_PID;
unsigned long PIDMillis=0;
unsigned long printMillis=0;
//============================================================

//============================================================================
=====
//==          inicijalizacija potrebnih
varijabli                              ==
//============================================================================
=====

int Start_up=0;
int prevStart_up=0;
#define PI 3.14159265358979323846264338327 95
int step_seq=0;
int safety_step_seq=0;
int rotation_seq=0;
int prevMODE=0;
int Direction=0;//ovu ne koristim vise kasnije izbris
int Direction_j=0; //can only be -1 or 1 or 0 if not walking
int Direction_i=0;
int Direction_rotation=0;

int walk=0;//beginig value
int walk_allow=1;//beginnig value

int prevAction=0;//0-either rotated or was in mode 0; if 1 then it has
walked
float visina=120;//wallking height

// Vremenske varijable
unsigned long previousMillis = 0;
unsigned long currentMillis;
unsigned long previousSafetyMillis;
int safety_seq=1;

float Time;

float visina_hoda;

int pos_x1;
int pos_y1;
int pos_x2;
int pos_y2;
int mod=0;
```

```
int counter_1;
int counter_2;


float i;//pomak u smjeru osi y
float j;//pomak u smjeru osi x
float z;//pomak u smjeru osi z
float p;//rotacija oko osi x
float r;//rotacija oko osi y
float yaw;//rotacija oko osi z


//noga1 s1=kuk1,s2=rame1,s3=lakat1  prednja desna
//noga2 s4=kuk2,s5=rame2,s6=lakat2  straznja desna
//noga3 s7=kuk3,s8=rame3,s9=lakat3  prednja liva
//noga4 s10=kuk4,s11=rame4,s12=lakat4  straznja liva
//ove vrijednosti se minjaju ovisno o tome kako spojis kabele na PCA9685
int kuk1=0,rame1=1,lakat1=2,kuk2=5,rame2=6,lakat2=7;
int kuk3=15,rame3=14,lakat3=13,kuk4=10,rame4=9,lakat4=8;


float i_tren;//trenutna pozicija
float j_tren;
float z_tren;

float r_tren;
float p_tren;
float yaw_tren;

float i_prev;
float j_prev;
float z_prev;

float r_prev;
float p_prev;
float yaw_prev;

float j_pocetni1,j_pocetni2,j_pocetni3,j_pocetni4;
float i_pocetni1,i_pocetni2,i_pocetni3,i_pocetni4;
float z_pocetni1,z_pocetni2,z_pocetni3,z_pocetni4;
float yaw_pocetni1,yaw_pocetni2,yaw_pocetni3,yaw_pocetni4;

float j_targ1,j_targ2,j_targ3,j_targ4;
float i_targ1,i_targ2,i_targ3,i_targ4;
float z_targ1,z_targ2,z_targ3,z_targ4;
float yaw_targ1,yaw_targ2,yaw_targ3,yaw_targ4;

float j_tren1,j_tren2,j_tren3,j_tren4;
float i_tren1,i_tren2,i_tren3,i_tren4;
float z_tren1,z_tren2,z_tren3,z_tren4;
float yaw_tren1,yaw_tren2,yaw_tren3,yaw_tren4;

float brzina_j1,brzina_i1,brzina_z1;
float brzina_j2,brzina_i2,brzina_z2;
float brzina_j3,brzina_i3,brzina_z3;
float brzina_j4,brzina_i4,brzina_z4;
float brzina_yaw1,brzina_yaw2,brzina_yaw3,brzina_yaw4;

//=========================================================================
==
//===       Senzor udaljenosti
```

```
//================================================================================
==

#define Signal_udaljenosti 25

void setup() {
  Serial.begin(115200);
  while(!Serial);
//  Serial.println("Start");
  //=========================================================
  //                MPU6050
  //=========================================================
  Wire.setClock(400000);
  Wire.begin();
  delay(250);
  Wire.beginTransmission(0x68);
  Wire.write(0x6B);
  Wire.write(0x00);
  Wire.endTransmission();
  for(RateCalibrationNumber=0;RateCalibrationNumber<2000;RateCalibrationNum
ber++){
    gyro_signals();
    RateCalibrationRoll+=RateRoll;
    RateCalibrationPitch+=RatePitch;
    RateCalibrationYaw+=RateYaw;
    delay(1);
  }
  RateCalibrationRoll/=2000;
  RateCalibrationPitch/=2000;
  RateCalibrationYaw/=2000;

  //=====================================================================
  //                RADIO
  //=====================================================================
  radio.begin();
  radio.setAutoAck(false);
  radio.openWritingPipe(address[1]); // adresa slanja podataka 00002
  radio.openReadingPipe(1,address[0]); // adresa primanja 00001
  //Set the transmission datarate
  radio.setDataRate(RF24_250KBPS); //(RF24_250KBPS|RF24_1MBPS|RF24_2MBPS)
  //Greater level = more consumption = longer distance
  radio.setPALevel(RF24_PA_HIGH);
//(RF24_PA_MIN|RF24_PA_LOW|RF24_PA_HIGH|RF24_PA_MAX)
  radio.startListening();

  pwm.begin();
  pwm.setPWMFreq(SERVO_FREQ);

  //==============================
  // Signal sa senzora udaljenosti
  pinMode(Signal_udaljenosti,INPUT);

}//end of setup

void loop() {


  currentMillis = millis();

  if(currentMillis-printMillis>1){
    printMillis=currentMillis;
```

```
//    Serial.print(K[1]);Serial.print(",");Serial.println(KalmanAnglePitch)
;
    Serial.print(KalmanAnglePitch);Serial.print(",");Serial.println(PITCH_P
ID);
  }

  if(currentMillis-PIDMillis > 2){
    PIDMillis=currentMillis;
//    ROLL_PID=PID(0,KalmanAngleRoll,0);
    PITCH_PID=PID(0,KalmanAnglePitch,1);
  }

  if(currentMillis-previousMillis > 10){

    previousMillis=currentMillis;
    if(Start_up==0){
      i_prev=0;
      j_prev=0;
      z_prev=120;
      r_prev=0;
      p_prev=0;
      yaw_prev=0;
      Start_up=1;
//      Serial.println("ROBI STARTAN");
    }
    else if(Start_up==1){

//    Serial.println(z_tren1);


      //============================
      //    MPU6050
      //============================
      gyro_signals();
      //calcualte the rotation rates
      RateRoll-=RateCalibrationRoll;
      RatePitch-=RateCalibrationPitch;
      RateYaw-=RateCalibrationYaw;

      kalman_1d(KalmanAngleRoll,KalmanUncertaintyAngleRoll,RateRoll,AngleRo
ll);
      KalmanAngleRoll=Kalman1DOutput[0];
      KalmanUncertaintyAngleRoll=Kalman1DOutput[1];

      kalman_1d(KalmanAnglePitch,KalmanUncertaintyAnglePitch,RatePitch,Angl
ePitch);
      KalmanAnglePitch=Kalman1DOutput[0];
      KalmanUncertaintyAnglePitch=Kalman1DOutput[1];
//      Serial.print("Roll angle [°]= ");
//      Serial.print(KalmanAngleRoll);
//      Serial.print("Pitch angle [°]= ");
//      Serial.println(KalmanAnglePitch);



      if (radio.available()) {
        radio.read(&dat,sizeof(Datoteka));
        previousSafetyMillis = currentMillis;
      }
```

```
      if (currentMillis - previousSafetyMillis > 500) {
        Safety_position();
//            Serial.println("*no remote data* ");

      }else{
        pos_x1=dat.pos_x1;
        pos_y1=dat.pos_y1;
        pos_x2=dat.pos_x2;
        pos_y2=dat.pos_y2;
        counter_1=dat.counter_1;
        counter_2=dat.counter_2;
      }

      if(dat.mod==2 && prevStart_up==0){//ovo dodano ako slučajno uđemo u
mod 2, a robot još nije prvi put uša u mod 0 ugl moremo daljinski pokrenuti
u kojem god ocemo modu i robot bi treba moci odma raditi
        z_tren=120;
        prevStart_up=1;
      }

      MODE(dat.mod);
    }//end of Start_up==1
  }//end of timed event
}//end of loop


void Algoritam_setanja(){
  float kut_zakreta=5;
  float duljina_koraka_i;
  float duljina_koraka_j;

//  if(Direction_rotation==1 || Direction_rotation==-1){
//      duljina_koraka_j=20;
//      duljina_koraka_i=20;
//  }else{
//      duljina_koraka_j=30;
//      duljina_koraka_i=30;
//  }

  duljina_koraka_j=30;
  duljina_koraka_i=30;



  kinematika(1,i_tren1,j_tren1,z_tren1,ROLL_PID,PITCH_PID,yaw_tren1);
  kinematika(4,i_tren4,j_tren4,z_tren4,ROLL_PID,PITCH_PID,yaw_tren4);

  kinematika(2,i_tren2,j_tren2,z_tren2,ROLL_PID,PITCH_PID,yaw_tren2);
  kinematika(3,i_tren3,j_tren3,z_tren3,ROLL_PID,PITCH_PID,yaw_tren3);



  if(walk==1 && prevAction==0 || walk==1 && prevAction==1){
    ROLL_PID=PID(0,KalmanAngleRoll,0);
    PITCH_PID=PID(0,KalmanAnglePitch,1);
    prevAction=1;
    safety_step_seq=0;
    if(step_seq==0){
      yaw_targ1=yaw_targ2=yaw_targ3=yaw_targ4=kut_zakreta*Direction_rotatio
n;
      brzina_yaw1=brzina_yaw4=(yaw_targ1-yaw_pocetni1)/Time;
```

```
      brzina_yaw2=brzina_yaw3=(yaw_targ2-yaw_pocetni2)/Time;
    }
    if(Direction_rotation==1){
      if(step_seq==0 && yaw_tren1<yaw_targ1){//rotiramo u desno
        yaw_tren1=yaw_tren4=yaw_tren1+brzina_yaw1;
        yaw_tren2=yaw_tren3=yaw_tren2+brzina_yaw2;

        kinematika(1,i_tren1,j_tren1,z_tren1,ROLL_PID,PITCH_PID,yaw_tren1);
        kinematika(4,i_tren4,j_tren4,z_tren4,ROLL_PID,PITCH_PID,yaw_tren4);

        kinematika(2,i_tren2,j_tren2,z_tren2,ROLL_PID,PITCH_PID,yaw_tren2);
        kinematika(3,i_tren3,j_tren3,z_tren3,ROLL_PID,PITCH_PID,yaw_tren3);
      }
      else if(step_seq==0 && yaw_tren1>=yaw_targ1){
        step_seq=1;
        z_targ1=0.7*visina_hoda;//dizemo nogu 1 i 4
        yaw_targ1=yaw_targ4=0;

        yaw_pocetni1=yaw_pocetni4=yaw_tren1;
        yaw_pocetni2=yaw_pocetni3=yaw_tren2;
      }
    }
    if(Direction_rotation==-1){
      if(step_seq==0 && yaw_tren1>yaw_targ1){//rotiramo u livo
        yaw_tren1=yaw_tren4=yaw_tren1+brzina_yaw1;
        yaw_tren2=yaw_tren3=yaw_tren2+brzina_yaw2;

        kinematika(1,i_tren1,j_tren1,z_tren1,ROLL_PID,PITCH_PID,yaw_tren1);
        kinematika(4,i_tren4,j_tren4,z_tren4,ROLL_PID,PITCH_PID,yaw_tren4);

        kinematika(2,i_tren2,j_tren2,z_tren2,ROLL_PID,PITCH_PID,yaw_tren2);
        kinematika(3,i_tren3,j_tren3,z_tren3,ROLL_PID,PITCH_PID,yaw_tren3);
      }
      else if(step_seq==0 && yaw_tren1<=yaw_targ1){
        step_seq=1;
        z_targ1=0.7*visina_hoda;//dizemo nogu 1 i 4
        yaw_targ1=yaw_targ4=0;

        yaw_pocetni1=yaw_pocetni4=yaw_tren1;
        yaw_pocetni2=yaw_pocetni3=yaw_tren2;
      }
    }
    if(Direction_rotation==0 && step_seq==0){
      step_seq=1;
      z_targ1=0.7*visina_hoda;//dizemo nogu 1 i 4
      yaw_pocetni1=yaw_pocetni4=yaw_tren1;
      yaw_pocetni2=yaw_pocetni3=yaw_tren2;
    }

    if(step_seq==1){//dizemo nogu 1 i 4
      brzina_z1=(z_targ1-z_pocetni1)/Time;
      brzina_yaw1=brzina_yaw4=(yaw_targ1-yaw_pocetni1)/Time;
    }
    if(step_seq==1 && z_tren1>z_targ1){ //odi je z_tren1>z_targ1 jer dizemo
nogu z os gleda prema doli
      z_tren1=z_tren4=z_tren1+brzina_z1;
      yaw_tren1=yaw_tren4=yaw_tren1+brzina_yaw1;

      kinematika(1,i_tren1,j_tren1,z_tren1,ROLL_PID,PITCH_PID,yaw_tren1);
      kinematika(4,i_tren4,j_tren4,z_tren4,ROLL_PID,PITCH_PID,yaw_tren4);
```

```
      kinematika(2,i_tren2,j_tren2,z_tren2,ROLL_PID,PITCH_PID,yaw_tren2);
      kinematika(3,i_tren3,j_tren3,z_tren3,ROLL_PID,PITCH_PID,yaw_tren3);
    }
    else if(step_seq==1 && z_tren1<=z_targ1){
      step_seq=2;
      z_pocetni1=z_tren1;
      j_pocetni1=j_tren1;
      j_pocetni2=j_tren2;

      i_pocetni1=i_tren1;
      i_pocetni2=i_tren2;
    }

    if(step_seq==2){//spustamo nogu 1 i 4
      z_targ1=visina_hoda;
      j_targ1=duljina_koraka_j*Direction_j;
      j_targ2=-duljina_koraka_j*Direction_j;

      i_targ1=duljina_koraka_i*Direction_i;
      i_targ2=-duljina_koraka_i*Direction_i;

      brzina_z1=(z_targ1-z_pocetni1)/Time;
      brzina_j1=(j_targ1-j_pocetni1)/Time;
      brzina_j2=(j_targ2-j_pocetni2)/Time;

      brzina_i1=(i_targ1-i_pocetni1)/Time;
      brzina_i2=(i_targ2-i_pocetni2)/Time;
    }

    if(step_seq==2 && z_tren1<z_targ1){// ovdje ovako jer spustamo nogu
      z_tren1=z_tren4=z_tren1+brzina_z1;
      j_tren1=j_tren4=j_tren1+brzina_j1;
      i_tren1=i_tren4=i_tren1+brzina_i1;

      j_tren2=j_tren3=j_tren2+brzina_j2;
      i_tren2=i_tren3=i_tren2+brzina_i2;

      kinematika(2,i_tren2,j_tren2,z_tren2,ROLL_PID,PITCH_PID,yaw_tren2);
      kinematika(3,i_tren3,j_tren3,z_tren3,ROLL_PID,PITCH_PID,yaw_tren3);

      kinematika(1,i_tren1,j_tren1,z_tren1,ROLL_PID,PITCH_PID,yaw_tren1);
      kinematika(4,i_tren4,j_tren4,z_tren4,ROLL_PID,PITCH_PID,yaw_tren4);
    }
    else if(step_seq==2 && z_tren1>=z_targ1){
      step_seq=3;
      z_pocetni1=z_pocetni4=z_tren1;
      j_pocetni1=j_pocetni4=j_tren1;
      i_pocetni1=i_pocetni4=i_tren1;

      z_pocetni2=z_pocetni3=z_tren2;
      j_pocetni2=j_pocetni3=j_tren2;
      i_pocetni2=i_pocetni3=i_tren2;
    }

    if(step_seq==3){//dizemo noge 2 i 3
      j_targ1=j_targ4=0;
      i_targ1=i_targ4=0;

      j_targ2=j_targ3=0;
      i_targ2=i_targ3=0;
      z_targ2=z_targ3=0.7*visina_hoda;
```

```
      yaw_targ2=yaw_targ3=0;

      brzina_j1=(j_targ1-j_pocetni1)/Time;
      brzina_j2=(j_targ2-j_pocetni2)/Time;

      brzina_i1=(i_targ1-i_pocetni1)/Time;
      brzina_i2=(i_targ2-i_pocetni2)/Time;

      brzina_z2=(z_targ2-z_pocetni2)/Time;

      brzina_yaw2=(yaw_targ2-yaw_pocetni2)/Time;
    }
    if(step_seq==3 && z_tren2>z_targ2){
      j_tren1=j_tren4=j_tren1+brzina_j1;
      j_tren2=j_tren3=j_tren2+brzina_j2;

      i_tren1=i_tren4=i_tren1+brzina_i1;
      i_tren2=i_tren3=i_tren2+brzina_i2;

      z_tren2=z_tren3=z_tren2+brzina_z2;

      yaw_tren2=yaw_tren3=yaw_tren2+brzina_yaw2;

      kinematika(1,i_tren1,j_tren1,z_tren1,ROLL_PID,PITCH_PID,yaw_tren1);
      kinematika(4,i_tren4,j_tren4,z_tren4,ROLL_PID,PITCH_PID,yaw_tren4);

      kinematika(2,i_tren2,j_tren2,z_tren2,ROLL_PID,PITCH_PID,yaw_tren2);
      kinematika(3,i_tren3,j_tren3,z_tren3,ROLL_PID,PITCH_PID,yaw_tren3);
    }
    else if(step_seq==3 && z_tren2<=z_targ2){
      step_seq=4;
      z_pocetni2=z_pocetni3=z_tren2;
    }

    if(step_seq==4){//spustamo nogu 2 i 3
      z_targ2=z_targ3=visina_hoda;
      brzina_z2=(z_targ2-z_pocetni2)/Time;
    }
    if(step_seq==4 && z_tren2<z_targ2){
      z_tren2=z_tren3=z_tren2+brzina_z2;

      kinematika(2,i_tren2,j_tren2,z_tren2,ROLL_PID,PITCH_PID,yaw_tren2);
      kinematika(3,i_tren3,j_tren3,z_tren3,ROLL_PID,PITCH_PID,yaw_tren3);

      kinematika(1,i_tren1,j_tren1,z_tren1,ROLL_PID,PITCH_PID,yaw_tren1);
      kinematika(4,i_tren4,j_tren4,z_tren4,ROLL_PID,PITCH_PID,yaw_tren4);
    }
    else if(step_seq==4 && z_tren2>=z_targ2){
      step_seq=0;
      prevAction=0;
      j_pocetni1=j_pocetni2=j_pocetni3=j_pocetni4=j_tren1;
      i_pocetni1=i_pocetni2=i_pocetni3=i_pocetni4=i_tren1;
      z_pocetni1=z_pocetni2=z_pocetni3=z_pocetni4=z_tren1;
      yaw_pocetni1=yaw_pocetni2=yaw_pocetni3=yaw_pocetni4=yaw_tren1;
    }
  }//walk==1 end

  if(walk==0 && prevAction==1 ||walk==0 && prevAction==2){//spustamo sve
noge doli i pamtimo ostale parametre
```

```
    walk_allow=0;//da ne dopusti unos podataka preko joysticka sve dok noge
nisu spustene jer ko ti je kriv što si ispustia joystick
      //bez ovoga zna zaštekat sve dok mu ne dopustim da spusti noge doli
    ROLL_PID=PID(0,KalmanAngleRoll,0);
    PITCH_PID=PID(0,KalmanAnglePitch,1);

    if(step_seq==1 && prevAction==1 || step_seq==2 &&
prevAction==1){//preuzima pozicije i sprema ih
      z_targ1=visina_hoda;

      i_pocetni1=i_pocetni4=i_tren1;
      j_pocetni1=j_pocetni4=j_tren1;
      z_pocetni1=z_pocetni4=z_tren1;
      yaw_pocetni1=yaw_pocetni4=yaw_tren1;

      i_pocetni2=i_pocetni3=i_tren2;
      j_pocetni2=j_pocetni3=j_tren2;
      z_pocetni2=z_pocetni3=z_tren2;
      yaw_pocetni2=yaw_pocetni3=yaw_tren2;
    }
    else if(step_seq==3 && prevAction==1||step_seq==4 && prevAction==1){
      z_targ2=visina_hoda;
      i_pocetni1=i_pocetni4=i_tren1;
      j_pocetni1=j_pocetni4=j_tren1;
      z_pocetni1=z_pocetni4=z_tren1;
      yaw_pocetni1=yaw_pocetni4=yaw_tren1;

      i_pocetni2=i_pocetni3=i_tren2;
      j_pocetni2=j_pocetni3=j_tren2;
      z_pocetni2=z_pocetni3=z_tren2;
      yaw_pocetni2=yaw_pocetni3=yaw_tren2;
    }

    prevAction=2;//ovo da ne ulazi konstantno u prohranu pozicija

    if(step_seq==1  || step_seq==2){
      if(safety_step_seq==0){
        brzina_z1=(z_targ1-z_pocetni1)/Time;
      }
      if(safety_step_seq==0 && z_tren1<z_targ1){
        z_tren1=z_tren4=z_tren1+brzina_z1;

        kinematika(1,i_tren1,j_tren1,z_tren1,ROLL_PID,PITCH_PID,yaw_tren1);
        kinematika(4,i_tren4,j_tren4,z_tren4,ROLL_PID,PITCH_PID,yaw_tren4);

        kinematika(2,i_tren2,j_tren2,z_tren2,ROLL_PID,PITCH_PID,yaw_tren2);
        kinematika(3,i_tren3,j_tren3,z_tren3,ROLL_PID,PITCH_PID,yaw_tren3);
      }
      else if(safety_step_seq==0 && z_tren1>=z_targ1){
        safety_step_seq=1;
        step_seq=0;
        prevAction=0;
        walk_allow=1;
        z_pocetni1=z_pocetni4=z_tren1;
      }
    }

    if(step_seq==3  || step_seq==4){
      if(safety_step_seq==0){
        brzina_z2=(z_targ2-z_pocetni2)/Time;
      }
```

```
      if(safety_step_seq==0 && z_tren2<z_targ2){
        z_tren2=z_tren3=z_tren2+brzina_z2;

        kinematika(2,i_tren2,j_tren2,z_tren2,ROLL_PID,PITCH_PID,yaw_tren2);
        kinematika(3,i_tren3,j_tren3,z_tren3,ROLL_PID,PITCH_PID,yaw_tren3);

        kinematika(1,i_tren1,j_tren1,z_tren1,ROLL_PID,PITCH_PID,yaw_tren1);
        kinematika(4,i_tren4,j_tren4,z_tren4,ROLL_PID,PITCH_PID,yaw_tren4);
      }
      else if(safety_step_seq==0 && z_tren2>=z_targ2){
        safety_step_seq=1;
        step_seq=0;
        prevAction=0;
        walk_allow=1;
        z_pocetni2=z_pocetni3=z_tren2;
      }
    }

    if(step_seq==0 && prevAction==2){// ovo ako nije ni jedna od ovih gori
      prevAction=0;
      walk_allow=1;

      i_pocetni1=i_pocetni4=i_tren1;
      j_pocetni1=j_pocetni4=j_tren1;
      z_pocetni1=z_pocetni4=z_tren1;
      yaw_pocetni1=yaw_pocetni4=yaw_tren1;

      i_pocetni2=i_pocetni3=i_tren2;
      j_pocetni2=j_pocetni3=j_tren2;
      z_pocetni2=z_pocetni3=z_tren2;
      yaw_pocetni2=yaw_pocetni3=yaw_tren2;
    }
  }//END of walk==0 && PrevAction=1
//
//Serial.print("prevAction= ");Serial.println(prevAction);
//Serial.print("walk= ");Serial.println(walk);
//Serial.print("Step_seq= ");Serial.println(step_seq);
//Serial.print("z_tren1= ");Serial.println(z_tren1);

}


float FILTER(float tren,float prev){
  float nova_vrijednost=tren*0.05+prev*0.95;
  return nova_vrijednost;
}


float KutuPuls(int metoda,float kut){
  if(metoda==1){
   int puls = map(kut,0, 180,SERVOMIN,SERVOMAX);// map angle of 0 to 180 to
Servo min and Servo max
   return puls;
   }
  if(metoda==2){
   int puls = map(kut,0, 180, SERVOMAX,SERVOMIN);// map angle of 0 to 180
to Servo min and Servo max
   return puls;
   }
}
```

```
//////////////////////kad koju metodu////
//metoda 1:
//lakat3
//lakat4
//rame3
//rame4
//kuk3
//kuk2
///
//metoda 2:
//lakat1
//lakat2

//rame1
//rame2
//kuk1
//kuk4


float PID(float Referenca, float PovratnaVeza, int KojiNagib){
  //KojiNagib se odnosi na to biramo mi roll ili pitch i ide Roll=0 Pitch=1
  if(prevMODE==0){y=0;}
  if(prevMODE==2){
    K[0]=0;
    K[1]=3.136;
    KD[0]=0;
    KD[1]=0.1725;
  }
  u = Referenca-PovratnaVeza;
  yP=K[KojiNagib]*u;

  yD=KD[KojiNagib]*(u-u_1[KojiNagib]);

  if(abs(u)>0.5){
    yI[KojiNagib]=yI[KojiNagib]+KI[KojiNagib]*u;
  }
  else if(abs(u)<0.5){
    yI[KojiNagib]=0;
  }
  y=yP+yI[KojiNagib]+yD;
  u_1[KojiNagib]=u;
  if(y>YMAX){
    y=YMAX;
    yI[KojiNagib]=y-yP-yD;
  }else if(y<YMIN){
    y=YMIN;
    yI[KojiNagib]=y-yP-yD;
  }

  return y;
}


void Safety_position() {
  pos_x1=127;
  pos_x2=127;
  pos_y1=127;
  pos_y2=127;
  counter_1=33;
  counter_2=129;
  dat.mod=0;
```

```
  MODE(dat.mod);
  }


void kinematika(int noga,float i, float j, float z, float roll, float
pitch, float yaw){
// zelim broj noge, x,y,z pozicije gdje zelim da mi stopalo noge bude u
odnosu na centralnu poziciju
//roll pitch iyaw

//#define uda_yaw 113.1 //mm
#define uda_pitch 113.1
//#define uda_pitch_2 113.1
//#define A_0 53
//#define A_1 49
#define L 100

//pretvorba unešenih kutova u radijane
roll=roll/180*PI;
pitch=pitch/180*PI;
yaw=yaw/180*PI;



//varijable potrebne za racun
//svi kutovi su u radijanima pa kasnije treba pon pretvoriti u stupnjeve
prilikom koristenja funkcije KutuPuls()

float I;
float J;
float hp=z;
//Serial.print("I=");Serial.println(I);
//Serial.print("J=");Serial.println(J);
//Serial.print("hp=");Serial.println(hp);
// varijable za yaw
float X1;
float Y1;
float X2;
float Y2;
float alfa_yaw;
float beta_yaw;
float R;
float i_yaw;
float j_yaw;

//varijable za pitch
float hr;
//float hr1;
//float hr2;

float j_pitch;
//float jl;

//varijable za roll
float h2;
//float C;
//float A;
//float Auk;
//float alfa_r;
//float beta_r;
float i_roll;
```

```
//varijable za translaciju po x osi
float C;
float A;
float h1;
float delta_xz;
float gama_xz;

//varijable za translaciju po y osi
float R1;
float alfa_yz;
float beta_yz;
float gama_yz;

//kutevi motora
float fi_kuk;
float fi_ram;
float fi_lak;

//rotacija oko osi z
if(noga==1){
  X1=102;
  Y1=-113.1;
  R=sqrt(X1*X1+Y1*Y1);
  alfa_yaw=atan(abs(Y1/X1));
  beta_yaw=alfa_yaw-yaw;
  X2=R*cos(beta_yaw);
  Y2=-R*sin(beta_yaw);
  j_yaw=Y2-Y1;
  i_yaw=X2-X1;
}
if(noga==2){
  X1=102;
  Y1=113.1;
  R=sqrt(X1*X1+Y1*Y1);
  alfa_yaw=atan(Y1/X1);
  beta_yaw=alfa_yaw+yaw;
  X2=R*cos(beta_yaw);
  Y2=R*sin(beta_yaw);
  j_yaw=Y2-Y1;
  i_yaw=X2-X1;
}
if(noga==3){
  X1=-102;
  Y1=-113.1;
  R=sqrt(X1*X1+Y1*Y1);
  alfa_yaw=atan(Y1/X1);
  beta_yaw=alfa_yaw+yaw;
  X2=-R*cos(beta_yaw);
  Y2=-R*sin(beta_yaw);
  j_yaw=Y2-Y1;
  i_yaw=X2-X1;
}
if(noga==4){
  X1=-102;
  Y1=113.1;
  R=sqrt(X1*X1+Y1*Y1);
  alfa_yaw=atan(abs(Y1/X1));
  beta_yaw=alfa_yaw-yaw;
  X2=-R*cos(beta_yaw);
  Y2=R*sin(beta_yaw);
```

```
  j_yaw=Y2-Y1;
  i_yaw=X2-X1;
}

//rotacija oko x osi
//prednja strana
if(noga==1||noga==3){
  hr = hp+uda_pitch*sin(pitch);
  j_pitch=-uda_pitch*(1-cos(pitch));
}
//straznja strana
if(noga==2||noga==4){
  hr = hp-uda_pitch*sin(pitch);
  j_pitch=uda_pitch*(1-cos(pitch));
}

//rotacija oko y osi;
//lijeva strana
if(noga==3||noga==4){
  h2=hr-53*sin(roll);
  i_roll=53*(1-cos(roll));
}
//desna strana
if(noga==1||noga==2){
  h2=hr+53*sin(roll);
  i_roll=-53*(1-cos(roll));
}

//translacija nogu u ravnini xz
I=i+i_yaw+i_roll;
//lijeva strana
if(noga==3 || noga==4){
  A=49-I;
  C=sqrt(h2*h2+A*A);
  h1=sqrt(C*C-49*49);
  delta_xz=atan(A/h2);
  gama_xz=atan(h1/49);
}
//desna strana
if(noga==1 || noga==2){
  A=49+I;
  C=sqrt(h2*h2+A*A);
  h1=sqrt(C*C-49*49);
  delta_xz=atan(A/h2);
  gama_xz=atan(h1/49);
}

//traslacija nogu yz ravnina
J=j+j_yaw+j_pitch;
R1=sqrt(h1*h1+J*J);
gama_yz=atan(J/h1);
alfa_yz=acos(1-(0.5*(R1*R1)/(L*L)));
beta_yz=(PI-alfa_yz)/2;

//Serial.print("gama_t=");Serial.print(gama_t);
//Serial.print("r1=");Serial.print(r1);
//Serial.print("alfa_t=");Serial.print(alfa_t);
//Serial.print("beta_t=");Serial.println(beta_t);


fi_kuk=(gama_xz+delta_xz)/PI*180;//kut u stupnjevima
```

```
fi_ram=(beta_yz+gama_yz)/PI*180;
fi_lak=alfa_yz/PI*180;

if(noga==1){

  pwm.setPWM(kuk1,0,KutuPuls(2,fi_kuk));
  pwm.setPWM(rame1,0,KutuPuls(2,fi_ram));
  pwm.setPWM(lakat1,0,KutuPuls(2,fi_lak));
}

if(noga==2){

  pwm.setPWM(kuk2,0,KutuPuls(1,fi_kuk));
  pwm.setPWM(rame2,0,KutuPuls(2,fi_ram));
  pwm.setPWM(lakat2,0,KutuPuls(2,fi_lak));
}

if(noga==3){
  pwm.setPWM(kuk3,0,KutuPuls(1,fi_kuk));
  pwm.setPWM(rame3,0,KutuPuls(1,fi_ram));
  pwm.setPWM(lakat3,0,KutuPuls(1,fi_lak));
}

if(noga==4){
  pwm.setPWM(kuk4,0,KutuPuls(2,fi_kuk));
  pwm.setPWM(rame4,0,KutuPuls(1,fi_ram));
  pwm.setPWM(lakat4,0,KutuPuls(1,fi_lak));
}

}


float mapfloat(long x, long in_min, long in_max, long out_min, long
out_max)
{
  return (float)(x - in_min) * (out_max - out_min) / (float)(in_max -
in_min) + out_min;
}


void MODE(int mod){
  if(mod == 0 && prevMODE == 0){
    i=map(pos_x1,0,255,-50,50);
    i=constrain(i,-50,50);
    i_tren=FILTER(i,i_prev);

    j=map(pos_y1,0,255,-50,50);
    j=constrain(j,-50,50);
    j_tren=FILTER(j,j_prev);

    z=map(counter_2,1,255,80,160);
//    Serial.print("z=");Serial.println(z);
    z=constrain(z,100,180);
    z_tren=FILTER(z,z_prev);

    r=map(counter_1,64,1,-20,20);
    r=constrain(r,-30,30);
    r_tren=FILTER(r,r_prev);

    p=map(pos_x2,0,255,-20,20);
    p=constrain(p,-20,20);
```

```
    p_tren=FILTER(p,p_prev);

    yaw=map(pos_y2,0,255,-20,20);
    yaw=constrain(yaw,-20,20);
    yaw_tren=FILTER(yaw,yaw_prev);

    kinematika(1,i_tren,j_tren,z_tren,r_tren,p_tren,yaw_tren);
    kinematika(4,i_tren,j_tren,z_tren,r_tren,p_tren,yaw_tren);
    kinematika(2,i_tren,j_tren,z_tren,r_tren,p_tren,yaw_tren);
    kinematika(3,i_tren,j_tren,z_tren,r_tren,p_tren,yaw_tren);
//    Serial.print("Roll");Serial.println(r);
    i_prev=i_tren;
    j_prev=j_tren;
    z_prev=z_tren;
    r_prev=r_tren;
    p_prev=p_tren;
    yaw_prev=yaw_tren;

  }//MODE_0 end

  if(mod==0 && prevMODE==2 || mod==0 && prevMODE==1){

    Time=map(dat.pos_x2,0,255,15,45);
    Time=constrain(Time,15,45);

    if(step_seq==0){
      z_targ1=0.7*visina_hoda;
      j_targ1=0;
      i_targ1=0;
      yaw_targ1=0;
      brzina_z1=(z_targ1-z_pocetni1)/Time;
      brzina_j1=(j_targ1-j_pocetni1)/Time;
      brzina_i1=(i_targ1-i_pocetni1)/Time;
      brzina_yaw1=(yaw_targ1-yaw_pocetni1)/Time;
    }
    if(step_seq==0 && z_tren1>z_targ1){//dizemo nogu 1
      z_tren1=z_tren1+brzina_z1;
      j_tren1=j_tren1+brzina_j1;
      i_tren1=i_tren1+brzina_i1;
      yaw_tren1=yaw_tren1+brzina_yaw1;

      kinematika(1,i_tren1,j_tren1,z_tren1,0,0,yaw_tren1);
    }
    else if(step_seq==0 && z_tren1<=z_targ1){
      z_targ1=visina_hoda;

      z_pocetni1=z_tren1;
      j_pocetni1=j_tren1;
      i_pocetni1=i_tren1;
      yaw_pocetni1=yaw_tren1;

      step_seq=1;
    }
    if(step_seq==1){//spustamo nogu 1 doli
      brzina_z1=(z_targ1-z_pocetni1)/Time;
    }
    if(step_seq==1 && z_tren1<z_targ1){
      z_tren1=z_tren1+brzina_z1;

      kinematika(1,i_tren1,j_tren1,z_tren1,0,0,yaw_tren1);
    }
```

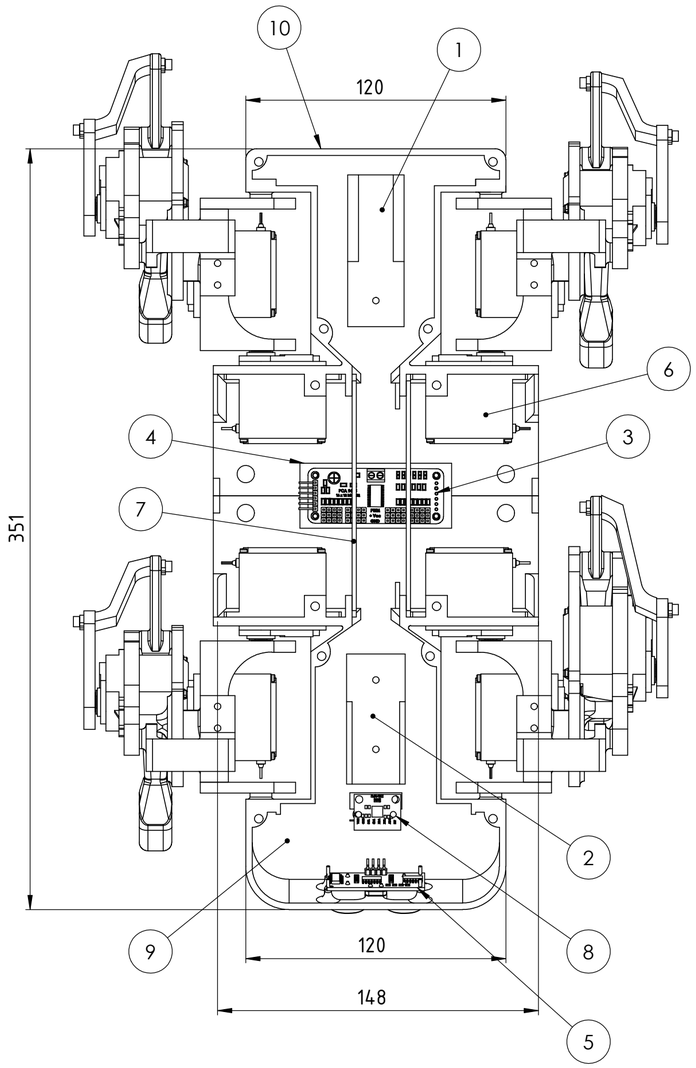
```
  else if(step_seq==1 && z_tren1>=z_targ1){
    z_pocetni1=z_tren1;

    z_targ4=0.7*visina_hoda;
    j_targ4=0;
    i_targ4=0;
    yaw_targ4=0;

    step_seq=2;
  }
  //dizemo nogu 4
  if(step_seq==2){
    brzina_z4=(z_targ4-z_pocetni4)/Time;
    brzina_j4=(j_targ4-j_pocetni4)/Time;
    brzina_i4=(i_targ4-i_pocetni4)/Time;
    brzina_yaw4=(yaw_targ4-yaw_pocetni4)/Time;
  }
  if(step_seq==2 && z_tren4>z_targ4){
    z_tren4=z_tren4+brzina_z4;
    j_tren4=j_tren4+brzina_j4;
    i_tren4=i_tren4+brzina_i4;
    yaw_tren4=yaw_tren4+brzina_yaw4;

    kinematika(4,i_tren4,j_tren4,z_tren4,0,0,yaw_tren4);
  }
  else if(step_seq==2 && z_tren4<=z_targ4){
    z_targ4=visina_hoda;

    z_pocetni4=z_tren4;
    j_pocetni4=j_tren4;
    i_pocetni4=i_tren4;
    yaw_pocetni4=yaw_tren4;

    step_seq=3;
  }
  //spustamo nogu 4 doli
  if(step_seq==3){
    brzina_z4=(z_targ4-z_pocetni4)/Time;
  }
  if(step_seq==3 && z_tren4<z_targ4){
    z_tren4=z_tren4+brzina_z4;

    kinematika(4,i_tren4,j_tren4,z_tren4,0,0,yaw_tren4);
  }
  else if(step_seq==3 && z_tren4>=z_targ4){
    z_pocetni4=z_tren4;

    z_targ2=0.7*visina_hoda;
    j_targ2=0;
    i_targ2=0;
    yaw_targ2=0;

    step_seq=4;
  }
  //dizemo nogu 2
  if(step_seq==4){
    brzina_z2=(z_targ2-z_pocetni2)/Time;
    brzina_j2=(j_targ2-j_pocetni2)/Time;
    brzina_i2=(i_targ2-i_pocetni2)/Time;
    brzina_yaw2=(yaw_targ2-yaw_pocetni2)/Time;
  }
```

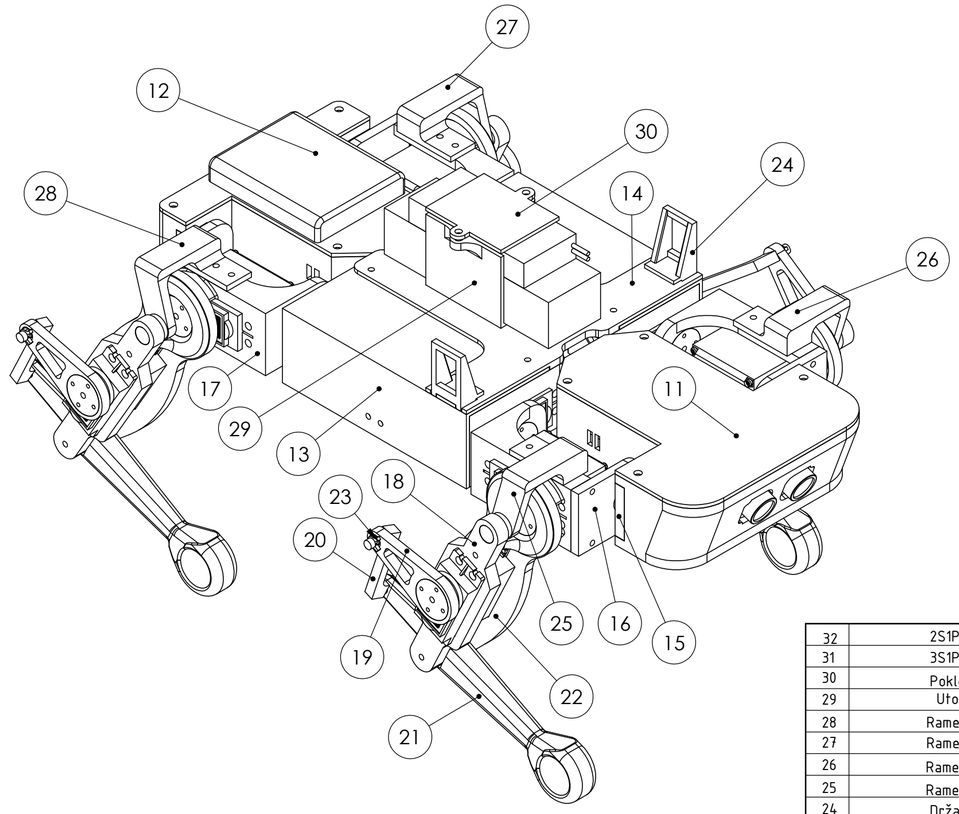
```
    if(step_seq==4 && z_tren2>z_targ2){
      z_tren2=z_tren2+brzina_z2;
      j_tren2=j_tren2+brzina_j2;
      i_tren2=i_tren2+brzina_i2;
      yaw_tren2=yaw_tren2+brzina_yaw2;

      kinematika(2,i_tren2,j_tren2,z_tren2,0,0,yaw_tren2);
    }
    else if(step_seq==4 && z_tren2<=z_targ2){
      z_targ2=visina_hoda;

      z_pocetni2=z_tren2;
      j_pocetni2=j_tren2;
      i_pocetni2=i_tren2;
      yaw_pocetni2=yaw_tren2;

      step_seq=5;
    }
    //spustamo nogu 2 doli
    if(step_seq==5){
      brzina_z2=(z_targ2-z_pocetni2)/Time;
    }
    if(step_seq==5 && z_tren2<z_targ2){
      z_tren2=z_tren2+brzina_z2;

      kinematika(2,i_tren2,j_tren2,z_tren2,0,0,yaw_tren2);
    }
    else if(step_seq==5 && z_tren2>=z_targ2){
      z_pocetni2=z_tren2;

      z_targ3=0.7*visina_hoda;
      j_targ3=0;
      i_targ3=0;
      yaw_targ3=0;

      step_seq=6;
    }
    //dizemo nogu 3
    if(step_seq==6){
      brzina_z3=(z_targ3-z_pocetni3)/Time;
      brzina_j3=(j_targ3-j_pocetni3)/Time;
      brzina_i3=(i_targ3-i_pocetni3)/Time;
      brzina_yaw3=(yaw_targ3-yaw_pocetni3)/Time;
    }
    if(step_seq==6 && z_tren3>z_targ3){
      z_tren3=z_tren3+brzina_z3;
      j_tren3=j_tren3+brzina_j3;
      i_tren3=i_tren3+brzina_i3;
      yaw_tren3=yaw_tren3+brzina_yaw3;

      kinematika(3,i_tren3,j_tren3,z_tren3,0,0,yaw_tren3);
    }
    else if(step_seq==6 && z_tren3<=z_targ3){
      z_targ3=visina_hoda;

      z_pocetni3=z_tren3;
      j_pocetni3=j_tren3;
      i_pocetni3=i_tren3;
      yaw_pocetni3=yaw_tren3;

      step_seq=7;
```

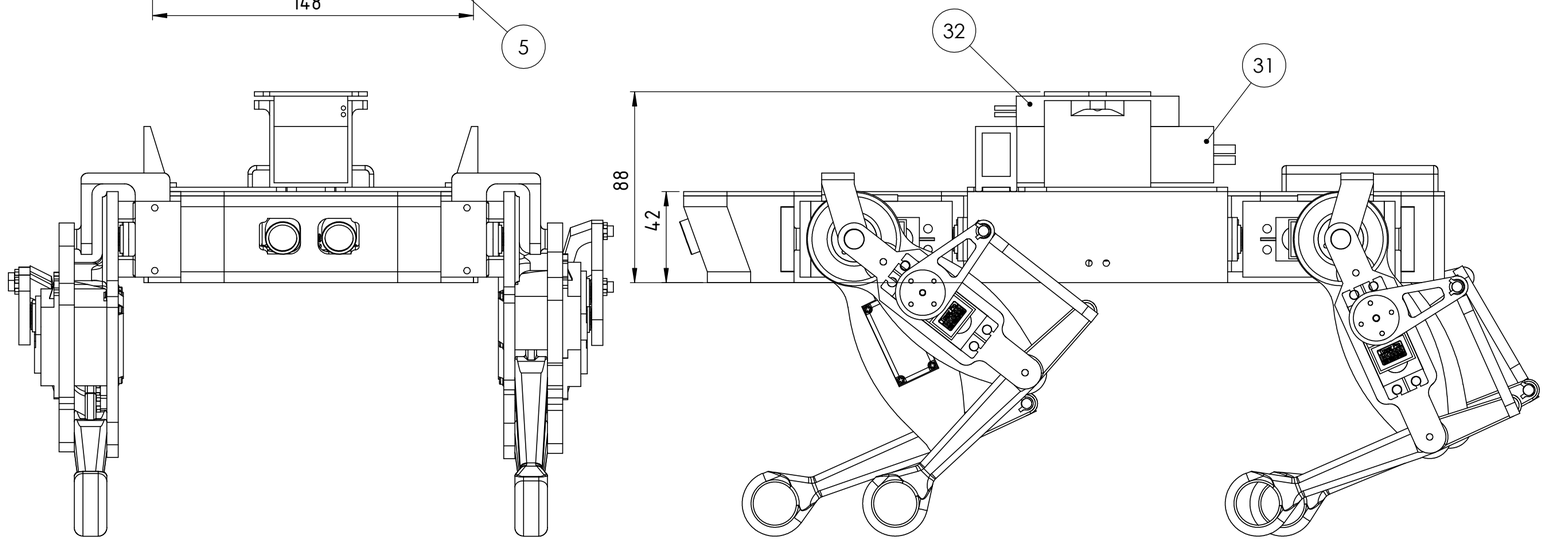
```
    }
    //spustamo nogu 3 doli
    if(step_seq==7){
      brzina_z3=(z_targ3-z_pocetni3)/Time;
    }
    if(step_seq==7 && z_tren3<z_targ3){
      z_tren3=z_tren3+brzina_z3;

      kinematika(3,i_tren3,j_tren3,z_tren3,0,0,yaw_tren3);
    }
    else if(step_seq==7 && z_tren3>=z_targ3){
      z_pocetni3=z_tren3;

      step_seq=8;
    }

    //sada moramo unjeti prijašne iznose da bi filter moga raditi
    if(step_seq==8){
      i_prev=i_tren1;
      j_prev=j_tren1;
      z_prev=z_tren1;

      r_prev=0;
      p_prev=0;
      yaw_prev=0;

      step_seq=0;
      prevMODE=0;
    }

  }//MODE_0 end

  if(mod==2 && prevMODE==0){
    j_pocetni1=j_pocetni2=j_pocetni3=j_pocetni4=j_tren;
    i_pocetni1=i_pocetni2=i_pocetni3=i_pocetni4=i_tren;
    z_pocetni1=z_pocetni2=z_pocetni3=z_pocetni4=z_tren;
    yaw_pocetni1=yaw_pocetni2=yaw_pocetni3=yaw_pocetni4=yaw_tren;

    j_tren1=j_tren2=j_tren3=j_tren4=j_tren;
    i_tren1=i_tren2=i_tren3=i_tren4=i_tren;
    z_tren1=z_tren2=z_tren3=z_tren4=z_tren;
    yaw_tren1=yaw_tren2=yaw_tren3=yaw_tren4=yaw_tren;

    prevMODE=2;
    step_seq=0;
    walk=0;
    walk_allow=1;
    prevAction=0;
  }
  else if(mod==2 && prevMODE==1){
    j_pocetni1=j_pocetni4=j_tren1;
    j_pocetni2=j_pocetni3=j_tren2;

    i_pocetni1=i_pocetni4=i_tren1;
    i_pocetni2=i_pocetni3=i_tren2;

    z_pocetni1=z_pocetni4=z_tren1;
    z_pocetni2=z_pocetni3=z_tren2;
    yaw_pocetni1=yaw_pocetni4=yaw_tren1;
    yaw_pocetni2=yaw_pocetni3=yaw_tren2;
```

```
    prevMODE=2;
    step_seq=0;
    walk=0;
    walk_allow=1;
    prevAction=0;
  }

  if(mod==2 && prevMODE==2){
    walk=0;




    if(dat.pos_x1>175 && walk_allow==1){walk=1; Direction_i=1;}
    else if(dat.pos_x1<80 && walk_allow==1) {walk=1; Direction_i=-1;}
    else{Direction_i=0;}

    if(digitalRead(Signal_udaljenosti)==LOW){//kada je nisko nema objekta
      if(dat.pos_y1>175&& walk_allow==1){walk=1; Direction_j=-1;}
      else if(dat.pos_y1<105 && walk_allow==1){walk=1; Direction_j=1;}
      else{Direction_j=0;}
    }
    else{
      if(dat.pos_y1<105 && walk_allow==1){walk=1; Direction_j=1;}
      else{Direction_j=0;}
    }

    if(dat.pos_y2>240 && walk_allow==1){walk=1;Direction_rotation=1;}
    else if(dat.pos_y2<10 && walk_allow==1){walk=1;Direction_rotation=-1;}
    else{Direction_rotation=0;}

//    Time=map(dat.pos_x2,0,255,15,40);
//    Time=constrain(Time,15,40);
    Time=map(dat.counter_1,1,64,40,15);
    Time=constrain(Time,12,40);


    visina_hoda=z_tren;

    Algoritam_setanja();
  }//END of mode2 i premode=2

  if(mod==1 && prevMODE==0){

    j_pocetni1=j_pocetni2=j_pocetni3=j_pocetni4=j_tren;
    i_pocetni1=i_pocetni2=i_pocetni3=i_pocetni4=i_tren;
    z_pocetni1=z_pocetni2=z_pocetni3=z_pocetni4=z_tren;
    yaw_pocetni1=yaw_pocetni2=yaw_pocetni3=yaw_pocetni4=yaw_tren;

    j_tren1=j_tren2=j_tren3=j_tren4=j_tren;
    i_tren1=i_tren2=i_tren3=i_tren4=i_tren;
    z_tren1=z_tren2=z_tren3=z_tren4=z_tren;
    yaw_tren1=yaw_tren2=yaw_tren3=yaw_tren4=yaw_tren;

    prevMODE=1;
    step_seq=0;
    walk=0;
    walk_allow=1;
    prevAction=0;
  }
  else if(mod==1 && prevMODE==2){
```

```
    j_pocetni1=j_pocetni4=j_tren1;
    j_pocetni2=j_pocetni3=j_tren2;

    i_pocetni1=i_pocetni4=i_tren1;
    i_pocetni2=i_pocetni3=i_tren2;

    z_pocetni1=z_pocetni4=z_tren1;
    z_pocetni2=z_pocetni3=z_tren2;
    yaw_pocetni1=yaw_pocetni4=yaw_tren1;
    yaw_pocetni2=yaw_pocetni3=yaw_tren2;

    prevMODE=1;
    step_seq=0;
    walk=0;
    walk_allow=1;
    prevAction=0;
  }
  else if(mod==1 && prevMODE==1){
    walk=0;

    if(dat.pos_x1>175 && walk_allow==1){walk=1; Direction_i=1;}
    else if(dat.pos_x1<80 && walk_allow==1) {walk=1; Direction_i=-1;}
    else{Direction_i=0;}

    if(digitalRead(Signal_udaljenosti)==LOW){//kada je nisko nema objekta
      if(dat.pos_y1>175&& walk_allow==1){walk=1; Direction_j=-1;}
      else if(dat.pos_y1<105 && walk_allow==1){walk=1; Direction_j=1;}
      else{Direction_j=0;}
    }
    else{
      if(dat.pos_y1<105 && walk_allow==1){walk=1; Direction_j=1;}
      else{Direction_j=0;}
    }

    if(dat.pos_y2>240 && walk_allow==1){walk=1;Direction_rotation=1;}
    else if(dat.pos_y2<10 && walk_allow==1){walk=1;Direction_rotation=-1;}
    else{Direction_rotation=0;}

    Time=map(dat.pos_x2,0,255,15,40);
    Time=constrain(Time,15,40);
    visina_hoda=z_tren;


    K[1]=mapfloat(counter_2,1,256,0.0,10.0);//testiram samo za pitch
    K[1]=constrain(K[1],0,10);
    K[0]=mapfloat(counter_1,1,256,0.0,10.0);//mijenjamo pojačanje za roll
    K[0]=constrain(K[0],0,10);




    Algoritam_setanja();
  }
}// END of function MODE
```







Poz.	Naziv dijela	Kom.	Crtež broj Norma	Materijal	Sirove dimenzije Proizvođač	Masa
32	2S1P LiPo baterija	1				
31	3S1P LiPo baterija	1				
30	Poklopac za baterije	1				
29	Utor za baterije	1				
28	Ramenska osovnia 2	1				
27	Ramenska osovnia 4	1				
26	Ramenska osovnia 3	1				
25	Ramenska osovnia 1	1				
24	Držač prekidača	2				
23	Uskočnik	8				
22	Noga dio 1	4				
21	Noga dio 3	4				
20	Poluga 2	4				
19	Poluga 1	4				
18	Noga dio 2	4				
17	Kuk	4				
16	Ploča kuka	4				
15	Nosač kuka	4				
14	Središnji poklopac	1				
13	Bočni pokrovna ploča	2				
12	Poklopac stražnji	1				
11	Poklopac prednji	1				
10	Straznji dio trupa	1				
9	Prednji dio trupa	1				
8	MPU6050	1				
7	Greda	2				
6	MG995	12				
5	HC-SR04	1				
4	Nosač PCA9685	1				
3	PCA9685	1				
2	CRODUINO CORE	1				
1	ESP32	1				

Broj naziva - code: 0035230730

	Datum	Ime i prezime	Potpis
Projektirao	14.2.2024.	Ivan Paleka	
Razradio	14.2.2024.	Ivan Paleka	
Crtao	14.2.2024.	Ivan Paleka	
Pregledao		Mladen Crnković	

FSB Zagreb
Katedra za strojarsku automatiku

ISO - tolerancije

Objekt:

Objekt broj:

R. N. broj:

Napomena:

Kopija

Materijal:

Masa:

Mjerilo originala: 1:2

Naziv: Sklop robota

Pozicija: 0

Format: A2

Listova: 1

Crtež broj: 0

List: 1

Design by CADLab