Samobalansirajuće vozilo

Herceg-Rušec, Matija

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:545792

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-23**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Matija Herceg-Rušec

Zagreb, 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Mladen Crneković, dipl. ing.

Student:

Matija Herceg-Rušec

Zagreb, 2020.

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

Zahvaljujem se profesoru Mladenu Crnekoviću na mentorstvu te pruženoj pomoći i mnogobrojnim savjetima koji su mi uvelike pomogli u izradi ovog završnog rada.

Također, zahvaljujem se obitelji i djevojci na potpori i strpljenju tijekom cijelog studija.

Matija Herceg-Rušec

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne 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:	
Ur.broj:	

ZAVRŠNI ZADATAK

Student: **MATIJA HERCEG-RUŠEC** Mat. br.: 0035208477

Naslov rada na hrvatskom jeziku: **SAMOBALANSIRAJUĆE VOZILO**

Naslov rada na engleskom jeziku: **SELFBALANCING VEHICLE**

Opis zadatka:

Samobalansirajuće vozilo predstavlja jedan od ideala minimalističke konstrukcije mehaničkog uređaja. Posljedica toga je nestabilan sustav pa je zadatak upravljačke jedinice stabilizacija sustava uz ostvarenje funkcije vožnje određenom brzinom u zadanom smjeru. To je moguće upotrebom 3-osnog akcelerometra i 3-osnog žiroskopa, mikrokontrolera i teorije vođenja sustava.

U radu je potrebno:

- definirati mehaničku strukturu samobalansirajućeg vozila
- odabrati motore, senzore i upravljački mikrokontroler
- izvesti matematički model sustava
- procijeniti vrijednost predloženog rješenja

Potrebno je navesti korištenu literaturu i ostale izvore informacija te eventualno dobivenu pomoć.

Zadatak zadan:

28. studenog 2019.

Zadatak zadao:

Prof.dr.sc. Mladen Crneković

Datum predaje rada:
1. rok: 21. veljače 2020.
2. rok (izvanredni): 1. srpnja 2020.
3. rok: 17. rujna 2020.

Predviđeni datumi obrane:
1. rok: 24.2. – 28.2.2020.
2. rok (izvanredni): 3.7.2020.
3. rok: 21.9. - 25.9.2020.

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

SADRŽAJ

SADRŽAJ I
POPIS SLIKA III
POPIS TABLICA IV
POPIS OZNAKA V
SAŽETAK VI
SUMMARY VII
1. UVOD 1
2. PRINCIP RADA SAMOBALANSIRAJUĆEG VOZILA 2
3. ODABIR AKTIVNIH KOMPONENTI 3
3.1. Motori 3
3.3. Senzori 5
3.4. Mikrokontroler 6
4. MEHANIČKA IZVEDBA ROBOTA 7
5. MATEMATIČKI MODEL SUSTAVA 9
5.1. Dinamika kotača 10
5.2. Dinamika tijela robota 11
5.3. Kombinacija jednadžbi 12
5.4. Prostor stanja 15
5.5. Dopuna jednadžbi za koračni motor 18
6. SIMULACIJA 20
6.1. Simulacija jednadžbi gibanja 20
6.2. PID regulacija 23
7. PROGRAM 26
7.1. Obrada signala iz senzora 26
7.2. Upravljanje koračnim motorima 27
7.3. PID regulacija 27
7.4. Osiguranje gašenja motora prilikom pada 28
8. MOBILNA APLIKACIJA ZA UPRAVLJANJE 29

9. TESTIRANJE I REZULTATI 32
10. PROCJENA VRIJEDNOSTI 34
11. ZAKLJUČAK 35
LITERATURA 36
PRILOZI 37

POPIS SLIKA

Slika 1. Nema17 koračni motor 3
Slika 2. Prikaz ovisnosti momenta o brzini vrtnje za koračni motor Nema17 4
Slika 3. Driver koračnih motora TB6600 5
Slika 4. Troosni akcelerometar i troosni žiroskop MPU6050 5
Slika 5. Mikrokontroler ESP32 6
Slika 6. Unutrašnjost samobalansirajućeg robota 7
Slika 7. Finalni izgled samobalansirajućeg robota 8
Slika 8. Sile i momenti na kotaču i tijelu robota 9
Slika 9. Sile i momenti na kotaču 10
Slika 10. Sile i momenti na tijelu robota 11
Slika 11. Odziv nereguliranog lineariziranog sustava na momentnu step pobudu 22
Slika 12. Odziv nereguliranog lineariziranog sustava na momentnu sinusnu pobudu 22
Slika 13. Blok shema PID regulatora 23
Slika 14. Odziv reguliranog sustava na sinusnu pobudu 25
Slika 15. QR kod za preuzimanje aplikacije 29
Slika 16. Finalni izgled aplikacije 30
Slika 17. Nagib izračunat samo akcelerometrom 32
Slika 18. Prikaz razlike u nefiltriranom (plavo) i filtriranom (crveno) signalu 32
Slika 19. Prikaz stabilizacije sustava nakon udarca 33
Slika 20. Prikaz nagiba prilikom promjene smjera kretanja robota 33

POPIS TABLICA

Tablica 1. Okvirne cijene komponenata robota..... 34

POPIS OZNAKA

Oznaka	Jedinica	Opis
m_K	kg	Masa kotača
m_T	kg	Masa robota svedena u težište
F_x	N	Sila u smjeru osi x
F_y	N	Sila u smjeru osi y
F_{tr}	N	Sila trenja između kotača i podloge
F_n	N	Sila dodira kotača i podloge
g	m/s^2	Ubrzanje sile teže
$x_1, \dot{x}_1, \ddot{x}_1$	m, m/s, m/s^2	Pozicija, brzina, ubrzanje po osi x_1
$x_2, \dot{x}_2, \ddot{x}_2$	m, m/s, m/s^2	Pozicija, brzina, ubrzanje po osi x_2
$y_1, \dot{y}_1, \ddot{y}_1$	m, m/s, m/s^2	Pozicija, brzina, ubrzanje po osi y_1
$y_2, \dot{y}_2, \ddot{y}_2$	m, m/s, m/s^2	Pozicija, brzina, ubrzanje po osi y_2
$\varphi, \dot{\varphi}, \ddot{\varphi}$	rad,rad/s,rad/s^2	Kut, kutna brzina, kutno ubrzanje tijela robota
$\varphi_K, \dot{\varphi}_K, \ddot{\varphi}_K$	rad,rad/s,rad/s^2	Kut, kutna brzina, kutno ubrzanje kotača
J_K	kgm^2	Moment inercije kotača, vratila i motora
J_T	kgm^2	Moment inercije tijela robota
M_K	Nm	Moment na kotaču
R_K, D_K	m	Radijus i promjer kotača
l	m	Udaljenost težišta od ishodišta k.s.

SAŽETAK

Ovaj završni rad prikazuje način rada i postupak izrade samobalansirajućeg robota na dva kotača. Prikazan je način odabira aktuatora i upravljačkih komponenti te sama konstrukcija samobalansirajućeg robota. Nakon odabira komponenata i konstrukcijskog rješenja, pojašnjeno je kako stvoriti matematički model zadanog sustava te ga simulirati u programskom paketu MATLAB. Nakon što je simulacijom dokazano da se sustav može ustabiliti korištenjem odabranih komponenata, napisan je program koji omogućuje stabilnost robota. Na kraju, prikazan je postupak izrade mobilne aplikacije koja omogućuje dodatno upravljanje robotom.

Tokom izrade rada korišteni su programski paketi SolidWorks, MATLAB, Arduino i Android Studio.

Ključne riječi: samobalansirajući robot, robot na dva kotača, dinamički sustav

SUMMARY

This final paper shows the procedure of making a self-balancing two-wheeled robot and explains the way this robot works. The methods of selecting actuators, control components and the construction of the self-balancing robot are presented. After selecting the components and the design solution, it was explained how to create a mathematical model of the given system and simulate it in the MATLAB software. After the simulation proved that the system can be stabilized by using the selected components, a program was written to provide stability for the robot. Finally, the process of creating a mobile application that allows additional robot control is presented. SolidWorks, MATLAB, Arduino and Android Studio software were used while working on the project.

Key words: self-balancing robot, two-wheeled robot, dynamic system

1. UVOD

Samobalansirajući robot predstavlja jednu od najzanimljivijih ideja u robotici - napraviti prirodno nestabilan sustav, u ovom slučaju obrnuto njihalo, stabilnim uz pomoć elektronike. Ljudsko tijelo se također može gledati kao obrnuto njihalo, gdje nesvjesno reakcijom u mišićima održavamo ravnotežu. Teško je opisati zašto se naginjemo u nekom smjeru i za koji iznos, jer to radimo prirodno na temelju iskustva. Međutim, kada je potrebno napraviti neki umjetan nestabilan sustav stabilnim, javljaju se mnogi problemi. Teško je na temelju iskustva reći za neki nepoznati sustav kolikom silom ili momentom treba djelovati u kojem zglobu da bi se postigla ravnoteža. Da bi se riješio problem stabilizacije takvog sustava, potrebna je kombinacija znanja iz senzorike, programiranja mikrokontrolera i teorije vođenja sustava. Zbog toga, izrada samobalansirajućeg robota idealan je projekt za studente mehatronike i robotike, jer ujedinjuje sva znanja stečena tijekom studija.

Za sada, primarna upotreba samobalansirajućih robota je u prijevozu ljudi, koju je popularizirao razvoj vozila Segway. Također, samobalansirajuće vozilo na dva kotača može biti korišteno u tvornicama za transport predmeta. U usporedbi s vozilima na četiri kotača, samobalansirajuće vozilo zauzima manje prostora te može raditi oštre okrete, a samim time kretati se u užim prostorima. Za izradu je korišteno manje materijala i manje aktuatora uz kompliciraniji algoritam za kontroliranje. Također, vidjeti robote na dva kotača kako obavljaju klasične zadatke je oku ugodno te pokazuje evoluciju u robotici.

2. PRINCIP RADA SAMOBALANSIRAJUĆEG VOZILA

Obrnuto njihalo se obično može naći u tri najčešća oblika: samobalansirajući robot, obrnuto njihalo na kolicima i obrnuto njihalo na linearnoj tračnici. Problem regulacije samo-balansirajućeg robota se tako može poistovjetiti s regulacijom obrnutog njihala. Robot se sastoji od limene konstrukcije, kotača, troosnog akcelerometra i troosnog žiroskopa, motora, upravljačkih sklopova za motore, mikrokontrolera, baterije i modula za napajanje.

Troosni akcelerometar i troosni žiroskop daju mikrokontroleru informaciju o trenutnom nagibu robota. Ta informacija je praktično neupotrebljiva zbog šuma i izrazite oscilatorne naravi akcelerometra, pa se treba primijeniti filtar. Pomoću komplementarnog filtra kombinira se informacija iz akcelerometra koja je točna (nema drift), ali jako oscilatorna s informacijom iz žiroskopa koja nije oscilatorna, ali ima drift. Takvom kombinacijom dobiva se gotovo idealan podatak o poziciji robota. S tako obrađenom informacijom iz senzora, mikrokontroler može stabilno regulirati sustav, bez neželjenih oscilacija.

Mikrokontroler na temelju informacije o nagibu robota provodi PID regulaciju sustava. Ulaz u regulator je nagib robota a izlaz iz PID regulatora je potrebna brzina u motorima robota kako bi se robot ispravio i postigao željeni nagib od nula stupnjeva.

Željena brzina koju odredi PID regulator se obrađuje i šalje na upravljačku jedinicu motora, gdje upravljačka jedinica na temelju dobivene informacije upravlja motorom.

Kako bi cijeli sustav radio, mora se podesiti nekoliko parametara. Vrlo je važno odrediti pravilnu brzinu uzorkovanja senzora nagiba, jer s premalo informacija u jedinici vremena sustav će imati zakašnjeli odziv, a s previše informacija u jedinici vremena će se procesor preopteretiti, te će također kasniti u obradi podataka. Također, važno je dobro namjestiti proporcionalnu, integracijsku i derivacijsku konstantu PID regulatora, kako bi se dobio čim stabilniji odziv robota. Proporcionalna komponenta direktno određuje koliko brzo će se motori vrtjeti u ovisnosti o nagibu robota, dok derivacijska i integracijska komponenta pokušavaju anulirati grešku u nagibu s obzirom na prošla i buduća mjerenja dodatnim povećanjem ili smanjivanjem brzine vrtnje motora.

3. ODABIR AKTIVNIH KOMPONENTI

Kao što je nabrojeno u prošlom poglavlju, samobalansirajući robot se sastoji od nekoliko aktivnih komponenti potrebnih za njegov rad. U ovom poglavlju će biti detaljnije opisane tehničke karakteristike svake od njih, zajedno s razlogom odabira istih.

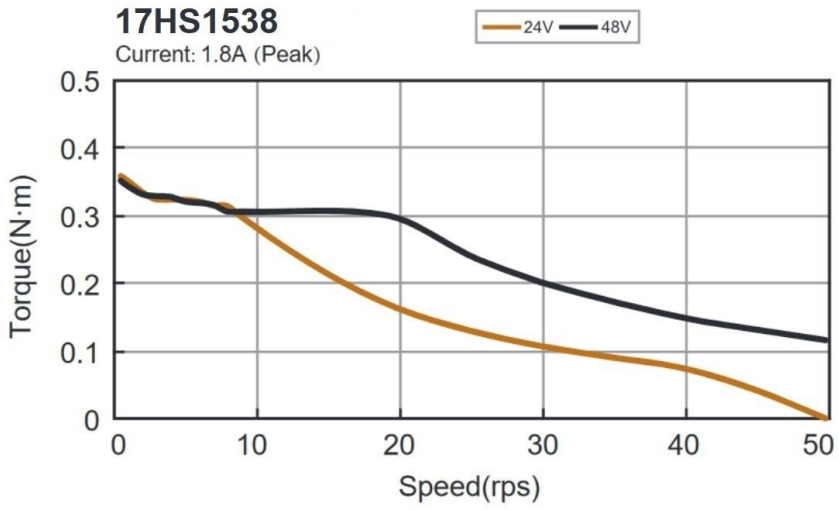
3.1. Motori

Za pogon su korišteni koračni motori Nema17. Koračni motori se jednostavno mogu upravljati u otvorenoj petlji, što olakšava samu izvedbu. Također, jedan od razloga odabira je taj što se koračnim motorima može vrlo jednostavno odrediti brzina vrtnje, koja je direktno povezana s horizontalnom brzinom na dnu inverznog njihala.

Slika 1. Nema17 koračni motor

Koračni motori rade na principu paljenja i gašenja susjednih zavojnica koje rezultiraju pomakom magnetiziranog rotora. Za rad im je potreban driver koji pretvara impulse s mikrokontrolera u potrebne strujne signale na zavojnicama motora. Koračni motor je osjetljiv na vibracije, te zbog toga ne može ispravno raditi na malom broju okretaja, pa treba voditi računa da se motor čim manje dovodi u to područje rada.

Kako koračni motor radi na principu promjena napona na zavojnicama, postoji granica do koje može pouzdano raditi. Pri velikim brzinama vrtnje impulsi za promjenu signala na zavojnicama se mijenjaju jako brzo, a time motor gubi sposobnost ostvarivanja momenta.

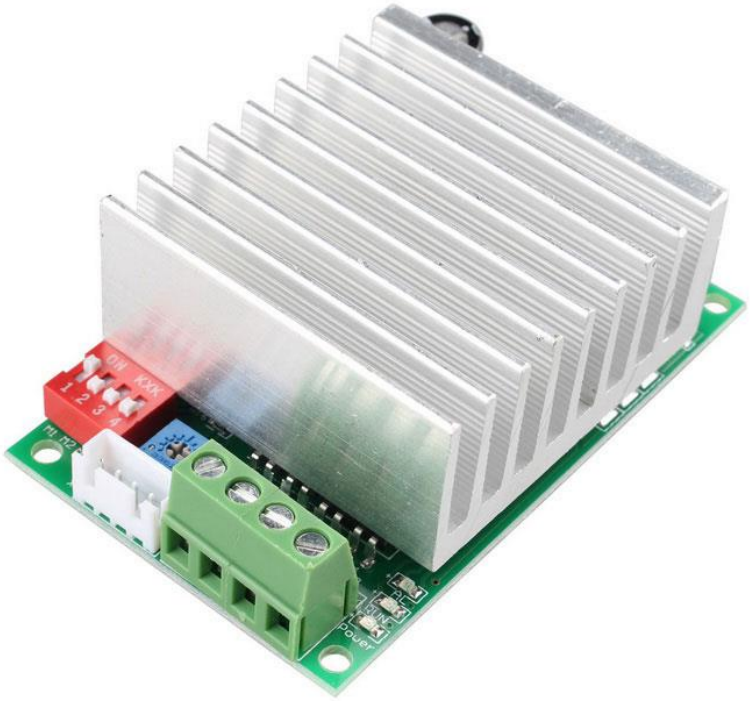


Slika 2. Prikaz ovisnosti momenta o brzini vrtnje za koračni motor Nema17

Ako se prekorači dopušteno opterećenje motora, motor ne prima nikakvu štetu, no događa se preskakanje koraka. Ako se zbog prevelikog momenta javi više preskočenih koraka, motor ispada iz faze, te zbog toga ne daje nikakav moment na izlazu sve dok se sustav ne zaustavi i koračni motor ponovo dovede u fazu rada.

3.2. Driver za motore

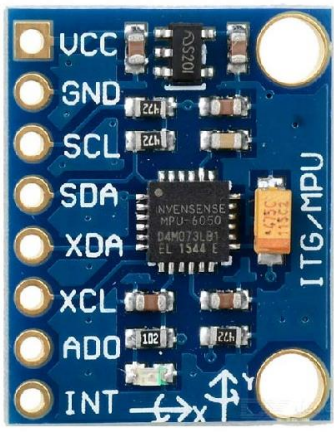
Za upravljanje koračnim motorom, kao što je već prije navedeno, potreban je driver. Pomoću mikrokontrolera na driver se šalju impulsi koje driver pretvara u strujne signale na zavojnicama motora. Driver omogućuje jednostavnu kontrolu motora pomoću samo dvije žice. Jedna žica prenosi impulse, a druga određuje smjer vrtnje motora. Na pločici drivera moguće je podesiti dopuštenu struju na zavojnicama motora, te pomoću sklopke podesiti opciju microsteppinga. Microstepping omogućuje dodatnu preciznost motora, kombinacijom uključivanja obiju zavojnica motora. Odabrani driver je TB6600.



Slika 3. Driver koračnih motora TB6600

3.3. Senzori

MPU6050 je troosni akcelerometar i troosni žiroskop, za hobi primjenu. Na mikrokontroler šalje informacije akcelerometra i žiroskopa koje je naknadno potrebno filtrirati i preračunati u stupnjeve, kako bi informacija bila čim jasnija. U radu s ovim senzorom, uspješnim se pokazao komplementarni filtar koji kombinira informacije akcelerometra i žiroskopa, uz niske zahtjeve procesiranja. Bez senzora takve vrste, regulacija samobalansirajućeg robota ne bi bila moguća.

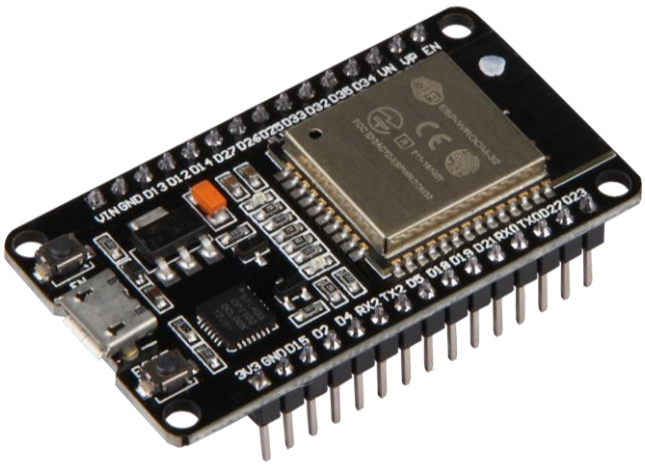


Slika 4. Troosni akcelerometar i troosni žiroskop MPU6050

3.4. Mikrokontroler

Centralni dio cijelog samobalansirajućeg robota je mikrokontroler. On vrši svu obradu podataka mjerenja, izračun PID regulacije, generira impulse za upravljanje koračnim motorom i vrši bluetooth komunikaciju s pametnim telefonima. Odabrana pločica ESP32 ima brzi takt procesora od 240 MHz, koji pomaže pri brzom očitavanju senzora i obradi podataka u realnom vremenu koje je potrebno za regulaciju dinamičkog sustava kao što je samobalansirajuće vozilo. Također, odabrani mikrokontroler je dvojezgreni te na sebi ima integriranu bluetooth komunikaciju. Dvije jezgre su korisne, jer se tako mogu razdvojiti procesi na mikrokontroleru. Na jednu jezgru je moguće podesiti samo rad motora, kako bi motori radili glatko, a na drugu jezgru je moguće primijeniti proračun nagiba robota.

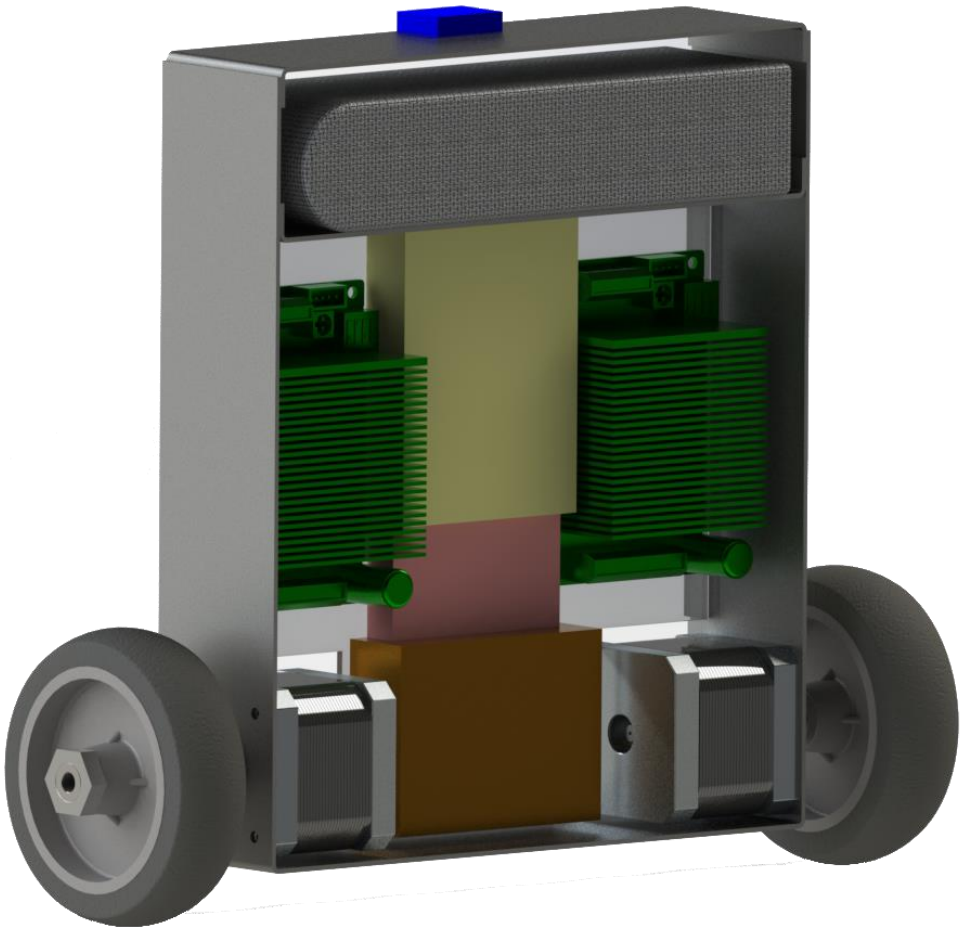
Treba pripaziti na napon na mikrokontroleru, jer zbog efikasnosti odabrani ESP32 radi na 3,3V, te ne može raditi s nekim komponentama koje zahtijevaju 5V napajanje. Mikrokontroler ESP32 može se također svrstati u hobi komponente, no zbog svoje velike procesorske snage, već ugrađenim bluetooth i WiFi modulom te mnogo ADC i DAC pinova može se koristiti u širokoj primjeni u raznim uređajima.



Slika 5. Mikrokontroler ESP32

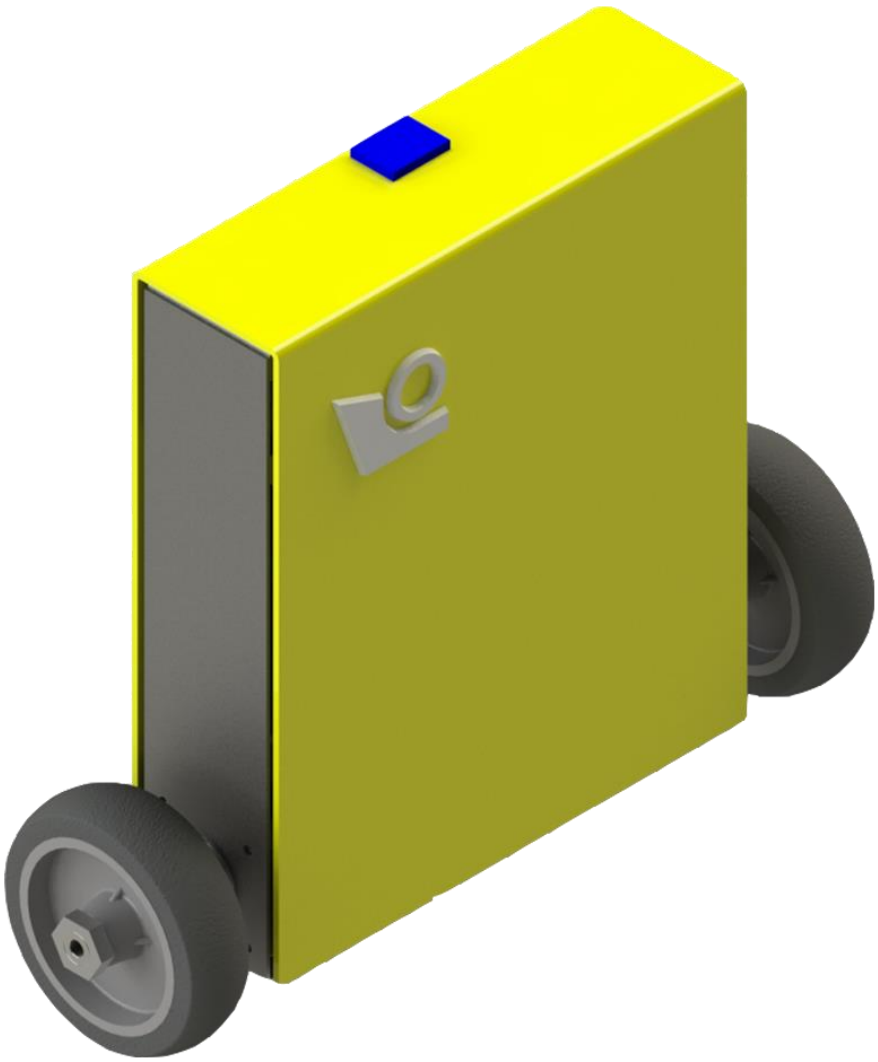
4. MEHANIČKA IZVEDBA ROBOTA

Samobalansirajući robot je mehanički izveden kao inverzno njihalo. Sastoji se od dva osnovna dijela: tijela robota i kotača. Kotači ispravljaju tijelo robota tako da se ono uvijek nalazi u vertikalnoj poziciji. U tijelu robota nalaze se svi ostali dijelovi. Motori su smješteni na dnu tijela robota, te su kotači direktno učvršćeni na osovine motora. Iznad svakog motora smješten je driver za njegovo upravljanje. Na sredini se nalaze mikrokontroler i sklop za stabilizaciju napona s 12V na 5V. Baterija je smještena na sam vrh tijela robota, te zbog svoje mase povećava inertnost robota. Veća inertnost na vrhu robota znači da će robotu biti potrebno više vremena da promijeni nagib, što olakšava regulaciju robota. No s druge strane, potreban je veći moment na kotačima da vrati sustav u ravnotežu, pa treba voditi računa o maksimalnom dopuštenom momentu motora, da ne dođe do preskakanja koraka.



Slika 6. Unutrašnjost samobalansirajućeg robota

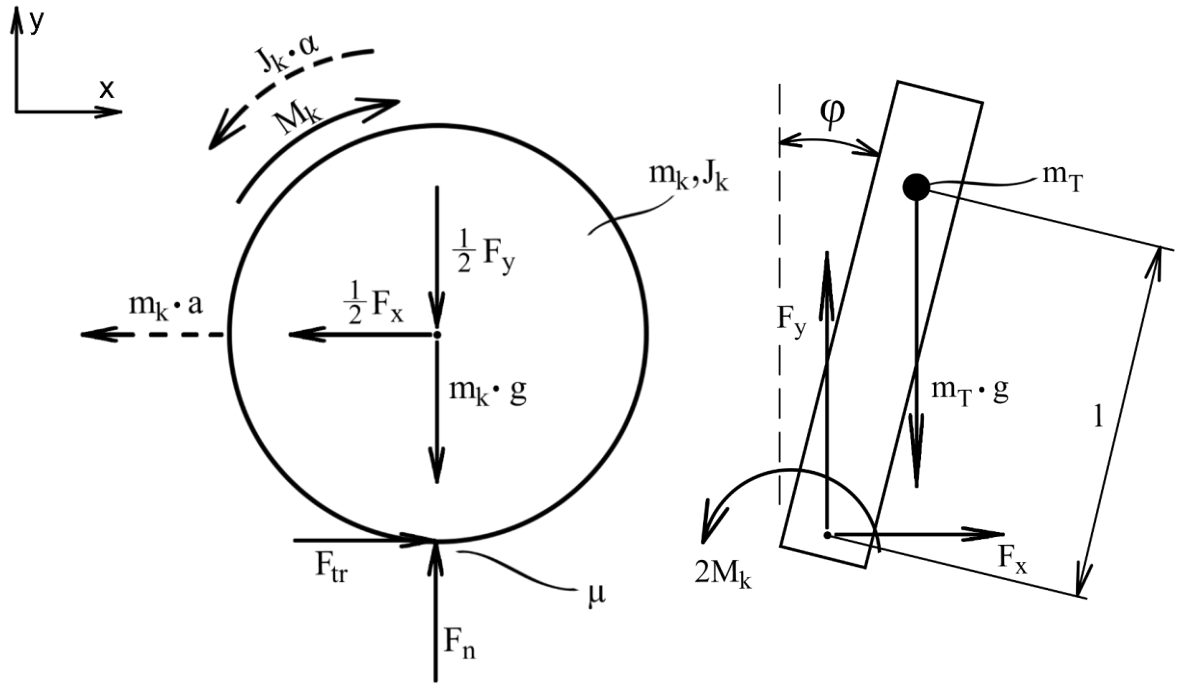
Troosni akcelerometar i troosni žiroskop je također smješten na samom vrhu konstrukcije robota. Razlog tome je što je akcelerometar jako osjetljive naravi. Ako bi se smjestio niže, vibracije koračnog motora bi imale veći utjecaj na čitanja, što nikako nije pogodno. Akcelerometar bi idealno bilo smjestiti blizu neke inertne mase, gdje bi sve vibracije bile prigušene. Zbog toga je pozicija akcelerometra u ovom robotu iznad baterije, koja je glavna inercijska masa u sustavu. Na kraju je izrađen poklopac za zaštitu komponenata prilikom pada, a također pridonosi ukupnom izgledu cijelog robota.



Slika 7. Finalni izgled samobalansirajućeg robota

5. MATEMATIČKI MODEL SUSTAVA

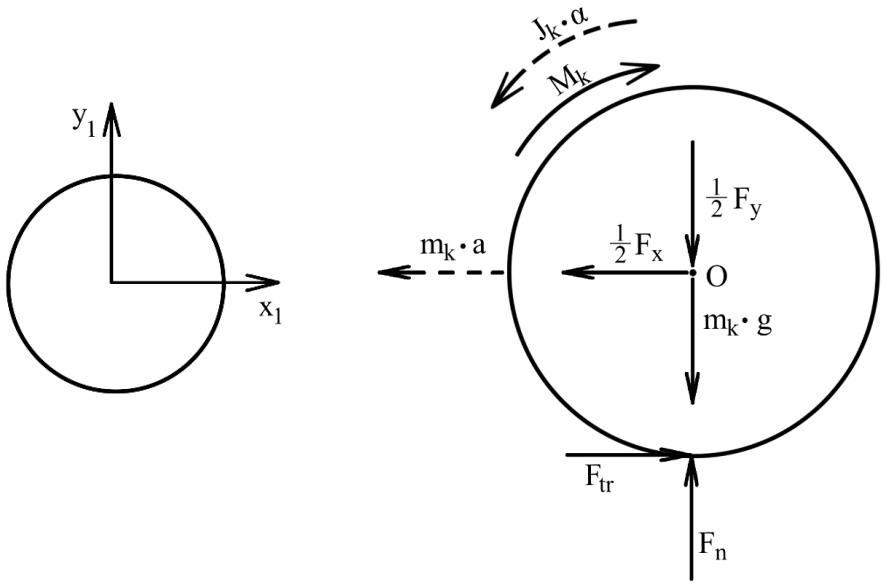
Samobalansirajući robot je sklop mehaničkih komponenata, te kao takav može biti opisan jednadžbama gibanja. Dobivene jednadžbe gibanja mogu biti simulirane u MATLAB-u, što uvelike olakšava razumijevanje sustava i podešavanje PID regulatora. Na slici ispod prikazane su sile i momenti koji djeluju na kotač (lijevo) i tijelo robota (desno).



Slika 8. Sile i momenti na kotaču i tijelu robota

Nakon što su sve sile i momenti definirani, valja pomoću diferencijalnih jednadžbi raspisati dinamiku kotača i tijela robota te ih kombinirati u jednadžbe gibanja. Konačne jednadžbe gibanja trebaju povezivati kutno ubrzanje zakreta robota oko osi kotača s ubrzanjem osi kotača u smjeru x-osi. Navedena ovisnost je korisna jer zakret robota možemo mjeriti senzorom, a na pomak kotača po x-osi utječemo aktuatorom. Potrebna brzina aktuacije biti će određena upravo putem ovisnosti dobivenih jednadžbama gibanja.

5.1. Dinamika kotača



Slika 9. Sile i momenti na kotaču

Na slici iznad prikazane su sve sile i momenti koji djeluju na kotač. Na os kotača djeluju sile F_x i F_y kao posljedica zglobne veze s tijelom robota. Kotač ima svoju težinu smještenu u centar mase, a kao posljedica mase javljaju se inercijske sile pomaka i rotacije. Na mjestu doticaja kotača s tlom javlja se sila trenja. Kotač je pogonjen motorom koji daje moment M_k. Zbog pojednostavljenja proračuna i malog utjecaja na rezultat zanemarit će se proklizavanje kotača ($\mu = 1$) i moment trenja u motoru ($b\dot{x} = 0$). Dobivaju se sljedeći izrazi:

$$\sum F_x = 0 \qquad -m_K \ddot{x}_1 - \frac{1}{2} F_x + F_{tr} = 0$$

$$m_K \ddot{x}_1 = -\frac{1}{2} F_x + F_{tr} \tag{5.1}$$

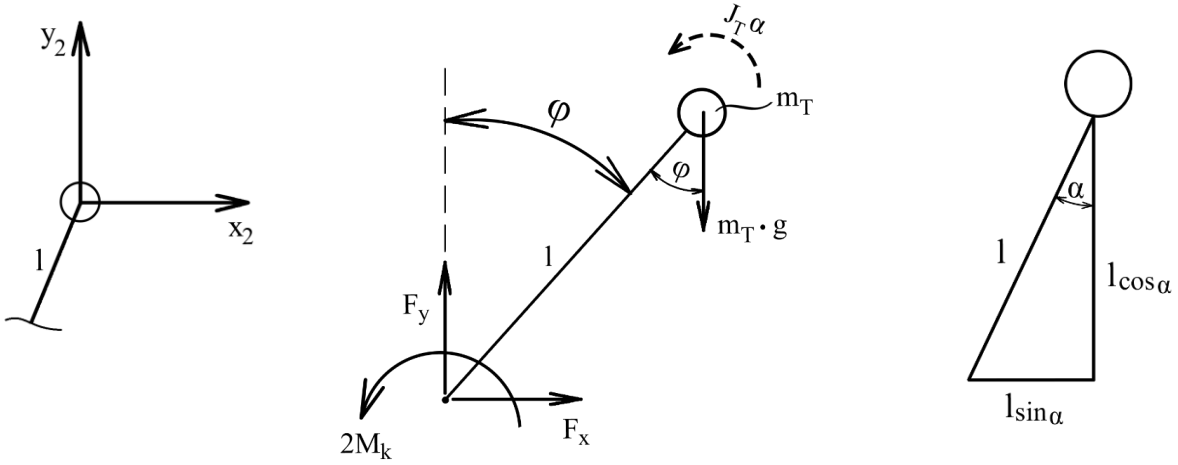
$$\sum F_y = 0 \qquad -\frac{1}{2} F_y - m_K g + F_n - m_K \ddot{y}_1 = 0$$

$$m_K \ddot{y}_1 = -\frac{1}{2} F_y - m_K g + F_N = 0 \quad (jer\ je\ \ddot{y}_1 = 0) \tag{5.2}$$

$$\sum M_0 = 0 \qquad J_K \ddot{\varphi}_K - M_K + F_{tr} R_K = 0$$

$$J_K \ddot{\varphi}_K = M_K - F_{tr} R_K \tag{5.3}$$

5.2. Dinamika tijela robota



Slika 10. Sile i momenti na tijelu robota

Na slici iznad prikazane su sve sile i momenti koji djeluju na tijelo robota. Robot ima dva kotača pa je ukupan moment na tijelo robota $2M_K$. U proračunskom modelu, masa robota je svedena u njegovo težište. Masa ima svoju težinu i pripadni inercijski moment koji se opire promjeni kuta robota. Dobivaju se sljedeći izrazi:

$$\sum F_x = 0 \qquad F_x = m_T \ddot{x}_2 \tag{5.4}$$

$$\sum F_y = 0 \qquad F_y - m_T g = m_T \ddot{y}_2 \tag{5.5}$$

$$\sum M_0 = 0 \qquad F_x l \cos\varphi - F_y l \sin\varphi + 2M_K + J_T \ddot{\varphi} = 0$$

$$J_T \ddot{\varphi} = -F_x l \cos\varphi + F_y l \sin\varphi - 2M_K \tag{5.6}$$

Uvrštavanjem izraza **(5.4)** i **(5.5)** u izraz **(5.6)** dobivamo:

$$J_T \ddot{\varphi} = -m_T \ddot{x}_2 l \cos\varphi + m_T g l \sin\varphi + m_T \ddot{y}_2 l \sin\varphi - 2M_K \tag{5.7}$$

5.3. Kombinacija jednadžbi

Prilikom sređivanja izraza potrebna su sljedeća pravila deriviranja:

$$(fg)' = fg' + f'g$$

$$\Big(f\big(g(x)\big)\Big)' = f'\big(g(x)\big)g'(x)$$

Prvi korak je povezati lokalne koordinatne sustave kotača i tijela robota. Povezivanje će se vršiti uz pomoć geometrije i trigonometrije. Prvo će se povezati koordinate lokalnih sustava x_1 i x_2. Postupak sređivanja glasi:

$$x_2 = x_1 + l\sin\varphi \ /'$$

$$\dot{x}_2 = \dot{x}_1 + l\cos\varphi \cdot \dot{\varphi} \ /'$$

$$\ddot{x}_2 = \ddot{x}_1 - l\sin\varphi \cdot (\dot{\varphi})^2 + l\cos\varphi \cdot \ddot{\varphi} \qquad \textbf{(5.8)}$$

Isti postupak može se provesti za koordinate y_1 i y_2, pa glasi:

$$y_2 = y_1 + l\cos\varphi \ /'$$

$$\dot{y}_2 = \dot{y}_1 - l\sin\varphi \cdot \dot{\varphi} = -l\sin\varphi \cdot \dot{\varphi}/'$$

$$\dot{y}_2 = 0 - l\sin\varphi \cdot \dot{\varphi} = -l\sin\varphi \cdot \dot{\varphi}/'$$

$$\ddot{y}_2 = -l\cos\varphi \cdot (\dot{\varphi})^2 - l\sin\varphi \cdot \ddot{\varphi} \qquad \textbf{(5.9)}$$

Sređivanjem izraza **(5.7)** dobiva se:

$$J_T\ddot{\varphi} = -2M_K + m_T g l\sin\varphi + m_T l(\underline{\ddot{y}_2\sin\varphi - \ddot{x}_2\cos\varphi})$$

Podcrtani izraz ne pomaže u rješavanju problema jer daje ovisnost u vezi s koordinatama lokalnog sustava u težištu tijela robota. Cilj je dobiti jednadžbu ovisnosti derivacije pomaka x_1 (pomak kotača) i kuta zakreta φ (nagib robota). Sređivanje podcrtanog izraza glasi:

$$\ddot{y}_2\sin\varphi - \ddot{x}_2\cos\varphi =$$

$$= (-l\cos\varphi\cdot(\dot{\varphi})^2 - l\sin\varphi\cdot\ddot{\varphi})\sin\varphi - (\ddot{x}_1 - l\sin\varphi\cdot(\dot{\varphi})^2 + l\cos\varphi\cdot\ddot{\varphi})\cos\varphi =$$

$$= \cancel{-l(\dot{\varphi})^2\sin\varphi\cos\varphi} - l\ddot{\varphi}\sin^2\varphi - \ddot{x}_1\cos\varphi + \cancel{l(\dot{\varphi})^2\sin\varphi\cos\varphi} - l\ddot{\varphi}\cos^2\varphi =$$

$$= -\ddot{x}_1\cos\varphi + l\ddot{\varphi}(\cos^2\varphi + \sin^2\varphi) =$$

$$= -\ddot{x}_1\cos\varphi + l\ddot{\varphi}\cdot 1 =$$

$$= -\ddot{x}_1\cos\varphi + l\ddot{\varphi} \qquad \textbf{(5.10)}$$

Nakon uvrštavanja izraza **(5.10)** u izraz **(5.7)** i sređivanja dobiva se:

$$J_T\ddot{\varphi} = -2M_K + m_T g l \sin\varphi - m_T l\ddot{x}_1 \cos\varphi - m_T l^2\ddot{\varphi}$$

$$(J_T + m_T l^2)\ddot{\varphi} = m_T g l \sin\varphi - m_T l\ddot{x}_1 \cos\varphi - 2M_K \quad \textbf{(5.11)}$$

Vidljivo je da je konačan izraz ovisnost kutne akceleracije robota i horizontalne akceleracije kotača. Taj izraz je koristan, te će biti nazvan prva jednadžba gibanja.

Sljedeća potrebna ovisnost je ona između kuta zakreta kotača i horizontalnog pomaka kotača. Ta ovisnost se može odrediti pomoću geometrije kotača, uz pretpostavku da nema proklizavanja. Izvod glasi:

$$\frac{\varphi_K r}{t} = \frac{2r\pi}{t} = \frac{s}{t} = \frac{x}{t}$$

$$\frac{\varphi_K}{t} = \frac{2\pi}{t} \cdot r = v$$

$$\omega = \frac{v}{r}$$

$$\dot{\varphi}_K = \frac{\dot{x}_1}{\frac{D_K}{2}}$$

$$\dot{\varphi}_K = 2\frac{\dot{x}_1}{D_K}\ /'$$

$$\ddot{\varphi}_K = 2\frac{\ddot{x}_1}{D_K} = \frac{\ddot{x}_1}{R_K} \quad \textbf{(5.12)}$$

Uvrštavanjem izraza **(5.1)** i **(5.12)** u izraz **(5.3)** dobiva se:

$$J_K \cdot \frac{\ddot{x}_1}{R_K} = M_K - R_K(m_K\ddot{x}_1 + \frac{1}{2}F_x)$$

$$\frac{J_K}{R_K}\ddot{x}_1 = M_K - R_K m_K\ddot{x}_1 - \frac{R_K}{2}\ F_x \quad \textbf{(5.13)}$$

Uvrštavanjem izraza **(5.4)** u izraz **(5.13)** dobiva se:

$$\frac{J_K}{R_K}\ddot{x}_1 = M_K - R_K m_K\ddot{x}_1 - \frac{R_K}{2}\ m_T\ddot{x}_2 \quad \textbf{(5.14)}$$

Uvrštavanjem izraza **(5.8)** u izraz **(5.14)** i sređivanjem dobiva se:

$$\frac{J_K}{R_K}\ddot{x}_1 = M_K - R_K m_K \ddot{x}_1 - \frac{R_K}{2}\, m_T(\ddot{x}_1 - l\sin\varphi\cdot(\dot{\varphi})^2 + l\cos\varphi\cdot\ddot{\varphi})$$

$$\frac{J_K}{R_K}\ddot{x}_1 = M_K - R_K m_K \ddot{x}_1 - \frac{1}{2}\, R_K m_T \ddot{x}_1 + \frac{1}{2} R_K m_T l\sin\varphi\cdot(\dot{\varphi})^2 - \frac{1}{2} R_K m_T l\cos\varphi\cdot\ddot{\varphi}$$

$$\left(\frac{J_K}{R_K} + R_K m_K + \frac{1}{2}\, R_K m_T\right)\ddot{x}_1 = M_K + \frac{1}{2}\, R_K m_T l\sin\varphi\cdot(\dot{\varphi})^2 - \frac{1}{2} R_K m_T l\cos\varphi\cdot\ddot{\varphi} \qquad \textbf{(5.15)}$$

Vidljivo je da je i ovaj konačan izraz ovisnost kuta nagiba robota i horizontalnog pomaka kotača. Taj izraz je također koristan, te će biti nazvan druga jednadžba gibanja.

Dobiveni sustav je nelinearan. Kako bi se sustav linearizirao, moraju se uvesti određene pretpostavke. Kada robot održava samo-ravnotežu sustav se nalazi u radnom području blizu nule, pa se može uvesti:

$$\sin\varphi \approx \varphi \qquad \ddot{x}_1\cdot\cos\varphi \approx \ddot{x}_1$$

Uvođenjem pretpostavki prva jednadžba gibanja sada glasi:

$$(J_T + m_T l^2)\ddot{\varphi} = m_T g l\varphi - m_T l\ddot{x}_1 - 2M_K \qquad \textbf{(5.16)}$$

Za drugu jednadžbu uvode se sljedeća pojednostavljenja:

$$(\dot{\varphi})^2 \approx 0 \qquad \cos\varphi\cdot\ddot{\varphi} \approx \ddot{\varphi}$$

Uvođenjem pretpostavki druga jednadžba gibanja sada glasi:

$$\left(\frac{J_K}{R_K} + R_K m_K + \frac{1}{2}\, R_K m_T\right)\ddot{x}_1 = M_K - \frac{1}{2}\, R_K m_T l\ddot{\varphi} \qquad \textbf{(5.17)}$$

5.4. Prostor stanja

Opći oblik prostora stanja glasi:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

U ovom sustavu ulazni podaci su vezani uz pomak x i kut zakreta φ. Vektor stanja x sastoji se od zavisnih veličina x i φ te njihovih derivacija. Izlazni vektor y sastoji se od željenih izlaznih veličina. U ovom slučaju bitno je dobivati jedino podatak o kutu robota, jer je cilj da taj kut bude jednak nuli. Opisane vrijednosti izgledaju ovako:

$$x = \begin{bmatrix} x \\ \dot{x} \\ \varphi \\ \dot{\varphi} \end{bmatrix} \qquad y = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \varphi \\ \dot{\varphi} \end{bmatrix} \qquad D = 0 \qquad \textbf{(5.18)}$$

Zbog kompleksnosti izraza **(5.16)** i **(5.17)** uvode se nove oznake:

$$E = J_T + m_T l^2 \qquad H = \left(\frac{J_K}{R_K} + R_K m_K + \frac{1}{2} R_K m_T\right)$$

$$F = m_T g l \qquad I = \frac{1}{2} R_K m_T l \qquad \textbf{(5.19)}$$

$$G = m_T l$$

Izrazi **(5.16)** i **(5.17)** uz uvrštene nove oznake **(5.19)** glase:

$$E\ddot{\varphi} = F\varphi - G\ddot{x}_1 - 2M_K \qquad \textbf{(5.20)}$$

$$H\ddot{x}_1 = M_K - I\ddot{\varphi} \qquad \textbf{(5.21)}$$

Iz prve jednadžbe općeg oblika prostora stanja vidljivo je da se vektor stanja x derivira. U vektoru stanja x su zavisne veličine x i φ te njihove derivacije prvog reda. Kako bi se dobile matrice koje zadovoljavaju prvu jednadžbu općeg oblika prostora stanja, potrebno je izraziti druge derivacije zavisnih veličina x i φ u ovisnosti o zavisnim veličinama i ulaznim veličinama. U ovom slučaju ulazna veličina sustava je moment kotača M_K.

Prvi korak je izlučivanje drugih derivacija zavisnih veličina iz izraza **(5.21)**. Dobiva se:

$$\ddot{\varphi} = \frac{M_K}{I} - \frac{H}{I}\ddot{x}_1 \tag{5.22}$$

$$\ddot{x}_1 = \frac{M_K}{H} - \frac{I}{H}\ddot{\varphi} \tag{5.23}$$

Uvrštavanjem izraza **(5.22)** u izraz **(5.20)** dobiva se prva valjana jednadžba:

$$\frac{E}{I}M_K - \frac{EH}{I}\ddot{x}_1 = F\varphi - G\ddot{x}_1 - 2M_K$$

$$\left(\frac{EH}{I} - G\right)\ddot{x}_1 = -F\varphi + \left(\frac{E}{I} + 2\right)M_K$$

$$\ddot{x}_1 = \frac{-F}{\frac{EH - GI}{I}}\varphi + \frac{\frac{E + 2\cdot I}{I}}{\frac{EH - GI}{I}}M_K$$

$$\ddot{x}_1 = \frac{-FI}{EH - GI}\varphi + \frac{E + 2\cdot I}{EH - GI}M_K \tag{5.24}$$

Uvrštavanjem izraza **(5.23)** u izraz **(5.20)** dobiva se druga valjana jednadžba:

$$E\ddot{\varphi} = F\varphi - \frac{G}{H}M_K + \frac{GI}{H}\ddot{\varphi} - 2M_K$$

$$\left(\frac{GI}{H} - E\right)\ddot{\varphi} = \left(\frac{G}{H} + 2\right)M_K - F\varphi$$

$$\ddot{\varphi} = \frac{\frac{G + 2H}{H}}{\frac{GI - EH}{H}}M_K - \frac{F}{\frac{GI - EH}{H}}\varphi$$

$$\ddot{\varphi} = \frac{G + 2H}{GI - EH}M_K - \frac{FH}{GI - EH}\varphi \tag{5.25}$$

Nakon što su raspisane valjane jednadžbe sustava direktno je moguće napisati prostor stanja primjenom pravila matričnog zapisa:

$$\begin{bmatrix} \dot{x}_1 \\ \ddot{x}_1 \\ \dot{\varphi} \\ \ddot{\varphi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{-FI}{EH-GI} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \dfrac{-FH}{GI-EH} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \dot{x}_1 \\ \varphi \\ \dot{\varphi} \end{bmatrix} + \begin{bmatrix} 0 \\ \dfrac{E+2I}{EH-GI} \\ 0 \\ \dfrac{G+2H}{GI-EH} \end{bmatrix} u$$

(5.26)

$$y = [0 \quad 0 \quad 1 \quad 0] \begin{bmatrix} x_1 \\ \dot{x}_1 \\ \varphi \\ \dot{\varphi} \end{bmatrix}$$

Odnosno, matrice A, B, C glase:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{-FI}{EH-GI} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \dfrac{-FH}{GI-EH} & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ \dfrac{E+2I}{EH-GI} \\ 0 \\ \dfrac{G+2H}{GI-EH} \end{bmatrix}$$

$$C = [0 \quad 0 \quad 1 \quad 0]$$

Time je prostor stanja za općeniti slučaj samobalansirajućeg robota određen. Sustav na svojem izlazu y prikazuje odziv na ulazni moment M_K. Izlaz sustava je kut φ.

5.5. Dopuna jednadžbi za koračni motor

Budući da je u ovom radu korišten koračni motor, ne vrijede ista pravila kao za običan DC motor. Običan DC motor je upravljan strujom koja rezultira momentom motora. Koračni motor je upravljan brojem koraka u sekundi što rezultira kutom brzinom. Do sada je razmatrana ovisnost kuta zakreta robota o momentu motora na ulazu, što se ne može postići koračnim motorom. Zbog toga je sustav potrebno napisati kao ovisnost kuta zakreta robota o kutnoj brzini vrtnje koračnog motora.

Prvi korak je sustav iz prostora stanja prebaciti u s-domenu. U sljedećem postupku prikazano je uvrštavanje i sređivanje izraza:

$$s^2 x_1 = \frac{-FI}{EH - GI}\varphi + \frac{E + 2I}{EH - GI}M_K \qquad /\cdot (EH - GI)$$
$$s^2 \varphi = \frac{-FH}{GI - EH}\varphi + \frac{G + 2H}{GI - EH}M_K \qquad /\cdot (EH - GI)$$

$$s^2 x_1 (EH - GI) = -FI\varphi + (E + 2I)M_K \qquad /\cdot (G + 2H)$$
$$s^2 \varphi (EH - GI) = FH\varphi - (G + 2H)M_K$$

$$s^2 x_1 (EH - GI)(G + 2H) = -FI(G + 2H)\varphi + (E + 2I)(G + 2H)M_K \qquad \textbf{(5.27)}$$
$$(G + 2H)M_K = FH\varphi - s^2 \varphi (EH - GI) \qquad \textbf{(5.28)}$$

Kako bi se eliminirao moment M_K iz sustava, uvrštava se izraz **(5.28)** u izraz **(5.27)**. Postupak uvrštavanja i sređivanja izgleda ovako:

$$s^2 x_1 (EH - GI)(G + 2H) = -FI(G + 2H)\varphi + (E + 2I)(FH\varphi - s^2 \varphi (EH - GI))$$
$$s^2 x_1 (EH - GI)(G + 2H) = -FI(G + 2H)\varphi + FH(E + 2I)\varphi - s^2 \varphi (EH - GI)(E + 2I)$$
$$s^2 x_1 (EH - GI)(G + 2H) = \varphi[-s^2 (EH - GI)(E + 2I) + FH(E + 2I) - FI(G + 2H)] \qquad \textbf{(5.29)}$$

Kao što je već i prije bilo razmatrano za kotač, uz pretpostavku da nema proklizavanja, vrijedi izraz koji povezuje kutnu brzinu i horizontalnu brzinu:

$$R_K \dot{\varphi}_K = \dot{x}_1$$

Prebacivanjem jednadžbe u s-domenu dobiva se:

$$sR_K \varphi_K = sx_1 \tag{5.30}$$

Uvrštavanjem izraza **(5.30)** u izraz **(5.29)** dobiva se:

$$s \cdot sR_K \varphi_K (EH - GI)(G + 2H) = \varphi[\cdots]$$

Konačna željena prijenosna funkcija izgleda ovako:

$$\frac{\varphi}{s\varphi_K} = \frac{-sR_K(EH - GI)(G + 2H)}{s^2(EH - GI)(E + 2I) + FI(G + 2H) - FH(E + 2I)} \tag{5.31}$$

Izraz **(5.31)** prikazuje ovisnost kuta zakreta robota φ o kutnoj brzini $s\varphi_K$ koračnog motora. Pomoću dobivene ovisnosti moguće je direktno simulirati odziv kuta zakreta robota na ulaznu brzinu vrtnje koračnog motora pomoću programskog paketa MATLAB. Također na simulacijskom modelu moguće je provesti regulaciju kako bi se dobile približne vrijednosti parametara regulatora potrebnih da se sustav ustabili. Više o tome se može pročitati u sljedećem poglavlju.

6. SIMULACIJA

Kako bi se potvrdila mogućnost regulacije specifične mehaničke konstrukcije, u ovom slučaju samobalansirajućeg robota, potrebno je prvotno provesti simulaciju. Ako su rezultati simulacije zadovoljavajući, prelazi se na fizičku izradu i programiranje. U suprotnom, ako rezultati nisu zadovoljavajući i sustav se ne može stabilizirati, potrebne su promjene u mehaničkoj konstrukciji ili u načinu regulacije. U ovom radu, simulacija dinamike sustava i regulacije istog će biti provedena pomoću programskog paketa MATLAB.

6.1. Simulacija jednadžbi gibanja

Na samom početku potrebno je definirati ulazne podatke simulacije. Podaci redom definirani su: masa tijela robota, masa kotača, ubrzanje sile teže, visina težišta, moment inercije tijela robota sveden na koordinatni sustav kotača, moment inercije kotača te promjer i radijus kotača.

```
%% Ulazni podaci
mT = 2.35628; %kg
mK = 0.08394; %kg
g = 9.81; %m/s^2
l = 0.05407; %m
JT = 0.01793029465; %kg*m^2
JK = 0.00005677435; %kg*m^2
DK = 0.075; %m
RK = 0.0375; %m
```

Nakon definicije ulaznih podataka, definirat će se pojednostavljenja **(5.19)** :

```
%% Varijable
E = JT+mT*l^2;
F = mT*g*l;
G = mT*l;
H = (JK/RK)+RK*mK+0.5*RK*mT;
I = 0.5*RK*mT*l;
```

Za određivanje prostora stanja, na ulazu je potrebno imati matrice A, B, C i D. Prvo će biti definirane prazne (null) matrice u koje će kasnije biti upisani članovi:

```
%% Matrice prostora stanja
A = zeros(4,4);
B = zeros(4,1);
C = zeros(1,4);
D = 0;
```

Sljedeći korak je popunjavanje matrica prostora stanja s varijablama vezanim za sustav samobalansirajućeg robota:

```
A(1,2) = 1;
A(2,3) = (-F*I)/(E*H-G*I);
A(3,4) = 1;
A(4,3) = (-F*H)/(G*I-E*H);

B(2,1) = (E+2*I)/(E*H-G*I);
B(4,1) = (G+2*H)/(G*I-E*H);

C(1,3) = 1;
```

Jednom kada su definirane osnovne matrice, može se odrediti prostor stanja:

```
%% Prostor stanja
t = 0:0.01:10;
N = length(t);

u = [1 zeros(1,N-1)];

x = [0;0;0;0];
for k = 1:N
    y(k) = C*x;
    x = A*x + B*u(k);
end
```

Na kraju se iz prostora stanja može i direktno dobiti prijenosna funkcija sustava pomoću koje se lako može prikazati odziv sustava i provesti simulacija regulacije. Kod glasi:

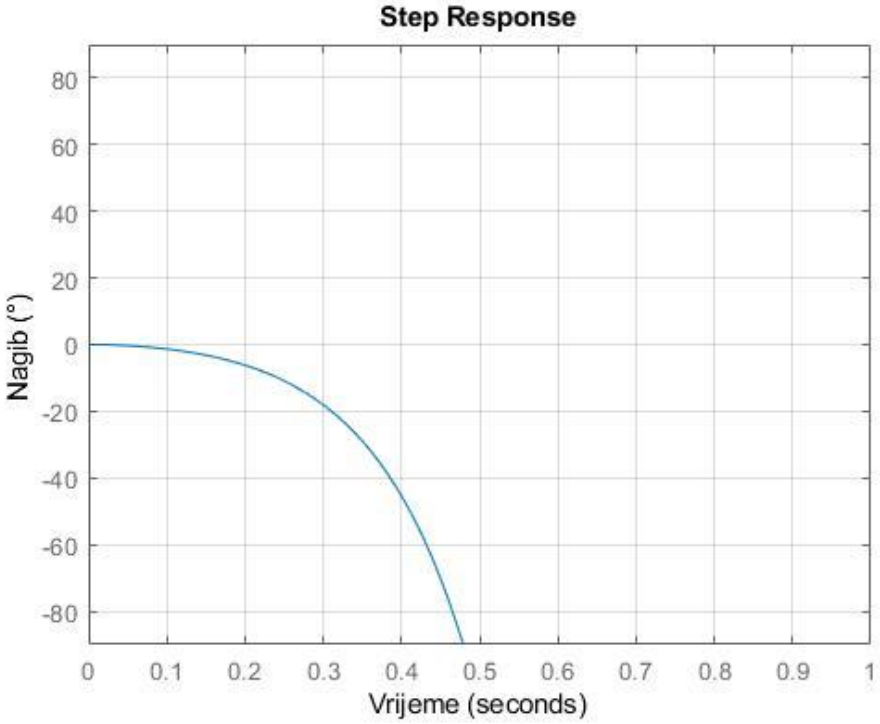
```
%% Prijenosna funkcija
[br,naz] = ss2tf(A,B,C,D);
yt = filter(br,naz,u);

G1 = tf(br, naz);
```

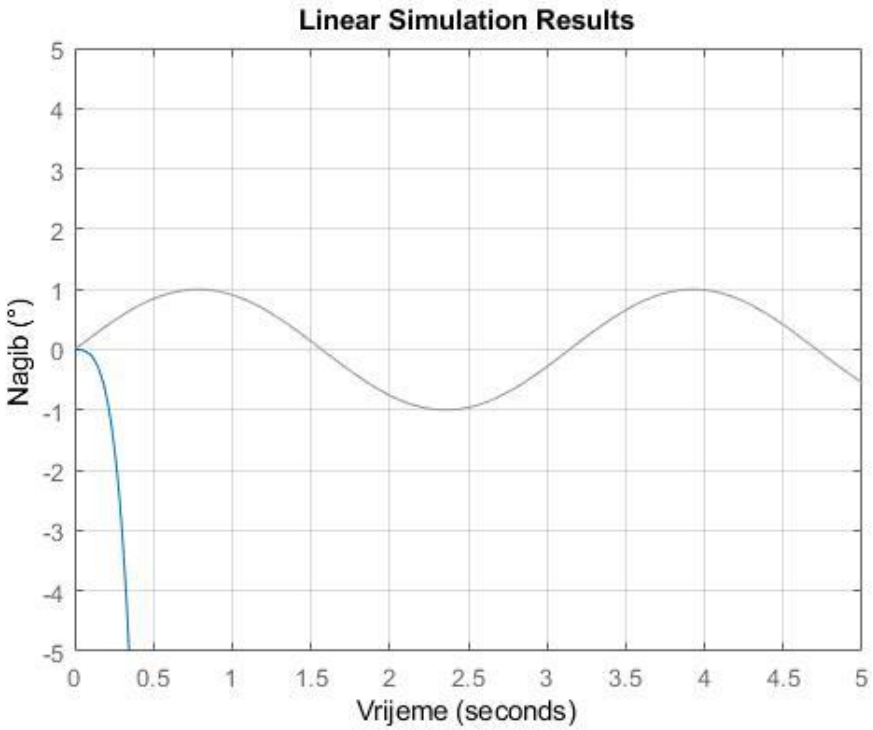
Pošto se radi o prirodno nestabilnom sustavu, odziv sustava bez ikakve regulacije je nestabilan i teži prevrtanju. Ta hipoteza je potvrđena korištenjem simulacije. Na slikama ispod prikazani su simulirani odzivi sustava.

Prva slika prikazuje kako se ponaša sustav, koji je inicijalno postavljen u stanje ravnoteže, na step pobudu. Nagib naglo raste, što znači da je sustav izgubio ravnotežu, a robot se prevrnuo. Jasno je vidljivo da je sustav sam po sebi nestabilan.

Na drugoj slici prikazano je ponašanje nereguliranog sustava na željenu sinusnu pobudu. I iz ove slike je jasno vidljivo da je sustav sam po sebi nestabilan i ne može pratiti sinusnu pobudu ako ne postoji regulacijski krug.



Slika 11. Odziv nereguliranog lineariziranog sustava na momentnu step pobudu



Slika 12. Odziv nereguliranog lineariziranog sustava na momentnu sinusnu pobudu

6.2. PID regulacija

U prošlom poglavlju je prikazano da je sustav nestabilan, te je potreban regulator za njegovu stabilnost. Koristit će se PID regulator za potrebe ovog završnog rada.

Slika 13. Blok shema PID regulatora

Prijenosna funkcija PID regulatora se može zapisati na sljedeći način:

$$Reg = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

Kada se otvoreni sustav G želi regulirati, za prijenosne funkcije sustava i regulatora vrijedi sljedeća formula zatvorenog regulacijskog sustava:

$$Krug = \frac{Reg \cdot G}{1 + Reg \cdot G}$$

Programski kod u MATLAB-u za regulaciju definiranog sustava iz prošlog poglavlja glasi:

```
%% PID regulacija
Kp = 5;
Ki = 20;
Kd = 1;

Reg_brojnik = [Kd Kp Ki];
Reg_nazivnik = [1 0];
Reg = tf(Reg_brojnik, Reg_nazivnik

Krug = feedback(G1*Reg,-1);
```

Budući da se koristi koračni motor, kojem nije moguće kontrolirati moment promjenom struje, kao što je to slučaj kod DC motora, mora se mijenjati logika upravljanja. Koračnom motoru je lako kontrolirati brzinu vrtnje, pa će se sustav regulirati pomoću brzine vrtnje koračnog motora. Da bi se to postiglo potrebno je koristiti prijenosnu funkciju **(5.31)** prilagođenu koračnom motoru te tu funkciju regulirati.

Prijenosna funkcija prilagođena koračnom motoru dobiva se pomoću sljedećeg koda:

```
%% TF koracnog motora
broj1 = -RK*(E*H-G*I)*(G+2*H);
broj2 = (E*H-G*I)*(E+2*I);
broj3 = F*I*(G+2*H)-F*H*(E+2*I);

brojnik = [0, broj1, 0];
nazivnik = [broj2, 0, broj3];

Gk = tf(brojnik, nazivnik)
```

Nakon što je definirana prijenosna funkcija, pomoću MATLAB-ovog alata „pidTuner“ moguće je lako podesiti parametre PID regulatora. Alat direktno iscrtava odziv sustava na željene parametre koji se podešavaju klizačima. Alat se pokreće ovako:

```
pidTuner(Gk,'pidf')
```

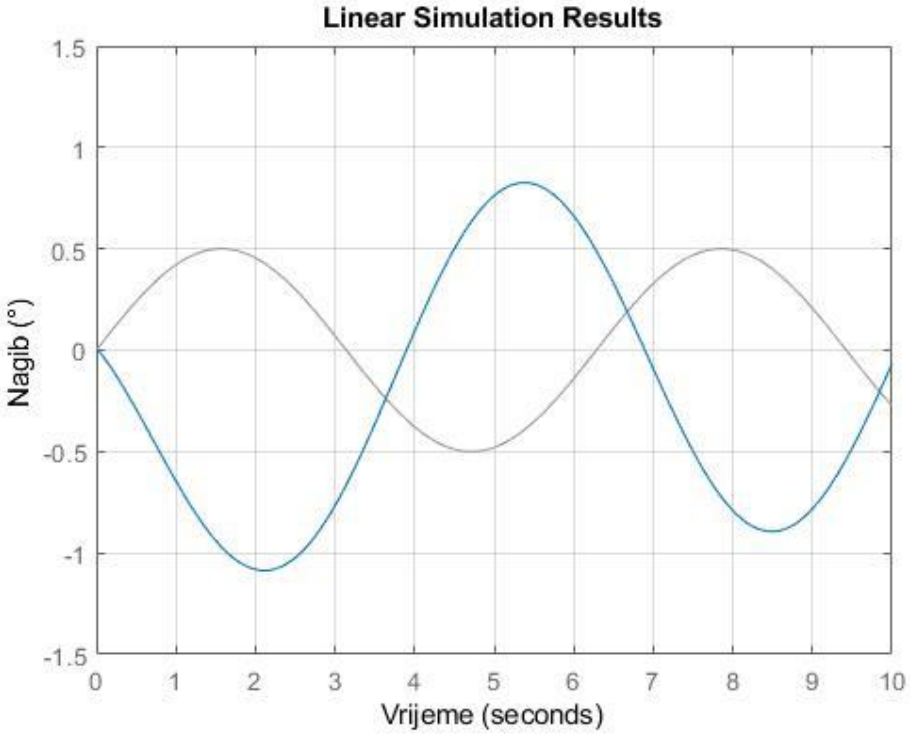
Tako određeni parametri uvrštavaju se ponovo u regulator. Kod glasi:

```
%% Regulacija koracnog motora
Kp = 125;
Ki = 200;
Kd = 0;

Reg_brojnik = [Kd Kp Ki];
Reg_nazivnik = [1 0];
Reg = tf(Reg_brojnik, Reg_nazivnik);

Krug_koracni_motor = feedback(Gk*Reg,-1);
```

Nakon što je određena prijenosna funkcija reguliranog kruga, može se iscrtati dijagram odziva kako bi se utvrdilo postiže li sustav stabilnost ili je sustav potrebno modificirati.



Slika 14. Odziv reguliranog sustava na sinusnu pobudu

Sustav je simuliran sinusnom pobudom jer će se tokom rada robota nagib mijenjati otprilike sinusoidalno između pozitivnih i negativnih vrijednosti nagiba. Iz odziva je vidljivo da se nagib nakon što pobuda krede u jednom smjeru počinje zbog inercije mijenjati u suprotnom, no promjena nagiba se vrlo brzo anulira zbog utjecaja regulatora, te se sustav vraća u željeni položaj. Sada, kada je vidljivo na temelju simulacije da je regulacija samobalansirajućeg robota moguća, može se krenuti u fizičku izvedbu i kreiranje programa.

7. PROGRAM

Kako bi samobalansirajući robot imao funkciju stabilizacije, vitalni dio je izrada programa mikrokontrolera. Program mora obavljati više funkcija: obradu signala iz senzora, upravljanje koračnim motorima, PID regulaciju i bluetooth komunikaciju s mobitelom. Cijeli program mora biti napravljen na efikasan način kako bi se procesorski resursi iskoristili najbolje moguće. Korišteni mikrokontroler ESP32 ima dvije jezgre, a time i mogućnost razdvajanja obrade podataka iz senzora i pokretanje koračnih motora, što je i učinjeno u programu. U daljnjem tekstu biti će opisani ključni dijelovi programa. Neće biti prikazani svi dijelovi programa, niti sve definicije varijabli. Cijeli kod može se naći u prilozima.

7.1. Obrada signala iz senzora

Postoje dva osnovna tipa filtra primjenjivih u ovom slučaju. To su Kalmanov filtar i komplementarni filtar. Kalmanov filtar je nešto kompleksniji za proračun, te je zbog toga korišten komplementarni filtar koji provodi jednostavan proračun. Taj proračun se provodi 500 puta u sekundi. Senzor očitava podatke iz akcelerometra i žiroskopa te ih sprema u varijable. Pomoću podataka iz akcelerometra izračunava se nagib koji je zbog naravi akcelerometra jako osjetljiv na vibracije. Između ciklusa očitavanja mjeri se vrijeme dt. Komplementarni filtar na kraju kombinira podatke iz akcelerometra i žiroskopa na temelju prošlih vrijednosti, vremena između dva čitanja i koeficijenta komplementarnog filtra zadanog na početku programa. Kod funkcije za očitavanje senzora prikazan je ispod:

```
void Ocitavanje_senzora(){
  while (i2cRead(0x3B, i2cData, 14));
  accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
  accY = (int16_t)((i2cData[2] << 8) | i2cData[3]);
  accZ = (int16_t)((i2cData[4] << 8) | i2cData[5]);
  gyroY = (int16_t)((i2cData[10] << 8) | i2cData[11]);

  double dt = (double)(micros() - timer) / 1000000; // Proracun delta_t
  timer = micros();

  pitch = (atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG) + ko-
rekcija_kuta;
  double gyroYrate = gyroY / 131.0;

  compAngleY = Koeficijent_filtra * (compAngleY + gyroYrate * dt) + (1-Koefi-
cijent_filtra) * pitch;
}
```

7.2. Upravljanje koračnim motorima

Brzina koračnih motora ovisi o broju impulsa poslanih od mikrokontrolera. Na driver koračnih motora šalju se impulsi. Svaki impuls znači da motor mora napraviti jedan korak. Za odabrani driver TB6600 već postoji gotov library od proizvođača za upravljanje motora. Pomoću već određenih funkcija, lako je definirati upravljač:

```
#include <AccelStepper.h>
```

Nakon toga definiraju se lijevi i desni koračni motor. Potrebno je definirati samo tri pina za upravljanje svakim motorom: pin za omogućivanje rada, pin gdje se daju impulsi koraka i pin gdje se određuje smjer vrtnje motora na temelju logičke nule ili jedinice. Kod je:

```
AccelStepper stepper_L(EN_L, CLK_L, CW_L);
AccelStepper stepper_D(EN_D, CLK_D, CW_D);
```

Pošto je poznato da koračni motori gube moment s povećanjem brzine, potrebno je ograničiti maksimalnu brzinu vrtnje motora. To se izvodi sljedećim funkcijama:

```
stepper_L.setMaxSpeed(5000);
stepper_D.setMaxSpeed(5000);
```

Na kraju, za kontroliranje koračnih motora korištene su dvije funkcije: jedna koja određuje kojom će se brzinom vrtjeti motor i druga koja pokreće koračni motor.

```
stepper_L.setSpeed(brzina_L);
stepper_D.setSpeed(brzina_D);
stepper_L.runSpeed();
stepper_D.runSpeed();
```

7.3. PID regulacija

Za PID regulator također postoji već gotov library od Arduina koji olakšava implementaciju regulacije. U isto vrijeme osigurava točnost koda i optimizaciju. Inicijalizacija izgleda ovako:

```
#include <PID v1.h>
```

Nakon toga definira se PID regulator koji će kontrolirati motore. Potrebno je definirati proporcionalnu, integracijsku i derivacijsku konstantu te smjer vrtnje motora nakon regulacije. Kada je to poznato, regulator se inicijalizira ovako:

```
PID PIDreg(&Input, &Output, &Setpoint, Kp, Ki, Kd, REVERSE);
```

Za ispravan rad regulatora, potrebno je podesiti još neke postavke. Setpoint je željeni nagib robota, koji je u ovom slučaju nula. Izlazne granice regulatora treba podesiti na područje od -5000 do 5000 jer je to područje u kojem rade koračni motori. Zadnja opcija koju je potrebno podesiti je vrijeme sampliranja regulatora. Vrijednost te varijable je jednaka vremenu sampliranja kod akcelerometra i žiroskopa. Kod izgleda ovako:

```
Input = 0;
Setpoint = 0;
PIDreg.SetMode(AUTOMATIC);
PIDreg.SetOutputLimits(-5000, 5000);
PIDreg.SetSampleTime(vrijeme_sampliranja);
```

Na kraju, za izračun regulacije potrebno je pozivati kod u kojem se definira vrijednost nagiba, te se na temelju toga izračunava potrebna brzina motora i piše se u varijable koje se šalju na driver koračnog motora kao impulsi. Kod izgleda ovako:

```
Input = compAngleY;
PIDreg.Compute();
brzina_L = Output;
brzina_D = -Output;
```

7.4. Osiguranje gašenja motora prilikom pada

S vremena na vrijeme, prilikom testiranja, moguće je da robot izgubi ravnotežu i padne. Prilikom pada, robot se naginje, a motori se nastoje vrtjeti čim brže. To može uzrokovati da se robot velikom brzinom zabije u nešto, nekoga ozlijedi ili nastane kvar na samom robotu. Kako bi se to spriječilo potrebno je implementirati provjeru nagiba robota, koja rezultira gašenjem motora pri većem nagibu. Kod izgleda ovako:

```
if(compAngleY > 30 || compAngleY < -30){
   if(start == false){
     Input = 0;
     Output = 0;
     Setpoint = 0;
   }else if(start == true){
     ESP.restart();
   }
 }
```

8. MOBILNA APLIKACIJA ZA UPRAVLJANJE

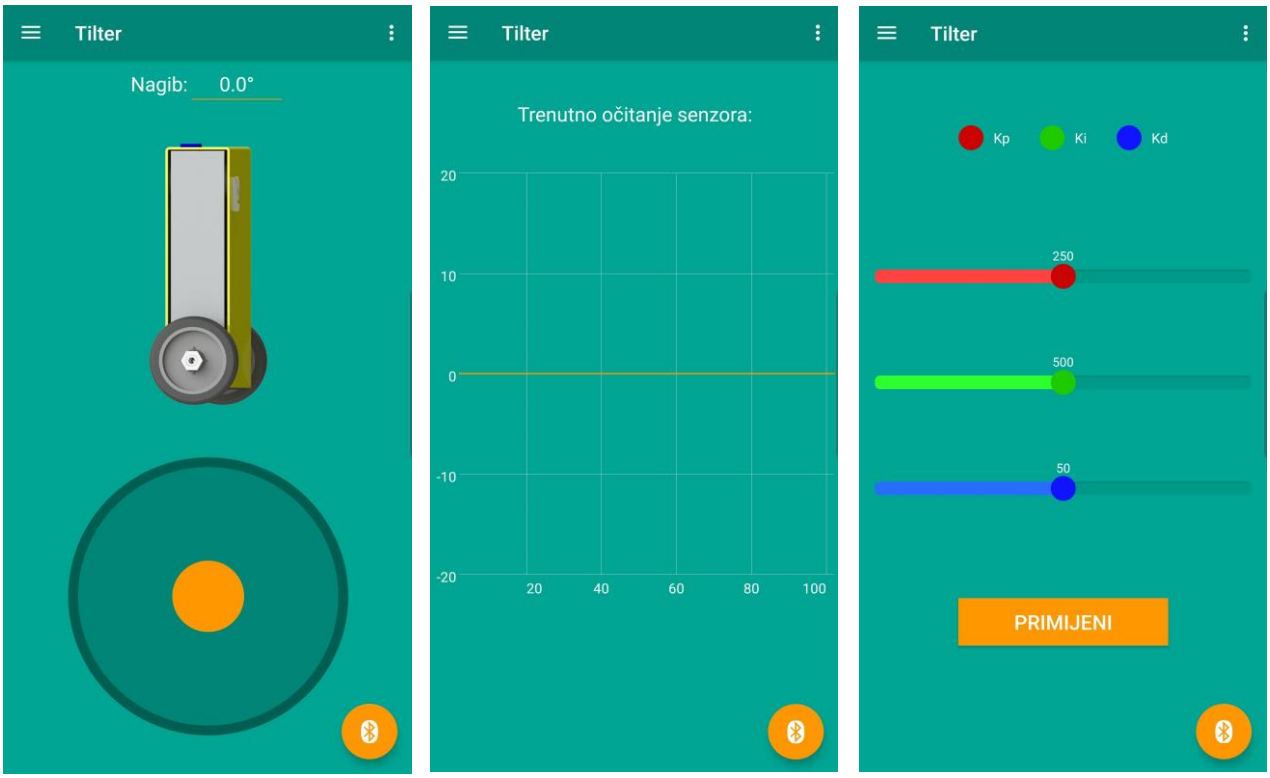
Kako bi se omogućile dodatne funkcionalnosti samobalansirajućeg robota, kreirana je aplikacija za kontrolu robota. Aplikacija je rađena u Android Studiu u programskom jeziku Java te je kompatibilna sa svim android pametnim telefonima koji imaju operacijski sustav android 6.0 ili više. Omogućuje kontrolu robota u prostoru pomoću upravljača, direktan prikaz nagiba robota na grafu te podešavanje parametara PID regulatora. Komunikacija se vrši putem bluetootha. Zbog svoje izrazite kompleksnosti i veličine, kod aplikacije neće biti komentiran u sklopu ovog rada, već će samo biti objašnjen način rada aplikacije. Kod glavnih programskih klasa može se naći u prilogu, a aplikacija se može preuzeti preko QR koda ispod.

Slika 15. QR kod za preuzimanje aplikacije

Izrada aplikacija u Android Studiu sastoji se od nekoliko dijelova. Grafički dio aplikacije se programira u xml-u. Tamo se određuje izgled svakog dijela aplikacije, zajedno s pozicijama i postavkama izgleda svih aktivnih elemenata. Osnovni dijelovi svake xml datoteke su layouti u koje se dodaju dalje drugi layouti ili aktivni elementi. Kao primjer, kućica za upis teksta se u xml-u određuje pomoću sljedećeg seta naredbi:

```
<android.support.v7.widget.AppCompatEditText
    android:id="@+id/kontroler_nagib"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center_vertical"
    android:focusable="false"
    android:lines="1"
    android:hint=" -- nagib -- "
    android:inputType="numberDecimal"
    android:digits="2"
    android:textColor="@color/bijela"
    android:textColorHint="@color/siva"
    android:textSize="20dp"
    android:textAlignment="center"
    app:backgroundTint="@color/colorAccent" />
```

Finalni izgled aplikacije može se vidjeti na slici ispod.



Slika 16. Finalni izgled aplikacije

Nakon što je definiran izgled aplikacije, potrebno je cijelu aplikaciju povezati pomoću programa u Java programskom jeziku. Glavni tip programske klase u Android Studiu naziva se activity. Prilikom pokretanja aplikacije, prvo se pokreće activity koji redom provodi inicijalizaciju svih dijelova programa i na zaslonu prikazuje predodređeni xml layout. Svako pokretanje activitya, čak i na najbržim mobitelima, je spor proces i stvara neugodno čekanje da se pokrene cijela aplikacija i prikaže zaslon. Kako bi se doskočilo tom problemu izmišljen je drugi zavisan oblik programske klase – fragment. Fragment se ne može pokrenuti bez activitya jer osnovne bazične funkcije aplikacije ne rade. Pravilan pristup je da se napravi jedan activity koji brine o vitalnim dijelovima aplikacije i komunikaciji s memorijom, a svi ostali dijelovi aplikacije se pokreću pomoću fragmenata. Fragmenti imaju gotovo trenutno učitavanje, te je korisničko iskustvo ugodnije.

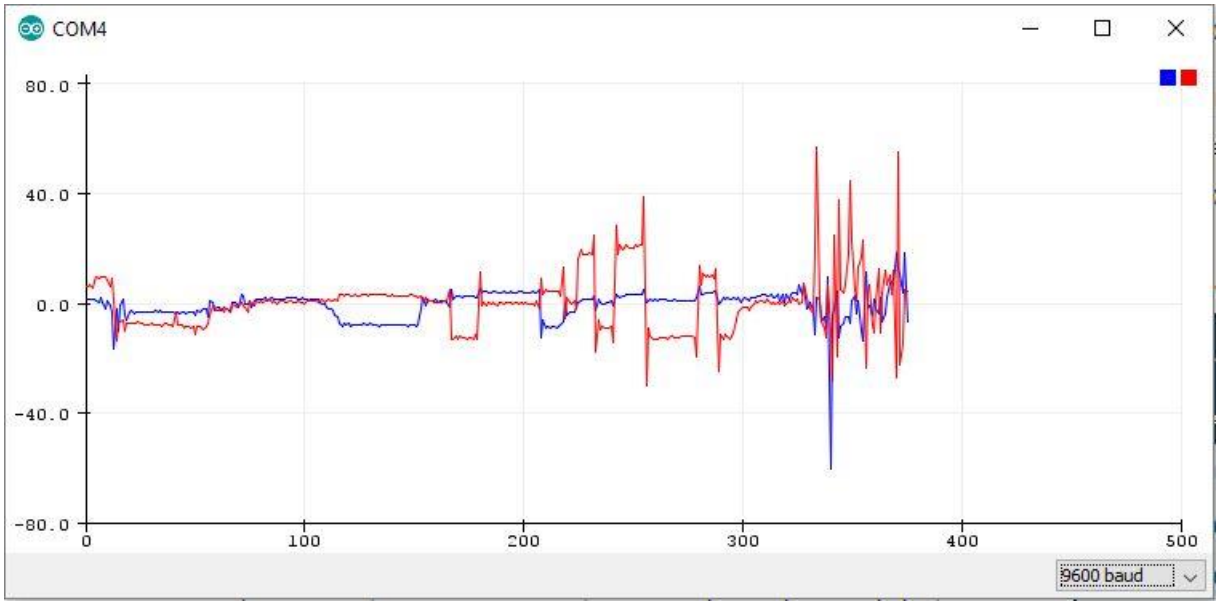
Aplikacija samobalansirajućeg robota ima samo jedan activity naziva MainActivity, a ostale klase su fragmenti i servisi. Postoji četiri fragmenta: About, Kontroler, PID i Senzori koji se svaki za sebe brine o određenom aspektu aplikacije. Npr. fragment PID brine o pokretanju xml datoteke zadužene za izgled ekrana na kojem su klizači za promjenu parametara PID regulatora. Također, obrađuje pozicije klizača te po pritisku tipke šalje informacije na bluetooth servis koji dalje prosljeđuje instrukcije prema robotu. Postoje još dvije klase servisa: Globals i BluetoothConnectionService. Servis Globals je svojevrsna baza globalnih varijabli koja omogućuje pristup globalnim varijablama iz bilo kojeg fragmenta aplikacije. Servis BluetoothConnectionService vrši svu obradu i pripremu podataka za slanje preko bluetootha te sadrži sav kod potreban za fluidnu komunikaciju između dva bluetooth uređaja. Kao primjer programa u Javi, prikazan je kod koji se aktivira pritiskom tipke PRIMIJENI na fragmentu ekrana za PID regulaciju:

```
button1.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {

            seekR = redSeekBar.getProgress();
            seekG = greenSeekBar.getProgress();
            seekB = blueSeekBar.getProgress();
            byte[] bytes = {(byte) 230, (byte) (seekR/2), (byte) (seekG/120),
(byte) (seekB), (byte) 230 };
            ((MainActivity) getActivity()).mBluetoothConnection.write(bytes);
        }
        return true;
    }
});
```

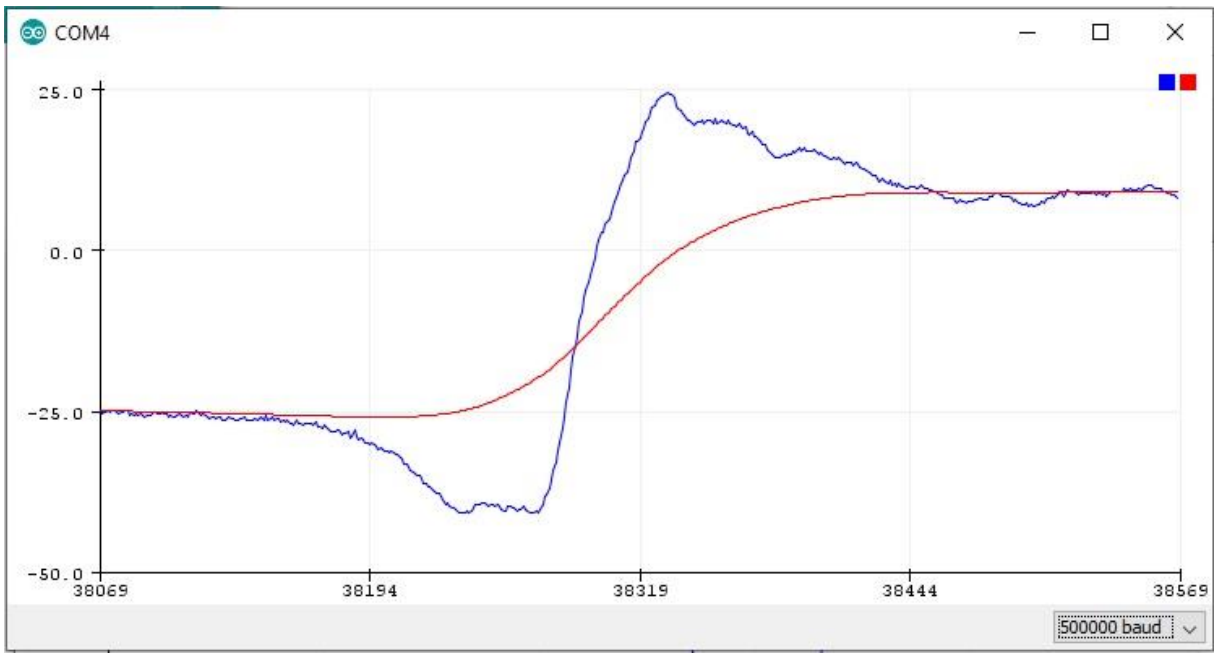
9. TESTIRANJE I REZULTATI

Prije primjene komplementarnog filtra, podaci o nagibu senzora po x i y osi imaju velike oscilacije zbog vibracija koje proizvode motori. Graf je vidljiv na slici ispod:



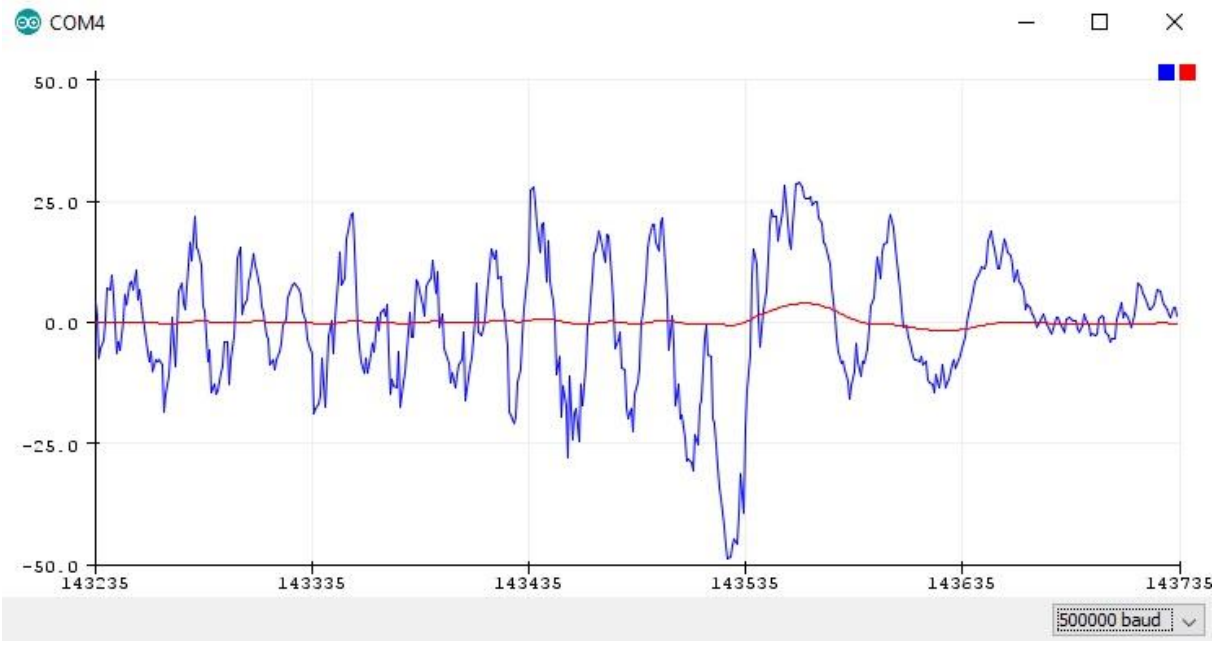
Slika 17. Nagib izračunat samo akcelerometrom

Nakon primjene komplementarnog filtra i pravilnog podešavanja, graf nagiba prilikom naglog trzaja tijela robota izgleda ovako:



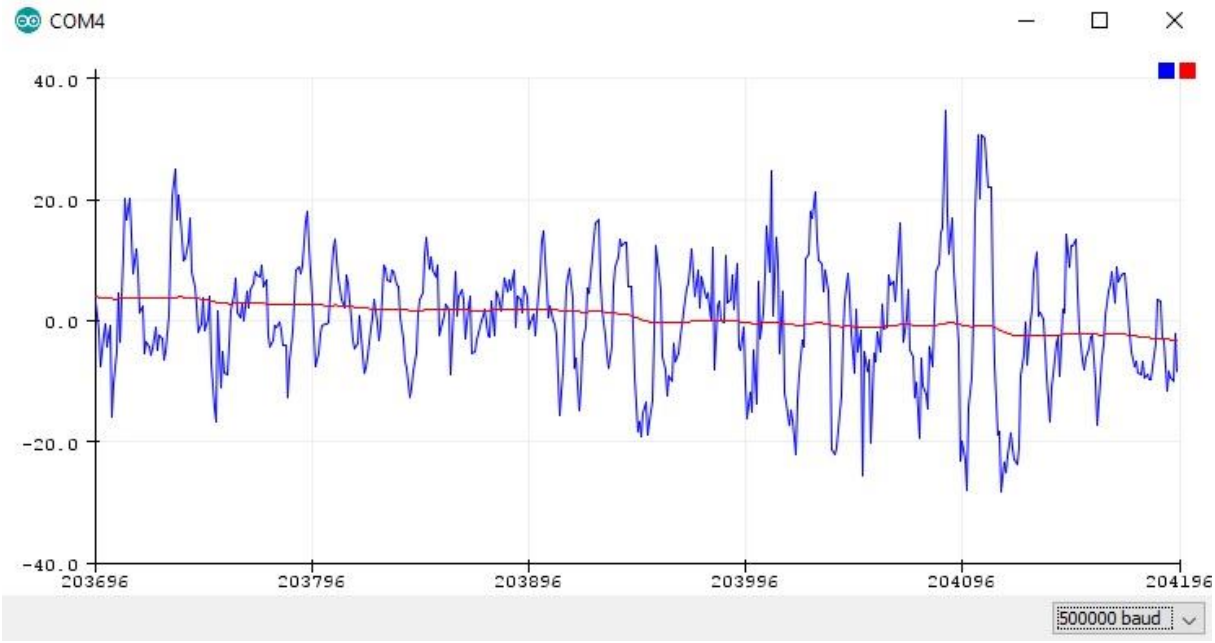
Slika 18. Prikaz razlike u nefiltriranom (plavo) i filtriranom (crveno) signalu

Prilikom nasumičnog laganog udarca na vrhu robota, zbog uspješno odrađene PID regulacije sustav se stabilizira. Odziv je prikazan grafom:



Slika 19. Prikaz stabilizacije sustava nakon udarca

Prilikom vožnje robota pomoću kontrolera na mobitelu, mijenja se željena vrijednost nagiba koju regulator nastoji držati. Tokom vožnje graf nagiba izgleda ovako:



Slika 20. Prikaz nagiba prilikom promjene smjera kretanja robota

10. PROCJENA VRIJEDNOSTI

Budući da je samobalansirajući robot fizički izveden, popis troškova je prikazan u tablici ispod. Navedene cijene su cijene na dan kupnje dijelova te mogu varirati s vremenom.

Tablica 1. Okvirne cijene komponenata robota

Komponenta	Cijena (kn)
Mikrokontroler ESP32	62,32
MPU6050 – akcelerometar i žiroskop	6,88
Nema17 motor – 2 komada	106,12
Driver TB6600 – 2 komada	100,12
Baterija Li-Po 4200mAh	190,64
BMS za bateriju	21,73
Kotač 75mm – 2 komada	30,00
Sklopka	3,2
Punjač za bateriju – 13,5V	152,35
Priključak za bateriju	4,50
Sklop za stabilizaciju napona na 5V	13,55
Čelični lim debljine 1,25mm	30,00
PVC ploča za poklopce	50,00
Žice	10,00
Vijci	5,00
Ukupno	786,41

11. ZAKLJUČAK

Ovim završnim radom prikazan je cijeli postupak pristupa problemu stabilizacije prirodno nestabilnog vozila na dva kotača. Na početku rada je dokazano da je sustav nestabilan te sam po sebi ne može održavati ravnotežu. Uvođenjem senzora koji daju informaciju o nagibu robota i pravilnim filtriranjem signala, robotu je omogućen uvid u trenutni nagib. Pomoću mikrokontrolera i dobivenog nagiba, provedena je regulacija sustava. Rezultati nakon regulacije pokazuju da je fizičku izvedbu robota moguće stabilizirati.

Postoji nekoliko poboljšanja koja je moguće izvesti na trenutnoj izvedbi samobalansirajućeg robota. Koračni motori Nema17 nemaju potrebe za velikim iznosima struje, pa je moguće koristiti manje izvedbe drivera umjesto korištenog drivera TB6600. Na koračne motore moguće je ugraditi enkodere koji bi davali informaciju o trenutnoj poziciji osovine motora. Time bi se spriječilo preskakanje koraka motora, te bi motori mogli postizati znatno veće momente pri velikim brzinama. Također, korištenjem enkodera mogle bi se eliminirati greške koje dovode do pomaka robota u prostoru, koji se trenutno događa jer robot nema povratnu informaciju giba li se on ili ne, već samo brine o tome da minimizira svoj nagib. Nakon uvođenja enkodera mogao bi se, umjesto PID regulatora, koristiti LQR regulator koji bi minimizirao greške u regulaciji. Za LQR regulator potrebno je poznavati linearne diferencijalne jednadžbe sustava, koje su za ovaj samobalansirajući robot izvedene u poglavlju 5.

Tokom izrade ovog završnog rada može se mnogo naučiti iz raznih područja mehatronike i robotike, od izrade matematičkog modela robota, simulacije dobivenog modela sustava, regulacije, programiranja mikrokontrolera, izrade mobilnih aplikacija, pa sve do konstruiranja fizičke izvedbe robotskog sustava te odabira aktivnih komponenti sustava. Sva ova znanja su vrlo korisna za daljnju edukaciju i budućeg inženjera mehatronike i robotike.

LITERATURA

[1] Silfer, T.: Control Synthesis for Balancing Robots, diplomski rad, 2018.

[2] Tomašić, T., Demetlika, A., Crneković, M.: Self-balancing mobile robot tilter, Transactions of famena XXXVI-3, 2012.

[3] https://fkeng.blogspot.com/2019/03/theory-and-design-of-two-wheels-self.html

[4] https://www.arduinolibraries.info/libraries/accel-stepper

[5] https://playground.arduino.cc/Code/PIDLibrary

[6] https://techtutorialsx.com/2018/03/09/esp32-arduino-serial-communication-over-bluetoth-hello-world

[7] https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/

[8] https://www.the-qrcode-generator.com/

[9] https://github.com/controlwear/virtual-joystick-android

[10] https://github.com/jjoe64/GraphView

[11] https://stackoverflow.com/questions/33461075/implement-bluetooth-connection-into-service-or-application-class-without-losing

[12] https://developer.android.com/guide/topics/connectivity/bluetooth

[13] https://grabcad.com/library/mks-tb6600-stepmot-driver-1

[14] https://grabcad.com/library/nema-17-stepper-motor-17hs4401-2

PRILOZI

I. CD-R disk
II. Kod mikrokontrolera
III. Kod android aplikacije
IV. Tehničke karakteristike koračnog motora Nema17
V. Tehničke karakteristike drivera za koračni motor TB6600
VI. Tehničke karakteristike akcelerometra i žiroskopa
VII. Tehničke karakteristike mikrokontrolera
VIII. Sklopni tehnički crtež robota

Kod mikrokontrolera:

```
#include <PID_v1.h>
#include <AccelStepper.h>
#include<Wire.h>
#include "BluetoothSerial.h"

TaskHandle_t Task1;
TaskHandle_t Task2;

BluetoothSerial SerialBT;

//Stepper motori (pinovi):
int EN_L = 1;
int CLK_L = 32;
int CW_L = 33;
int EN_D = 1;
int CLK_D = 25;
int CW_D = 26;

//MPU6050 + Komplementarni filtar:
double accX, accY, accZ;
double pitch;
double gyroY;
double compAngleY; // Kut izračunat korištenjem komplementarnog filtra
uint32_t timer;
uint8_t i2cData[14]; // Buffer za I2C podatke

//Brzina stepper motora:
int brzina_L;
int brzina_D;

//PID regulator:
double Setpoint, Input, Output;

//Konstante
double Kp=230, Ki=25000, Kd=0.01;
double Koeficijent_filtra = 0.997;
double korekcija_kuta = -2.3;
int vrijeme_sampliranja = 2;
int vrijeme_regulacije = 10;
int vrijeme_slanja_bt = 20;
int vrijeme_primanja_bt = 10;

AccelStepper stepper_L(EN_L, CLK_L, CW_L);
AccelStepper stepper_D(EN_D, CLK_D, CW_D);
PID PIDreg(&Input, &Output, &Setpoint, Kp, Ki, Kd, REVERSE);

int brojac = 0;
String procitani_bajt;
int bt_kut = 0;
int bt_snaga = 0;
bool regulator_primanje = false;
double reg_p = 0;
double reg_i = 0;
double reg_d = 0;
int zadnja_reakcija_kontrolera = 0;
boolean start = false;
```

```
void setup()
{
  Serial.begin(500000);
  delay(100);

  //Gyro kod:
  Wire.begin();
  i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) = 1000Hz
  i2cData[1] = 0x00; // Disable FSYNC and set 260 Hz Acc filtering, 256 Hz
Gyro filtering, 8 KHz sampling
  i2cData[2] = 0x00; // Set Gyro Full Scale Range to ±250deg/s
  i2cData[3] = 0x00; // Set Accelerometer Full Scale Range to ±2g
  while (i2cWrite(0x19, i2cData, 4, false)); // Write to all four registers
  while (i2cWrite(0x6B, 0x01, true)); // disable sleep mode
  delay(100);

  //Bluetooth device name
  SerialBT.begin("ESP-Tilter");
  delay(100);

  //Multitasking:
  disableCore0WDT();
  xTaskCreatePinnedToCore(
                    Task1code,   /* Task function. */
                    "Task1",     /* name of task. */
                    10000,       /* Stack size of task */
                    NULL,        /* parameter of the task */
                    200,           /* priority of the task */
                    &Task1,      /* Task handle to keep track of created task
                    0);          /* pin task to core 0 */

  xTaskCreatePinnedToCore(
                    Task2code,   /* Task function. */
                    "Task2",     /* name of task. */
                    10000,       /* Stack size of task */
                    NULL,        /* parameter of the task */
                    0,           /* priority of the task */
                    &Task2,      /* Task handle to keep track of created task
                    0);          /* pin task to core 0 */


  //Stepper kod:
  stepper_L.setMaxSpeed(5000);
  stepper_D.setMaxSpeed(5000);
  delay(100);

  //PID kod:
  Input = 0;
  Setpoint = 0;
  PIDreg.SetMode(AUTOMATIC);
  PIDreg.SetOutputLimits(-5000, 5000);
  PIDreg.SetSampleTime(vrijeme_sampliranja);

  delay(500);
}
```

```
void loop()
{
  if(start == true){
    stepper_L.setSpeed(brzina_L);
    stepper_D.setSpeed(brzina_D);
    stepper_L.runSpeed();
    stepper_D.run-
Speed();
  }
}

void Task1code( void * pvParameters ){
  for(;;){
    uint32_t vrijeme_privremeno = millis();
    if ((vrijeme_privremeno % vrijeme_sampliranja) == 0) {
      Ocitavanje_senzora();
    }
    if ((vrijeme_privremeno % vrijeme_regulacije) == 0) {

      if(start == false && pitch > -0.5 && pitch < 0.5){
        start = true;
      }
      if(compAngleY > 30 || compAngleY < -30){
        if(start == false){
          Input = 0;
          Output = 0;
          Setpoint = 0;
        }else if(start == true){
          ESP.restart();
        }
      }

      if(start == true){
        Input = compAngleY;
      }
      PIDreg.Compute();
      if(Output < 50 && Output > -50){
        Output = 0;
      }
      brzina_L = Output - cos(bt_kut*DEG_TO_RAD)*bt_snaga*5;
      brzina_D = -Output - cos(bt_kut*DEG_TO_RAD)*bt_snaga*5;
    }
  }
}

void Task2code( void * pvParameters ){
  for(;;){
      if ((millis() % (vrijeme_slanja_bt)) == 0) {

        SerialBT.print((compAngleY),2);
        SerialBT.print("/");

      }
```

```
      if ((millis() % (vrijeme_primanja_bt)) == 0) {
        if (SerialBT.available()) {
            brojac++;
            procitani_bajt = SerialBT.read();
            if(procitani_bajt == "230"){
              if(regulator_primanje == false){
                regulator_primanje = true;
              }else{
                regulator_primanje = false;
              }
              brojac = 0;
            }

            if(regulator_primanje == true){
              if(brojac == 1){
                reg_p = procitani_bajt.toDouble()*2;
              }
              if(brojac == 2){
                reg_i = procitani_bajt.toDouble()*120;
              }
              if(brojac == 3){
                reg_d = procitani_bajt.toDouble()/1000;
                brojac = 0;

                PIDreg.SetTunings(reg_p, reg_i, reg_d);
                PIDreg.Compute();

                Serial.println("Reg_Kp:"+String(PIDreg.GetKp(),3));
                Serial.println("Reg_Ki:"+String(PIDreg.GetKi(),3));
                Serial.println("Reg_Kd:"+String(PIDreg.GetKd(),3));
              }
            }
            if(regulator_primanje == false){
              if(brojac == 1){
                bt_kut = procitani_bajt.toInt()*2;
              }
              if(brojac == 2){
                bt_snaga = procitani_bajt.toInt();
                brojac = 0;

                double Zeljeni_Setpoint =
sin(bt_kut*DEG_TO_RAD)*bt_snaga*0.05;

                if((Setpoint-Zeljeni_Setpoint) < 0){
                  Setpoint = Setpoint + 0.1;
                }else if((Setpoint-Zeljeni_Setpoint) > 0){
                  Setpoint = Setpoint - 0.1;
                }

                zadnja_reakcija_kontrolera = millis();

              }
            }

          }
```

```
          if((millis() - zadnja_reakcija_kontrolera) > 100){
            if(Setpoint < 0){
              Setpoint = Setpoint + 0.01;
            }else if(Setpoint > 0){
              Setpoint = Setpoint - 0.01;
            }
            bt_snaga = 0;
            bt_kut = 0;
          }
      }
    }
}


void Ocitavanje_senzora(){
  while (i2cRead(0x3B, i2cData, 14));
  accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
  accY = (int16_t)((i2cData[2] << 8) | i2cData[3]);
  accZ = (int16_t)((i2cData[4] << 8) | i2cData[5]);
  gyroY = (int16_t)((i2cData[10] << 8) | i2cData[11]);

  double dt = (double)(micros() - timer) / 1000000; // Proracun delta_t
  timer = micros();

  pitch = (atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG) + ko-
rekcija_kuta;
  double gyroYrate = gyroY / 131.0;

  compAngleY = Koeficijent_filtra * (compAngleY + gyroYrate * dt) + (1-Koefi-
cijent_filtra) * pitch;

  //Serial.print(pitch);
  //Serial.print(" | ");
  //Serial.println(compAngleY);
}
```

Kod android aplikacije:

build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.app.gevtech.tilter"
        minSdkVersion 23
        targetSdkVersion 28
        versionCode 2
        versionName "Tilter v1.1"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnit-
Runner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-opti-
mize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support:support-v4:28.0.0'
    implementation 'com.android.support:design:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    implementation 'io.github.controlwear:virtualjoystick:1.10.1'
    implementation 'com.jjoe64:graphview:4.2.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-
core:3.0.2'
}
```

About.java

```
package com.app.gevtech.tilter;

import android.content.Context;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;



public class About extends Fragment {
    // TODO: Rename parameter arguments, choose names that match
    // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
    private static final String ARG_PARAM1 = "param1";
    private static final String ARG_PARAM2 = "param2";

    // TODO: Rename and change types of parameters
    private String mParam1;
    private String mParam2;


    public About() {
        // Required empty public constructor
    }

    /**
     * Use this factory method to create a new instance of
     * this fragment using the provided parameters.
     *
     * @param param1 Parameter 1.
     * @param param2 Parameter 2.
     * @return A new instance of fragment About.
     */
    // TODO: Rename and change types and number of parameters
    public static About newInstance(String param1, String param2) {
        About fragment = new About();
        Bundle args = new Bundle();
        args.putString(ARG_PARAM1, param1);
        args.putString(ARG_PARAM2, param2);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            mParam1 = getArguments().getString(ARG_PARAM1);
            mParam2 = getArguments().getString(ARG_PARAM2);
        }
    }
```

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_about, container, false);
    }

    // TODO: Rename method, update argument and hook method into UI event
    public void onButtonPressed(Uri uri) {

    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);

    }

    @Override
    public void onDetach() {
        super.onDetach();
    }

}
```

BluetoothConnectionService.java

```
package com.app.gevtech.tilter;

import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.util.Log;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;

/**
 * Bluetooth connection class (Library)
 */

public class BluetoothConnectionService {
    private static final String TAG = "BluetoothConnectionServ";

    private static final String appName = "MYAPP";

    private static final UUID MY_UUID_INSECURE =
            UUID.fromString("8ce255c0-200a-11e0-ac64-0800200c9a66");

    private final BluetoothAdapter mBluetoothAdapter;
    Context mContext;

    private AcceptThread mInsecureAcceptThread;

    private ConnectThread mConnectThread;
    private BluetoothDevice mmDevice;
    private UUID deviceUUID;
    ProgressDialog mProgressDialog;

    String poruka = "";
    String incomingMessage = "";

    Globals g = Globals.getInstance();

    private ConnectedThread mConnectedThread;

    public BluetoothConnectionService(Context context) {
        mContext = context;
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        start();
    }


    /**
     * This thread runs while listening for incoming connections. It behaves
     * like a server-side client. It runs until a connection is accepted
     * (or until cancelled).
     */
    private class AcceptThread extends Thread {
```

```
        // The local server socket
        private final BluetoothServerSocket mmServerSocket;

        public AcceptThread(){
            BluetoothServerSocket tmp = null;

            // Create a new listening server socket
            try{
                tmp = mBluetoothAdapter.listenUsingInsecureRfcommWithService-
Record(appName, MY_UUID_INSECURE);

                Log.d(TAG, "AcceptThread: Setting up Server using: " +
MY_UUID_INSECURE);
            }catch (IOException e){
                Log.e(TAG, "AcceptThread: IOException: " + e.getMessage() );
            }

            mmServerSocket = tmp;
        }

        public void run(){
            Log.d(TAG, "run: AcceptThread Running.");

            BluetoothSocket socket = null;

            try{
                // This is a blocking call and will only return on a
                // successful connection or an exception
                Log.d(TAG, "run: RFCOM server socket start.....");

                socket = mmServerSocket.accept();

                Log.d(TAG, "run: RFCOM server socket accepted connection.");

            }catch (IOException e){
                Log.e(TAG, "AcceptThread: IOException: " + e.getMessage() );
            }

            //talk about this is in the 3rd
            if(socket != null){
                connected(socket,mmDevice);
            }

            Log.i(TAG, "END mAcceptThread ");
        }

        public void cancel() {
            Log.d(TAG, "cancel: Canceling AcceptThread.");
            try {
                mmServerSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "cancel: Close of AcceptThread ServerSocket failed.
" + e.getMessage() );
            }
        }

    }
```

```
    /**
     * This thread runs while attempting to make an outgoing connection
     * with a device. It runs straight through; the connection either
     * succeeds or fails.
     */
    private class ConnectThread extends Thread {
        private BluetoothSocket mmSocket;

        public ConnectThread(BluetoothDevice device, UUID uuid) {
            Log.d(TAG, "ConnectThread: started.");
            mmDevice = device;
            deviceUUID = uuid;
        }

        public void run(){
            BluetoothSocket tmp = null;
            Log.i(TAG, "RUN mConnectThread ");

            // Get a BluetoothSocket for a connection with the
            // given BluetoothDevice
            try {
                Log.d(TAG, "ConnectThread: Trying to create InsecureRfcommSoc-
ket using UUID: "
                        +MY_UUID_INSECURE );
                tmp = mmDevice.createRfcommSocketToServiceRecord(deviceUUID);
            } catch (IOException e) {
                Log.e(TAG, "ConnectThread: Could not create InsecureRfcommSoc-
ket " + e.getMessage());
            }

            mmSocket = tmp;

            // Always cancel discovery because it will slow down a connection
            mBluetoothAdapter.cancelDiscovery();

            // Make a connection to the BluetoothSocket

            try {
                // This is a blocking call and will only return on a
                // successful connection or an exception
                mmSocket.connect();

                Log.d(TAG, "run: ConnectThread connected.");
            } catch (IOException e) {
                // Close the socket
                try {
                    mmSocket.close();
                    Log.d(TAG, "run: Closed Socket.");
                } catch (IOException e1) {
                    Log.e(TAG, "mConnectThread: run: Unable to close connec-
tion in socket " + e1.getMessage());
                }
                Log.d(TAG, "run: ConnectThread: Could not connect to UUID: " +
MY_UUID_INSECURE );
            }
```

```
            //will talk about this in the 3rd video
            connected(mmSocket,mmDevice);
        }
        public void cancel() {
            try {
                Log.d(TAG, "cancel: Closing Client Socket.");
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "cancel: close() of mmSocket in Connectthread fai-
led. " + e.getMessage());
            }
        }
    }



    /**
     * Start the chat service. Specifically start AcceptThread to begin a
     * session in listening (server) mode. Called by the Activity onResume()
     */
    public synchronized void start() {
        Log.d(TAG, "start");

        // Cancel any thread attempting to make a connection
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }
        if (mInsecureAcceptThread == null) {
            mInsecureAcceptThread = new AcceptThread();
            mInsecureAcceptThread.start();
        }
    }

    /**
     AcceptThread starts and sits waiting for a connection.
     Then ConnectThread starts and attempts to make a connection with the ot-
her devices AcceptThread.
     **/

    public void startClient(BluetoothDevice device,UUID uuid){
        Log.d(TAG, "startClient: Started.");

        //initprogress dialog
        mProgressDialog = ProgressDialog.show(mContext,"Connecting Bluetooth"
                ,"Please Wait...",true);

        mConnectThread = new ConnectThread(device, uuid);
        mConnectThread.start();
    }

    /**
     Finally the ConnectedThread which is responsible for maintaining the
BTConnection, Sending the data, and
     receiving incoming data through input/output streams respectively.
     **/
    private class ConnectedThread extends Thread {
```

```
        private final BluetoothSocket mmSocket;
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket) {
            Log.d(TAG, "ConnectedThread: Starting.");

            mmSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            //dismiss the progressdialog when connection is established
            try{
                mProgressDialog.dismiss();
            }catch (NullPointerException e){
                e.printStackTrace();
            }


            try {
                tmpIn = mmSocket.getInputStream();
                tmpOut = mmSocket.getOutputStream();
            } catch (IOException e) {
                e.printStackTrace();
            }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run(){
            byte[] buffer = new byte[1024];  // buffer store for the stream

            int bytes; // bytes returned from read()

            // Keep listening to the InputStream until an exception occurs
            while (true) {
                // Read from the InputStream
                try {
                    bytes = mmInStream.read(buffer);
                    incomingMessage = new String(buffer, 0, bytes);
                    poruka += incomingMessage;
                    for(int i = 0; i < poruka.length(); i++){
                        if(poruka.charAt(i) == '/'){
                            g.setData(poruka.substring(0, i));
                            poruka = poruka.substring(i+1);
                        }

                    }
                } catch (IOException e) {
                    break;
                }
            }
        }

        //Call this from the main activity to send data to the remote device
        public void write(byte[] bytes) {
```

```
            try {
                mmOutStream.write(bytes);
            } catch (IOException e) {
            }
        }

        /* Call this from the main activity to shutdown the connection */
        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) { }
        }
    }

    private void connected(BluetoothSocket mmSocket, BluetoothDevice mmDevice)
{
        Log.d(TAG, "connected: Starting.");

        // Start the thread to manage the connection and perform transmissions
        mConnectedThread = new ConnectedThread(mmSocket);
        mConnectedThread.start();
    }

    /**
     * Write to the ConnectedThread in an unsynchronized manner
     *
     * @param out The bytes to write
     * @see ConnectedThread#write(byte[])
     */
    public void write(byte[] out) {
        //perform the write
        mConnectedThread.write(out);
    }

}
```

Globals.java

```
package com.app.gevtech.tilter;

public class Globals{
    private static Globals instance;

    // Global variable
    private String data = "0";

    // Restrict the constructor from being instantiated
    private Globals(){}

    public void setData(String d){
        this.data=d;
    }
    public String getData(){
        return this.data;
    }

    public static synchronized Globals getInstance(){
        if(instance==null){
            instance=new Globals();
        }
        return instance;
    }
}
```

Kontroler.java

```
package com.app.gevtech.tilter;

import android.content.Context;
import android.os.Bundle;
import android.os.Handler;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.animation.RotateAnimation;
import android.widget.EditText;
import android.widget.ImageView;

import io.github.controlwear.virtual.joystick.android.JoystickView;

import static java.lang.Math.abs;


public class Kontroler extends Fragment {

    View view;

    String ispis = "0|0";
    int kut = 0;
    int snaga = 0;

    String nagib;
    float kut_nagiba = 0;
    float kut_prosli = 0;
    EditText kontroler_nagib;
    ImageView animacija;

    Globals g = Globals.getInstance();

    public Kontroler() {
        // Required empty public constructor
    }

    // TODO: Rename and change types and number of parameters
    public static Kontroler newInstance(String param1, String param2) {
        Kontroler fragment = new Kontroler();
        Bundle args = new Bundle();
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {

        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
```

```
        // Inflate the layout for this fragment
        view = inflater.inflate(R.layout.fragment_kontroler, container,
false);

        JoystickView joystick = view.findViewById(R.id.joystickView);
        joystick.setOnMoveListener(new JoystickView.OnMoveListener() {
            @Override
            public void onMove(int angle, int strength) {

                ispis = angle+"|"+strength;
                kut = angle;
                snaga = strength;
            }
        });

        kontroler_nagib = view.findViewById(R.id.kontroler_nagib);
        animacija = view.findViewById(R.id.animacija);


        final Handler handler = new Handler();
        final int delay = 10; //milliseconds

        handler.postDelayed(new Runnable() {
            public void run() {

                if(!ispis.equals("0|0")) {
                    try {
                        byte[] bytes = {(byte) (kut/2) , (byte) snaga};
                        ((MainActivity) getActivity()).mBluetoothConnec-
tion.write(bytes);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }

                try {

                    kut_nagiba = Float.parseFloat(g.getData());

                    if(abs(kut_nagiba) < 360) {

                        nagib = kut_nagiba + "°";
                        kontroler_nagib.setText(nagib);

                        final RotateAnimation rotateAnim = new RotateAnima-
tion(kut_prosli, kut_nagiba,
                                RotateAnimation.RELATIVE_TO_SELF, 0.49f,
                                RotateAnimation.RELATIVE_TO_SELF, 0.8f);
                        rotateAnim.setDuration(20);
                        rotateAnim.setFillAfter(true);
                        animacija.setAnimation(rotateAnim);
                        rotateAnim.start();

                        kut_prosli = kut_nagiba;
                    }
```

```
                } catch (Exception e) {
                    e.printStackTrace();
                }

                handler.postDelayed(this, delay);
            }
        }, delay);

        return view;
    }


    @Override
    public void onAttach(Context context) {
        super.onAttach(context);

    }

    @Override
    public void onDetach() {
        super.onDetach();

    }

}
```

MainActivity.java

```
package com.app.gevtech.tilter;


import android.Manifest;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.pm.ActivityInfo;
import android.content.res.ColorStateList;
import android.net.Uri;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentTransaction;
import android.util.Log;
import android.view.View;
import android.support.v4.view.GravityCompat;
import android.support.v7.app.ActionBarDrawerToggle;
import android.view.MenuItem;
import android.support.design.widget.NavigationView;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.WindowManager;
import java.util.Set;
import java.util.UUID;

public class MainActivity extends AppCompatActivity
        implements NavigationView.OnNavigationItemSelectedListener {

    private static final String TAG = "MainActivity";


    DrawerLayout drawer;
    NavigationView navigationView;
    FloatingActionButton fab;

    Boolean povezano = false;

    BluetoothAdapter mBluetoothAdapter;
    BluetoothConnectionService mBluetoothConnection;

    private static final UUID MY_UUID_INSECURE =
            UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    BluetoothDevice mBTDevice;


    //The BroadcastReceiver that listens for bluetooth broadcasts
    private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
```

```
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();

            if (BluetoothDevice.ACTION_ACL_CONNECTED.equals(action)) {
                povezano = true;
                fab.callOnClick();
            }
            else if (BluetoothDevice.ACTION_ACL_DISCONNECTED.equals(action)) {
                povezano = false;
                fab.callOnClick();
            }
        }
    };


    @Override
    protected void onDestroy() {
        Log.d(TAG, "onDestroy: called.");
        super.onDestroy();
        unregisterReceiver(mReceiver);
    }



    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, Win-
dowManager.LayoutParams.FLAG_FULLSCREEN);
        setContentView(R.layout.activity_main);

        getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_AD-
JUST_NOTHING);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        super.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_POR-
TRAIT);

        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);


        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

        IntentFilter filter = new IntentFilter();
        filter.addAction(BluetoothDevice.ACTION_ACL_CONNECTED);
        filter.addAction(BluetoothDevice.ACTION_ACL_DISCONNECTED);
        this.registerReceiver(mReceiver, filter);


        fab = findViewById(R.id.fab);
        fab.setBackgroundTintList(ColorStateList.valueOf(getResources().getCo-
lor(R.color.colorAccent)));
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Povezivanje . . . . . .", Snackbar.LEN-
GTH_LONG)
```

```
                        .setAction("Action", null).show();

                if (povezano) {
                    Snackbar.make(view, "Povezano.", Snackbar.LENGTH_LONG)
                            .setAction("Action", null).show();
                    fab.setBackgroundTintList(ColorStateList.valueOf(getReso-
urces().getColor(R.color.colorPrimary)));
                } else {
                    checkBTPermissions();
                    enableBT();

                    if(mBluetoothAdapter.isEnabled()){
                        Set<BluetoothDevice> pairedDevices = mBluetoothAdap-
ter.getBondedDevices();
                        if (pairedDevices.size() > 0) {
                            for (BluetoothDevice device : pairedDevices) {
                                if (device.getName().equals("ESP-Tilter")) {
                                    mBTDevice = device;
                                    mBTDevice.createBond();
                                    mBluetoothConnection = new BluetoothCon-
nectionService(MainActivity.this);
                                    startConnection();
                                    break;
                                }
                            }
                        }

                    }

                    Snackbar.make(view, "Neuspješno povezivanje. Je li uređaj
uparen?", Snackbar.LENGTH_LONG)
                            .setAction("Action", null).show();
                    fab.setBackgroundTintList(ColorStateList.valueOf(getReso-
urces().getColor(R.color.colorAccent)));
                }
            }
        });

        drawer = findViewById(R.id.drawer_layout);
        navigationView = findViewById(R.id.nav_view);
        ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
                this, drawer, toolbar, R.string.navigation_drawer_open,
R.string.navigation_drawer_close);
        drawer.addDrawerListener(toggle);
        toggle.syncState();
        navigationView.setNavigationItemSelectedListener(this);
        navigationView.setItemIconTintList(null);

        navigationView.getMenu().getItem(0).setChecked(true);
        loadFragment(new Kontroler());
    }

    //create method for starting connection
//***remember the conncction will fail and app will crash if you haven't pai-
red first
    public void startConnection(){
        startBTConnection(mBTDevice,MY_UUID_INSECURE);
```

```
    }

    /**
     * starting chat service method
     */
    public void startBTConnection(BluetoothDevice device, UUID uuid){
        Log.d(TAG, "startBTConnection: Initializing RFCOM Bluetooth Connec-
tion.");

        mBluetoothConnection.startClient(device,uuid);
    }


    public void enableBT(){
        if(mBluetoothAdapter == null){
            Log.d(TAG, "enableDisableBT: Does not have BT capabilities.");
        }
        if(!mBluetoothAdapter.isEnabled()){
            Log.d(TAG, "enableDisableBT: enabling BT.");
            Intent enableBTIntent = new Intent(BluetoothAdapter.ACTION_REQU-
EST_ENABLE);
            startActivity(enableBTIntent);
        }

    }

    /**
     * This method is required for all devices running API23+
     * Android must programmatically check the permissions for bluetooth. Put-
ting the proper permissions
     * in the manifest is not enough.
     *
     * NOTE: This will only execute on versions > LOLLIPOP because it is not
needed otherwise.
     */
    private void checkBTPermissions() {
        int permissionCheck = this.checkSelfPermission("Manifest.permis-
sion.ACCESS_FINE_LOCATION");
        permissionCheck += this.checkSelfPermission("Manifest.permission.AC-
CESS_COARSE_LOCATION");
        if (permissionCheck != 0) {

            this.requestPermissions(new String[]{Manifest.permission.AC-
CESS_FINE_LOCATION, Manifest.permission.ACCESS_COARSE_LOCATION}, 1001); //Any
number
        }
    }


    @Override
    public void onBackPressed() {
        if (drawer.isDrawerOpen(GravityCompat.START)) {
            drawer.closeDrawer(GravityCompat.START);
        } else {
            super.onBackPressed();
        }
    }
```

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is pre-
sent.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_1) {
            Intent shareIntent = new Intent(Intent.ACTION_SEND);
            shareIntent.setType("text/plain");
            shareIntent.putExtra(Intent.EXTRA_SUBJECT, "Tilter robot");
            String shareMessage = "\nSkini aplikaciju za kontroliranje robota
:\n\n";
            shareMessage = shareMessage +
"https://play.google.com/store/apps/details?id=" + BuildConfig.APPLICATION_ID
+ "\n\n";
            shareIntent.putExtra(Intent.EXTRA_TEXT, shareMessage);
            startActivity(Intent.createChooser(shareIntent, "Odaberi :"));
            return true;
        } else if (id == R.id.action_2) {
            Intent intent = new Intent(Intent.ACTION_SENDTO, Uri.fromParts(
                    "mailto", "matijaherceg8@gmail.com", null));
            intent.putExtra(Intent.EXTRA_SUBJECT, "Tilter robot");
            startActivity(Intent.createChooser(intent, "Kontaktiraj kreatora
:"));
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    @Override
    public boolean onNavigationItemSelected(MenuItem item) {
        // Handle navigation view item clicks here.
        int id = item.getItemId();

        if (id == R.id.nav_1) {
            loadFragment(new Kontroler());
            fab.show();
        } else if (id == R.id.nav_2) {
            loadFragment(new Senzori());
            fab.show();
        } else if (id == R.id.nav_4) {
            loadFragment(new About());
            fab.hide();
        } else if (id == R.id.nav_6) {
            loadFragment(new PID());
```

```
            fab.show();
        }

        drawer.closeDrawer(GravityCompat.START);
        return true;
    }

    private void loadFragment(Fragment fragment) {
        // create a FragmentManager
        FragmentManager fm = getSupportFragmentManager();
        // create a FragmentTransaction to begin the transaction and replace
the Fragment
        FragmentTransaction fragmentTransaction = fm.beginTransaction();
        // replace the FrameLayout with new Fragment
        fragmentTransaction.replace(R.id.fragmentLayout, fragment);
        fragmentTransaction.commit(); // save the changes
    }

}
```

PID.java

```
package com.app.gevtech.tilter;

import android.annotation.SuppressLint;
import android.bluetooth.BluetoothDevice;
import android.content.Context;
import android.content.IntentFilter;
import android.net.Uri;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v4.app.Fragment;
import android.util.TypedValue;
import android.view.LayoutInflater;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;
import android.view.ViewTreeObserver;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.TextView;

import java.nio.charset.Charset;


public class PID extends Fragment {
    // TODO: Rename parameter arguments, choose names that match
    // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
    View view;
    Button button1;

    int seekR, seekG, seekB;
    SeekBar redSeekBar, greenSeekBar, blueSeekBar;

    TextView text_seekbar_red, text_seekbar_green, text_seekbar_blue;

    // TODO: Rename and change types of parameters
    private String mParam1;
    private String mParam2;


    public PID() {
        // Required empty public constructor
    }

    /**
     * Use this factory method to create a new instance of
     * this fragment using the provided parameters.
     *
     * @param param1 Parameter 1.
     * @param param2 Parameter 2.
     * @return A new instance of fragment PID.
     */
    // TODO: Rename and change types and number of parameters
    public static PID newInstance(String param1, String param2) {
        PID fragment = new PID();
        Bundle args = new Bundle();

        fragment.setArguments(args);
```

```
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {

        }
    }

    @SuppressLint("ClickableViewAccessibility")
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup con-
tainer, Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        view = inflater.inflate(R.layout.fragment_pid, container, false);


        redSeekBar = view.findViewById(R.id.red);
        greenSeekBar = view.findViewById(R.id.green);
        blueSeekBar = view.findViewById(R.id.blue);

        text_seekbar_red = view.findViewById(R.id.text_seekbar_red);
        text_seekbar_green = view.findViewById(R.id.text_seekbar_green);
        text_seekbar_blue = view.findViewById(R.id.text_seekbar_blue);

        button1 = view.findViewById(R.id.button1);

        view.getViewTreeObserver().addOnGlobalLayoutListener(new ViewTree-
Observer.OnGlobalLayoutListener() {
            @SuppressLint("NewApi")
            @Override
            public void onGlobalLayout() {
                //now we can retrieve the width and height
                int height = view.getHeight();

                int visina_teksta_gumba = height / 35;

                button1.setTextSize(TypedValue.COMPLEX_UNIT_PX, visina_tek-
sta_gumba);

                updateBackground();

                text_seekbar_red.setTextSize(TypedValue.COMPLEX_UNIT_PX, red-
SeekBar.getHeight()/2);
                text_seekbar_green.setTextSize(TypedValue.COMPLEX_UNIT_PX,
greenSeekBar.getHeight()/2);
                text_seekbar_blue.setTextSize(TypedValue.COMPLEX_UNIT_PX, blu-
eSeekBar.getHeight()/2);

                view.getViewTreeObserver().removeOnGlobalLayoutListener(this);

            }
        });

        button1.setOnTouchListener(new View.OnTouchListener() {
```

```
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    button1.setBackgroundColor(getActivity().getColor(R.co-
lor.buttonHighlight));

                    seekR = redSeekBar.getProgress();
                    seekG = greenSeekBar.getProgress();
                    seekB = blueSeekBar.getProgress();
                    try {
                        byte[] bytes = {(byte) 230, (byte) (seekR/2), (byte)
(seekG/120), (byte) (seekB), (byte) 230 };
                        ((MainActivity) getActivity()).mBluetoothConnec-
tion.write(bytes);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                else if (event.getAction() == MotionEvent.ACTION_UP) {
                    button1.setBackgroundColor(getActivity().getColor(R.co-
lor.colorAccent));
                }

                return true;
            }
        });

        updateBackground();

        redSeekBar.setOnSeekBarChangeListener(seekBarChangeListener);
        greenSeekBar.setOnSeekBarChangeListener(seekBarChangeListener);
        blueSeekBar.setOnSeekBarChangeListener(seekBarChangeListener);

        return view;
    }

    // TODO: Rename method, update argument and hook method into UI event
    public void onButtonPressed(Uri uri) {

    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);

    }

    @Override
    public void onDetach() {
        super.onDetach();
    }

    private SeekBar.OnSeekBarChangeListener seekBarChangeListener
            = new SeekBar.OnSeekBarChangeListener()
    {

        @Override
```

```
        public void onProgressChanged(SeekBar seekBar, int progress,
                                      boolean fromUser) {
            // TODO Auto-generated method stub
            updateBackground();

        }
        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {
            // TODO Auto-generated method stub
        }

        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {
            // TODO Auto-generated method stub
        }
    };

    private void updateBackground()
    {
        seekR = redSeekBar.getProgress();
        seekG = greenSeekBar.getProgress();
        seekB = blueSeekBar.getProgress();

        int val1 = redSeekBar.getWidth() - redSeekBar.getPaddingLeft() - red-
SeekBar.getPaddingRight();
        text_seekbar_red.setText(String.valueOf(seekR));
        text_seekbar_red.setX(redSeekBar.getPaddingLeft() - text_seek-
bar_red.getWidth()/2 + val1 * redSeekBar.getProgress() / redSeekBar.getMax());

        int val2 = greenSeekBar.getWidth() - greenSeekBar.getPaddingLeft() -
greenSeekBar.getPaddingRight();
        text_seekbar_green.setText(String.valueOf(seekG));
        text_seekbar_green.setX(greenSeekBar.getPaddingLeft() - text_seek-
bar_green.getWidth()/2 + val2 * greenSeekBar.getProgress() / greenSeekBar.get-
Max());

        int val3 = blueSeekBar.getWidth() - blueSeekBar.getPaddingLeft() -
blueSeekBar.getPaddingRight();
        text_seekbar_blue.setText(String.valueOf((float)seekB/1000));
        text_seekbar_blue.setX(blueSeekBar.getPaddingLeft() - text_seek-
bar_blue.getWidth()/2 + val3 * blueSeekBar.getProgress() / blueSeekBar.get-
Max());

    }

}
```

Senzori.java

```
package com.app.gevtech.tilter;

import android.content.Context;
import android.net.Uri;
import android.os.Bundle;
import android.os.Handler;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.series.DataPoint;
import com.jjoe64.graphview.series.LineGraphSeries;
import java.util.Random;
import static java.lang.Math.abs;


public class Senzori extends Fragment {

    View view;
    GraphView graph;
    private static final Random RANDOM = new Random();
    private LineGraphSeries<DataPoint> series;
    private int lastX = 0;

    Globals g = Globals.getInstance();
    float kut_nagiba = 0;

    public Senzori() {
        // Required empty public constructor
    }

    // TODO: Rename and change types and number of parameters
    public static Senzori newInstance(String param1, String param2) {
        Senzori fragment = new Senzori();
        Bundle args = new Bundle();
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {

        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        view = inflater.inflate(R.layout.fragment_senzori, container, false);


        graph = view.findViewById(R.id.graph);
        series = new LineGraphSeries<DataPoint>();
```

```
        series.setColor(getActivity().getColor(R.color.colorAccent));

        graph.addSeries(series);
        graph.getViewport().setYAxisBoundsManual(true);
        graph.getViewport().setMinY(-20);
        graph.getViewport().setMaxY(20);
        graph.getViewport().setXAxisBoundsManual(true);
        graph.getViewport().setMinX(0);
        graph.getViewport().setMaxX(100);
        graph.getGridLabelRenderer().setGridColor(getActivity().getColor(R.co-
lor.bijela));
        graph.getGridLabelRenderer().setHorizontalLabelsColor(getActi-
vity().getColor(R.color.bijela));
        graph.getGridLabelRenderer().setVerticalLabelsColor(getActivity().get-
Color(R.color.bijela));

        final Handler handler = new Handler();
        final int delay = 20; //milliseconds

        handler.postDelayed(new Runnable() {
            public void run() {

                addEntry();

                handler.postDelayed(this, delay);
            }
        }, delay);

        return view;
    }
    private void addEntry() {
        try {
            kut_nagiba = Float.parseFloat(g.getData());
        }catch (Exception e) {
            e.printStackTrace();
        }

        if(abs(kut_nagiba) < 360) {
            series.appendData(new DataPoint(lastX++, kut_nagiba), true, 200);
        }
    }
    // TODO: Rename method, update argument and hook method into UI event
    public void onButtonPressed(Uri uri) {

    }
    @Override
    public void onAttach(Context context) {
        super.onAttach(context);

    }
    @Override
    public void onDetach() {
        super.onDetach();
    }

}
```

Tehničke karakteristike koračnog motora Nema17:

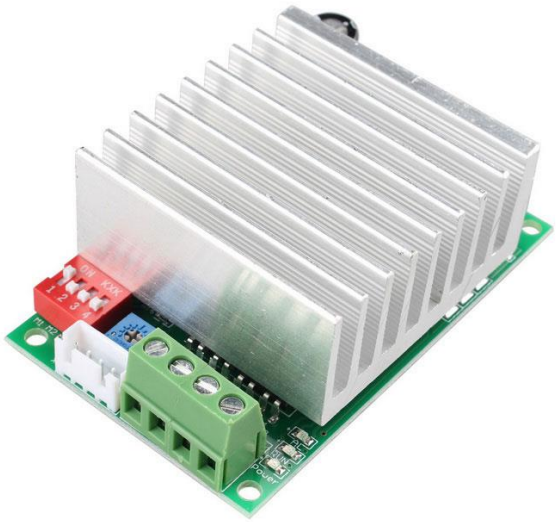
Tehničke karakteristike	
Moment držanja (Ncm)	42
Moment otpora – isključen motor (Ncm)	1,5
Moment inercije rotora ($kgcm^2$)	0,057
Masa (g)	230
Struja kroz zavojnicu (A)	1,5
Otpor zavojnice (ohm)	2,1
Induktivitet zavojnice (mH)	5,0

Signali i boje žica	
Zavojnica A - početak	Crvena
Zavojnica A - završetak	Plava
Zavojnica B - početak	Zelena
Zavojnica B - završetak	Crna

Više informacija na poveznici:

http://www.autoflexible.com/file_upload/product/attach/NEMA%2017.pdf

Tehničke karakteristike drivera za koračni motor TB6600:



Tehničke karakteristike	
Maksimalni napon na ulazu (V)	45
Maksimalna struja na izlazu (A)	4,5
Dimenzije (mm)	50 x 50 x 23
Mogućnost mikro koraka	1/1, 1/2, 1/4, 1/8, 1/16

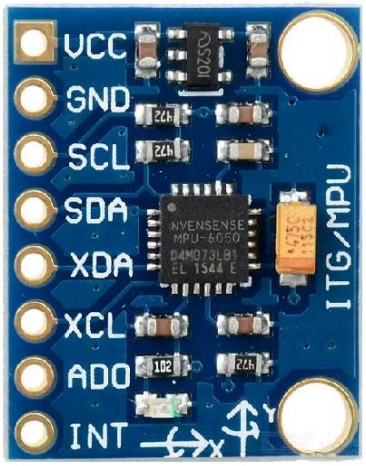
Dodatne karakteristike
Automatska regulacija struje
Automatsko isključivanje sklopa na previsokoj temperaturi
Isključivanje sklopa pri nedostatnom naponu
Zaštita od prevelikih struja
Optoizolacija ulaznih priključaka

Više informacija na poveznici:

https://aws.robu.in/wp-content/uploads/2017/11/TB6600HG_datasheet_en_20160610-1.pdf

https://aws.robu.in/wp-content/uploads/2017/11/TB6600-stepper-motor-Driver-Controller-ROBU.IN_.jpg

Tehničke karakteristike akcelerometra i žiroskopa:



Opis korištenih priključaka	
VCC	Ulazni napon sklopa (3,3 – 5 V)
GND	Priključak mase sustava
SCL	Priključak za clock impulse pri I2C komunikaciji
SDA	Priključak za prijenos informacija pri I2C komunikaciji

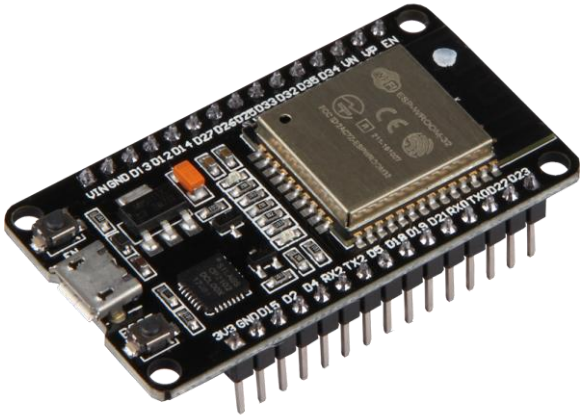
Tehničke karakteristike
Mikroelektromehanički 3 osni akcelerometar i 3 osni žiroskop
Ulazni napon: 3,3 – 5 V
Komunikacijski protokol: I2C
16-bitni analogno-digitalni pretvarači
Programibilna I2C adresa
Integrirani senzor temperature

Više informacija na poveznici:

https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

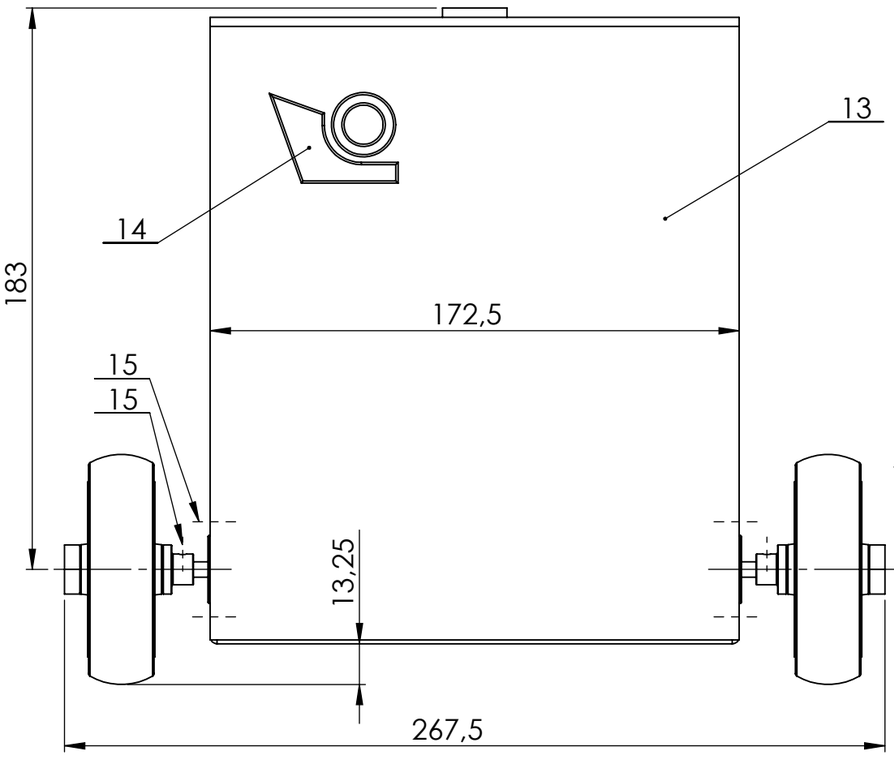
Tehničke karakteristike mikrokontrolera:

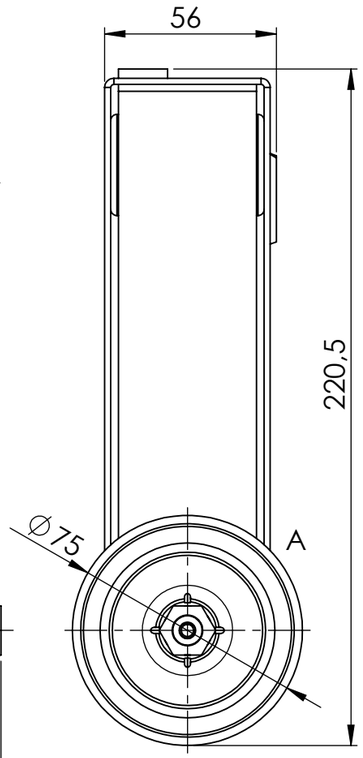


Tehničke karakteristike	
Mikroprocesor	Tensilica Xtensa LX6
Maksimalna frekvencija	240 MHz
Radni napon	3,3 V
Broj jezgri	2
Analogni ulazi	16 (12-bitni)
Digitalno-analogni pretvarači	2 (8-bitni)
Digitalni ulazno-izlazni priključci	39
Dopuštena struja na ulazima/izlazima	40 mA
Dopuštena struja na 3,3V priključku	50 mA
SRAM	520 KB
Komunikacijski priključci	SPI (4), I2C (2), I2S (2), CAN, UART (3)
Wi-Fi	802.11 b/g/n
Bluetooth	V4.2 - BLE + Classic

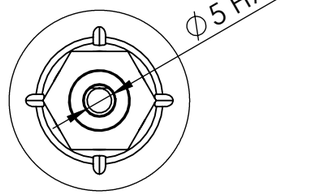
Više informacija na poveznici:

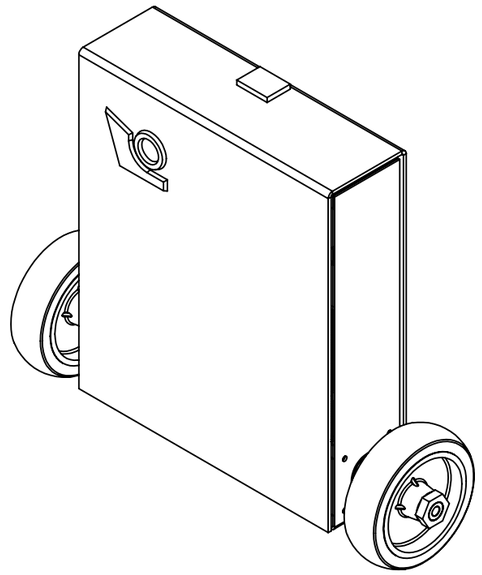
https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

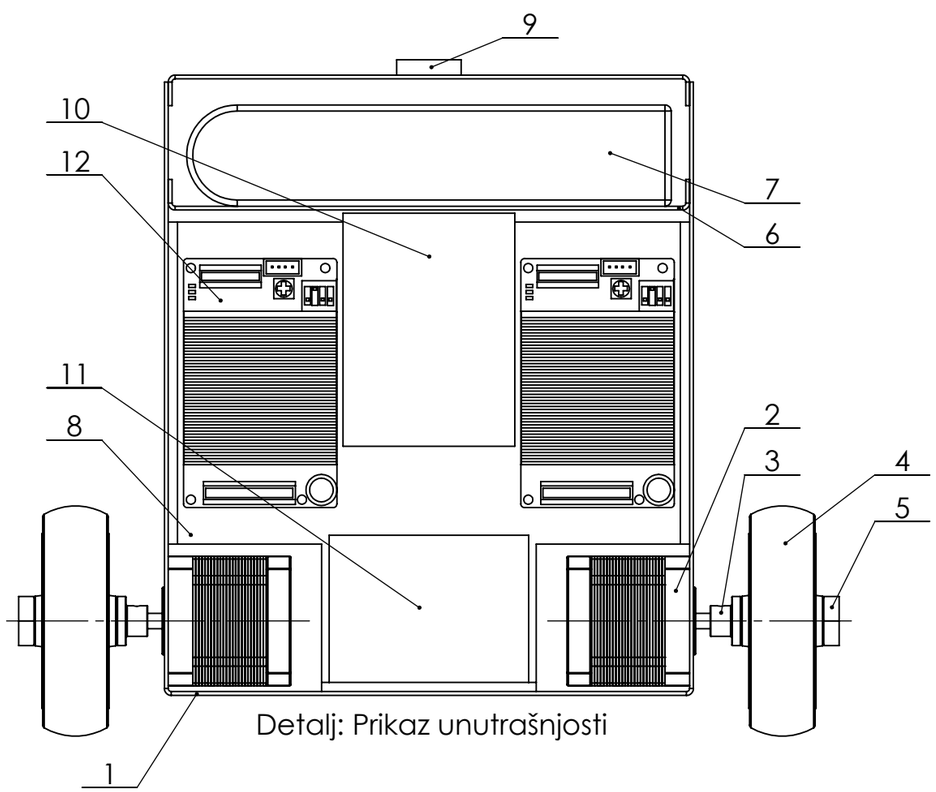




Detalj A
Mjerilo 1 : 1







Detalj: Prikaz unutrašnjosti

Poz.	Naziv dijela	Kom.	Crtež broj Norma	Materijal	Sirove dimenzije Proizvođač	Masa
15	Vijak M3 x 8	10	ISO 4762			
14	FSB logo	1				
13	Poklopac robota	1		PVC		
12	Driver koračnog motora TB6600	2				
11	Pločica stabilizatora napona	1				
10	Pločica s mikrokontrolerom ESP32	1				
9	Senzor MPU6050	1				
8	Podloga za montažu komponenata	1		PVC		
7	Baterija + BMS	1				
6	Ojačanje kućišta	2		X12CrMnNiN17-7-5		
5	Matica M10	2	ISO 4036			
4	Kotač 75mm	2				
3	Adapter	2		X12CrMnNiN17-7-5		
2	Nema17 koračni motor	2				
1	Kućište robota	1		X12CrMnNiN17-7-5		

Broj naziva - code		Datum	Ime i prezime	Potpis
	Projektirao	10.02.20.	Matija Herceg-Rušec	
	Razradio	10.02.20.	Matija Herceg-Rušec	
	Crtao	10.02.20.	Matija Herceg-Rušec	
	Pregledao	10.02.20.	dr.sc. Mladen Crneković	
	Mentor		dr.sc. Mladen Crneković	

FSB Zagreb

ISO - tolerancije		Objekt:	Objekt broj:
Ø5 H7/h7	0,012 / -0,012		R. N. broj:
		Napomena:	Kopija
		Materijal:	Masa: 2,7 kg
		Mjerilo originala: M 1:2	Naziv: Sklop robota
		Crtež broj: R-0-1000-00	Pozicija: 00
			Format: A3
			Listova: 1
			List: 1

Design by CADLab

0 10 20 30 40 50 60 70 80 90 100