Samobalansirajući robot

Plavšić, Nemanja

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET OSIJEK

Sveučilišni studij

SAMOBALANSIRAJUĆI ROBOT

Diplomski rad

Nemanja Plavšić

Osijek, 2016.

ELEKTROTEHNIČKI FAKULTET OSIJEK
Sveučilište Josipa Jurja Strossmayera u Osijeku

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek,

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Nemanja Plavšić
Studij, smjer:	Diplomski studij, Procesno računarstvo
Mat. br. studenta, godina upisa:	D-675R, 2013.
Mentor:	doc.dr.sc. Davor Vinko
Sumentor:	-
Predsjednik Povjerenstva:	
Član Povjerenstva:	
Naslov diplomskog rada:	Samobalansirajući robot
Primarna znanstvena grana rada:	Automatizacija i robotika
Sekundarna znanstvena grana (ili polje) rada:	Elektronika
Zadatak diplomskog rada:	Razviti sustav balansiranja robota te ga implemtirati u ATMEGA32 mikroupravljač i testirati na izrađenom prototupu.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 Postignuti rezultati u odnosu na složenost zadatka: 3 Jasnoća pismenog izražavanja: 2 Razina samostalnosti: II

Potpis sumentora:	Potpis mentora:

Dostaviti:

1. Studentska služba

U Osijeku, godine

Potpis predsjednika Odbora:

SADRŽAJ

1. UVOD

U ovom će radu biti opisan sustav balansiranja robota koji funkcionira po principu obrnutog njihala. Sagledat će se rješavanje problema sa gledišta izrade gdje je bitno težište u svrsi balansiranja, kao i elektroničkog sklopovlja, izbora motora i programiranja.

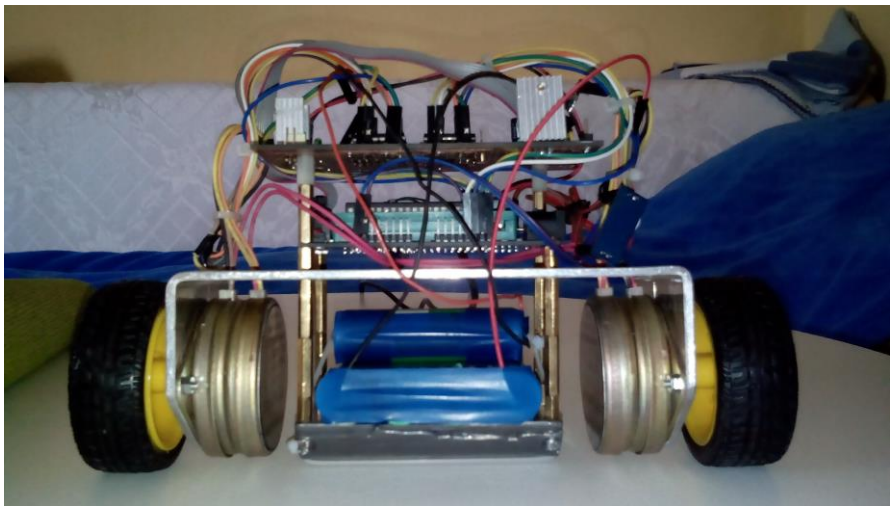
U drugom poglavlju su opisane komponente koje su se koristile, mikrokontoroler kao glavna jedinica za obradu podataka sa senzora i slanje izlaznih vrijednosti na aktuatore. Objašnjen je princip rada koračnih motore te njihova primjena i kontrola. Zatim, komunikacija s MPU6050 senzorom, njegovo povezivanje sa mikrokontrolerom i načinom kako se preko njega dobijaju vrijednost nagiba robota.

U trećem poglavlju je opisana shema i izrada štampane pločice koja je potrebna kako bi motori dobili dovoljno snage za pokretanje i regulatori napona koji osiguravaju izvor energije za mikrokotroler i motore.

U četvrtom poglavlju je opisan sustav balansiranja, način na koji su se sve komponente povezale i dijelovi programskog koda koji upravlja svim komponentama.

U zadnjem poglavlju je obrađena analiza rada samobalansirajućeg robota. Prikazan je graf balansiranja i graf odziva PD regulatora, te su opisane mane sustava i načini kako se može poboljšati balansiranje samog sustava.

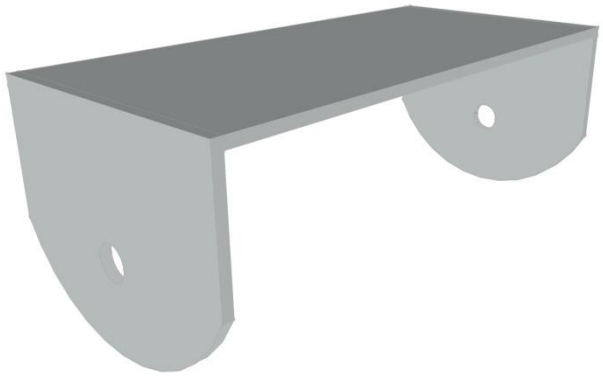
2. KOMPONENTE ROBOTA



Slika 2.1. Izgled samobalansirajućeg robota

2.1. Šasija robota

Šasija robota je napravljena od aluminija s dimenzijama 165x80x63 mm koja se vidi na slici 2.2. Napravljena je prema zahtjevima motora, njihovog položaja te položaja mikrokontrolera i napajanja. Cilj je postići čvrsto kućište u kojem je poželjno spuštanje težišta, jer će se s nižim težištem robot lakše balansirati. Zbog ograničenja motora dodan je uteg od olova u podnožju šasije ispod osi motora, kako bi se olakšalo balansiranje robota.



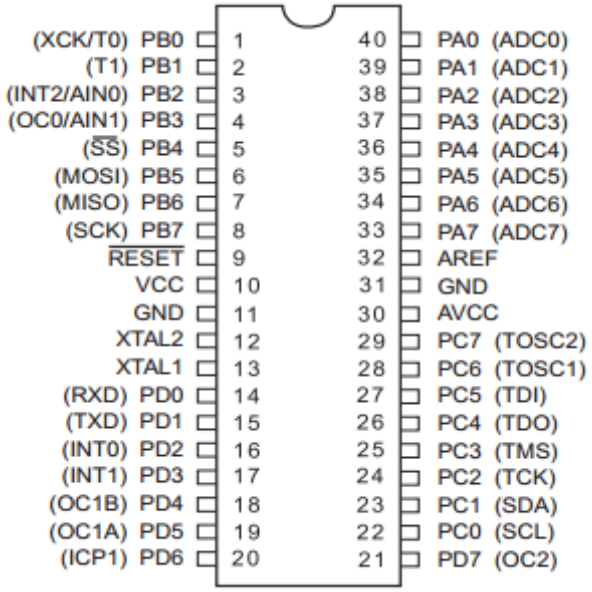
Slika 2.2. – Šasija robota

2.2. Atmel AVR ATMEGA 32

ATMEGA32 je 8-bitni AVR mikrokontroler s 32 KB programibilne flash memorije. Baziran je na naprednoj RISC arhitekturi sa 131 instrukcijom. Pregled nožica ATMEGA32 može se vidjeti na slici 2.3. [1]

Neke od značajki:

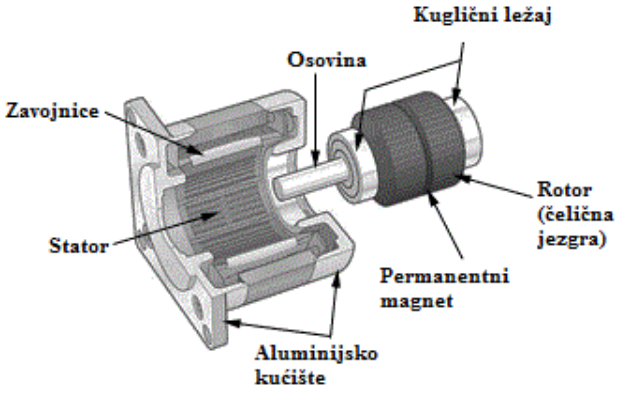
- 1024 Byte EEPROM
- 2 KByte SRAM
- JTAG sučelje
- Dva 8 bitna timer/counter-a i jedan 16 bitni timer/counter
- 8 kanalni, 10 bitni ADC
- I^2C serijska sabirnica
- 32 programibilne ulazno/izlazne linije
- 2.7 V – 5.5 V napon rada
- 0 – 16 MHz brzine



Slika 2.3. Pregled nožica ATMEGA32 mikrokontrolera [1]

2.3. Koračni motori

Koračni motori su motori bez četkica koji dijele pun krug na broj koraka jednake veličine. Koračni motori se sastoje od zavojnica i permanentnog magneta u sredini. Broj koraka ovisi o broju polova u permanenom magnetu. Na slici 2.4. su prikazani dijelovi koračnog motora.



Slika 2.4. Dijelovi koračnog motora [2]

Značajke motora:

- Korak : 7.5°
- Napon : 12V
- Fazna struja : 0.33A
- Fazni otpor : 36Ω
- Fazni induktivitet : 20mH
- Zaustavna sila 800g/cm

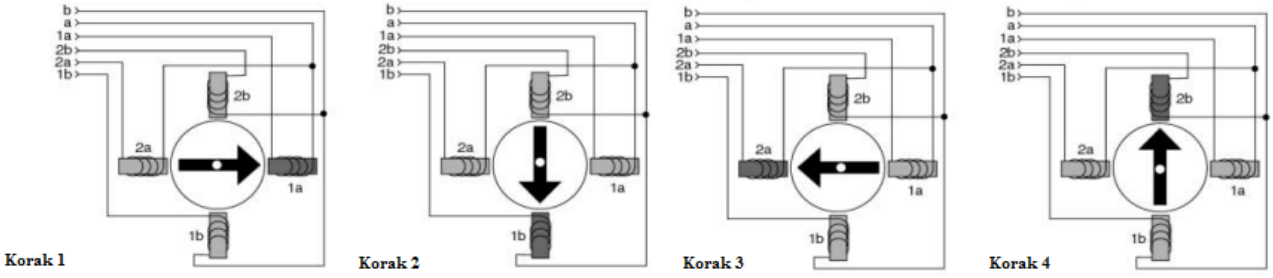
Motori ostvaruju rotaciju tako što slijedno puštamo struju kroz zavojnice. Motori se inače sastoje od dvije zavojnice kroz koje propuštamo struju određenim redoslijedom te ostvarimo rotaciju. Taj način ostvarivanja rotacije se zove „puni korak“ sa propuštanjem struje kroz jednu zavojnicu u trenutku, što se može vidjeti na slici 2.5., a sekvence paljenja faza u tablici 2.1.

Drugi način je propuštanje struje kroz dvije zavojnice što isto rezultira jednakim brojem koraka po jednom krugu samo što je veći moment sile. Ovaj pristup se može vidjeti na slici 2.6., a sekvence paljenja faza u tablici 2.2.

Mješavinom ova dva načina se može udvostručiti broj koraka koračnog motora te dobiti još sporija i preciznija rotacija. Taj način se zove „polu korak“ i može se vidjeti na slici 2.7., a sekvence paljenja faza u tablici 2.3. [3]

Tablica 2.1. Sekvence paljenja koračnih motora za puni korak s jednom fazom

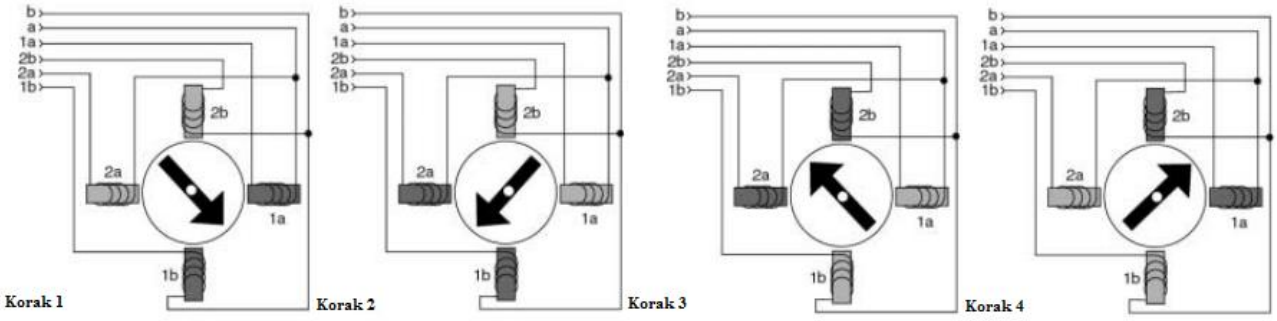
Korak	1	2	3	4
1A	1	0	0	0
2A	0	1	0	0
1B	0	0	1	0
2B	0	0	0	1



Slika 2.5. Puni korak (struja prolazi kroz jednu fazu) [3]

Tablica 2.2. Sekvence paljenja koračnih motora za puni korak s dvije faze

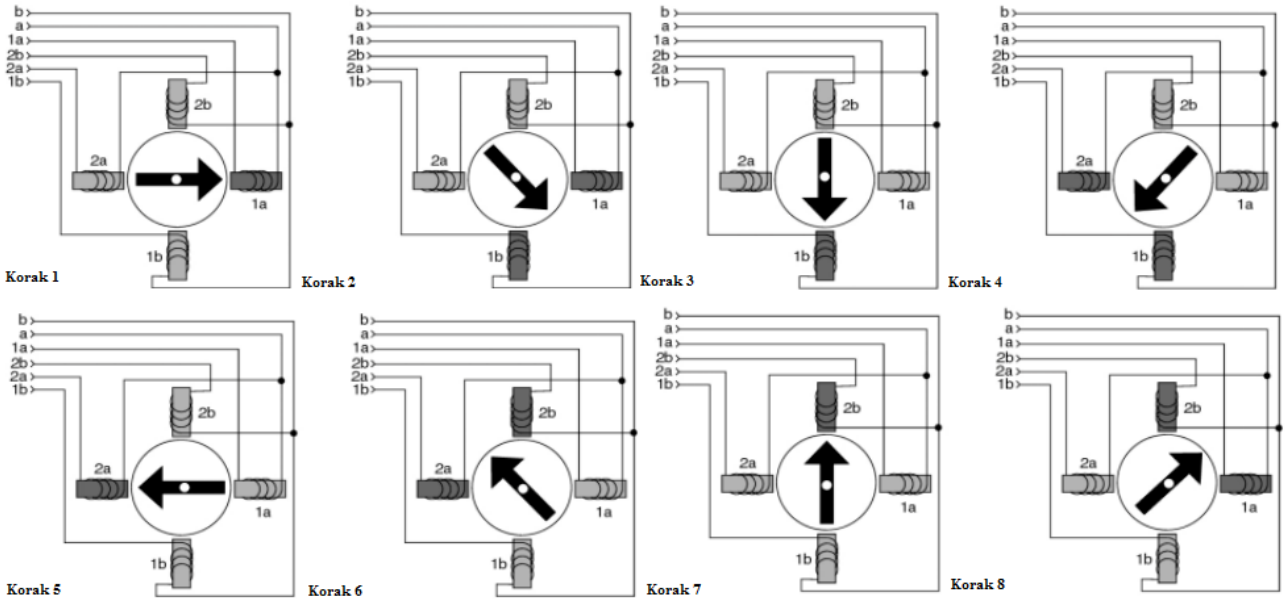
Korak	1	2	3	4
1A	1	0	0	1
2A	1	1	0	0
1B	0	1	1	0
2B	0	0	1	1



Slika 2.6. Puni korak (struja prolazi kroz dvije faze) [3]

Tablica 2.3. Sekvence paljenja koračnih motora za polu korak

Korak	1	2	3	4	5	6	7	8
1A	1	1	0	0	0	0	0	1
2A	0	1	1	1	0	0	0	0
1B	0	0	0	1	1	1	0	0
2B	0	0	0	0	0	1	1	1



Slika 2.7. Polu korak [3]

2.4. MPU6050

MPU6050 je senzor koji u sebi ima 3 osi za žiroskop i 3 osi za akcelerometar prema kojem se može odrediti položaj u prostoru. Izgled MPU6050 sklopa može se vidjeti na slici 2.8.

Slika 2.8. MPU6050 [4]

Sadrži dva 16-bitna AD konvertera za digitalizaciju žiroskopa i akcelerometra. Za precizno praćenje brzih i sporih kretnji MPU6050 omogućava korisniku skaliranje vrijednosti žiroskopa od $\pm 250, \pm 500, \pm 1000$ i $\pm 2000\,°/sec$ i za akcelerometar $\pm 2g, \pm 4g, \pm 8g$ i $\pm 16g$.Komunikacija se viši preko I^2C sabirnice u paketima od 8 bita. Izlazne nožice MPU6050 se mogu vidjeti u tablici 2.4. [5]

Tablica 2.4. Opis nožica sa MPU6050 čipa

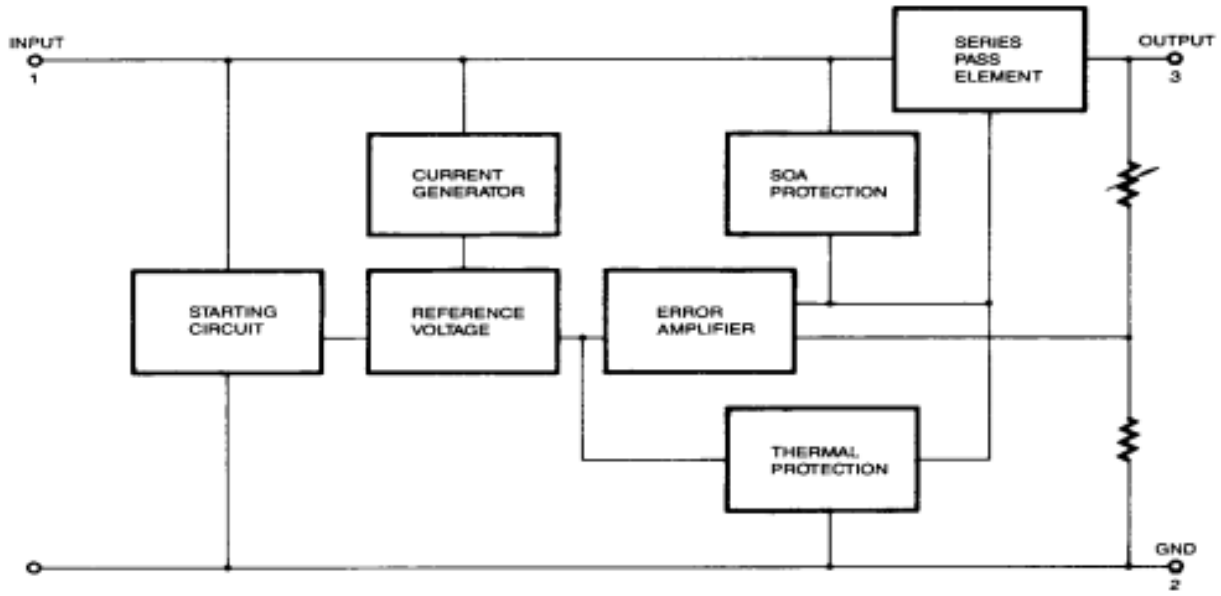
MPU6050	OPIS
VCC	Napajanje od 2.375V – 3.46V
GND	Uzemljenje
SCL	I^2C serijski signal takta
SDA	I^2C serijski podaci
XDA	Podaci za spajanja vanjskih senzora
XCL	Signal takta za spajanje vanjskih senzora
AD0	Slave adresa najmanje značajnog bita
INT	Interupt za digitalni izlaz

2.5. Regulatori napona LM7805, LM317 iAMS1117-3.3

LM7805 je pozitivni regulator napona na 5V. Dolazi u TO-220 pakovanju sa 3 nožice što ga čini korisnim u velikom broju primjena. Interni blok dijagram LM7805 se može vidjeti na slici 2.9., a tipično spajanje na slici 2.10. [6]

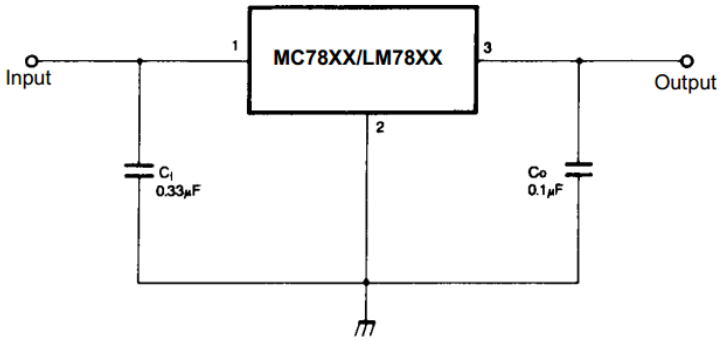
Značajke:

- Maksimalna izlazna struja do 1A
- Izlazni napon 5V
- Zaštita od toplinskog preopterećenja
- Zaštita od kratkog spoja



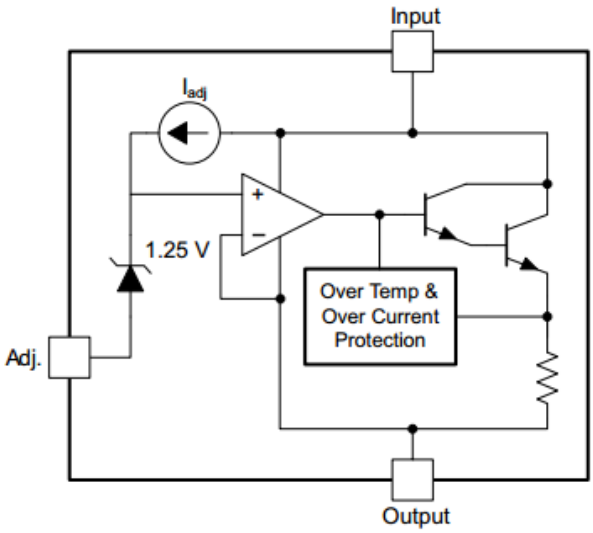
Slika 2.9. Interni dijagram LM7805 regulatora napona [6]

Na ovakav način se ostvari stabilni napon od 5V koji može propuštati struje do 1A u cilju napajanja mikrokontrolera i MPU6050 senzora.



Slika 2.10. Primjena LM7805 regulatora napona na 5V [7]

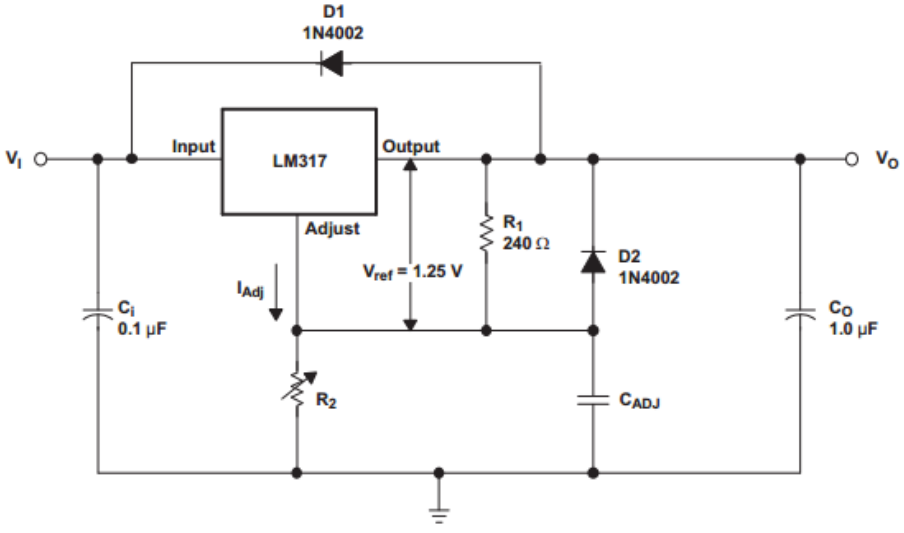
LM317 je pozitivni prilagodljivi regulator napona sposoban provesti 1.5 A struje kroz raspon napona od 1.25V do 37V. Dolazi u pakovanju TO-220 s 3 nožice isto kao i LM7805. Interni blok dijagram LM317 se može vidjeti na slici 2.11., a tipično spajanje na slici 2.12.



Slika 2.11. Blok dijagram LM317 pozitivnog prilagodljivog regulatora napona [7]

Značajke:

- Izlazni napon u rasponu od 1.25V do 37V
- Izlazna struja do 1.5A
- Zaštita od toplinskog opterećenja
- Zaštita od kratkog spoja



Slika 2.12. Primjena LM317 prilagodljivog regulatora napona [7]

Jednadžba prema kojoj se računa željeni izlazni napon se može vidjeti u (2-1).

$$V_o = V_{REF}\left(1 + \frac{R_2}{R_1}\right) + (I_{ADJ} * R_2) \quad (2\text{-}1)$$

gdje je:

- V_o – željeni izlazni napon
- V_{REF} – referentni napon, tipična vrijednost 1.25V
- R_1 i R_2 – otpornici čiji odnos određuje razinu izlaznog napona
- I_{ADJ} – tipična vrijednost 50μA i obično se zanemaruje

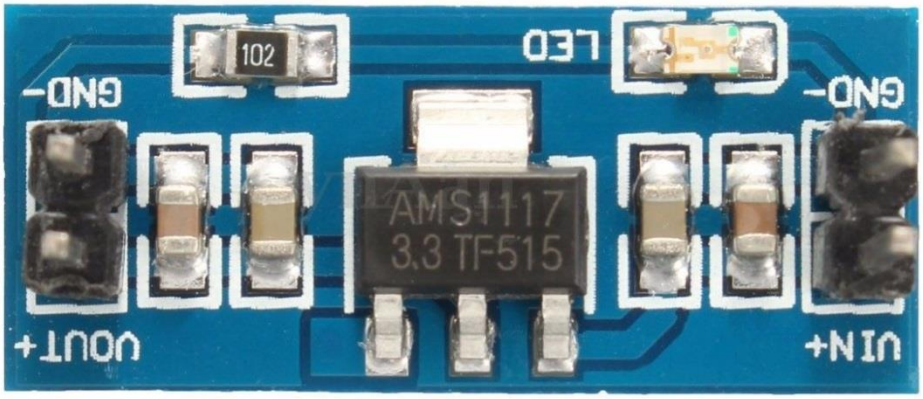
Kako bi se dobio izlazni napon od 12 V, koji je potreban za napajanje motora, u jednadžbu (2-1) se uvrste vrijednosti za V_o = 12V, R_2 = 4700 Ω, V_{REF} = 1.25 V i I_{ADJ} = 50μA te dobijemo vrijednost otpornika R_1 = 611,5 Ω. Budući da ne postoji otpornik od 611 Ω, mogu se uzeti otpornici od 510 Ω i 100 Ω čiji spoj u seriju daje otpornost od 610 Ω, što je približno željenoj vrijednosti. [7]

AMS1117- 3.3 je fiksni regulator napona sa 5.0V na 3.3 V koji je dizajniran za struje do 800mA i dolazi u SOT-223 pakovanju.

Značajke:

- Izlazni napon 3.3 V
- Izlazna struja do 800mA
- Linijska regulacija: 0.2% max.
    - sposobnost održavanja konstantnog izlaznog napona unatoč promjenama ulaznog napona.
- Regulacija opterećenja 0.4% max.
    - sposobnost za održavanje konstantne razine napone (ili struje) unatoč promjenama opterećenja na izvoru (kao što je promjena otpora).
- Zaštita od toplinskog opterećenja
- Zaštita od kratkog spoja

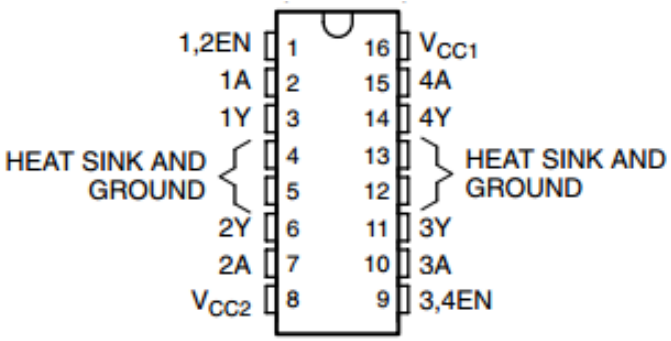
Ovaj regulator je potreban kako bi se napajao MPU6050 senzor na 3.3V, a izgled razvojne pločice sa AMS1117-3.3 regulatorom napona se može vidjeti na slici 2.13. [8]



Slika 2.13. Razvojna pločica sa AMS1117-3.3 regulatorom napona

2.6. L293D H-most driver

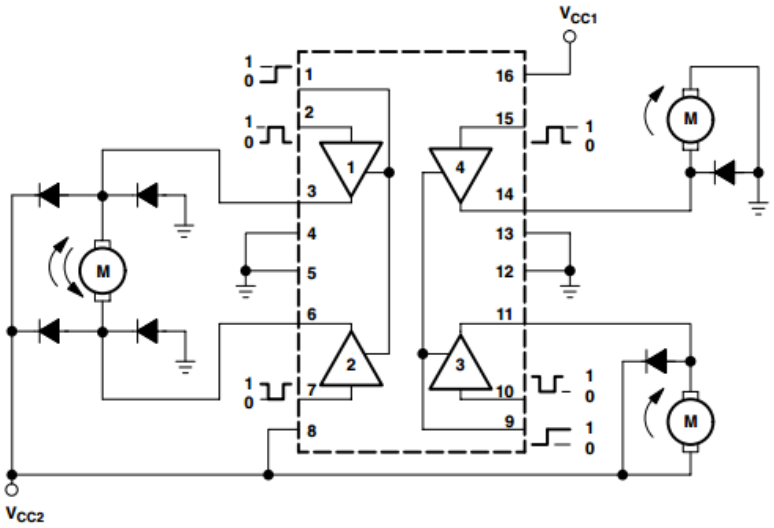
L293D je četverostruki polu H-most koji je dizajniran za struje do 600mA i raspon napona od 4.5V – 36V. Na slici 2.15. se mogu vidjeti izvodi nožica. [9]



Slika 2.14. Izvodi nožica L293D polu H-mosta [9]

Značajke:

- V_{cc1} napajanje čipa za logičke operacije od 4.5V do 7V
- V_{cc2} vanjsko napajanje za motore od V_{cc1} do 36V
- Kontinuirana struja do maksimalno 600mA



Slika 2.15. Shema spajanja L293D polu H-mosta [9]

Slika 2.15. prikazuje načine kontrole DC motora sa L293D čipom koji se upravljaju samo sa dvije žice.

Preko L293D čipa se mogu upravljati i koračni motori koji se upravljaju preko četiri žice, po dva para žica koji su krajevi zavojnice. Kako pojedini čip ima 4 ulaza i 4 izlaza, on je u mogućnosti upravljati samo sa jednim koračnim motorom.

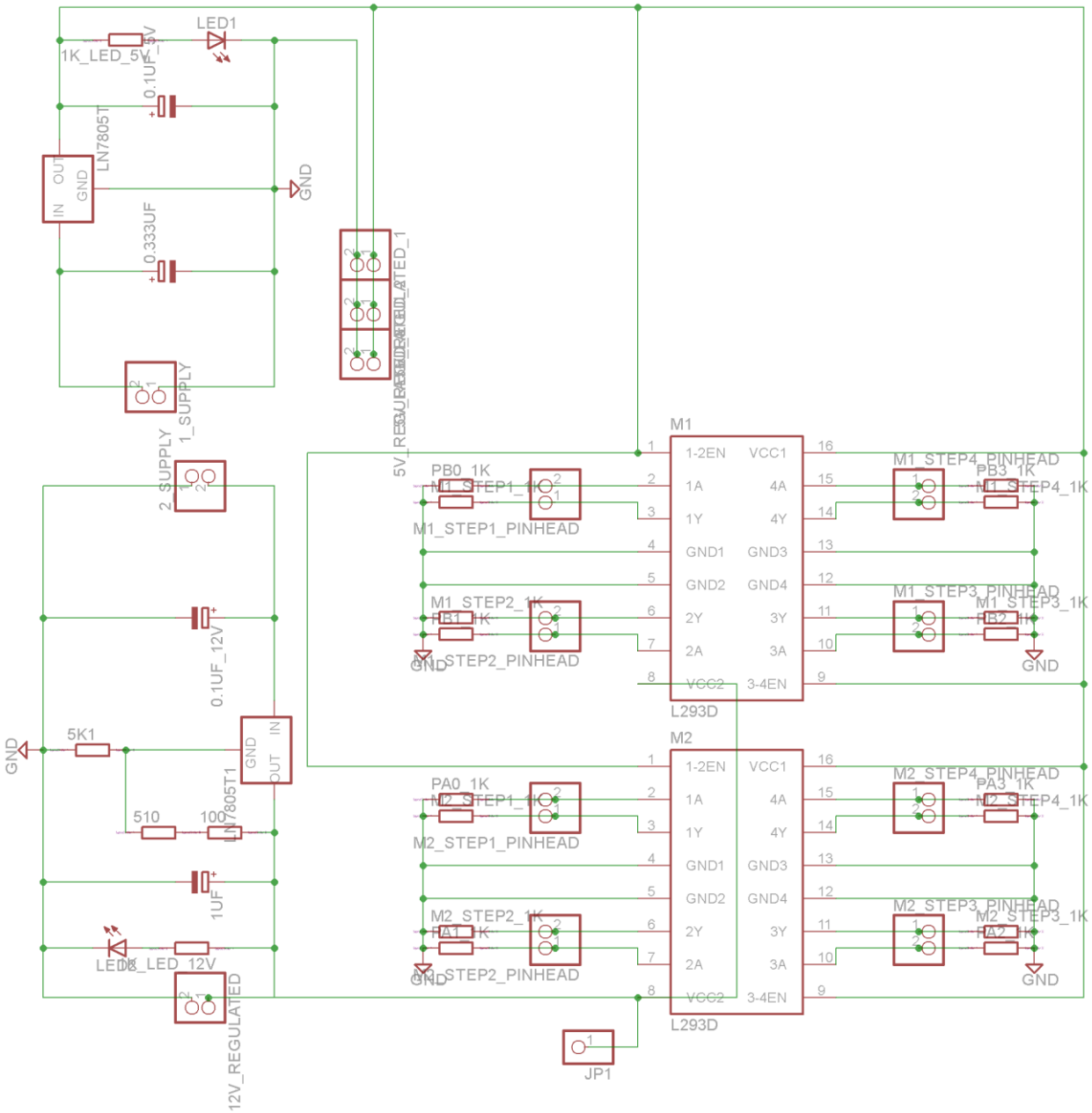
Na četiri ulazna pina se postavljaju izlazi sa mikrokontorlera preko kojeg se vrši upravljanje, a na izlazne pinove se postavljaju izvodi sa motora. U tablici 2.5. su prikazani izlazi sa mikrokontrolera koji upravljaju sa motorima.

Tablica 2.5. Izlazni signali sa mikrokontrolera koji upravljaju sa motorima

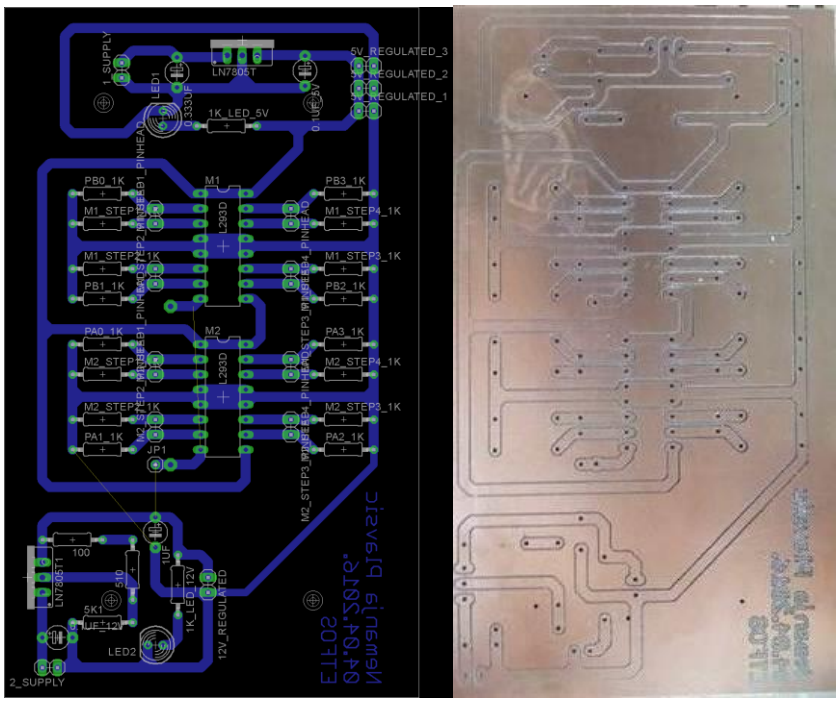
Port pin	**PA0**	**PA1**	**PA2**	**PA3**	**PB0**	**PB1**	**PB2**	**PB3**
Motor 1	1A	2A	1B	2B	X	X	X	X
Motor 2	X	X	X	X	1A	2A	1B	2B

3. IZRADA ŠTAMPANE PLOČICE

Mikrokontroler ne može pružiti dovoljno snage za napajanje motora, te se s toga pravi pogonska pločica sa L293D čipovima koji će osigurati dovoljan napon i struju kako bi motori mogli raditi. Koristit će se koračni motori koji po koraku povuku 330mA struje, a L293D mogu kontinuirano pružiti do 600mA struje. Na pločici su ugrađeni regulatori napona LM7805 za napajanje mikrokontrolera na 5V i LM317 za napajanje motora na 12V. Shema pogonske pločice se može vidjeti na slici 3.1., a dizajn štampane pločice i razvijene pločice na slici 3.2.



Slika 3.1. Shema pogonske pločice

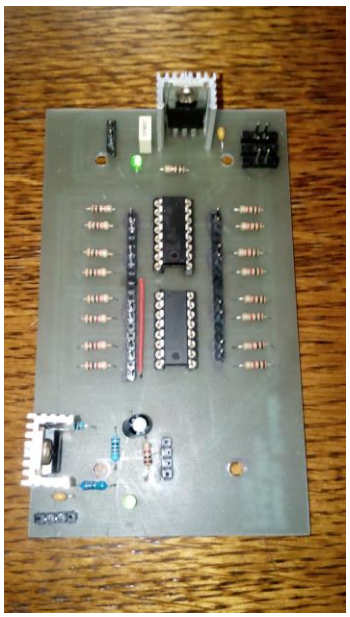


a) Dizajn štampane pločice u programskom alatu Eagle

b) Razvijena štampana pločica na vitroplastu

Slika 3.2. Štampana pločica

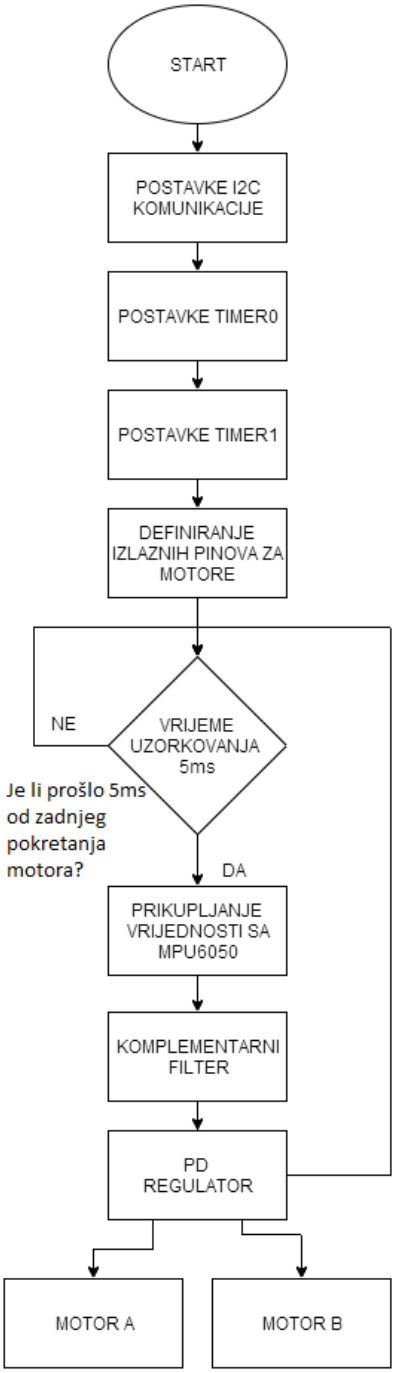
Nakon što se poleme sve komponente na izrađenu pogonsku pločicu, koja se može vidjeti na slici 3.3., imamo kontrolu motora sa 8 ulaza i 8 izlaza i 6 pinova za napajanje dodatnih sklopova na 5V. Na svaki ulazni i izlazni pin kod L239D čipova postavljeni su pull-down otpornici od 1 kΩ koji osiguravaju spuštanje signala na logičku nulu.



Slika 3.3. Izrađena pogonska pločica

4. SUSTAV BALANSIRANJA

Robot vrši balansiranje tako što mikrokontroler uzima vrijednosti nagiba robota periodično svakih 5 ms s MPU6050 žiroskopa i akcelerometra preko I^2C serijske sabirnice. Podaci se prvo filtriraju u komplementarnom filtru jer su vrijednosti s akcelerometra jako osjetljive te imaju puno šuma. Zatim se vrijednost kuta šalje PD regulatoru koji izračunava smjer i brzinu kretanja motora koji se kreću sukladno vrijednostima PD regulatora. Blok dijagram sustava se može vidjeti na slici 4.1.



Slika 4.1. Blok dijagram sustava balansiranja

4.1. I2C serijska komunikacija sa MPU6050

Mikrokontroler komunicira sa MPU6050 preko I^2C serijske sabirnice preko dvije nožice(SDA i SCL) što su na ATMEGA32 mikrokontroleru na portu B pinovi broj 23 i 22.
Komunikacija se ostvaruje tako što se generira signal takta preko mikrokontrolera na MPU6050 te se šalje start bit u svrhu pokretanja komunikacije. Nakon toga se šalje adresa uređaja u paketu od 8 bita i ACK bit koji potvrđuje uspješno primljenu adresu. Sljedeći 8-bitni paket su postavke unutrašnjih registara te se onda primaju podaci sa vrijednostima žiroskopa i akcelerometra na mikrokontroler. Frekvencija uzorkovanja podataka sa MPU6050 je 40 kH. Na slici 4.2. se može vidjeti C programski kod preko kojeg je ostvarena I^2C serijska komunikacija. [10]

```
float MPU6050_ReadAccel(int axis)//x = 0; y = 1; z = 2
{
    char reg = axis * 2 + 59;   //Accelerometar output  x=59    y=61    z=63
    float factor = 16384; //datasheet str. 13
    int val = 0;
    float float_val = 0;
    char ret = 0;

    ret = TWIM_ReadRegister(reg);
    val = ret << 8;

    ret = TWIM_ReadRegister(reg+1);
    val += ret;

    if (val & 1<<15)
    val -= 1<<16;

    float_val = val;

    float_val = float_val / factor;

    return float_val;
}
```

```
float MPU6050_ReadGyro(int axis)//x = 0; y = 1; z = 2
{
    char reg = axis * 2 + 67;   //Gyroscope output  x=67    y=69    z=71
    float factor = 131; // datasheet str. 12
    int val = 0;
    float float_val = 0;
    char ret = 0;

    ret = TWIM_ReadRegister(reg);
    val = ret << 8;

    ret = TWIM_ReadRegister(reg+1);
    val += ret;

    if (val & 1<<15)
    val -= 1<<16;


    float_val = val;

    float_val = float_val / factor;

    return float_val;
}
```

Slika 4.2. Funkcije za čitanje vrijednosti sa akcelerometra i žiroskopa

4.2. Računanje kuta sa akcelerometra

Podaci sa MPU6050 dobiveni preko I^2C serijske sabirnice su vrijednosti u radijanima koji se trebaju obraditi kako bi dobili vrijednost kuta u stupnjevima i odrediti kvadrant u kojem se nalazi. Način na koji se odredi u kojem se kvadrantu nalazi dobije se preko funkcije *atan2*koja prima dva argumenta za točno izračunavanje kvadranta u kojem se nalazi kut. Na slici 4.3. se nalazi C programski kod koji prima „sirove“ vrijednosti sa MPU6050, a kao izlaz daje vrijednosti kuta u stupnjevima.

```
float getAngleValues(float rawAccY, float rawAccZ)
{
    //Racunanje vrijednosti sa Accelerometra
    float rawAngle = (RAD_to_DEG * atan2(rawAccZ, rawAccY)) - 90;
    return rawAngle;
}
```

Slika 4.3. Dobijanje kuta u stupnjevima

4.3. Komplementarni filtar

Zbog velikog šuma u očitanjima akcelerometra potrebna je filtracija signala. Komplementarni filtar je dobar zbog svoje jednostavnosti i kratkog vremena izvođenja. Filtar izračunava trenutnu vrijednost kuta uzimajući vrijednosti sa žiroskopa, jer on nije toliko osjetljiv na šum. Vrijednost sa žiroskopa množi sa vremenom uzorkovanjaΔtte dobije trenutni kut. Zbrajajući trenutni kut s prethodno očitanim kutom možemo vidjeti gdje se trenutno nalazi robot. Također se uzima i vrijednost sa akcelerometra te se preko pojačanja (*eng*. Gain) kontrolira koliki udio vjerovanja se daje vrijednostima sa žiroskopa, a koliko sa akcelerometra.[11]
Jednadžba komplementarnog filtra se možete vidjeti u (4-1), a njegovu implementaciju u C programskom jeziku na slici 4.4.

$$\theta(t) = Gain * (\theta(t - \Delta t) + \omega(t) * dt) + (1 - Gain) * \varphi(t) \quad (4\text{-}1)$$

gdje je:

- $\theta(t)$ filtrirana vrijednost kuta u trenutku t
- $\theta(t - \Delta t)$ prethodno filtrirana vrijednost kuta
- $\omega(t)$ vrijednost sa žirokopa u trenutku t
- $\varphi(t)$ vrijednost sa akcelerometra u trenutku t
- $Gain$ je pojačanje komplemetarnog filtera
- Δt je vrijeme uzotkovanja

Za pojačanje je uzeta vrijednost od 0.92 jer u stacionarnom stanju daje najprecizniju vrijednost kuta što se može vidjeti u tablici 4.1.
Do koeficijenata se došlo tako što se uzela srednja vrijednost dijeljenja kuta dobivenog sa žiroskopa sa filtriranim kutom komplementarnim filtrom. Jednadžba po kojoj se izračunavao koeficijent preciznosti se može vidjeti u jednadžbi (4-2).

$$Q = \frac{1}{n}\sum_{i=1}^{n}\frac{\varphi_i}{\theta_i} \tag{4-2}$$

gdje je:

- n je ukupni broj podataka koji se obrađuje
- φ_i je i-ti podatak vrijednosti sa akcelerometra
- θ_i je i-ti podatak filtrirane vrijednosti
- Q je koeficijent preciznosti

Mjerenje je izvršeno nakon što se postavio senzor na robota. Robot postavio na 0° te se počelo izvršavati mjerenje kuta sa akcelerometra.

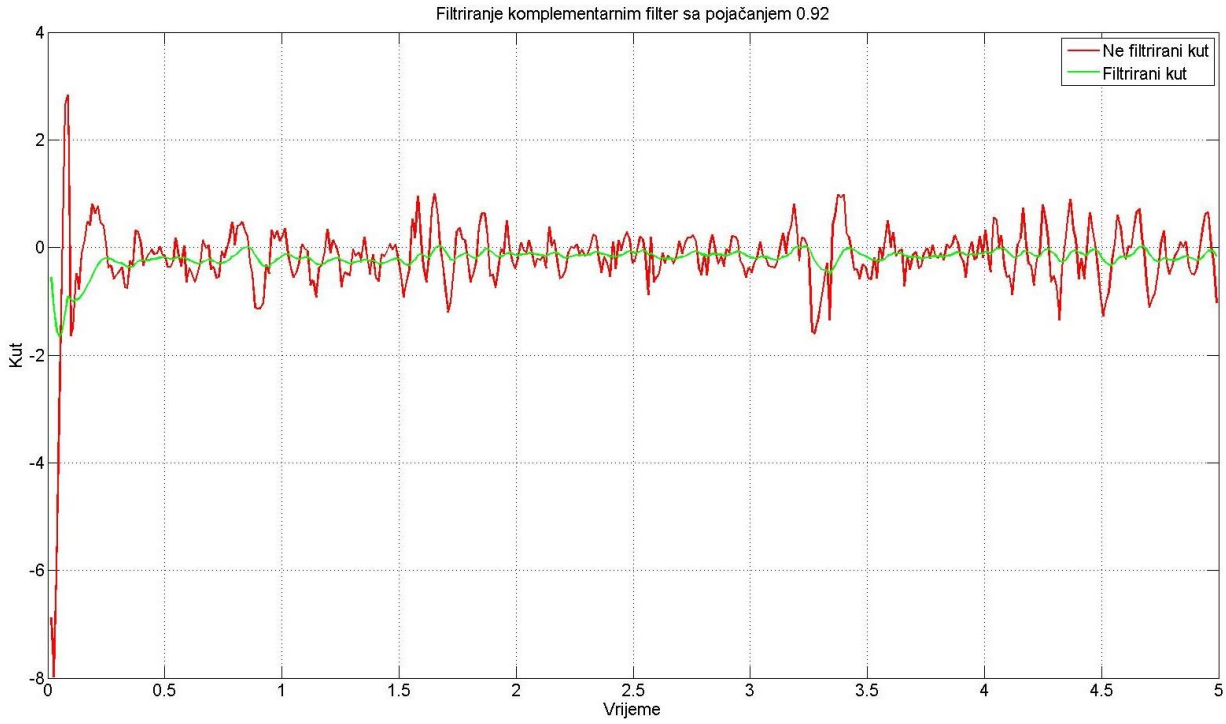
Tablica 4.1. Određivanje pojačanje komplementarnog filtra

Pojačanje	Koeficijent preciznosti (Q)
0,99	0.0018
0,98	0.0013
0,97	0.0011
0,96	0.0010
0,95	0.0010
0,94	0.0009
0,93	0.0009
0,92	0.0008
0,91	0.0009
0,90	0.0012

```
float ComplementaryFilter(float newAngle, float newRate, float Gain, float oldAngle, float dt)
{
    float filteredAngle = Gain * (oldAngle + newRate * dt) + (1-Gain) * newAngle;
    return filteredAngle;
}
```

Slika 4.4. Implementacija komplementarnog filtra u C jeziku

Na slici 4.5. se može vidjeti mjerenje kuta bez filtra i filtrirani kut komplementarnim filtrom.



Slika 4.5. Komplementarni filtar sa pojačanjem 0.92

4.4. PD regulator

PD regulator je proporcionalno derivacijski regulator koji kao ulaz prima kut nagiba robota, a na izlazu daje vrijednost koja će kontrolirati brzinu motora. Jednadžba PD regulatora se može vidjeti u (4-2), a implementacija PD regulatora u C programskom jeziku se može vidjeti na slici 4.6.

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt} \tag{4-2}$$

gdje je:

- $u(t)$ upravljačka veličina
- $e(t)$ funkcija pogreške
- K_p pojačanje proporcionalnog djelovanja
- K_d pojačanje derivacijskog djelovanja

Funkcija pogreške $e(t)$uzima vrijednost nagiba robota i oduzima je od željene vrednosti, što je u ovom slučaju 0°.

Derivacijsko djelovanje funkcije pogreške $\frac{de(t)}{dt}$ uzima trenutnu vrijednost funkcije pogreške $e(t)$ i oduzima je sa prethodnom vrijednosti funkcije pogreškle $e(t - \Delta t)$.

```
float angleKp, angleKd;
int OCRvalue = 0;

angleKp = 20;
angleKd = 10;
OCRvalue = 312;

int angleError = (reqAngle - currentAngle);

AngleError_D = (angleError - prevAngle);
prevAngle = angleError;

int anglePD = -((angleKp * angleError) + (angleKd * AngleError_D));

int PWMduty = constrain(anglePD, -(OCRvalue-156), (OCRvalue-156));
```

Slika 4.6. PD regulator implementiran u C programski jezik

4.5. Kontrola koračnih motora

Koračni motori se pokreću tako što se uključuju zavojnice određenim slijedom, čekajući između koraka određeno vrijeme. Kontrola brzine se jedino može vršiti mijenjajući vremenski razmak između koraka. To se može ostvariti *Output Compare* prekidnom rutinom koja se pokreće za zadani vremenski period kroz vrijeme koje se zadaje u registru *OCR*. U prekidnu rutinu se postavi counter koji broji korake i ukazuje na sljedeći korak.

Postavke timera za *Output Compare* način rada se može vidjeti na slici 4.7., a funkcija za kontrolu motora na puni korak i polu korak na slici 4.8.

```
void PWMinit_timer1()
{
    TIMSK |= (1<<OCIE1A); //Output compare registar A interupt enable
    TCCR1B |= (1<<WGM12) | (1<<CS12) | (1<<CS10); //prescale 1024
    TCNT1 = 0; // set timer1 to 0;

}
```

Slika 4.7. Postavke timera za Ouptut Compare način rada

```
void StepperMotorB(uint8_t MODE, uint8_t Direction)
{
    if (MODE == FullStep)
    {
        if (Direction == Forward)
        {
            stepB++;
            stepCountB++;

            if (stepB > 4)
            {
                stepB = 1;
            }

        }
        else if (Direction == Backward)
        {
            stepB--;
            stepCountB--;
            if (stepB < 1)
            {
                stepB = 4;
            }
        }

        if (stepB == 1)
        {
            PORTB = 9;
        }
        else if (stepB == 2)
        {
            PORTB = 12;
        }
        else if (stepB == 3)
        {
            PORTB = 6;
        }
        else if (stepB == 4)
        {
            PORTB = 3;
        }

    }
```

```
    else if (MODE == HalfStep)
    {
        if (Direction == Forward)
        {
            halfstepB++;
            stepCountB += 0.5;

            if (halfstepB > 8)
            {
                halfstepB = 1;
            }
        }
        else if (Direction == Backward)
        {
            halfstepB--;
            stepCountB -= 0.5;

            if (halfstepB < 1)
            {
                halfstepB = 8;
            }
        }

        if (halfstepB == 1)
        {
            PORTB = 8;
        }
        else if (halfstepB == 2)
        {
            PORTB = 12;
        }
        else if (halfstepB == 3)
        {
            PORTB = 4;

        }
        else if (halfstepB == 4)
        {
            PORTB = 6;
        }
        else if (halfstepB == 5)
        {
            PORTB = 2;
        }
        else if (halfstepB == 6)
        {
            PORTB = 3;
        }
        else if (halfstepB == 7)
        {
            PORTB = 1;
        }
        else if (halfstepB == 8)
        {
            PORTB = 9;
        }
    }
}
```

Slika 4.8. Funkcija za kontrolu motora

Funkcija prima varijablu *MODE* koji određuje rotaciju punim korakom ili polukorakom i *Direction* koja određuje smjer kretanja motora. Step varijabla vodi računa koji je trenutni puni korak i koji je sljedeći korak, a varijabla halfstep vodi računa koji je trenutni polu korak i koji je sljedeći polu korak. Sa varijablom *stepCount* se kontrolira koliko je ukupno napravljeno koraka, ako se prebacuje u međuvremenu između modova upravljanja koračnih motora.

4.6. Balansiranje

Programski kod funkcije koja obavlja balansiranje se može vidjeti na slici 4.9. Funkcija sadrži PD regulatora koji kao ulaz prima vrijednosti željenog kuta, trenutnog kuta i vremena

uzorkovanja. Na izlazu daje vrijednost *OCR* registra koji definira vrijeme čekanja između koraka koračnih motora.

```
void motorControl(float reqAngle, float currentAngle, float dt)
{
    float angleKp, angleKd;
    int OCRvalue = 0;

    angleKp = 20;
    angleKd = 10;
    OCRvalue = 312;

    int angleError = (reqAngle - currentAngle);

    AngleError_D = (angleError - prevAngle);
    prevAngle = angleError;

    int anglePD = -((angleKp * angleError) + (angleKd * AngleError_D));

    int PWMduty = constrain(anglePD, -(OCRvalue-156), (OCRvalue-156));

    if (PWMduty < 0 && currentAngle < -2 && currentAngle > -40) //Backward -angle
    {
        motorDirectionA = Backward;
        motorDirectionB = Backward;

        OCR1A = OCRvalue - constrain(abs(PWMduty),0,(OCRvalue-156));
    }
    else if (PWMduty > 0 && currentAngle > 2 && currentAngle < 40) //Forward + angle
    {
        motorDirectionA = Forward;
        motorDirectionB = Forward;

        OCR1A = OCRvalue - constrain(abs(PWMduty),0,(OCRvalue-156));
    }
    else
    {
        motorSTOP();
        prevAngle = 0;
        AngleError_I = 0;
        PWMduty = 0;
        motorDirectionA = Stop;
        motorDirectionB = Stop;
    }
}
```

Slika 4.9. Funkcija koja izvršava balansiranje

Prije nego što se vrijednosti dobivene sa PD regulatora pošalju na motore, provjerava se da li se robot nalazi u dopuštenom području. Robot je u dopuštenom području ako je nagib robota veći od 2° i manji od 40° prema pozitivnom smjeru i manji od -2° i veći od -40° u negativnom smjeru. Za kutove manje od 2° i veće od -2° se smatra da je robot u balansu. U slučaju da se robot nalazi u dopuštenom području postavlja se smjer kretanja motora varijablom

motorDirection te se računa vrijednost *OCR* registra. U slučaju da se robot ne nalazi u dopuštenom području sve vrijednosti kontrole se resetiraju te se aktuatori nalaze u stanju mirovanja.

Na slici 4.10. se nalazi programski kod glavne petlje koja se ponavlja svakih 5ms koji je određen konstantom *Ts*. U početku petlje se dohvaćaju vrijednosti sa MPU6050, filtriraju se vrijednosti sa žiroskopa nisko-propusnim filtrom koji služe za preciznije određivanje kuta komplementarnim filtrom. Filtrirane vrijednosti se šalju funkciji *motorControl* gdje se računaju vrijednosti PD regulatora i šalje odziv na motore.

```
while(1)
{

    if (seconds >= Ts)
    {
        loop_dt = seconds;
        seconds = 0;

        rawAngle = getAngleValues(MPU6050_ReadAccel(1), MPU6050_ReadAccel(2)); //Angle in DEG

        //filtriranje
        rawRate = LowPassFilter(MPU6050_ReadGyro(1), rawRate, 0.97); //Rate in DEG/second
        compAngle = ComplementaryFilter(rawAngle, rawRate, 0.92, compAngle, loop_dt);

        motorControl(0, compAngle, loop_dt);

    }//If
}//While
```

Slika 4.10. Glavna petlja programa

5. ANALIZA SUSTAVA BALANSIRANJA

Ovaj sustav je stalno aktivan, tj u rijetkim slučajevima on dođe u stacionarno stanje. Razlog togaje brzina kojom se robot može kretati zbog grubog skakanja između koraka. Koračni motori nisu u stanju obavljati preciznije pomake od 3.75° po koraku što je kod sustava balansiranja bitno. Tako se isto ne može ostvariti ni veća brzina od31.25 RPM, jer ako se prebrzo uključuju sljedeće sekvence koraka događa se preskakanje između koraka zbog fizičkih nedostataka u izradi samih motora.

Važno je napomenuti kako ovako dizajniran robot neće pasti sve dok ne pređe kut nagiba od $\pm 40°$, tj sve dok ne prijeđe u nedopušteno područje.

5.1. Analiza odziva kuta

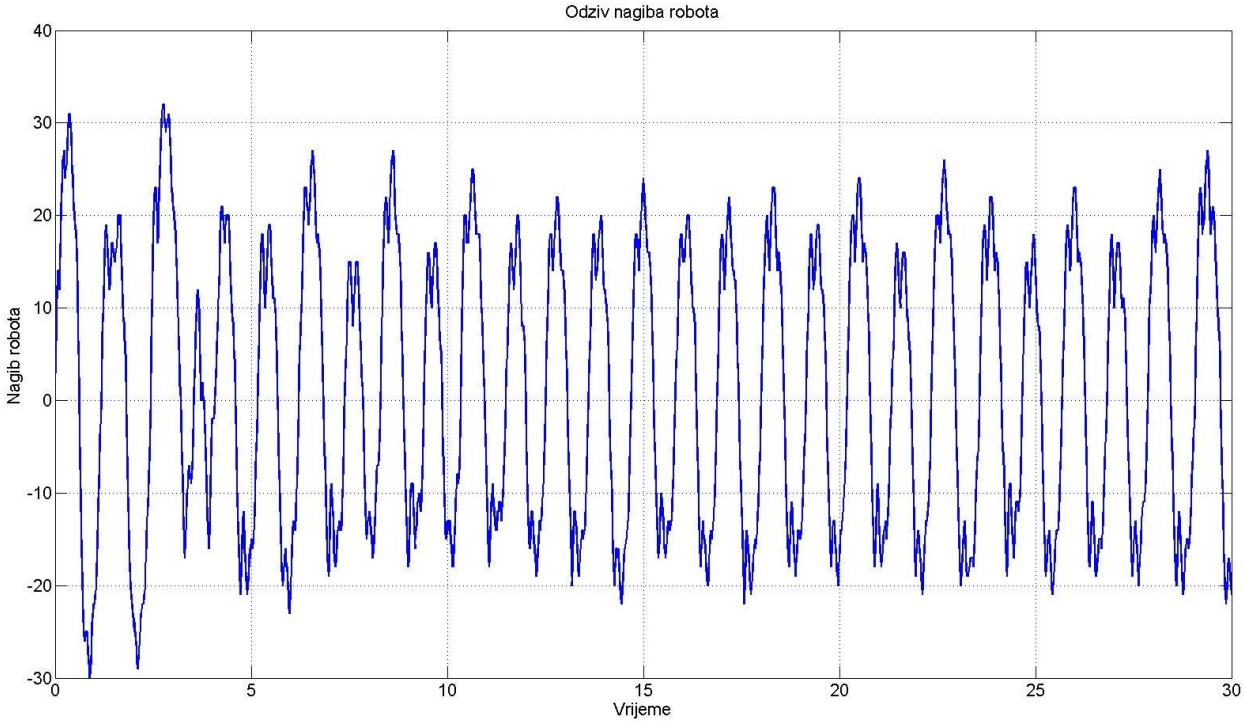
U svrhe analize je napravljeno dvadeset mjerenja koja se mogu vidjeti u tablici 5.1. Mjerenje je izvršeno na način da se robot nagne na kut od 30° i pusti da se balansira. U tablici vidimo kako se robot svega devet puta uspio izbalansirati unutar intervala od trideset sekundi. U ostalim mjerenjima robot nije pao, nego je i dalje pokušavao doći u stanje balansa.

Tablica 5.1. Mjerenja odziva kuta robota

Redni broj mjerenja	Uspiješno balansiranje	Vrijeme do balansa [s]	Vrijeme mjerenja [s]
1	NE	-	30
2	NE	-	30
3	NE	-	30
4	DA	8,46	30
5	NE	-	30
6	DA	6,86	30
7	DA	8,44	30
8	DA	4,11	30
9	NE	-	30
10	NE	-	30
11	DA	4,44	30
12	DA	3,34	30
13	NE	-	30
14	NE	-	30

15	DA	16,34	30
16	DA	15,8	30
17	NE	-	30
18	DA	11,78	30
19	NE	-	30
20	NE	-	30

Na slici 5.1. je prikazan odziv kuta za prvo mjerenje prema tablici 5.1. na kojem vidimo kako robot pokušava doći u stanje balansa.

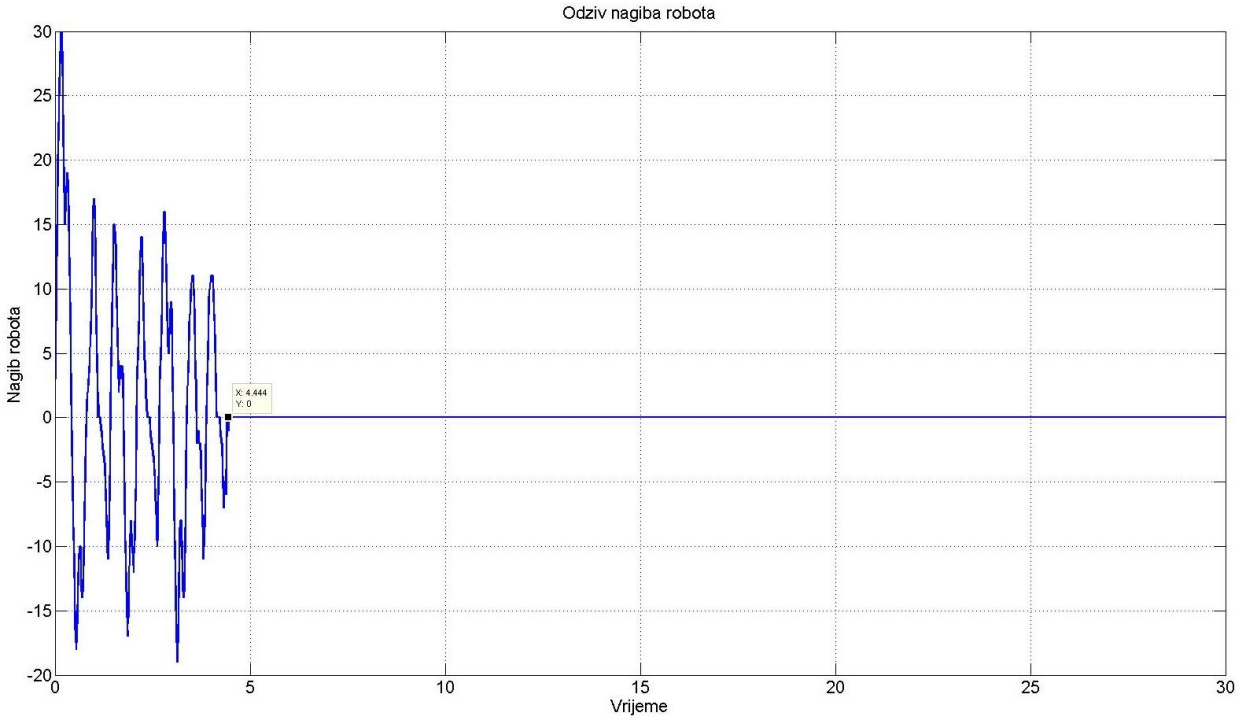


Slika 5.1. Odziv nagiba robota gdje se robot nije uspio izbalansirati

Na slici 5.1. je prikazan odziv kuta za jedanaesto mjerenje prema tablici 5.1. na kojem vidimo kako se robot uspio izbalansirati za početni kut od 30° za svega 4.44 s.

Razlog ovakve razlike u mjerenjima za isti početni kut su motori. Na slikama se vide male oscilacije pri krajnjim vrijednostima kuta koje su uzrokovane promjenom smjera koračnih motora.

Koračni motori korišteni u ovom radu imaju malu rezoluciju, svega 7.5° po koraku, gdje se metodom polukoraka svelo na 3.75° po koraku što je 96 koraka za puni krug. Kad bi se koristili motori sa rezolucijom od 200 koraka ili više, oscilacije bi se znatno smanjile ili u potpunosti nestale.

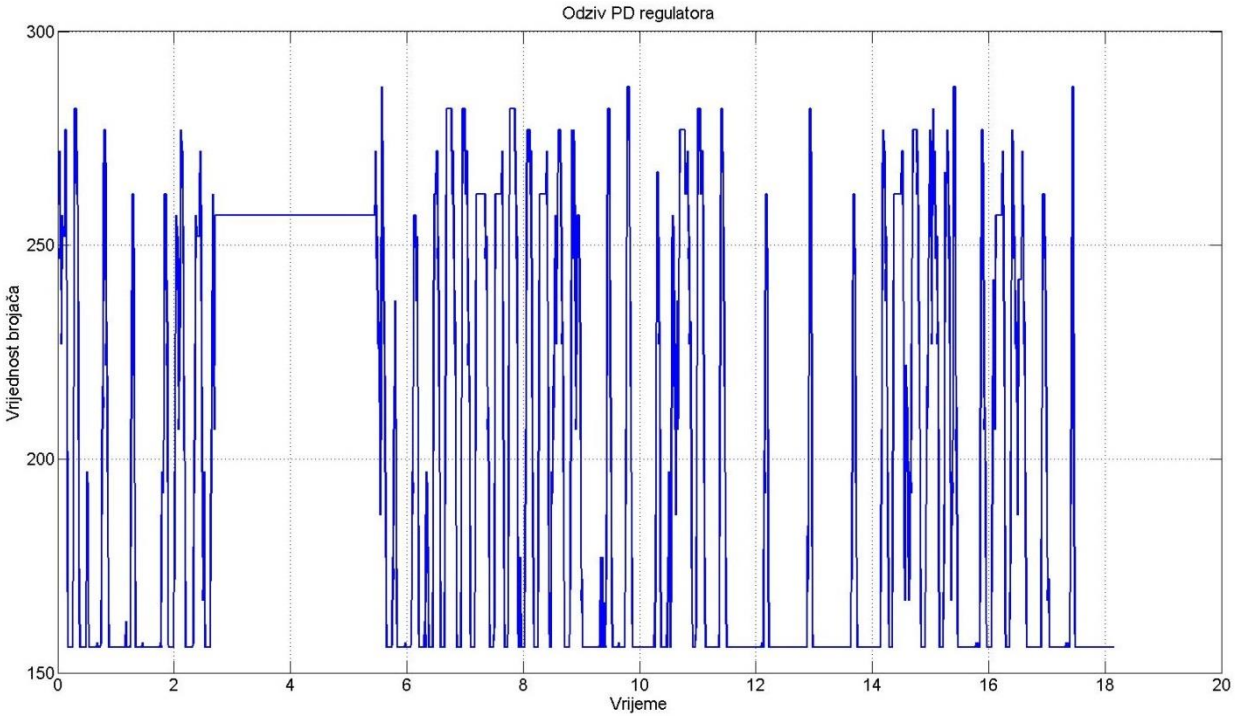


Slika 5.2. Odziv nagiba robota gdje se robot uspio izbalansirati za 4.44 s

5.2. Analiza odziva PD regulatora

PD regulator kao ulaz uzima vrijednosti kuta, te preko njega određuje kojom brzinom i u kojem smjeru treba djelovati. Ako se na ulazu PD regulatora dobije negativnu vrijednost kuta onda se motori okreću isto u negativnom smjeru, točnije robot se pomiče prema nuli. Vrijednost na izlazu PD regulatora je vrijednost brojača koji određuje brzinu okretanja motora, točnije to je vrijeme između svakog koraka. Najveća brzina kojom se motor može okretati je 31.25 RPM-a što je u vrijednosti brojača jednako 156 otkucaja. Vrijednosti brojača od 156 otkucaja predstavlja vrijeme od 0.019968 s, te se nakon tog vremena prebacuje na sljedeći korak. Najmanja vrijednost brojača je 312 otkucaja, što predstavlja vrijeme od 0.039936 s. Vremena iznad 40ms karakteriziraju veće oscilacije u okretajima koje nisu pogodne za kretanje robota.

Na slici 5.4. se vidi odziv PD regulatora te se može primijetiti kako se najčešće motori kreću najvećom brzina, što je ukazatelj da motori nisu dovoljno brzi te je to ograničenje u dizajnu ovog robota.



Slika 5.4. Odziv PD regulatora

6. ZAKLJUČAK

Najveći izazovi u izgradnji samobalansirajućeg robota su bili određivanje težišta i izbor motora. Težište je jako bitna stavka ako motori nisu dovoljno jaki ni brzi da se uspiju ispuniti zahtjevi balansiranja.

Za izbor motora se mogu uzeti i DC motori sa zupčastim prijenosom kojim se smanjuje broj okretaja u minuti i povećava moment sile. Treba se uzeti u obzir da DC motori imaju zaustavni put koji traje određeni vremenski period te da ne mogu naglo promijeniti smjer vrtnje. Za rješenje tog problema bi bila potrebna elektromagnetna kočnica. U tim slučajevima su koračni motori bolji jer po svom dizajnu imaju elektromagnetnu kočnicu koja ujedno i omogućava kretanje ako se pale zavojnice određenim redoslijedom te se tako mogu zaustaviti na određenom koraku bez zaustavnog puta. Motori koji su korišteni u ovom radu imaju prilično velik korak i nisu u mogućnosti izvršavati preciznije kretanje. To se može postići jedino mikro koracima za koje je potrebno drugačije sklopovlje koje nije obrađeno u ovom radu.

Prvobitni plan je bio izraditi robota pogonjenog DC motorima i upravljati ga sustavom upravljanja postavljenjem polova. Trebao se balansirati, kretati te okretati oko svoje osi. Zbog manjka financijskih sredstava za boljim i kvalitetnijim komponentama uspješno se ostvarilo jedino balansiranje po jednoj osi.

LITERATURA

[1] Atmel - ATMEGA32.pdf

[2] Dijelovi koračnog motora - http://www.minebea.co.jp/english/news/press/2012/__icsFiles/afieldfile/2015/01/29/press12062603_en.gif (pristup ostvaren u lipnju 2016. godine)

[3] Atmel - Driving Unipolar Stepper Motors Using C51C251.pdf

[4] MPU6050 - http://aws.robu.in/wp-content/uploads/2014/12/mpu_-_6050_gyro_sensor.png (pristup ostvaren u lipnju 2016. godine)

[5] InvenSence - PS-MPU-6000A-00v3.4.pdf

[6] Fairchild semiconductor -MC78XXLM78XXMC78XXA - 3-Terminal Voltage Regulator.pdf

[7] Texas Instruments - LM317 3-Terminal Adjustable Regulator.pdf

[8] Advanced Monolithic Systems AMS1117.pdf

[9] QUADRUPLE HALF-H DRIVERS - L293, L293D

[10] I2C - http://physudo-e.blogspot.hr/2013/09/read-acceleration-sensor-mpu-6050-with.html (pristup ostvaren u kolovozu 2015. godine)

[11] Komplementarni filtar - http://www.pieter-jan.com/node/11(pristup ostvaren u kolovozu 2015. godine)

SAŽETAK

Ideja je bila izraditi samobalansirajući robot koji će preko Atmega32 mikrokontrolera komunicirati sa MPU6050 žiroskopom i akcelerometrom čije se vrijednosti šalju na PD regulator čiji se odziv šalje na aktuatore. Cilj zadatka je konstruirati robota i napisati čitav programski kod u C programskom jeziku. Potrebno je ostvariti komunikaciju sa MPU6050 te filtrirati dobivene signale komplementarnim filtrom. Zatim, realizirati PD regulator te podesiti parametre kako bi se dobilo željeno dinamičko vladanje. Za aktuatore je potrebno izraditi štampanu pločicu sa L293D čipom i napisati programski kod za kontrolu. Robot uspješno vrši balansiranje prema zamišljenom planu.

Ključne riječi: Samobalansirajući robot, Atmega32, MPU6050, žiroskop, akcelerometar, L293D, H-most, koračni motori, PD regulator, komplementarni filtar

Self-balancing robot

Abstract

The idea was to build self-balancing robot which will communicate to MPU6050 gyroscope and accelerometer with Atmega32 microcontroller which values are then sent to PD controller whose output is sent to actuators. Main goal is to construct the robot and to write program code in C programming language. It is necessary to communicate with MPU6050 and filter the values with complementary filter. Furthermore it is needed to design PD controller and adjust the parametars to get desired dynamic behavior. Printed circuit board with L293D chips was needed for the actuators and to write a programing code forcontrol of the motors. Every goal that was defined in the beginning wasachieved and we have a stable and functional self-balancing robot.

Key words: Self-balancing robot, Atmega32, MPU6050, gyroscope, accelerometer, L293D, H-bridge, stepper motors, PD controller, complementary filter

ŽIVOTOPIS

Nemanja Plavšić rođen je 19.07.1990. godine u Vukovaru. Nakon osnovnoškolskog obrazovanja upisuje Tehničku školu Nikole Tesle u Vukovaru gdje je stekao zvanje tehničara za računalstvo. U rujnu 2009. godine upisuje sveučilišni studij računarstva na Elektrotehničkom fakultetu u Osijeku, gdje je 2013. godine stekao zvanje inženjera računarstva. Iste godine upisuje diplomski studij procesnog računarstva na Elektrotehničkom fakultetu u Osijek koji još pohađa.

U slobodno vrijeme volontira u udruzi „Vukovarske iskrice – udruga za pomoć osobama sa intelektualnim teškoćama“ gdje održava računala i web stranicu.

Potpis:

_____

PRILOZI

- CD medij s:
  - Projekt dizajna pogonske pločice u programskom alatu Eagle
  - Literatura sa popisa literature
  - Projekt samobalansirajućeg robota u programskom alatu Atmel Studio
  - Slike korištene u diplomskom radu
  - Diplomski rad
- Programski kod u C programskom jeziku:

```
//*****************************************************************************
//Samobalansirajući robot
//Student:          Nemanja Plavšić
//Elektrotehnički fakultet Osijek
//*****************************************************************************

#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <string.h>
#include <avr/interrupt.h>
#include <math.h>

#define addr 0x68
#define DEG_to_RAD 0.017453292
#define RAD_to_DEG 57.29577951
#define TRUE 1
#define FALSE 0
#define constrain(AMT,LOW,HIGH) ((AMT)<(LOW) ? (LOW) : ((AMT)>(HIGH) ? (HIGH):(AMT)))
#define Ts 0.005 //delata_time in  [s] 5ms
#define Forward 2
#define Backward 1
#define Stop 0
#define FullStep 3
#define HalfStep 4

//*****************************************************************************
//TWI State codes
//*****************************************************************************
// General TWI Master staus codes
#define TWI_START                   0x08  // START has been transmitted
#define TWI_REP_START               0x10  // Repeated START has been transmitted
#define TWI_ARB_LOST               0x38  // Arbitration lost

// TWI Master Transmitter staus codes
#define TWI_MTX_ADR_ACK               0x18  // SLA+W has been tramsmitted and ACK received
#define TWI_MTX_ADR_NACK            0x20  // SLA+W has been tramsmitted and NACK received
#define TWI_MTX_DATA_ACK            0x28  // Data byte has been tramsmitted and ACK received
#define TWI_MTX_DATA_NACK            0x30  // Data byte has been tramsmitted and NACK received

// TWI Master Receiver staus codes
#define TWI_MRX_ADR_ACK               0x40  // SLA+R has been tramsmitted and ACK received
#define TWI_MRX_ADR_NACK            0x48  // SLA+R has been tramsmitted and NACK received
#define TWI_MRX_DATA_ACK            0x50  // Data byte has been received and ACK tramsmitted
#define TWI_MRX_DATA_NACK            0x58  // Data byte has been received and NACK tramsmitted

// TWI Slave Transmitter staus codes
#define TWI_STX_ADR_ACK               0xA8  // Own SLA+R has been received; ACK has been returned
#define TWI_STX_ADR_ACK_M_ARB_LOST    0xB0  // Arbitration lost in SLA+R/W as Master; own SLA+R has been received; ACK has been
```

```
returned
#define TWI_STX_DATA_ACK            0xB8  // Data byte in TWDR has been transmitted; ACK has been received
#define TWI_STX_DATA_NACK            0xC0  // Data byte in TWDR has been transmitted; NOT ACK has been received
#define TWI_STX_DATA_ACK_LAST_BYTE    0xC8  // Last data byte in TWDR has been transmitted (TWEA = "0"); ACK has been received

// TWI Slave Receiver staus codes
#define TWI_SRX_ADR_ACK                0x60  // Own SLA+W has been received ACK has been returned
#define TWI_SRX_ADR_ACK_M_ARB_LOST    0x68  // Arbitration lost in SLA+R/W as Master; own SLA+W has been received; ACK has been
returned
#define TWI_SRX_GEN_ACK                0x70  // General call address has been received; ACK has been returned
#define TWI_SRX_GEN_ACK_M_ARB_LOST    0x78  // Arbitration lost in SLA+R/W as Master; General call address has been received; ACK
has been returned
#define TWI_SRX_ADR_DATA_ACK        0x80  // Previously addressed with own SLA+W; data has been received; ACK has been returned
#define TWI_SRX_ADR_DATA_NACK        0x88  // Previously addressed with own SLA+W; data has been received; NOT ACK has been
returned
#define TWI_SRX_GEN_DATA_ACK        0x90  // Previously addressed with general call; data has been received; ACK has been
returned
#define TWI_SRX_GEN_DATA_NACK        0x98  // Previously addressed with general call; data has been received; NOT ACK has been
returned
#define TWI_SRX_STOP_RESTART        0xA0  // A STOP condition or repeated START condition has been received while still addressed
as Slave

// TWI Miscellaneous status codes
#define TWI_NO_STATE                0xF8  // No relevant state information available; TWINT = "0"
#define TWI_BUS_ERROR                0x00  // Bus error due to an illegal START or STOP condition

#define TWIM_READ    1
#define TWIM_WRITE   0


//Function prototypes***********************************************************
float getAngleValues(float rawAccY, float rawAccZ);
float KalmanFilter(float newAngle, float newRate, float dt);
//float getDistance(void);
//float getVelocity(void);
void USART_send(char data);
void USART_message(char *data);
uint8_t USART_receive(void);
float MedianFilter(float a, float b, float c);
float ComplementaryFilter(float newAngle, float newRate, float Gain, float oldAngle, float dt);
void resetPIDvalues(void);
void resetEncoderValues(void);
float LowPassFilter (float rawData, float filtData, float Gain);
float MPU6050_ReadAccel(int axis);
float MPU6050_ReadGyro(int axis);
void StepperMotorA(uint8_t MODE, uint8_t Direction);
void StepperMotorB(uint8_t MODE, uint8_t Direction);
void motorControl(float reqAngle, float currentAngle,float dt);
//*******************************************************************************

//***************************** VARIABLES *********************************
float rKkr = 1;
float rDkr = 0.00;
float rIkr = 0.0;

//for measurment

int i_Angle, i_Time, i_PD;
char c_Angle[10], c_Time[10], c_PD[10];

//*************************************************************************************************************************
************
char recieveData;
int recieveFlag = 0;
int iKkr, iDkr, iIkr;
char cKkr[10];
char cDkr[10];
char cIkr[10];
```

```
float requestedPosition = 0;
float requestedVelocity = 0;
float requestedAngle = 0;
float requestedRate = 0;
float filterdData[4] = {0, 0, 0, 0};
float filterdDataPID[4] = {0, 0, 0, 0};
float compAngle = 0;
int filterdTPS[2] = {0, 0};
int filterdENC[2] = {0, 0};
float currentPosition = 0;
float currentVelocity = 0;
//*****************************************************
//PID variables*********************************************
int AngleError_I = 0;
float AngleError_D = 0;
float prevAngle = 0;
//*****************************************************
//Time variables*******************************************
float loop_dt = 0;
volatile float seconds = 0;
volatile float sendSeconds = 0;
//*****************************************************
//Gyroscope, Accelerometar and Encoder variables*****************
float acc_x = 0;
float acc_y = 0;
float acc_z = 0;
float gyro_x = 0;
float gyro_y = 0;
float gyro_z = 0;
float rawPosition = 0;
float rawVelocity = 0;
float rawAngle = 0;
float rawRate = 0;
//Stepper motor variables**************************************
uint8_t halfstepA = 1;
uint8_t halfstepB = 1;
uint8_t stepA = 1;
uint8_t stepB = 1;
float stepCountA = 1;
float stepCountB = 1;
uint8_t motorDirectionA = 0;
uint8_t motorDirectionB = 0;
uint8_t motorMode = 0;
//**********************INTERUPTS *****************************

ISR(TIMER0_OVF_vect)
{
          TCNT0 = 225; // 30 to overflow 1ms
          seconds += 0.001;
          sendSeconds += 0.001;
          if (sendSeconds > 30)
          {
                    sendSeconds = 0;
          }


          if (TCNT1 > OCR1A)
          {
                    TCNT1 = 0;

          }
}

ISR(TIMER1_COMPA_vect)
{
          TCNT1 = 0;
          StepperMotorA(motorMode,motorDirectionA);
          StepperMotorB(motorMode,motorDirectionB);
```

```
}
//************************ FUNCTIONS ***************************

//TWIM - Code developed by PhysUdo*****************************
uint8_t TWIM_Init (uint32_t TWI_Bitrate)
{

          //Set TWI bitrate
          //If bitrate is too high, then error return

          TWBR = ((F_CPU/TWI_Bitrate)-16)/2;
          if (TWBR < 11) return FALSE;

          return TRUE;
}
uint8_t TWIM_Start (uint8_t Address, uint8_t TWIM_Type)//1 = read, 0 = write
{
          uint8_t        twst;

          //Send START condition

          TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);

          // Wait until transmission completed

          while (!(TWCR & (1<<TWINT)));

          //Check value of TWI Status Register. Mask prescaler bits.

          twst = TWSR & 0xF8;
          if ((twst != TWI_START) && (twst != TWI_REP_START)) return FALSE;

          // Send device address

          TWDR = (Address<<1) + TWIM_Type;
          TWCR = (1<<TWINT)|(1<<TWEN);

          //Wait until transmission completed and ACK/NACK has been received

          while (!(TWCR & (1<<TWINT)));

          //Check value of TWI Status Register. Mask prescaler bits.

          twst = TWSR & 0xF8;
          if ((twst != TWI_MTX_ADR_ACK) && (twst != TWI_MRX_ADR_ACK)) return FALSE;

          return TRUE;
}
void TWIM_Stop (void)
{

          //Send stop condition

          TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);

          //Wait until stop condition is executed and bus released

          while (TWCR & (1<<TWINT));
}
uint8_t TWIM_Write (uint8_t byte)
{
          uint8_t   twst;

          // Send data to the previously addressed device

          TWDR = byte;
          TWCR = (1<<TWINT)|(1<<TWEN);
```

```
          // Wait until transmission completed

          while (!(TWCR & (1<<TWINT)));

          // Check value of TWI Status Register. Mask prescaler bits

          twst = TWSR & 0xF8;
          if (twst != TWI_MTX_DATA_ACK) return 1;

          return 0;
}
uint8_t TWIM_ReadAck (void)
{
          TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWEA);
          while (!(TWCR & (1<<TWINT)));

          return TWDR;
}
uint8_t TWIM_ReadNack (void)
{
          TWCR = (1<<TWINT)|(1<<TWEN);
          while(!(TWCR & (1<<TWINT)));

          return TWDR;
}

void TWIM_WriteRegister(char reg, char value)
{
          TWIM_Start(addr, TWIM_WRITE); // set device address and write mode
          TWIM_Write(reg);
          TWIM_Write(value);
          TWIM_Stop();
}

char TWIM_ReadRegister(char reg)
{
          TWIM_Start(addr, TWIM_WRITE);
          TWIM_Write(reg);
          TWIM_Stop();

          TWIM_Start(addr, TWIM_READ); // set device address and read mode
          char ret = TWIM_ReadNack();
          TWIM_Stop();
          return ret;
}

float MPU6050_ReadAccel(int axis)//x = 0; y = 1; z = 2
{
          char reg = axis * 2 + 59;     //Acceleromatar output        x=59      y=61      z=63
          //char AFS_SEL = TWIM_ReadRegister(28);
          //float factor = 1<<AFS_SEL;
          float factor = 16384; //datasheet str. 13
          int val = 0;
          float float_val = 0;
          char ret = 0;

          ret = TWIM_ReadRegister(reg);
          val = ret << 8;

          ret = TWIM_ReadRegister(reg+1);
          val += ret;

          if (val & 1<<15)
          val -= 1<<16;

          float_val = val;
```

```
          float_val = float_val / factor;

          return float_val;
}

float MPU6050_ReadGyro(int axis)//x = 0; y = 1; z = 2
{
          char reg = axis * 2 + 67;     //Gyroscope output  x=67      y=69      z=71
          //char FS_SEL = TWIM_ReadRegister(27);
          //float factor = 1<<FS_SEL; // factor = 1 * 2^FS_SEL
          float factor = 131; // datasheet str. 12
          int val = 0;
          float float_val = 0;
          char ret = 0;

          ret = TWIM_ReadRegister(reg);
          val = ret << 8;

          ret = TWIM_ReadRegister(reg+1);
          val += ret;

          if (val & 1<<15)
          val -= 1<<16;


          float_val = val;

          float_val = float_val / factor;

          return float_val;
}
//***********************************************************
void MPU6050_Config()
{
          TWIM_Init(100);
          TWIM_WriteRegister(27, 0); //FS_SEL = 0 Gyro_Config +-250 °/sec
          TWIM_WriteRegister(28, 0); //AFS_SEL = 0 Acc_Config +-2g
          TWIM_WriteRegister(107, 0); // Wake up from sleep mode
}

//USART functions*************************************
void USART_init(uint16_t baud)
{
          uint16_t baudPrescaler;
          baudPrescaler = (F_CPU/(16UL*baud))-1;
          UBRRH = (uint8_t)(baudPrescaler>>8);
          UBRRL = (uint8_t)baudPrescaler;
          UCSRB |= (1<<RXEN)|(1<<TXEN)|(1<<RXCIE); //omogući primanje i slanje podataka
          UCSRC |= (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
}

void USART_send(char data)
{
          while(!(UCSRA & (1<<UDRE)));
          UDR=data;
}

void USART_message(char *data)
{
          while(*data != '\0')
          USART_send(*data++);
}

uint8_t USART_receive(void)
{
          while(!(UCSRA & (1<<RXC)));
          return UDR;
}
```

```
//*****************************************************


//Complementary Filter*********************************
float ComplementaryFilter(float newAngle, float newRate, float Gain, float oldAngle, float dt)
{
          float filteredAngle = Gain * (oldAngle + newRate * dt) + (1-Gain) * newAngle;
          return filteredAngle;
}
//*****************************************************

//Low Pass Filter**************************************
float LowPassFilter (float rawData, float filtData, float Gain)
{
          float S = rawData * Gain + filtData * (1-Gain);

          return S;
}
//*****************************************************



//Median filter****************************************
float MedianFilter(float a, float b, float c)
{
          float middle;

          if ((a <= b) && (a <= c))
          {
                    middle = (b <= c) ? b : c;
          }
          else if ((b <= a) && (b <= c))
          {
                    middle = (a <= c) ? a : c;
          }
          else
          {
                    middle = (a <= b) ? a : b;
          }
          return middle;
}
//*****************************************************



//Get Measure Values***********************************
float getAngleValues(float rawAccY, float rawAccZ)
{
          //Racunanje vrijednosti sa Accelerometra u stupnjevima
          float rawAngle = (RAD_to_DEG * atan2(rawAccZ, rawAccY)) - 90;
          return rawAngle;
}

//*****************************************************

//Timers initialisation********************************
void Timer0_Init(void)
{
          TCCR0 |= (1<<CS02); // prescalar = 256
          TCNT0 = 225; //30 do overflow-a što je 1 [ms]
          TIMSK |= (1<<TOIE0); //Set TOIE bit // enable overflow interupt
}
//
//
void PWMinit_timer1()
{
          TIMSK |= (1<<OCIE1A);
          TCCR1B |= (1<<WGM12) | (1<<CS12) | (1<<CS10); //prescale 1024
```

```
          TCNT1 = 0;

}


//*****************************************************

void StepperMotorB(uint8_t MODE, uint8_t Direction)
{
          if (MODE == FullStep)
          {
                    if (Direction == Forward)
                    {
                              stepB++;
                              stepCountB++;

                              if (stepB > 4)
                              {
                                        stepB = 1;
                              }

                    }
                    else if (Direction == Backward)
                    {
                              stepB--;
                              stepCountB--;
                              if (stepB < 1)
                              {
                                        stepB = 4;
                              }
                    }

                    if (stepB == 1)
                    {
                              PORTB = 9;
                    }
                    else if (stepB == 2)
                    {
                              PORTB = 12;
                    }
                    else if (stepB == 3)
                    {
                              PORTB = 6;
                    }
                    else if (stepB == 4)
                    {
                              PORTB = 3;
                    }

          }

          //HALF_STEP mode *****************
          else if (MODE == HalfStep)
          {
                    if (Direction == Forward)
                    {
                              halfstepB++;
                              stepCountB += 0.5;

                              if (halfstepB > 8)
                              {
                                        halfstepB = 1;
                              }
                    }
                    else if (Direction == Backward)
                    {
                              halfstepB--;
                              stepCountB -= 0.5;
```

```
                        if (halfstepB < 1)
                        {
                                halfstepB = 8;
                        }
                }

                if (halfstepB == 1)
                {
                        PORTB = 8;
                }
                else if (halfstepB == 2)
                {
                        PORTB = 12;
                }
                else if (halfstepB == 3)
                {
                        PORTB = 4;

                }
                else if (halfstepB == 4)
                {
                        PORTB = 6;
                }
                else if (halfstepB == 5)
                {
                        PORTB = 2;
                }
                else if (halfstepB == 6)
                {
                        PORTB = 3;
                }
                else if (halfstepB == 7)
                {
                        PORTB = 1;
                }
                else if (halfstepB == 8)
                {
                        PORTB = 9;
                }
        }
}
void StepperMotorA(uint8_t MODE, uint8_t Direction)
{
        if (MODE == FullStep)
        {
                if (Direction == Forward)
                {
                        stepA++;
                        stepCountA++;

                        if (stepA > 4)
                        {
                                stepA = 1;
                        }

                }
                else if (Direction == Backward)
                {
                        stepA--;
                        stepCountA--;
                        if (stepA < 1)
                        {
                                stepA = 4;
                        }
                }

                if (stepA == 1)
```

```
                {
                        PORTA = 3;
                }
                else if (stepA == 2)
                {
                        PORTA = 6;
                }
                else if (stepA == 3)
                {
                        PORTA = 12;
                }
                else if (stepA == 4)
                {
                        PORTA = 9;
                }

        }

        //HALF_STEP mode *****************
        else if (MODE == HalfStep)
        {
                if (Direction == Forward)
                {
                        halfstepA++;
                        stepCountA += 0.5;

                        if (halfstepA > 8)
                        {
                                halfstepA = 1;
                        }
                }
                else if (Direction == Backward)
                {
                        halfstepA--;
                        stepCountA -= 0.5;

                        if (halfstepA < 1)
                        {
                                halfstepA = 8;
                        }
                }

                if (halfstepA == 1)
                {
                        PORTA = 9;
                }
                else if (halfstepA == 2)
                {
                        PORTA = 1;
                }
                else if (halfstepA == 3)
                {
                        PORTA = 3;
                }
                else if (halfstepA == 4)
                {
                        PORTA = 2;
                }
                else if (halfstepA == 5)
                {
                        PORTA = 6;
                }
                else if (halfstepA == 6)
                {
                        PORTA = 4;
                }
                else if (halfstepA == 7)
                {
```

```
                                PORTA = 12;
                        }
                        else if (halfstepA == 8)
                        {
                                PORTA = 8;
                        }
                }

}

void motorSTOP()
{
        PORTA = 0x00;
        PORTB = 0x00;
}

void motorControl(float reqAngle, float currentAngle, float dt)
{
        float angleKp, angleKd;
        int OCRvalue = 0;

        angleKp = 20;
        angleKd = 5;
        OCRvalue = 312;

        int angleError = (reqAngle - currentAngle);

        AngleError_D = (angleError - prevAngle);
        prevAngle = angleError;

        int anglePD = -((angleKp * angleError) + (angleKd * AngleError_D));

        int PWMduty = constrain(anglePD, -(OCRvalue-156), (OCRvalue-156));

        if (PWMduty < 0 && currentAngle < -2 && currentAngle > -40) //Backward -angle
        {
                motorDirectionA = Backward;
                motorDirectionB = Backward;

                OCR1A = OCRvalue - constrain(abs(PWMduty),0,(OCRvalue-156));
        }
        else if (PWMduty > 0 && currentAngle > 2 && currentAngle < 40) //Forward + angle
        {
                motorDirectionA = Forward;
                motorDirectionB = Forward;

                OCR1A = OCRvalue - constrain(abs(PWMduty),0,(OCRvalue-156));
        }
        else
        {
                motorSTOP();
                prevAngle = 0;
                AngleError_I = 0;
                PWMduty = 0;
                motorDirectionA = Stop;
                motorDirectionB = Stop;
        }
}

int main(void)
{
        //INITIALISATION
        USART_init(19200);
        MPU6050_Config();
        Timer0_Init();
        PWMinit_timer1();

        DDRB |= (1<<PB0) | (1<<PB1) | (1<<PB2) | (1<<PB3); // Setting up left motor control pins as output
```

```
        DDRA |= (1<<PA0) | (1<<PA1) | (1<<PA2) | (1<<PA3); // Setting up right motor control pins as output

        motorMode = HalfStep;
        sei();

        while(1)
        {

                if (seconds >= Ts)
                {
                        loop_dt = seconds;
                        seconds = 0;

                        rawAngle = getAngleValues(MPU6050_ReadAccel(1), MPU6050_ReadAccel(2)); //Angle in DEG

                        //filtriranje
                        rawRate = LowPassFilter(MPU6050_ReadGyro(1), rawRate, 0.97); //Rate in DEG/second
                        compAngle = ComplementaryFilter(rawAngle, rawRate, 0.93, compAngle, loop_dt);

                        motorControl(0, compAngle, loop_dt);

                }//If
        }//While
}//Main
```