Decentralizirana regulacija struje skupine električnih istosmjernih motora putem Weba

Koren, Leon

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Leon Koren

Zagreb, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:

Dr. sc. Tihomir Žilić, dipl. ing.

Student:

Leon Koren

Zagreb, 2018.

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: **Leon Koren** Mat. Br.: 0035196867

Naslov rada na hrvatskom jeziku: **Decentralizirana regulacija struje skupine električnih istosmjernih motora putem Weba**

Naslov rada na engleskom jeziku: **Decentralized Web control of the electrical current of a group of DC motors**

Opis zadatka:

Radi se o skupini istosmjernih (DC) nezavisno uzbuđenih kolektorskih motora te istodobnoj strujnoj regulaciji svakog od njih tzv. decentralizirana regulacija struje skupine električnih DC motora. Zadavanje referentnih vrijednosti struje svakog motora zasebno bit će realizirano putem Web portala, za što će trebati mikroračunalo s postavljenim mrežnim okvirom web2py (sadrži server). Električni motori opremeljni su senzorima struje i enkoderima te su spojeni s mikrokontrolerom. Isti mikrokontroleri šalju naponske upravljačke signale na motore. Mikrokontroleri također imaju pokrenute servere za bežičnu komunikaciju s mikroračunalom. Za programiranje mikroračunala koristit će se Python programski jezik, a za mikrokontrolere C/C++ jezik.

U ovom radu, potrebno je:

1. Proučiti postojeće senzore električne struje, odabrati, i realizirati mjerenja armaturnih struja motora,
2. Matematičko modelirati skupinu nezavisno uzbuđenih istosmjernih motora,
3. Projektirati decentralizirani regulator struje za skupinu motora,
4. Programirati mikroračunalo u Python jeziku za komunikaciju s osobnim računalom (preko Interneta) te mikrokontrolerima (bežično),
5. Programirati upravljački zakon u mikroračunalo/mikrokontrolere,
6. Postaviti servere na mikrokontrolerima za bežičnu komunikaciju s mikroračunalom
7. Programirati mikrokontrolere u C/C++ programskom jeziku za primanje signala sa senzora motora te za slanje upravljačkih napona na motore
8. Izraditi WEB portal na mikroračunalu koristeći web2py mrežnom okviru preko kojeg bi se zadavale referentne upravljačke vrijednosti struje motora.

Zadatak zadan:

30. studenog 2017.

Zadatak zadao:

Doc.dr.sc. Tihomir Žilić

Rok predaje rada:
1. rok: 23. veljače 2018.
2. rok (izvanredni): 28. lipnja 2018.
3. rok: 21. rujna 2018.

Predviđeni datumi obrane:
1. rok: 26.2. - 2.3. 2018.
2. rok (izvanredni): 2.7. 2018.
3. rok: 24.9. - 28.9. 2018.

Predsjednik Povjerenstva:

Izv. prof. dr. sc. Branko Bauer

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

Zahvaljujem se mentoru, doc. dr. sc. Tihomiru Žiliću, na stručnim savjetima i pruženoj pomoći pri izradi ovoga rada.

Zahvalio bih se također svojoj obitelji na potpori koju mi je pružila tijekom studija.

Leon Koren

SADRŽAJ

SADRŽAJ . . . . . **I**

POPIS SLIKA . . . . . **III**

POPIS OZNAKA . . . . . **IV**

1. UVOD **1**

2. TEORIJSKA OSNOVA **2**
   2.1 Teorija regulatora 3
      2.1.1 PI regulator 3
   2.2 Osnove integriranih "*embedded*" sustava 4
      2.2.1 Espressif ESP32 razvojna pločica 5
   2.3 Arduino 6
      2.3.1 Arduino IDE 6
   2.4 World Wide Web (WWW) 7
      2.4.1 Povijest WWW-a 7
      2.4.2 Web stranica 7
      2.4.3 Web2Py 7

3. IZRADA MAKETE **8**
   3.1 Izrada nosača motora 8
   3.2 Izrada senzora brzine 9
   3.3 Izrada upravljačkog sklopa 10
      3.3.1 H-most 10
      3.3.2 Senzor struje 11
   3.4 Izrada mehaničkog opterećenja 12
   3.5 Izračun parametara motora 13
      3.5.1 Izrada modela motora u Simulink-u 13
      3.5.2 Snimanje karakterističnih odziva motora 15
      3.5.3 Estimacija parametara 16

4. PI REGULATOR **17**
   4.1 Kontinuiran PI regulator 17
      4.1.1 Simulink model 18
      4.1.2 Izračun parametara regulatora 19
   4.2 Diskretni PI regulator 21

5. KOD UNUTAR MIKROKONTROLERA **22**
   5.1 Paralelno procesiranje 22
   5.2 Web poslužitelj 23
   5.3 PI regulator 24
   5.4 Slanje podataka u databazu 25

6. WEB STRANICA **27**
   6.1 Model 27
   6.2 Kontroler (*Controllers*) 28
   6.3 Pogled (*View*) 29
   6.4 Statične datoteke (*Static*) 29
      6.4.1 *Javascript* datoteka s funkcijama 30
   6.5 Izgled Web stranice 32

7. ZAKLJUČAK **33**

8. LITERATURA **34**

9. PRILOG **35**
9.1 Web stranica 35
9.1.1 Datoteka *default.py* 35
9.1.2 Datoteka *functions.js* 37
9.2 Arduino kod 42
9.3 Sheme sklopa 50
9.3.1 Upravljački sklop 50
9.3.2 Senzor brzine 51
9.4 Prikaz postava 52

POPIS SLIKA

Slika 1. Blok dijagram sklopa . . . 2
Slika 2. PI regulator . . . 3
Slika 3. ESP32 SoC . . . 4
Slika 4. ESP32 funkcijski blok dijagram . . . 5
Slika 5. Arduino IDE s osnovnim funkcijama . . . 6
Slika 6. Model nosača u SolidWorks-u . . . 8
Slika 7. Disk s prorezima . . . 9
Slika 8. Senzor struje . . . 9
Slika 9. L298N H-most modul . . . 10
Slika 10. ADS1115 AD pretvornik . . . 11
Slika 11. Inertno opterećenje motora . . . 12
Slika 12. Model armature motora . . . 13
Slika 13. Model mehanike motora . . . 14
Slika 14. Odziv struje . . . 15
Slika 15. Odziv brzine vrtnje . . . 15
Slika 16. Estimacija parametara . . . 16
Slika 17. Primjer analognog regulatora . . . 17
Slika 18. Model s kontinuiranim regulatorom . . . 18
Slika 19. Upis zahtjeva za regulaciju . . . 19
Slika 20. Izračun parametara u Simulink-u . . . 20
Slika 21. Podešen odziv sustava . . . 20
Slika 22. Ulazna stranica . . . 32
Slika 23. Rezultati eksperimenta . . . 32

POPIS OZNAKA

b	Koeficijent viskoznog trenja
f	Frekvencija
F_r	Koeficijent trenja
I	Struja
J	Inercija rotora motora
J_u	Inercija opterećenja
K_a	Konstanta armature
K_e	Konstanta elektromotorne sile
K_i	Integralna konstanta PI regulatora
K_m	Konstanta momenta
K_p	Proporcionalna konstanta PI regulatora
L_a	Induktivitet armature
N	Brzina vrtnje u okr/s
R_a	Otpor armature
T_a	Vremenska konstanta armature
T_i	Vremenska konstanta PI regulatora
X_l	Reaktancija armature
ω	Brzina vrtnje u rad/s

1. UVOD

Danas su regulatori suštinski dio svakog automatskog sustava. Njihova namjena je pospješiti odnosno omogućiti točnost i brzinu odziva nekog sustava u realnom okruženju.

Povijesno gledano regulatori se koriste još od antike, no prva prava analiza je provedena od strane J.C. Maxwella 1868. godine u radu ”On Governors”. Unutar Maxwellovog rada prvi put možemo vidjeti analizu centrifugalnog regulatora. Važnost samog rada je u tome što Maxwell zaključuje kako realan sustav ima kašnjenja koja mogu dovesti do nestabilnosti.

Kasnije otkrićem električne struje dolazi do izuma električnih motora te se vrlo brzo shvaća da je sam rad motora usko vezan uz njihovu kontrolu.

Teorija regulacije električnih motora i danas je vrlo složeno područje koje je teško shvatiti bez realnih primjera. Kako bi se olakšalo shvaćanje izrađuju se različite makete na kojima se vrši testiranje te izučavanje.

Sama izrada takvih maketa je većinom vrlo skupa te je mnogo bolje rješenje projektirati sustav na kojem može raditi više ljudi. Takvi sustavi danas postoje u vidu računalnih simulacija, no vrlo rijetko simulacija može obuhvatiti potpuno zadan problem.

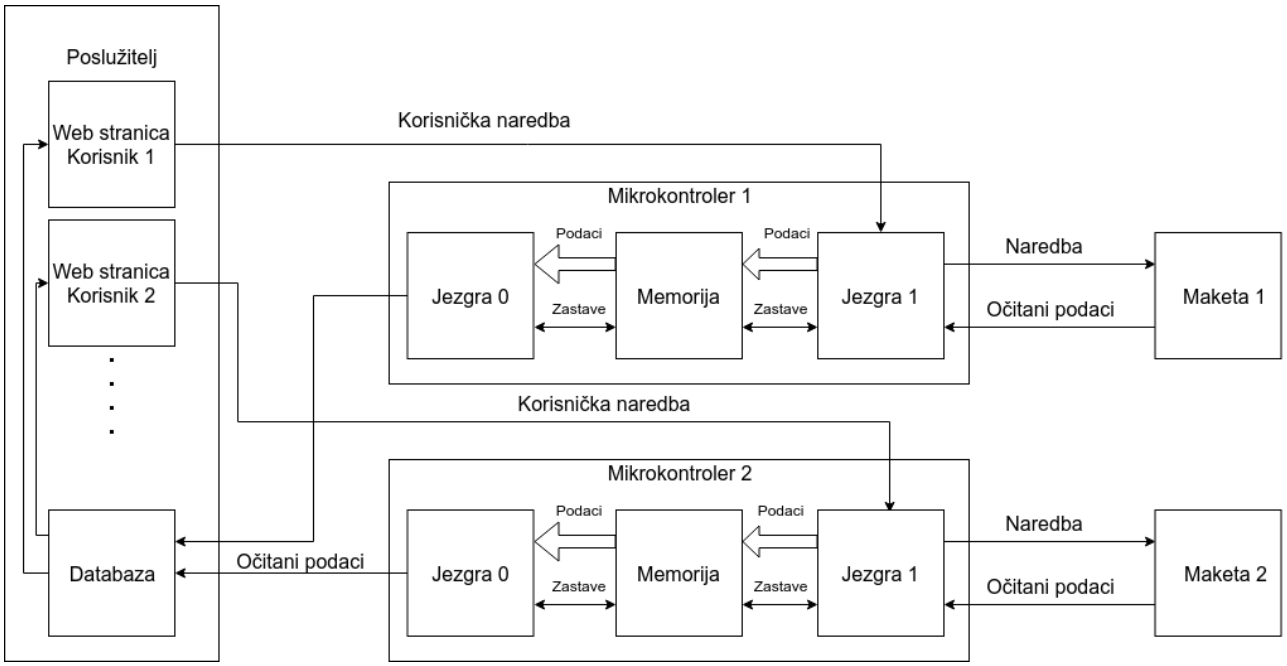
Slijedom prije navedenih činjenica logično rješenje je dizajnirati sustav koji u približno realnom vremenu može izvršiti zadan zadatak te prikazati rezultate. Kao logičan medij za izvršenje takvog zadatka nameće se internet.

Spojem realne makete s internet sučeljem tako dobivamo svestran sustav sposoban obuhvatiti pozitivne strane realnog i virtualnog svijeta. No, kao i svaki drugi sustav ima svoje nedostatke, najizraženiji od njih je da se gubi mogućnost direktnog dodira sa sustavom. Drugim riječima sam korisnik vlastita osjetila zamjenjuje umjetnim senzorskim sustavima.

2. TEORIJSKA OSNOVA

Unutar ovog dijela ukratko će biti predstavljeni svi dijelovi koji će se koristiti. Prvotno će biti obrađena teorija regulacije kao i specifičan regulator korišten unutar ovog rada. Nakon toga bit će predstavljen mikrokontroler te naposljetku ukratko objašnjena izrada web stranice u programskom okruženju Web2Py.

Radi lakšeg praćenja cijelog sklopa prvotno će biti prikazan blok dijagram te ukratko objašnjen princip spajanja više maketi.



Slika 1. Blok dijagram sklopa

Prikazani dijagram pokazuje način spajanja svih komponenti unutar sustava. Dodatno je moguće uočiti kako za korištenje više maketa nije potrbno raditi promjene samog sustava nego samo izraditi dodatnu maketu i pridodati ju sustavu. Takvi sustavi su decentralizirani jer svaki može funkcionirati kao jedinka.

Shodno prijašnim zaključcima unutar ovoga rada biti će obrađen sustav s jednom maketom.

2.1. Teorija regulatora

Uvodni dio ovog rada ukratko je objasnio svrhu regulatora te njihovu korisnost u realnom svijetu. Bilo je rečeno da je prvo službeno obrazloženje regulatora je sadržano unutar rada J.C.Maxwella u radu ”On Governors”. Nakon njegovoga rada počinje ozbiljno zanimanje za regulacijske sustave te već 1877. godine dva znanstvenika (E.J. Routh i A. Hurwitz) opisuju stabilnost sustava pomoću diferencijalnih jednadžbi koji danas poznajemo kao Routh-Hurwitz teorem.

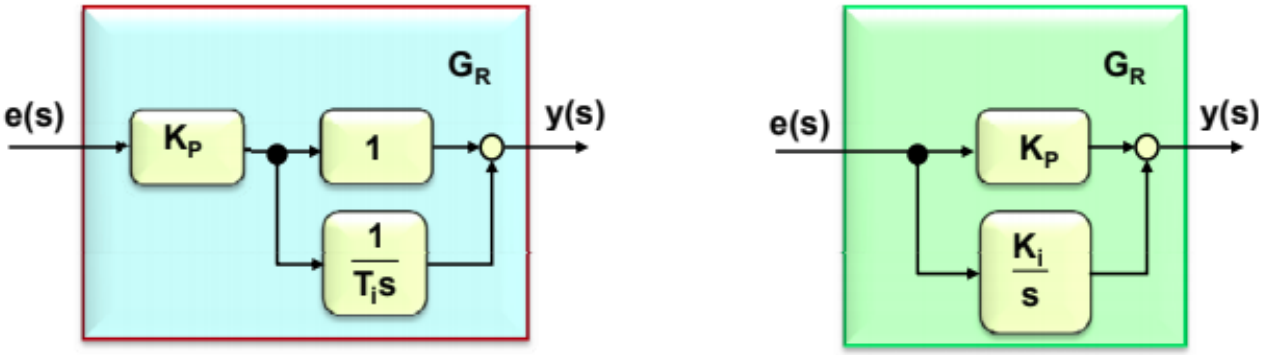
Regulacijski sustav je ”sustav koji koristi povratnu petlju kako bi formirao devijacijsku veličinu te s njom upravlja sustavom na takav način da se devijacijska veličina svede na nulu”[1]. Kraće rečeno regulatori služe za regulaciju neke sistemske vrijednosti te pospješivanje i ubrzavanje rada nekog sustava. Važno je napomenuti da su regulatori dio zatvorene petlje tj. regulator pokušava umanjiti odnosno svesti na nulu razliku između mjerene i zadane vrijednosti.

Regulatori se dijele prema arhitekturama upravljačkih članova koje posjeduju pa tako razlikujemo: osnovne tipove(P,I,D), njihove kombinacije(PI,PD,PID) te složene(adaptivni, neuronske mreže, LQR...). Danas se pretežno koriste složeni regulatori pošto sama implementacija istih nije vrlo zahtjevna, a daju mnogo bolje rezultate.

Unutar ovog rada bit će korišten PI regulator pa će on biti mnogo detaljnije objašnjen.

2.1.1. PI regulator

PI regulator se sastoji od proporcionalnog i integralnog člana u paralelnom spoju. Danas je jedan od najkorištenijih regulatora pošto je jednostavniji za implementaciju od PID regulatora, a daje vrlo dobar odziv. Prednosti su potpuna eliminacija greške u stacionarnom stanju te relativno brz odziv, no samo integralno djelovanje unosi dodatnu nestabilnost u sustav. Većinom se koriste za regulaciju sustava koji u sebi ne posjeduju integralno djelovanje kako ne bi dodatno narušili stabilnost.



Slika 2. PI regulator

Gore navedene blok dijagrame možemo prikazati u obliku prijenosne funkcije:

$$G(s) = K_p * (1 + \frac{1}{T_i s}), \tag{2.1}$$

$$G(s) = K_p + \frac{K_i}{s}. \tag{2.2}$$

Pošto je vidljivo iz gore navedenih prijenosnih funkcija kako su integralno pojačanje i vremenska konstanta međusobno zavisne vrijednosti:

$$K_i = \frac{K_p}{T_i}. \tag{2.3}$$

Regulator se potpunosti može definirati s dva parametra koja se dobivaju nekom metodom sinteze.

Sama sinteza regulatora iziskuje definiranje željenog ponašanja sustava kao što su: vrijeme porasta, vrijeme smirivanja, postotni prebačaj, vlastite prigušene frekvencije te trajno odstupanje vrijednosti.

Postupak sinteze regulatora bit će prikazan kasnije na realnom primjeru regulacije struje DC motora.

2.2. Osnove integriranih "*embedded*" sustava

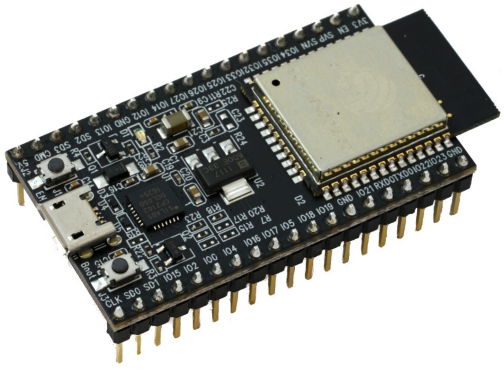
Interakciju između realnog i virtualnog svijeta vrlo često nam omogućuju i integrirani sustavi, kako je suštinski dio ovog rada baš takav, ovdje će biti obrađene osnove ovakvih sustava.

Povijesno gledano prvi integrirani sustav bio je onaj na Apollo misijama dizajniran 1960-ih godina na MIT sveučilištu. Sam razvoj kroz povijest bio je usko vezan uz razvoj ostale računalne opreme.

Danas su integrirani sustavi sastavni dio mnogih tehnologija koje se koriste: od uređaja za krajnje korisnike do industrijskih pogona i sofisticiranih autonomnih sustava.

Prednosti takvih sustava su niska cijena te mali broj popratnih dijelova. Najveći nedostatak je što iziskuju vrlo veliko poznavanje programiranja u nekom od nižih jezika pa njihov razvoj većinom mogu raditi školovani inženjeri. Sljedeći nedostatak je da se svako servisiranje uređaja koji posjeduju integrirani sustavi sastoji od zamjene cijelog sustava.

Unutar ovog rada bit će korištena posebna skupina integriranih sustava tzv. sistem na čipu ("*system on chip*") ili skraćeno SoC. Ovakav sustav posjeduje mikrokontroler te svu prateću periferiju na jednoj tiskanoj pločici te se harverski smatra crnom kutijom. Specifično bit će korišten ESP32 proizvođača Espressif.



Slika 3. ESP32 SoC

2.2.1. Espressif ESP32 razvojna pločica

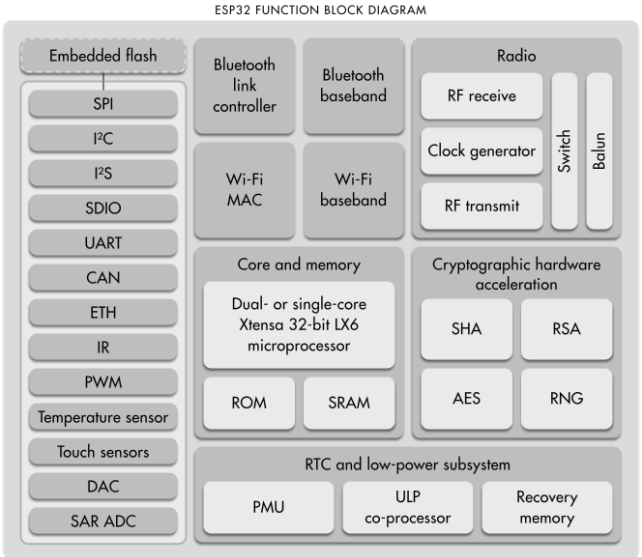
Kako je već bilo navedeno ESP32 je podskupina integriranih sustava takozvani sustavi na čipu. Primarna funkcija ovakvih sustava je razvoj tehnologije u području interneta stvari ("*Internet of things*") ili skraćeno IoT.

Čip posjeduje dvije procesorske jezgre harvardske arhitekture Xtensa LX6, svake sposobne postići frekvenciju do 240MHz. Sam čip je u mogućnosti adresirati do 4GB memorije što samo po sebi govori da radi sa 32-bit sabirnicom. Dodatna periferija čipa:

- WiFi modul
- Bluetooth 4.2 modul
- 12-bitni A/D pretvornik
- 8-bitni D/A pretvornik
- Brojač realnog vremena (RTC)
- Serijska komunikacija
- I^2C komunikacija
- SPI komunikacija
- "*Touch*" senzor
- Temperaturni senzor
- Hall senzor

Programiranje se može raditi pomoću razvojnoga okruženja izdanoga od strane proizvođača ili unutar Arduino okruženja. Iako čip posjeduje tzv. RTOS ("*Real-time operating system*"), kako bi postigli višu razinu apstrakcije moguće je staviti neki drugi operacijski sustav kao naprimjer mikro Python.

Unutar ovog rada bit će korišteno Arduino razvojno okruženje te WiFi mogućnosti čipa kako bi se postigla komunikacija s krajnjim korisnikom preko Web stranice.



Slika 4. ESP32 funkcijski blok dijagram

2.3. Arduino

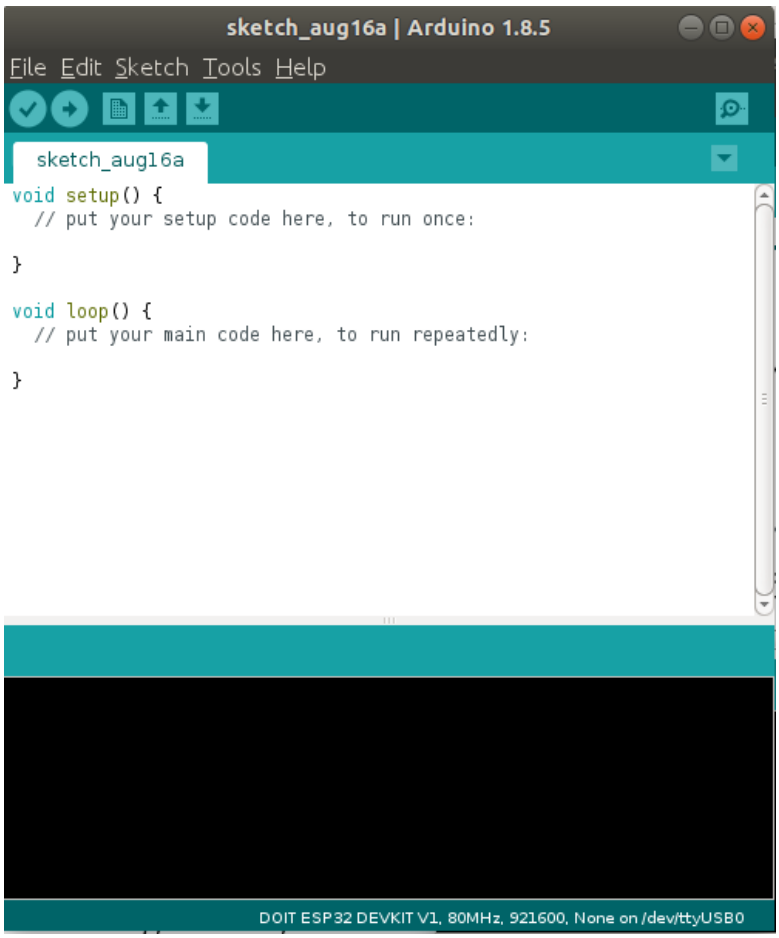
Općenito gledano Arduino je potpuno hardversko i programsko rješenje dizajnirano kako bi početnicima olakšalo učenje programiranja mikrokontrolera. Kao takvo pokriva sveukupno rješenje. Pošto u sklopu ovog rada neće biti korišten službeni mikrokontroler tvrtke Arduino AG nego samo dio koji obuhvaća integrirano razvojno okruženje (”*IDE*”) taj dio će i podrobnije biti objašnjen.

2.3.1. Arduino IDE

Arduino IDE je vrlo jednostavno programsko rješenje koje početnicima omogućuje lagano učenje programiranja, a naprednijim korisnicima vrlo brz razvoj programa. Sam program sastoji se od tekstualnog uređivača za pisanje koda te dodatnih modula koji omogućuju lagano kompiliranje te nasnimavanje čipa.

Pošto je cijeli sustav otvorenog koda mnoge knjižnice su napisane od strane samog proizvođača ili programerske zajednice te je svima dostupan na korištenje. Dodatna prednost ovakvog sistema je da hardver koji nije službeno dio Arduino repozitorija bude ugrađen te podržan od strane Arduino IDE paketa.

Unutar programskog paketa primarno se koristi C++ programski jezik te ponekad C te se u programiranju koriste sva pravila koja vrijede u tima dvama jezicima. Svaki program napisan unutar Arduino IDE naziva se ”*sketch*” te se na računalu sprema pod ekstenzijom ”*.ino*”. Najjednostavniji kod se sastoji od samo dvije funkcije, *setup()* koja se izvršava samo jednom na početku i služi za početne definicije i uključivanje svih modula koji će biti korišteni, te funkcije *loop()* koja se izvršava ciklički i sadrži glavni dio koda zadužen za izvršavanje programirane funkcionalnosti.



Slika 5. Arduino IDE s osnovnim funkcijama

2.4. World Wide Web (WWW)

World Wide Web ili skraćeno nazvan Web je informacijsko virtualno mjesto na kojem su dokumenti i ostali resursi indentificirani preko uniformnih resursnih lokatora (URL), međusobno povezani pomoću hiperlinkova dostupno na Internet-u[2].

Unutar ovog rada suštinski dio interakcije s korisnikom bit će preko Web-a, pa je vrlo važno objasniti osnovu ovog sustava.

2.4.1. Povijest WWW-a

Web je izumljen 1989. godine od strane engleskog znanstvenika Tim Berners-Lee, koji je 1990. godine isprogramirao prvi web preglednik na institutu CERN u Švicarskoj, no sam koncept Web-a i Web preglednika postao je dostupan širim masama tek sredinom 1991. godine.

Sljedeći vrlo važan korak za postojanje današnjeg sustava je odluka CERN-a 1993. godine da Web bude besplatan i dostupan svima potpuno bez bilo kakvih ograničenja. Takvom odlukom Web polako postaje najviše korišten medij za izmjenu informacija i znanja kavim ga danas poznajemo.

2.4.2. Web stranica

Web stranica je dio Web-a na kojem se nalaze relevantni podaci za korisnika te kojima se pristupa pomoću URL adrese.

Glavna podjela prema tipu podataka je na statičke i dinamičke. Danas je većina stranica kojima pristupa korisnik barem u jednom dijelu dinamična.

Stranice su napisane u posebno razvijenom programskom kodu HTML ("*Hyper-text markup language*") uz dodatak CSS-a ("*Cascading Style Sheets*"). U novije vrijeme dolazi do potrebe stranica prilagođenih korisniku pa se koriste viši programski jezici koji interaktivno generiraju HTML. Neki od tih programskih jezika su PHP, Python, Ruby itd.

Unutar ovog rada bit će korišten programski jezik Python te potpuno programsko rješenje Web2Py.

2.4.3. Web2Py

Web2Py je programsko rješenje za izradu te objavljivanje web stranica napisano u Python programskom jeziku od strane Massima Di Pierra 2007. godine. Dizajniran je kako bi ubrzao izradu web stranica na način da se mnogi osnovni moduli ne moraju izrađivati iz početka nego su dostupni unutar samog alata.

Prvotno je zamišljen kao edukacijski sustav temeljen na već postojećim sustavima sličnoga tipa, no s vremenom postaje vrlo dobar alat za mnoge koji žele brzo izraditi interaktivnije stranice.

Prednosti ovakvoga sustava su potpuna implementacija vrlo apstraktnog programskog jezika te s druge strane mogućnost rada sa svima starijim programskim jezicima korištenim u razvoju web stranica. Nedostatak su već unaprijed definirani moduli čime se smanjuje broj mogućnosti ponuđenih programeru, no ipak to nije ograničenje pošto se moduli mogu stvarati od strane samog programera.

3. IZRADA MAKETE

Motor korišten u maketi je izvađen iz nefunkcionalnog Epson printera te su same specifikacije nepoznate tj. podaci proizvođača bili su veoma šturi opisujući samo osnovne vrijednosti. Kako bi bilo moguće projektirati regulator potrebno je eksperimentalno utvrditi parametre motora koji nisu dani u podacima.

Naravno sljedeći problem je da motor nije bio pričvršćen te nije imao nikakav senzor brzine pa je prije utvrđivanja samih parametara bilo potrebno dizajnirati postavu s dodatnim senzorom brzine(Prilog 9.4).

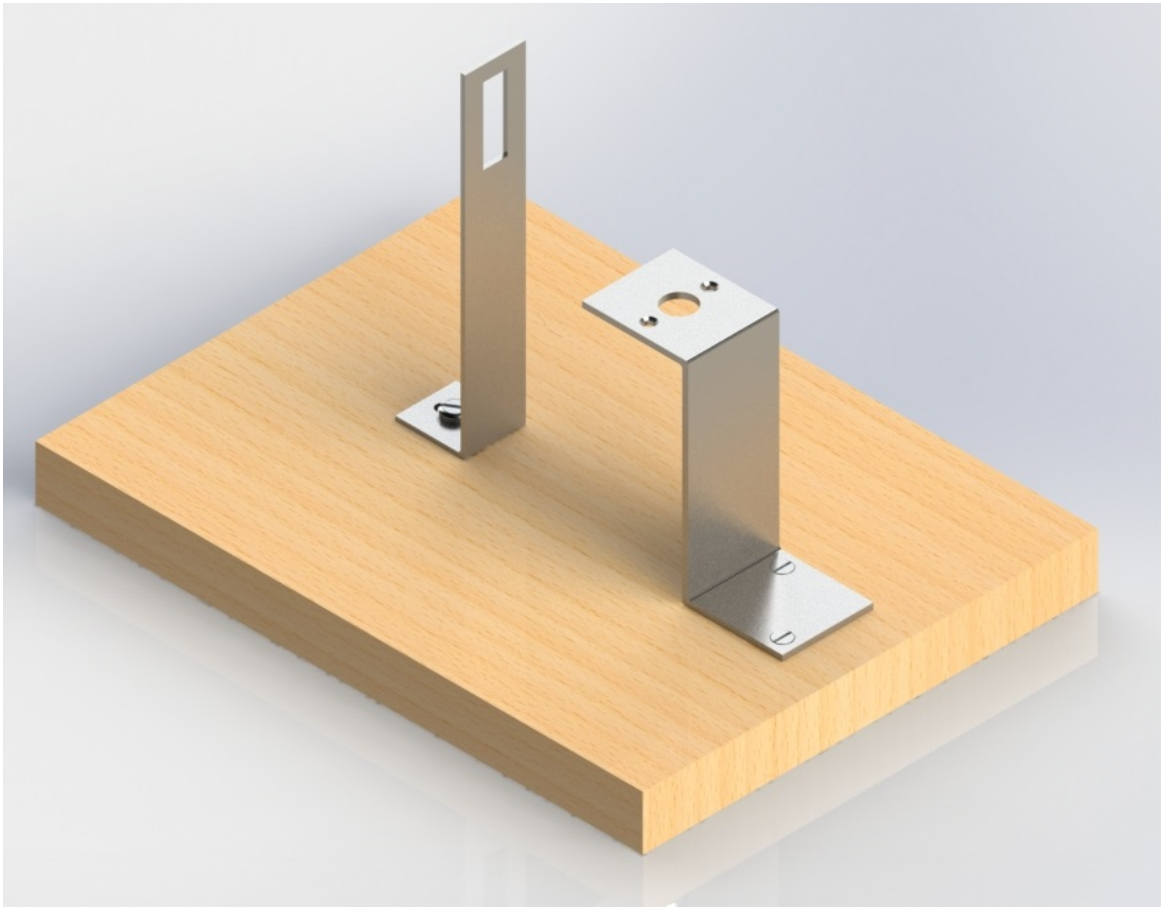
Shodno prije navedenim činjenicama u nastavku će biti prikazana izrada postave kao i senzora brzine te sam postupak određivanja parametara.

3.1. Izrada nosača motora

Prvotni cilj izrade nosača bio je zadržati jednostavnost makete pa se tako sam nosač sastoji od drvene ploče i limene trake. Motor je pričvršćen u vertikalnoj orijentaciji na sam nosač.

Nosač je dizajniran unutar programskog paketa SolidWorks kako bi svaka mjera bila prilagođena veličini motora te kako se kasnije ne bi moralo raditi dodatne prilagodbe.

Na samom nosaču motora pričvršćen je i nosač senzora brzine te kontrolna elektronika sa svim komponentama potrebnim za potpunu funkcionalnost cijele makete.



Slika 6. Model nosača u SolidWorks-u

3.2. Izrada senzora brzine

Pošto je za senzor brzine izabran inkrementalni enkoder, bilo je potrebno izraditi disk s prorezima te potrebnu elektroniku koja daje pravokutni signal.

Disk s prorezima izrađen je iz starog CD-a tako što je s njega skinuta metalna folija te zalijepljen papir s izrezanim prorezima. Za dovoljnu preciznost odabrano je 256 proreza tj. 8-bitna rezolucija.

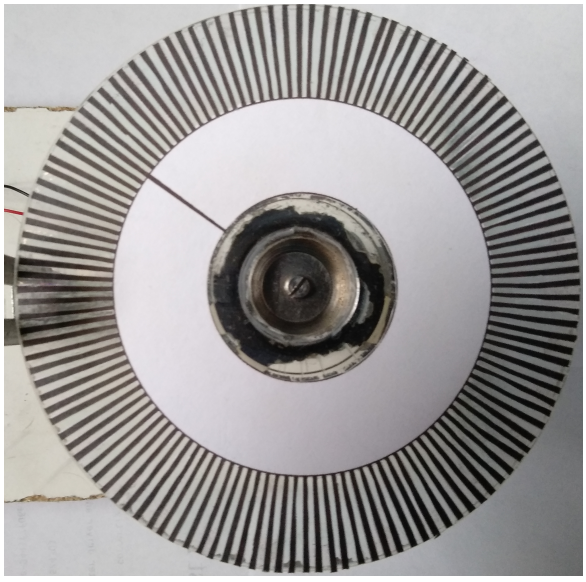
Sam senzor se sastoji od optokaplera te operacijskog pojačala u spoju komparatora s varijabilnim referentnim naponom(Prilog 9.3.2). Svrha komparatora je stabilizirati te pojačati napon na vrijednosti dovoljne za očitavanje. Dodatno je na izlaz komparatora stavljena zener dioda kako bi limitirala napon na 3.3V što je maksimalna vrijednost ulaza mikrokontrolera.

Nakon izrade senzora funkcionalnost je provjerena pomoću osciloskopa te je brzinu moguće očitati preko frekvencije kvadratnog signala prema formuli:

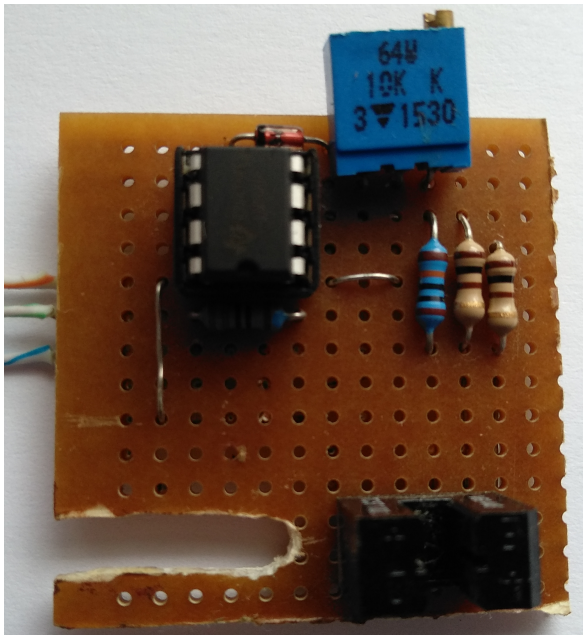
$$\omega = \frac{f}{256} * 2\pi. \tag{3.4}$$

Također moguće je izraziti i brzinu u okretajima u minuti pomoću formule:

$$N = \frac{f}{256} * 60. \tag{3.5}$$



Slika 7. Disk s prorezima



Slika 8. Senzor brzine

3.3. Izrada upravljačkog sklopa

Najvažniji dio makete je upravljački sklop. Sastoji se od mikrokontrolera, H-mosta te senzora struje(Prilog 9.3.1).

Mikrokontroler je objašnjen u uvodnom dijelu ovog rada te će kasnije biti objašnjen algoritam i programski kod korišten za upravljanje pa će u ovom dijelu detaljnije biti razrađeni H-most i senzor struje.

Sheme spajanja senzora struje te upravljačke pločice dane su u prilogu.

3.3.1. H-most

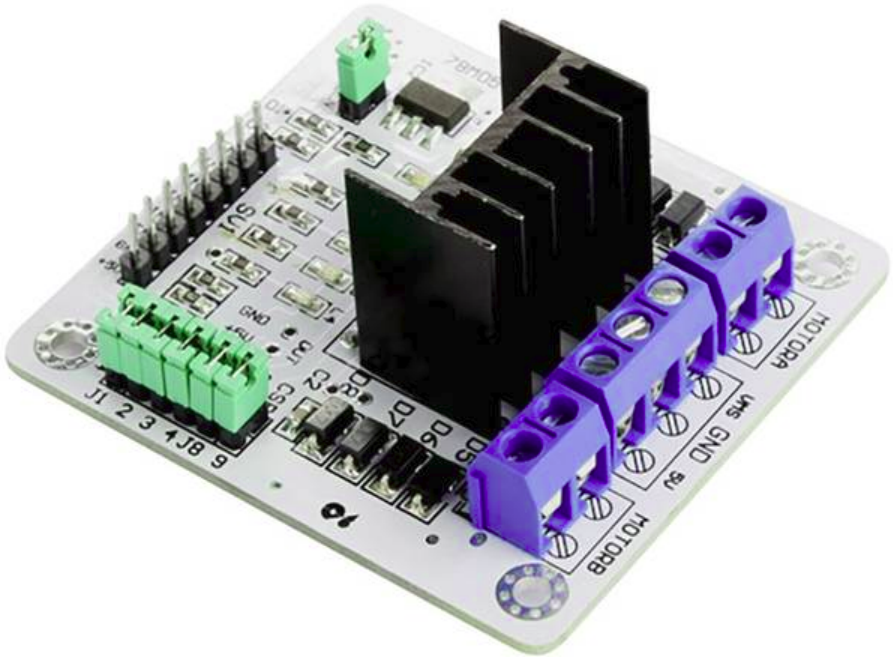
H-most je sklop sastavljen od četiri tranzistora ili MOSFET-a zadužen za kontrolu motora preko PWM signala.

Ulazni PWM signal uključuje i isključuje parove tranzistora te tako mijenja smjer struje koja protiče kroz motor. Shodno tome frekvencija upravljačkog signala bira se tako da perioda bude manja od električne vremenske konstante motora. Takvim odabirom frekvencije dobiva se stabilna struja jer se sama armatura motora ponaša kao filtar prvoga reda.

Prednost ovog sklopa je upravljanje DC motorom uz vrlo male gubitke u samom sklopu, no pošto se radi o prekidačkom sklopu njegovim djelovanjem uvodi se dodatan šum na frekvenciji ulaznog PWM signala. Iako je prije navedeno da sama armatura motora filtrira dio šuma kako bi dodatno kondicionirali signal uputno je dodati filter prvog reda nakon senzora struje.

Uzevši u obzir sve prije navedeno vrlo je lako zaključiti da je H-most najkorišteniji sklop u upravljanju DC motorima.

Unutar ovog rada bit će korišten integrirani H-most L298N, kojeg je moguće pribaviti kao poseban modul sa svim pripadajućim elektroničkim komponentama na jednoj tiskanoj pločici kao standardan modul.



Slika 9. L298N H-most modul

3.3.2. Senzor struje

Senzor struje kako samo ime govori omogućuje mjerenje struje unutar nekog strujnog kruga. Postoji više izvedbi senzora ovisno o njihovoj primjeni i načinu rada. Najviše korišteni senzori danas su:

- Senzor s Hall efektom
- Shunt otpornik
- Strujni transformator
- Senzor struje s optičkim vlaknom

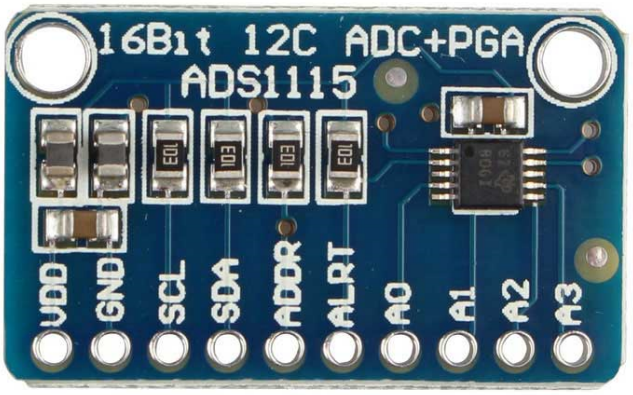
Unutar ovog rada detaljnije će biti objašnjeni Hall senzor i shunt otpornik. Razlog obrade ova dva senzora je da su oni najkorišteniji u trenutnim sklopovima te međusobno pokrivaju vrlo velik dio područja mjerenja struje.

Hall je 1879. godine otkrio ako kroz neki metal unutar magnetskog polja teče konstantna struja, okomito na tok struje javit će se Hallov napon proporcionalan jačini magnetskog polja. Kako je opće poznato da se prolaskom struje kroz vodič inducira magnetsko polje oko tog vodiča, Hall efekt se može koristiti za mjerenje struje unutar vodiča bez direktnog kontakta sa strujom. Samim time, takav način omogućuje mjerenje vrlo visokih struja, no nedostatak takvog mjerenja je mali inducirani napon za male struje.

Shunt otpornik za razliku od Hall senzora funkcionira na vrlo jednostavnom principu pada napona uslijed prolaska struje kroz vodič. Pad napona ovisi o vrijednosti otpora te je proporcionalan struji. Prednost ovog načina mjerenja je njegova jednostavnost i vrlo mala cijena pošto je za mjerenje struje potrebno imati voltmetar i otpornik malog otpora. Nedostatak je nemogućnost galvanskog odvajanja mjernog kruga od mjerenog te zagrijavanje otpornika kod prolaska struje što može narušiti točnost mjerenja.

Kako bi se podaci mogli vrlo točno isčitati bit će primijenjen vanjski AD pretvornik ADS1115(Prilog 9.3.1). Razlog primjene samog vanjskog AD konvertera je veća točnost kao i mogućnost diferencijalnog mjerenja što je vrlo velika prednost kod ispitivanja napona na shunt otporniku. Sljedeća prednost ovakvog mjerenja je sama zaštita mikrokontrolera u slučaju prenapona te se uz to smanjuje sama količina obrađenih podataka unutar čipa.

Uzevši sve nabrojeno unutar ovog rada bit će primijenjen ovakav AD konverter sa shunt otpornikom nazivne vrijednosti 0.82Ω.



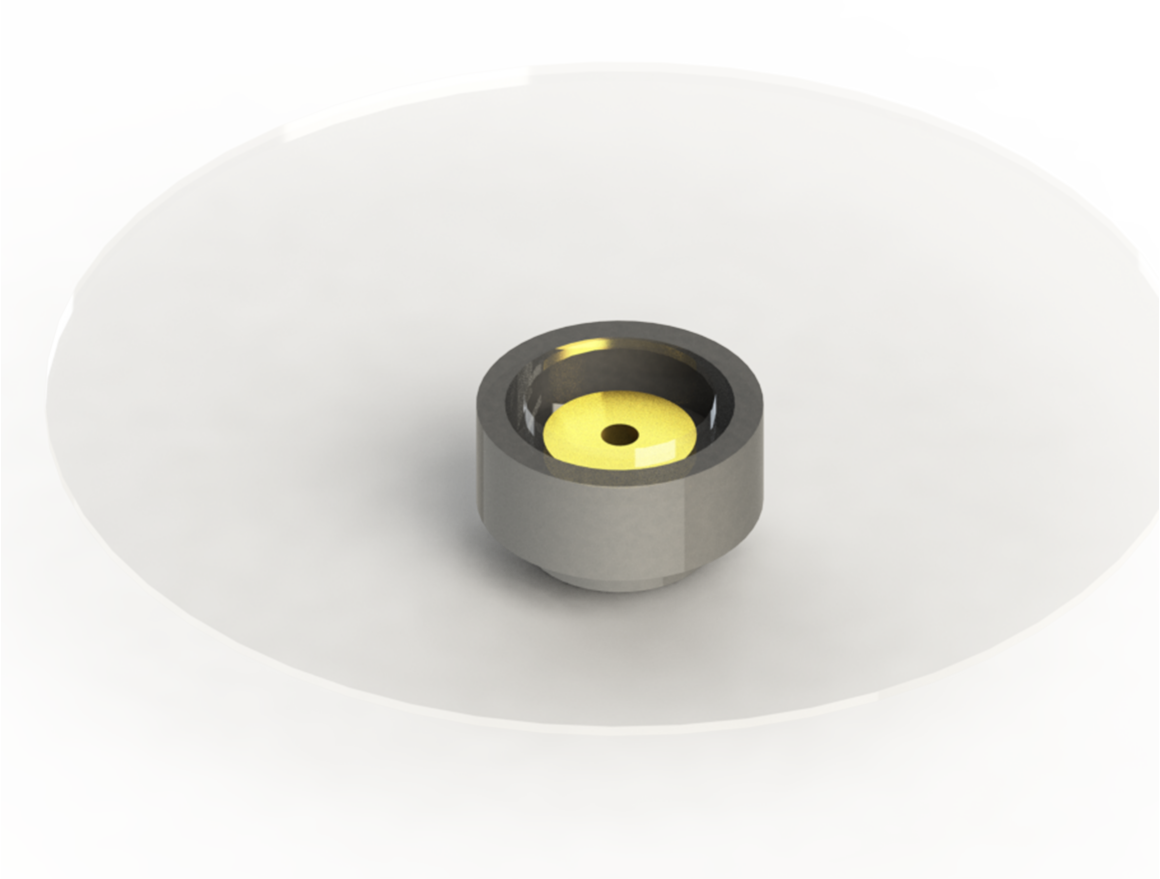
Slika 10. ADS1115 AD pretvornik

3.4. Izrada mehaničkog opterećenja

Pošto u realnosti niti jedan motor neće biti neopterećen poželjno je dodati mehaničko opterećenje. Unutar ovog rada primijenjeno je inertno opterećenje u vidu zamašnjaka na vratilu motora.

Naravno važno je napomenuti da u stvarnoj primjeni motor nikada neće biti opterećen samo inertnim opterećenje, već i nekim statičkim ili dinamičkim momentom, no ipak dodavanjem ovog opterećenja sama maketa je realističnija te bolje primjenjiva u izučavanju spektra praktičnih problema.

Zamašnjak je dizajniran unutar SolidWorks programskog paketa te je na njega dodan disk s prorezima u svrhu mjerenja brzine vrtnje. Naknadno je pomoću istog programskog paketa izračunat moment inercije samog zamašnjaka koji će kasnije biti primijenjen u simulaciji samog motora.



Slika 11. Inertno opterećenje motora

3.5. Izračun parametara motora

Kako je navedeno u uvodnom dijelu ovog poglavlja unutar podataka proizvođača nisu sadržani svi parametri motora potrebni za simuliranje te izradu regulatora. Shodno tome u ovom dijelu bit će prikazan način eksperimentalnog utvrđivanja prametara.

Prvotno je potrebno izraditi model motora unutar programskog paketa Simulink te nakon toga pomoću modula za estimaciju parametara odrediti iste.

3.5.1. Izrada modela motora u Simulink-u

Prvo će biti izrađen model armature motora pomoću kojeg se simulira ovisnost struje o ulaznom naponu. Kako bi se definirali parametri armature (pojačanje i vremenska konstanta) potrebno je odrediti otpor i induktivitet armature.

Otpor armature može se odrediti jednostavnim spajanjem ommetra na motor te iščitavanjem srednje vrijednosti otpora. Vrijednost otpora nije uvijek ista zbog komutatorskih četkica.

Induktivitet je moguće odrediti preko formule za induktivnu reaktanciju:

$$X_L = \frac{V}{I} = 2\pi f L_a. \tag{3.6}$$

Rješavanjem jednadžbe po L_a dobiva se:

$$L_a = \frac{V}{2\pi f I}. \tag{3.7}$$

Za izračun ovih vrijednosti potrebno je upotrijebiti sinusni izmjenični napon te se tada napon i struja mogu uzimati kao RMS vrijednosti koje je moguće očitati s bilo kojeg voltmetra odnosno ampermetra. Pomoću gore navedenih podataka može se odrediti pojačanje i vremenska konstanta armature:

$$K_a = \frac{1}{R_a}, \tag{3.8}$$

$$T_a = \frac{L_a}{R_a}. \tag{3.9}$$

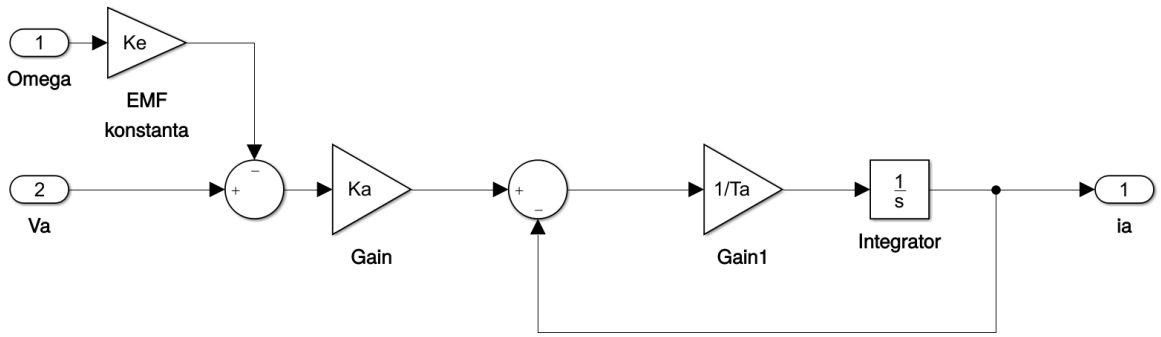
Pomoću formule:

$$K_a u_a = T_a \dot{i_a} + i_a \tag{3.10}$$

te ako je:

$$u_a = V_a - K_e \omega \tag{3.11}$$

dobiva se sljedeća blok shema.



Slika 12. Model armature motora

Model armature se maskira kao jedan blok te se pristupi izradi mehaničkog dijela modela motora.

Pošto unutar mehaničkoga modela motora postoje parametri koje je mnogo teže odrediti jednostavnim pokusom kako je to bilo kod armature svi parametri će biti približno definirani kako bi omogućili pokretanje simulacije. Shodno tome dobivanje parametara bit će objašnjeno samo putem modula za estimaciju.

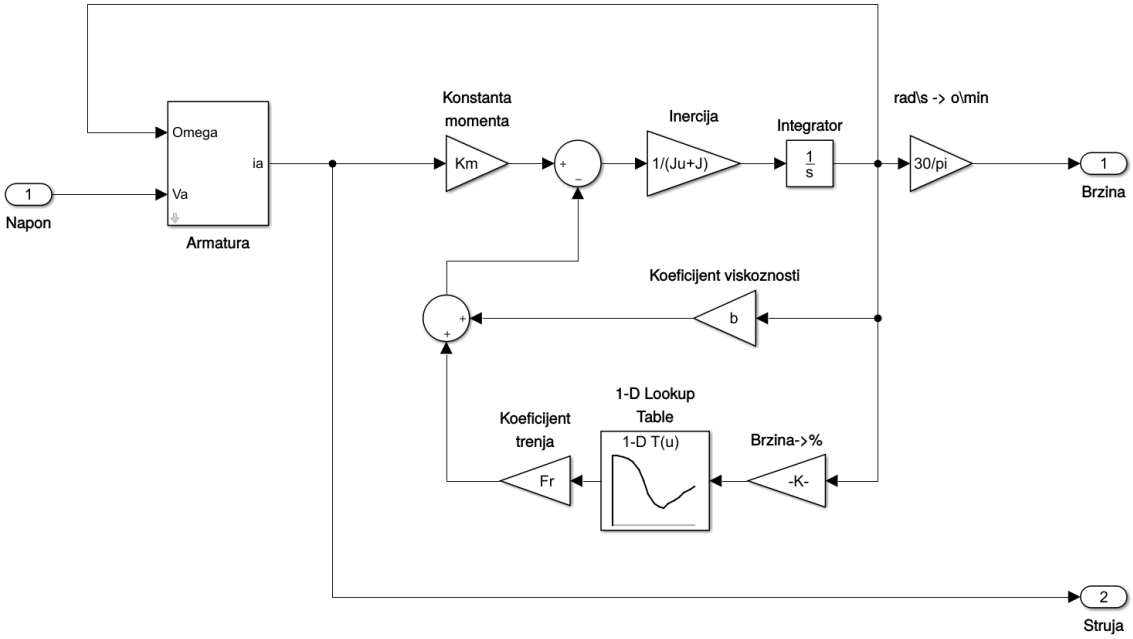
Kako bi se mogao izraditi model mehanike motora prvotno treba postaviti neke zakonitosti. Množenjem struje armature s konstantom momenta K_m dobva se moment, nakon toga od punog momenta oduzimaju se sva opterećenja te se nakon toga množi s inverzom momenta inercije. Integracijom ove varijable dobiva se brzina u radijanima u sekundi. Množenjem brzine s konstantom elektromotorne sile dobivamo povratni elektromotorni napon koji se oduzima od ulaznog napona. Shodno tome sam motor je automatski sustav s regulirajućom povratnom petljom.

Unutar ovog rada motor će biti opterećen samo statičkim i dinamičkim trenjem te viskoznim prigušenjem, no pošto nisu vrijednosti poznate konstante će biti određene pomoću estimacije.

Sustav prvotno može biti prikazan pomoću diferencijalne jednadžbe:

$$K_m i_a = b\omega - F_{truk}\omega + (Ju + J)\dot{\omega} \tag{3.12}$$

gdje je sa F_{truk} obuhvaćen cijeli model trenja.



Slika 13. Model mehanike motora

Kako je vidljivo trenje je modelirano tako da je zbrojen model Coulombovog i Stribeckovog trenja te je implementirano kao 1-D "*lookup*" tablica.

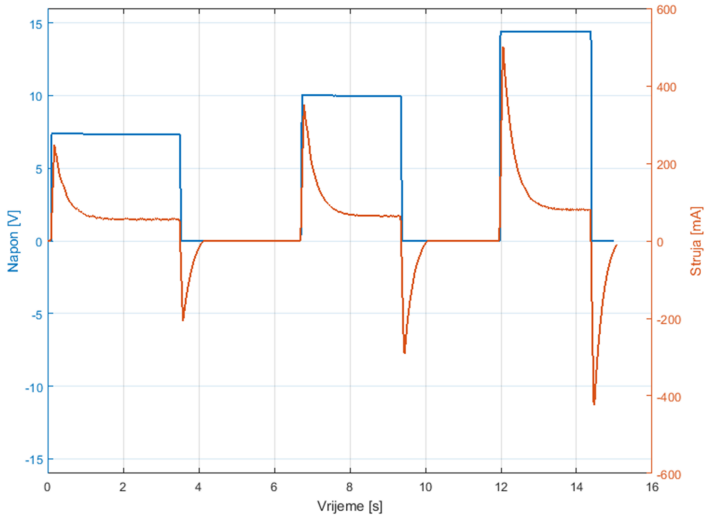
Posljednji dio samog modela je dizajnirati senzor struje tako da odgovara onom korištenom kod eksperimentalnog snimanja podataka.

3.5.2. Snimanje karakterističnih odziva motora

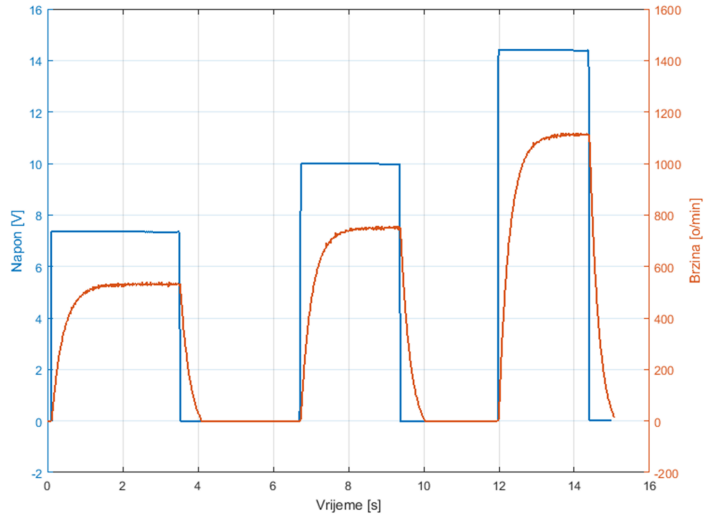
Odzivi motora snimljeni su pomoću mikrokontrolera koji će kasnije biti korišten u upravljanju samim motorom. Ulazni napon zadavan je s laboratorijskim napajanjem kao slijed više step funkcija različitih amplituda.

Unutar mikrokontrolera je jednostavan program napisan u Arduino razvojnom okruženju. Program uzima vrijednosti sa senzora struje, brzine te napona konvertira ih po potrebi te sprema u vlastiti RAM. Nakon spremljenog određenog broja točaka akvizicija staje te se podaci serijskom komunikacijom prenose na matično računalo.

Jednom kada su podaci prikazani unutar prozora za serijsku komunikaciju, mogu se kopirati te spremiti u običan tekstualan dokument. Programski paket Matlab može spremljene podatke učitati u vlastitu memoriju te omogućiti daljnji rad s istima. Uz snimljene podatke važno je spremiti vremenski interval akvizicije istih kako bi izvedba simulacije bila moguća.



Slika 14. Odziv struje



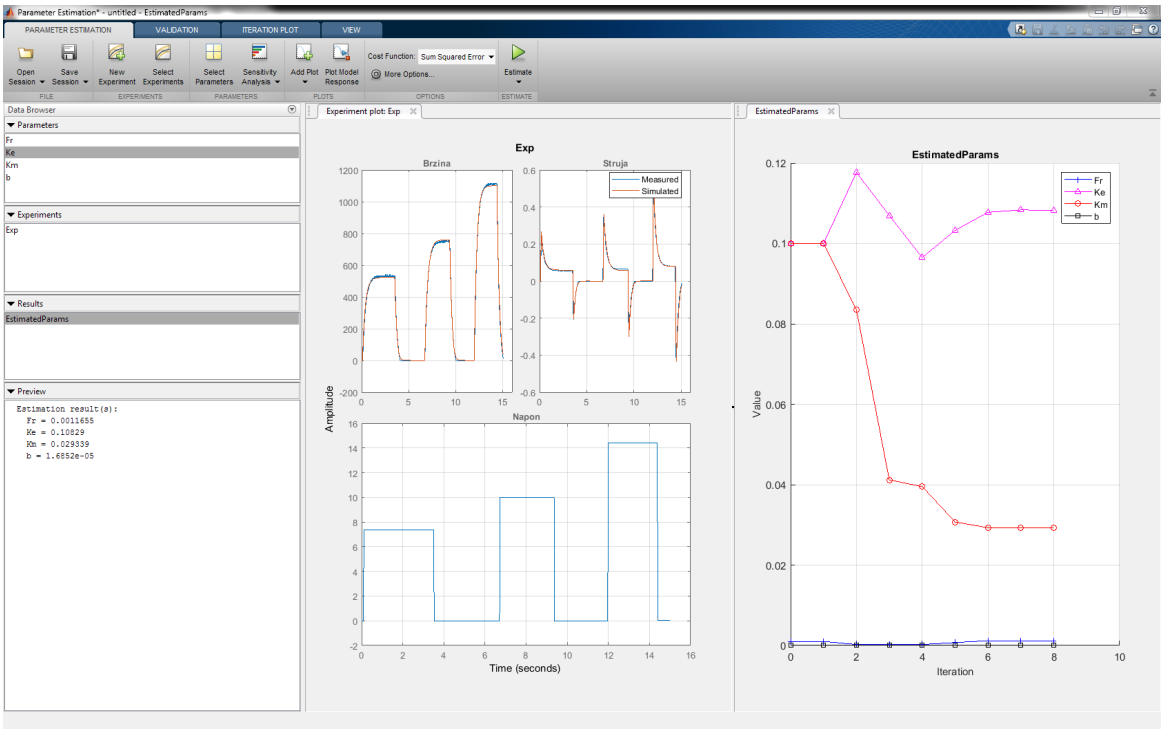
Slika 15. Odziv brzine vrtnje

3.5.3. Estimacija parametara

Predhodno je navedeno da se unutar programskog paketa Matlab nalazi modul za estimaciju parametara nekog sustava ako je poznat odziv istog. Unutar ovog dijela ukratko će biti objašnjen način dobivanja parametara pomoću ovog modula. Modul funkcionira unutar Simulink alata tako što radi pertubaciju parametara pokušavajući smanjiti RMS pogrešku između zadanih odziva i simuliranih odziva. Kako bi sam sustav bolje estimirao parametre vrlo je važno imati dovoljno dobar model sustava te odrediti sve parametre koje je moguće. Drugim riječima iako je sama estimacija vrlo dobar postupak, ipak ima svoje granice te može dati pogrešne rezultate.

Postupak estimacije započinje izradom modela unutar Simulink-a što je izrađeno ranije. Na model je potrebno dodati ulaze i izlaze kako bi sustav znao koje se vrijednosti mjere. Sljedeći korak je otvaranje modula za estimaciju te dodavanje polja u kojima se nalaze eksperimentalno dobiveni podaci. Prije samog pokretanja sustav će izvršiti simulaciju, zbog čega je bilo potrebno proizvoljno odabrati parametre te prikazati razliku između realnih podataka i simulacije. Unutar samog modula moguće je dodatno mijenjati funkciju cilja te još dodatne parametre, no za potrebe ovog rada sve će biti standardno.

Kada se pokrene estimacija sustav će ispisivati prema određenoj funkciji cilja vrijednost greške, kada se greška više ne mijenja sustav staje i predpostavi da su parametri dovoljno dobri. Ako nakon estimacije ne dolazi do poklapanja simuliranih vrijednosti i stvarnih podataka, vrlo vjerojatno postoji greška negdje u modelu, parametrima ili je sam model previše jednostavan da kvalitetno opiše dinamiku sustava.



Slika 16. Estimacija parametara

Iz (Slike 16.) vidljivo je da nakon 9 koraka estimacija staje, ali i da se mjereni i simulirani podaci vrlo dobro poklapaju (iznos RMS greške je 0.1205), pa se može zaključiti da parametri dobro oslikavaju stvarne karakteristike motora.

4. PI REGULATOR

Sljedeći logičan dio ovog rada je dizajniranje PI regulatora struje unutar programskog paketa Simulink, te nakon toga izrada istog unutar C++ programskog jezika kako bi mogao biti primijenjen u Arduino razvojnom okruženju.

Svaki mikrokontroler je prema definiciji digitalan sustav tj. radi u diskretnim stanjima pa je shodno tome potrebno sva kašnjenja akvizicije i proračune vrijednosti regulatora obuhvatiti simulacijom. Sustav se prema tome može dizajnirati na dva različita načina, kao kontinuirani sustav s dodatnim blokovima koji opisuju sva kašnjenja ili kao diskretni regulator s AD i DA pretvaračima i vremenom uzorkovanja jednakim onom unutar mikrokontrolera.

Kako bi se mogle prikazati prednosti i nedostaci jednog i drugog sistema simulacije u nastavku bit će obrađena oba pristupa.

4.1. Kontinuiran PI regulator

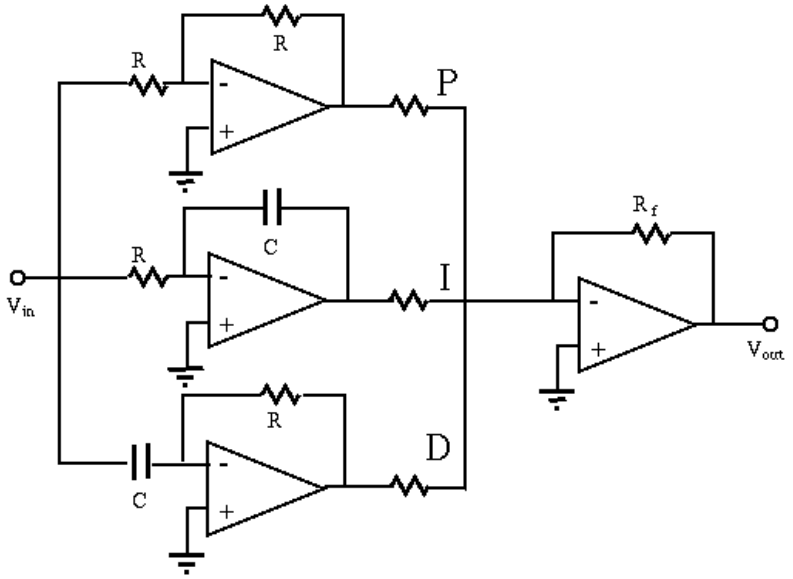
Kontinuirani PI regulator predpostavlja kako unutar sustava nema diskretizacije drugim riječima procesne vrijednosti su poznate u svim trenucima te u svim amplitudama. Tipično ovakav način izrade simulacije te samog regulatora bio bi izvediv kod analognog sklopovlja sa specijalnim čipovima samo za namjenu reguliranja nekog procesa.

Prednosti ovakvog regulatora su jednostavnija izrada simulacijskog modela kao i proračuna parametara samog regulatora. Kontinuirani regulatori su mnogo bliži stvarnom svijetu pošto u realnosti nema digitalnih stanja. Sam proračun ovakvih regulatora je vrlo dobro definiran te postoje uputsva korak po korak kako namjestiti parametre.

Nedostaci su korištenje specijaliziranih čipova ili neke druge analogne tehnologije u svrhu regulacije sustava. Ovakav način dizajniranja limitira broj mogućnosti te mogućnost preinaka samog regulatora, da bi se promijenili parametri mora se mijenjati dio sklopa.

Primjena ovakvih regulatora u današnje vrijeme sve je rjeđa jer postoje kvalitetne teorije izrade diskretnih regulatora za kontinuirane procese te su cijene mikrokontrolera vrlo niske.

Primjer analognog PID regulatora je izrađen iz operacijskih pojačala svako od njih u specifičnom spoju proporcionalnog, integralnog ili derivacijskog djelovanja. Kako bi se promijenili parametri regulatora potrebno je promijeniti vrijednosti otpora i/ili kapaciteta.



Slika 17. Primjer analognog regulatora

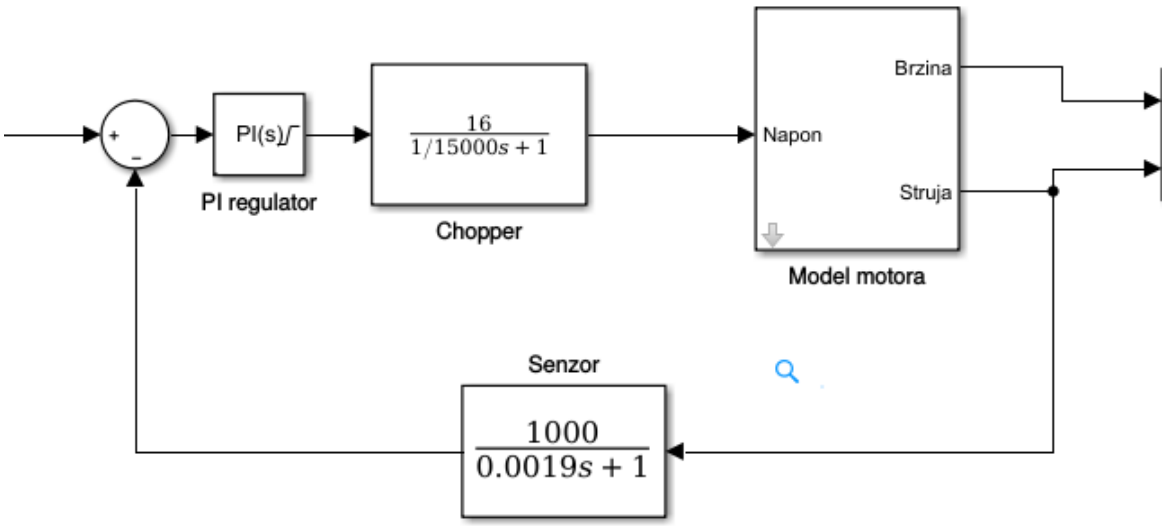
4.1.1. Simulink model

Uvodno je bilo rečeno da svako kašnjenje pretvorbe i akvizicije modeliramo kao član s kašnjenjem prvog reda.

Unutar ovog rada se koristi AD konverter za akviziciju struje motora, kao upravljački sustav koristi se PWM signal te, kako je uvodno spomenuto, H-most.

Kod dizajniranja članova koji zamjenjuju ulazno izlazne diskretne module kao pojačanje uzima se konverzijski faktor te kao vremenska konstanta samo kašnjenje modula, ako se radi o AD konverteru uzima se vremenska konstanta pretvorbe, a ako se radi o PWM generatoru upravljačkog signala uzima se inverz frekvencije.

Problem ovakvog načina izrade je što se i dalje radi o kontinuiranom modelu s dodanim kašnjenjima pa sam regulator tek približno opisuje ponašanje stvarnog. Prema tome parametri dobiveni na ovakav način neće uvijek dobro opisati stvarni sustav.



Slika 18. Model s kontinuiranim regulatorom

Iz samog modela vidljivi su blokovi koji prikazuju kašnjenja prije navedenih dijelova sklopa, pa tako možemo zaključiti da sama akvizicija vrijednosti struje traje 1.9ms te da je frekvencija PWM signala 15kHz. Uz vremenske konstante možemo primijetiti da je pojačanje senzora sklopa 1000 tj. da mjerimo struju u mA te iz istog razloga referentnu vrijednost zadajemo isto u mA. Sljedeće vidljivo pojačanje je kod sklopa H-most tzv. "*chopper*", koje u ovom slučaju iznosi 16 pošto je napon napajanja sklopa 16V.

Sama svrha ovako detaljno razrađenog modela je jednostavnije određivanje parametre samog regulatora što će biti predstavljeno u sljedećem dijelu.

4.1.2. Izračun parametara regulatora

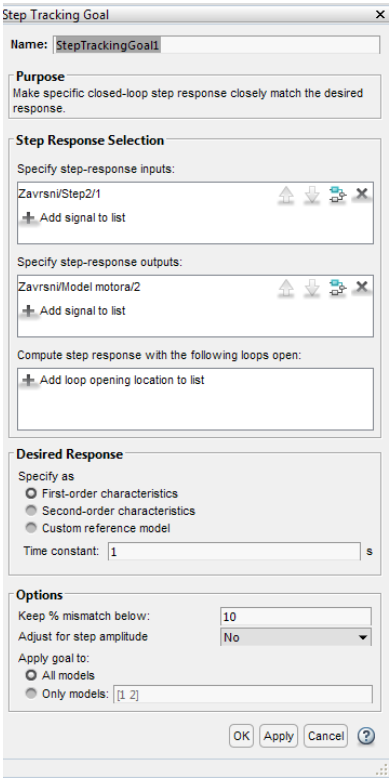
Izračunavanje parametara je izvedivo na više načina; neki od njih spadaju u standardnu teoriju regulacije i primjenjuju se već dugi niz godina kao što su na primjer Ziegler-Nicholsova metoda sinteze regulatora ili Takahashi metoda za diskretne regulatore. Sljedeći način je s automatskim programskim alatima za optimiziranje regulatora kao što je Matlab modul "*Control System Tuner*" koji nam omogućuje da zadavanjem zahtjeva na neki sistem dizajniran unutar Simulink sustava odredimo približno optimalan regulator.

Govoreći o dva različita pristupa sintezi regulatora vidljiva je i glavna razlika među njima: kod metoda sinteze iz teorije regulacije kako bi se izračunali parametri regulatora potrebno je eksperimentalno izmjeriti sam odziv ili dovesti sustav u neko vrlo točno određeno stanje, što u većini slučajeva vodi do pogrešaka realnih mjerenja. S druge strane sam model unutar nekog simulacijskog sustava neće imati greške mjerenja te će sustav moći ispitati mnogo veći broj mogućih rješenja pa će sam dizajner sustava imati mnogo više mogućnosti nametnuti strože zahtjeve na sustav, no točnost samog modela u krajnjem slučaju ovisi samo o točnosti izmjerenih podataka putem kojih je taj sustav dizajniran.

Prethodno prikazanim podacima dolazi se do zaključka da bez obzira koliko dobar i optimiziran sustav ispada unutar simulacije ili kod samog proračuna, već početna pretpostavka ponašanja sustava nije izričito točna pa će i sami parametri ponekad biti vrlo daleko od optimalnih. Kao takvi će mnogo puta ovisiti o točno određenim uvjetima u kojima se nalazi sustav.

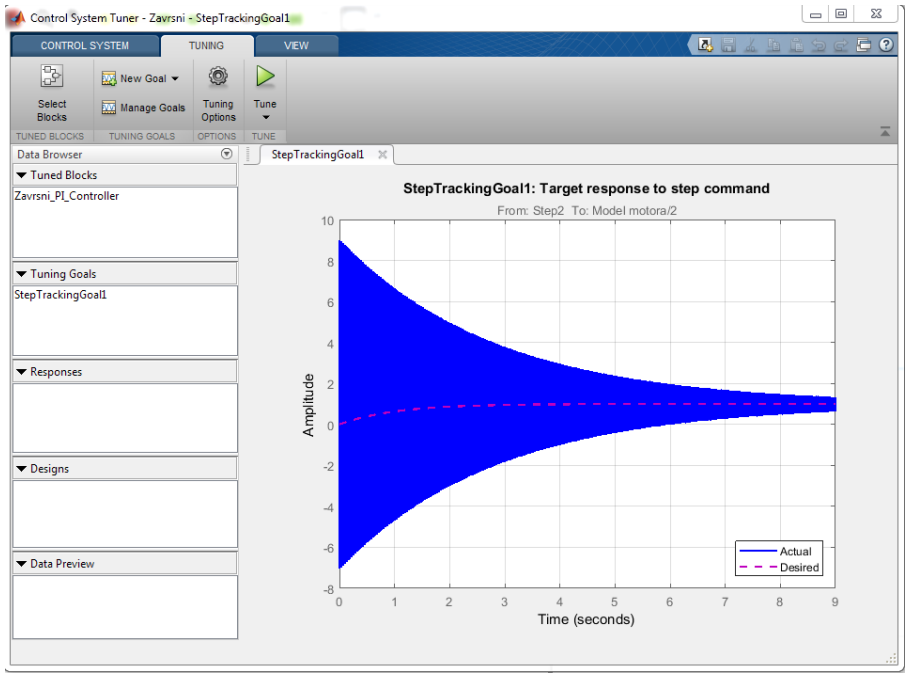
Unutar ovog rada parametri regulatora bit će određeni pomoću paketa Simulink pošto je već dizajniran cijeli model te je moguće primijeniti mnogo širi spektar zahtjeva na regulaciju.

Prvotno se odabire cilj koji se želi postići, u ovom slučaju praćenje step funkcije te se nakon toga unose podaci o ulaznom i izlaznom signalu. Posljednje se unose zahtjevi na odziv. Tako je moguće odabrati funkciju prvog ili drugog reda te dodatne parametre kao što su vrijeme rasta i postotni prebačaj.



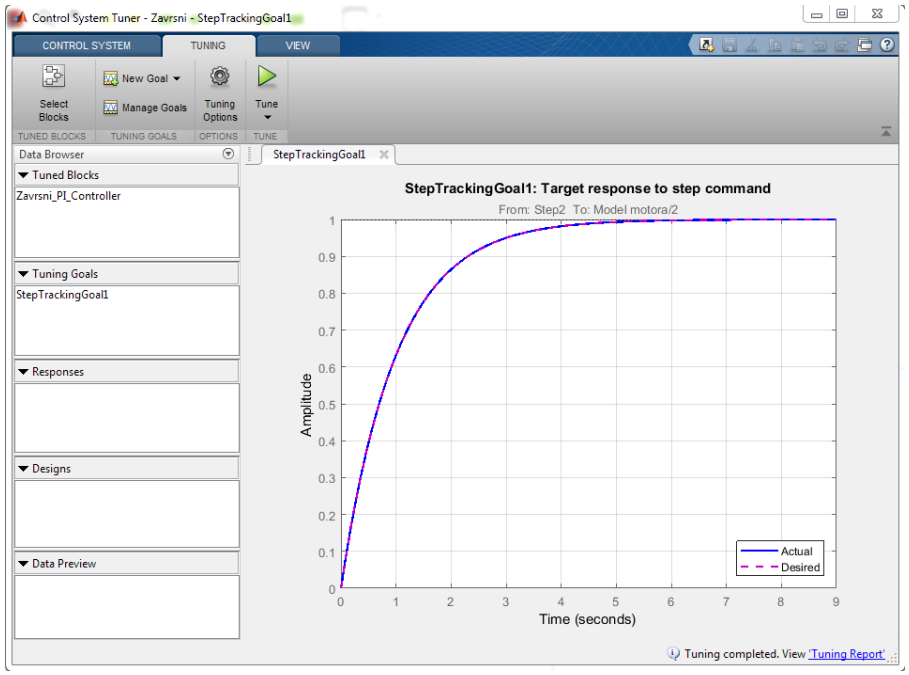
Slika 19. Upis zahtjeva za regulaciju

Sljedeći korak je potvrditi unos tako da će sustav automatski generirati simulirani odziv te zahtjevani odziv. Ako je korisnik zadovoljan zahtjevanim odzivom može se pritisnuti tipka ”*Tune*”, a sustav pronalazi parametre za koje će odziv biti što bliže zadanome.



Slika 20. Izračun parametara u Simulink-u

Sustav će prikazati odziv kao na slici niže.



Slika 21. Podešen odziv sustava

4.2. Diskretni PI regulator

Ranije je bilo napomenuto kako sami mikrokontroleri rade kao digitalni sustav tj. ulazne i izlazne vrijednosti diskretizirane su po vremenu i po vrijednosti. Kako bi bilo moguće opisati takve sustave unutar teorije regulacije izvedena je Z-transformacija, koja je formalno ista kao i S-transformacija u kontinuiranoj domeni.

Primjena diskretnih regulatora danas je vrlo raširena kako je već ranije napomenuto pa je vrlo uputno posvetiti i njima dio svakog razlaganja na temu regulacije. Prednosti ovakvih sustava su isti kao i kod mikrokontrolera tj. integriranih sustava. Takve sustave je vrlo lako prenamjeniti samo pomoću jednostavne promjene programa, za razliku od analognih sustava kod kojih je bilo potrebno mijenjati dijelove ili cijeli strujni krug. Nedostatak ovakve regulacije je teže matematičko opisivanje samog sistema te izračun parametara samog regulatora. Dodatan nedostatak općenito diskretnih sustava je da se samom diskretizacijom stvara efekt niskopropusnog filtra s vremenskom konstantom jednakom periodu uzorkovanja.

Iako je unutar ovog rada obrađen takoreći diskretan sustav zbog velike brzine akvizicije podataka te izdavanja novih naredbi samo proučavanje se može izvesti pomoću kontinuirane teorije sustava. Ipak ako bi se htjelo izvesti parametre sustava pomoću diskretne metode uputno je koristiti Takahashi metodu.

Takahashi metoda određivanja parametara je vrlo bliska Ziegler-Nicholsonovoj metodi za otvoreni krug. Postupak koji se primjenjuje je jednak, no dodatno treba uzimati u obzir periodu uzorkovanja.

Ako bi se htjeli odrediti sami parametri regulatora unutar Simulink programskog alata bilo bi potrebno odrediti neke postavke same simulacije; prvotno odabrati neku metodu rješavanja diferencijalnih jednadžbi za diskretne sustave. Nakon toga je potrebno sve blokove zamijeniti diskretnim blokovima te upisati periode uzorkovanja.

Kao posljednja solucija postoje i takozvani hibridni sustavi kod kojih je sama staza kontinuirana dok je regulator diskretan, sama izmjena signala omogućena je pomoću blokova ”*Zero-Order Hold*”. Ovakvi sustavi se ne mogu linearizirati unutar Simulink programskog alata stoga se parametri regulatora određuju nekom od metoda za kontinuirane sustave te se rade dodatne prilagodbe.

Iako je sam mikrokontroler diskretni sustav, zbog vrlo malog perioda akvizicije tj. velike brzine uzorkovanja u ovom radu bit će primijenjen kontinuirani regulator.

5. KOD UNUTAR MIKROKONTROLERA

Unutar ovog dijela bit će objašnjen dio po dio koda te neki osnovni principi primijenjeni u izradi istog. Pošto će u prilogu biti prikazan cijelokupan kod, unutar ovog dijela kod prikaza kodova neće biti naglaska na toku samog programa nego više na pojedinačnim modulima.

Kako bi se postigla potpuna funkcionalnost samog koda bilo je potrebno upotrijebiti obje jezgre mikrokontrolera te će i kao prvi dio biti prikazan način razdjeljivanja procesa na dvije jezgre.

Jezgra 0 izvršava: slanje podataka u databazu.

Jezgra 1 izvršava: primanje naredbe, regulaciju sustava te akviziciju podataka.

5.1. Paralelno procesiranje

Naziv paralelno procesiranje unutar hrvatskog jezika podrazumijeva više različitih definicija, no važno je razdvojiti prividno paralelno procesiranje od stvarnog.

Prividno paralelno procesiranje je moguće unutar svakog procesora na način da se više procesa izvršava sekvencijalno, no važno je napomenuti da se ovakvim načinom ne postiže izvršavanje dvaju procesa u isto vrijeme. Ovakav način izvođenja je vrlo koristan kod stolnih računala ili poslužitelja kod kojih krajnji korisnik nije dovoljno brz da primijeti kontinuiranu izmjenu procesa na istoj jezgri, no u sklopu ovog rada postoji dio akvizicije signala koji mora uvijek biti u realnom vremenu te bez zastajkivanja. Shodno tome bit će primijenjeno stvarno paralelno procesiranje.

Stvarno paralelno procesiranje je moguće samo kod procesora koji posjeduju dvije ili više jezgre koje su u stanju obavljati iste operacije, drugim riječima procesor mora posjedovati dvije ili više "*CPU*" jedinice, pošto neki posjeduju i "*FPU*" jedinice koje služe samo za "*floating-point*" operacije. U većini slučajeva unutar podataka proizvođača definiran je broj jezgri.

Paralelno procesiranje neće raditi samo ako neki procesor posjeduje više procesorskih jezgri nego je potrebno programski uključiti rad obiju jezgri te dodatno definirati koja jezgra izvršava koji proces.

```
void setup(){
        xTaskCreatePinnedToCore(
                runcore0,           // Funkcija na jezgri 0
                "runcore0",         // Ime funkcije na jezgri 0
                10000,              // Veličina stack-a dodjeljenog procesu
                NULL,               // Parametri procesa
                1,                  // Prioritet procesa
                &xTask1,            // Handle za proces 0
                0);                 // Broj jezgre na kojoj se izvršava proces
        xTaskCreatePinnedToCore(
                runcore1,           // Funkcija na jezgri 1
                "runcore1",         // Ime funkcije na jezgri 1
                10000,              // Veličina stack-a dodjeljenog procesu
                NULL,               // Parametri procesa
                1,                  // Prioritet procesa
                &xTask2,            // Handle za proces 1
                1);                 // Broj jezgre na kojoj se izvršava proces
}
void runcore0(void * parameter){
        .......                     // Kod koji se izvršava na jezgri 0
}
void runcore1(void * parameter){
        .......                     // Kod koji se izvršava na jezgri 1
}
```

Kod 1. Paralelno procesiranje

5.2. Web poslužitelj

Funkcionalnost samog sustava ovisi o mogućnosti komuniciranja preko interneta. Iako samo ime sugerira da se radi o pravom Web poslužitelju sa svim mogućnostima koje u današnje vrijeme isti posjeduju, ovaj poslužitelj je ipak vrlo jednostavniji.

Jedina namjena unutar ovog rada je da zaprimi slijed podataka poslan kao argument unutar hiper-linka. Način slanja samih podataka na ovaj način bit će obrazložen unutar poglavlja koje se bavi izradom Web stranice i sučelja. Kada se na njih pošalje zahtjev u vidu hiper-linka, većina poslužitelja odgovara sa složenom stranicom. Ovaj Web poslužitelj odgovara jednostavnim odgovorom unutar standarda internet komunikacije, da su podaci zaprimljeni te da klijent može prekinuti komunikaciju.Takvim pristupom štedi se na memoriji i procesorskom vremenu.

Server je unutar mikrokontrolera pokrenut na *jezgri 1*, koja je kako će kasnije biti vidljivo zadužena i za akviziciju podataka sa senzora(Slika 1.).

```
//Pruotni poziv biblioteka za normalan rad Web poslužitelja
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

//Uključivanje WiFi modula
void setup(){
        .....
        WiFi.begin(NET_NAME, PASS);
        .....
}

//Paljenje i konfiguracija unutar jezgre 1

void runcore1(void * parameter){
        ....
        server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
                {
                        int paramsNr = request->params();
                        for(int i=0;i<paramsNr;i++)
                                {
                                        AsyncWebParameter* p = request->getParam(i);
                                        if(p->name()=="napon")
                                        {
                                                napon_cont=p->value();
                                                flag1=true;Serial.println(flag1);
                                        }
                                }
                        request->send(200, "text/plain", "");
                });
        ....
        }
```

Kod 2. Web poslužitelj

5.3. PI regulator

Najvažniji dio koda je implementacija PI regulatora unutar samog mikrokontrolera. Integralno djelovanje regulatora je implementirano pomoću konačne sume te je dodatno stavljen "*anti-windup*" algoritam, koji funkcionira tako što blokira upravljačku veličinu na zadanoj vrijednosti. Implementacija ovog algoritma postignuta je pomoću "*if*" uvjeta koji otkrivaju trenutnu vrijednost upravljačkog signala te prema tome blokiraju prevelike vrijednosti.

Izlaz algoritma ograničen je rasponom od negativne do pozitivne vrijednosti sto, što se kasnije pretvara u dva PWM signala suprotnih polariteta, kojima se ostvaruje upravljanje motora.

Dodatni dio algoritma je implementacija predznaka mjerene struje. Pošto se pomoću senzora struje mjeri uvijek apsolutna vrijednost, algoritam pomoću predznaka upravljačke veličine te uz predpostavku da je kašnjenje odziva struje vrlo malo, mijenja predznak mjerene veličine.

```
// Učitanje struje
float struja = ads.readADC_Differential_2_3()*0.125*1.185;

//Izračun greške
float i_err=i_ref-struja*coeff/1000.0;

//Suma integralnog djelovanja
integral_i=integral_i+ki*i_err;

//Izračun reference
float u_ref=kp*i_err+integral_i;

//"Anti-windup" clamping metoda
if(u_ref>100){u_refa=100;u_refb=0;coeff=1;integral_i=100/ki;}
if(u_ref>0 && u_ref<=100){u_refa=u_ref;u_refb=0;coeff=1;}
if(u_ref==0){u_refa=0;u_refb=0;}
if(u_ref<-100){u_refa=0;u_refb=100;coeff=-1;integral_i=-100/ki;}
if(u_ref<0 && u_ref>=-100){u_refa=0;u_refb=-u_ref;coeff=-1;}

//Zadavanje postotne vrijednosti PWM signala
set_duty(u_refa,u_refb);
```

Kod 3. PI regulator

Iz same implementacije je vidljivo da se radi o vrlo jednostavnom algoritmu, no samom jednostavnošću dobiva se velika brzina izračuna upravljačkih vrijednosti. Kao što je bilo ranije napomenuto, ako se vrlo brzo izračunavaju upravljačke vrijednosti, sustav možemo smatrati kontinuiranim.

5.4. Slanje podataka u databazu

Slanje podataka implementirano je na način da ne smeta samoj akviziciji pomoću *jezgre 1*. Kako bi se omogućila kontinuiran akvizicija podataka u trenutku slanja podataka, akvizicija se mora vršiti u drugu varijablu. U tu svrhu inicijalizirane su po dvije varijable sličnih imena koje se naizmjenično pune i prazne podacima, drugim riječima kada jedna varijabla sprema podatke druga šalje prethodne podatke u databazu.

Podaci koji se šalju na poslužiteljsku databazu zapakirani su unutar "*JSON*" formata, jer je taj format zasigurno jedan od najkorištenijih standardnih formata unutar internet sustava. Kada se podaci zapakiraju u jednu varijablu pomoću funkcije "*POST*" šalju se na poslužitelj koji ih dodatno obrađuje te sprema u databazu. Sama manipulacija nad podacima sa poslužiteljske strane bit će obrazložena u sljedećem poglavlju.

Nakon što su podaci poslani u databazu, varijable za trenutno spremanje podataka se brišu te su spremne za ponovno pisanje. Zamjena varijabli definirana je brojem podataka u njima koji je definiran od strane programera.

```
//Zamjena je definirana brojem podataka (svakih 500)
if(500*k==i){
        k++;
        flag_send=true;
        if(flag_sw){flag_sw=false;}
        else{flag_sw=true;}
}
........
//Ovisno o stanju zastave flag_sw upis podataka u jednu od varijabli
if(flag_sw){
        brzina_array1+=String(brzina)+",";
        struja_array1+=String(struja)+",";
        naredba_array1+=String(i_ref*1000)+",";
        vrijeme_array1+=String(vrm)+",";
}
else{
        brzina_array+=String(brzina)+",";
        struja_array+=String(struja)+",";
        naredba_array+=String(i_ref*1000)+",";
        vrijeme_array+=String(vrm)+",";
}
```

Kod 4. Izmjena varijabli kod akvizicije

U nastavku će biti prikazano kako ista zastava koja je prikazana u gore navedenom kodu utječe na pakiranje podataka te na njihovo slanje na poslužitelj.

Pošto su u prijašnjem kodu prikazane sve varijable koje se koriste kod akvizicije, unutar sljedećeg koda bit će prikazana samo jedna kako bi se uputilo na dio u kojem se varijable koriste.

Dodatno bit će prikazan i dio koda zadužen za definiranje kraja podataka koji je implementiran na način da se nakon svih podataka tj. nakon kraja eksperimenta, pošalje jedan red u kojem su sve vrijednosti nule osim polja vremena koje je namjerno uzeto kao negativna jedinica. Ovakvim načinom definiranja kraja slanja podataka smanjuje se količina utrošenog procesorskog vremena na samom poslužitelju jer nakon što su primljeni svi podaci skripta više ne mora vršiti osvježavanje dijagrama.

```
//Slanje zadnjeg reda
http.begin(SITE);
http.addHeader("Content-Type", "application/json");
// Slanje podataka pomoću POST
http.POST("{\"struja\":[0],\"brzina\":[0],\"vrijeme\":[-1],\"naredba\":[0]}");
http.end();

//Varijable unutar if uvjeta su zamjenjene
if(flag_sw){
        //Brisanje suvišnog zareza unešenog u kodu kod akvizicije
        struja_array.remove(struja_array.length()-1);
        ........

        //Pakiranje svih vrijednosti unutar jedne varijable
        JSONmessageBuffer="{\"struja\":["+struja_array+
                          "],\"brzina\":["+brzina_array+
                          "],\"vrijeme\":["+vrijeme_array+
                          "],\"naredba\":["+naredba_array+"]}";

        //Brisanje vrijednosti varijabla
        struja_array="",brzina_array="",naredba_array="",vrijeme_array="";
        }
else
        {
        struja_array1.remove(struja_array1.length()-1);
        ........

        JSONmessageBuffer="{\"struja\":["+struja_array1+
                          "],\"brzina\":["+brzina_array1+
                          "],\"vrijeme\":["+vrijeme_array1+
                          "],\"naredba\":["+naredba_array1+"]}";

        struja_array1="",brzina_array1="",naredba_array1="",vrijeme_array1="";
        }

//Slanje podataka
http.begin(SITE);
http.addHeader("Content-Type", "application/json");
http.POST(JSONmessageBuffer);
http.end();
```

Kod 5. Slanje podataka

6. WEB STRANICA

Stranica će biti, kako je i u uvodnom dijelu rečeno, izrađena unutar razvojnog sistema Web2Py. Razvojni sustav može biti pokrenut na računalu na kojem se izrađuje sama aplikacija te kasnije učitana na sam poslužitelj ili je moguće sam razvojni sustav pokretati na poslužitelju na kojem će biti kasnije pokrenuta aplikacija.

Za potrebe ovog rada razvojni sustav bit će pokrenut na poslužitelju, no to neće uvelike promijeniti kod. Jedina razlika će biti u databazi koja će se koristiti, primarno Web2Py koristi SQLite databazu, no u slučaju ovog rada koristi se MySQL. Promjenom databaze neće se promijeniti kod pošto Web2Py koristi DAL ("*Database Abstraction Layer*") modul, koji omogućuje komunikaciju sa različitim bazama podataka uz pomoć iste Python sintakse.

Razvojno okruženje koristi princip "*Model-View-Controller*". Cilj ovakvog pristupa je razdijeliti dio koda koji se izvršava na poslužitelju od dijela koji vidi korisnik kod samog razvoja aplikacije. Pod svakim dijelom može biti sadržano više datoteka, no sve obavljaju istu primarnu funkciju definiranu dijelom kojem pripadaju. Pošto je ovakva podjela najjednostavnija u smislu same aplikacije i opis iste bit će podijeljen na ovakav način.

6.1. Model

Unutar ovog dijela primarno postoje datoteke "*menu.py*" i "*db.py*". Za potrebe ovog rada koristiti će se samo "*db.py*" datoteka unutar koje se definira baza podataka i tablica koja će se koristiti.

```
db.define_table('podaci',Field('struja','double'),Field('brzina','double')....)
```

Kod 6. Definiranje tablice u databazi

Prije navedena naredba definira tablicu imena *podaci* te njezine stupce naziva *struja*, *napon* u kojima će se nalaziti podaci tipa *double*. Iako sama tablica ima više definiranih stupaca naredba je potpuno ista za svaki sljedeći stupac koji se definira pa nije potrebno ispisati cijelu naredbu.

Kada je potrebna promjena tipa databaze tj. ne želi se koristiti baza podataka zadana od strane početne aplikacije promjena iste mora se izvršiti u posebnoj datoteci koja se nalazi pod privatim datotekama. Ova datoteka imena "*appconfig.ini*", zadužena je za osnovnu konfiguraciju same aplikacije. Unutar te datoteke moguće je promijeniti osnovne podatke o aplikaciji, kao što su ime aplikacije i autora, opis same aplikacije i drugo. Dodatne postavke unutar ove datoteke definiraju link na kojem će aplikacija potražiti samu databazu. Sama sintaksa bit će prikazana u nastavku uz zamjenu privatnih podataka s općim ključnim riječima.

```
; App configuration
[app]
name        = Welcome
author      = Your Name <you@example.com>
description = a cool new app

; db configuration
[db]
uri       = mysql://username:password@host_ip/table
migrate   = true
pool_size = 10
```

Kod 7. Konfiguracijska datoteka

6.2. Kontroler (*Controllers*)

Datoteke pisane unutar ovog dijela služe za upravljanje poslužiteljem stranice "*Server-sided code*". Unutar ovog dijela postoje datoteke koje nose nazive po mapama sadržanima u *View* dijelu. Pisane su u Python programskom jeziku te sadrže funkcije koje se izvršavaju kada korisnik pozove određenu stranicu. Unutar funkcija može biti bilo kakav kod koji omogućuje dodatnu prilagodljivost stranica korisniku i zahtjevima istog. Primarna datoteka koja se koristi u ovom dijelu je *default.py* te unutar nje obavezno mora biti sadržana funkcija *index* pošto kada pozovemo stranicu bez dodatnih argumenata sustav automatski poziva ovu stranicu.

U sklopu ovog rada kontroler smo primijenili kako bi omogućili upisivanje i obradu podataka iz mikrokontrolera u databazu te slanje naredbe na mikrokontroler. Usto kako bi bilo moguće podatke prikazati na grafovima prvotno ih treba prilagoditi formatu koji graf kao zasebni modul može prikazati.

Pošto će sam kod biti dan u prilogu ovdje će biti ukratko prikazane samo neke funkcije koje su važne za daljnje objašnjenje.

Prva funkcija koja će biti prikazana je *ucitaj_struja()* te je njezina namjena učitati podatke iz databaze, prilagoditi ih ispisu na dijagramu te zapakirati u JSON format i poslati pregledniku.

```
def ucitaj_struja():

        #Učitaj podatke iz databaze i spremi kao riječnik pod varijablom podaci
        podaci=db().select(db.podaci.struja,db.podaci.vrijeme).as_json()
        podaci=json.loads(podaci)

        #Zahtjev canvas.js grafova da podaci budu spremljeni u JSON s x i y varijablama
        for i in podaci:
                i['x']=i.pop('vrijeme')
                i['y']=i.pop('struja')

        #Makni posljednji podatak (oznaka završetka eksperimenta)
        podaci=json.dumps(podaci[0:len(podaci)-1])

        #Dodaj zaglavlje u odgovor i pošalji podatke
        response.headers["Access-Control-Allow-Origin"] = '*'
        return podaci
```

Kod 8. Prikaz funkcije za učitavanje vrijednosti

U prvom redu vidljiv je način korištenja *DAL* modula. Primjećuje se standardnost sintakse o kojoj je bilo govora ranije kada je modul i opisivan. Sljedeći dio je učitavanje podataka, koje se u ovom kodu obavlja preko modula *JSON*. Cijela svrha ovog modula je da pretvori tekstualnu varijablu u strukturirani rječnik kako bi se moglo unutar *for* petlje zamijeniti nestandardni nazivi polja sa standardnima.

Posljedni dio same funkcije je vraćanje zaglavlja i podataka. Zaglavlje je potrebno zbog izbjegavanja greške u zahtjevu preglednika za podacima, no sam način nije najsigurnije rješenje.

6.3. Pogled (*View*)

Pogled kako mu i samo ime govori je jedini dio aplikacije koji korisnik stvarno može vidjeti. Služi većinom kako bi se u njega programirao ”kostur” stranice pomoću *HTML* jezika. Iako se unutar web preglednika može smatrati kao statični kod važno je primijetiti razliku između jednog i drugog. Većinu pogleda unutar ovakvih aplikacija generira nekakav kod koji je na poslužiteljskoj strani, u ovom slučaju Python pa zbog toga niti sam dizajner stranice neće znati potpuni kod koji se nalazi u trenutnom pogledu.

Samim time što se većina koda unutar pogleda generira u trenutku zahtjeva za nekom stranicom te pošto je i sam kod promjenjiv pod utjecajem vanjskih datoteka (npr. *JavaScript*), pisanje istog kod izrade aplikacije je svedeno na minimum te neće ovdje biti dodatno prikazivano.

Jedini važan dio koji ostaje unutar ovog koda je interaktivna promjena veličine stranice i općenito objekta na njoj pomoću modula zvanog *Bootstrap*. Funkcionalnost se definira dodavanjem specifičnih klasa unutar elemenata dokumenta. Mogućnosti ovog modula se ne zaustavljaju na promjeni dimenzija okvira i elemenata, nego je moguće sakrivati ili prikazivati neke elemente u odnosu na veličinu pogleda ili dodavati interaktivni meni.

Svaka dodatna funkcionalnost pogleda kako je na primjer prije spomenuti *Bootstrap* ili dinamički grafovi dobiva se pomoću koda koji se nalazi u statičkom dijelu datoteka. Njihov način rada bit će definiran u sljedećem dijelu.

6.4. Statične datoteke (*Static*)

Statične datoteke su skup datoteka koje sudjeluju u stvaranju potpunog pogleda. Većinom se mogu podijeliti na uređivačke datoteke (*CSS*), funkcijske datoteke (*JS*), slike, te standardne stranice pogrešaka (*HTML*). Kako je vidljivo iz nabrojenog sam pojam statičke datoteke nije ograničen na vrstu, nego ovisi o ponašanju iste. Drugim riječima datoteke koje se ne mijenjaju između sesija možemo definirati kao statičke.

Najvažnije statične datoteke su (*JS*) čija je namjena interakcija s korisnikom na već učitanoj web stranici, pošto se sam kod izvršava unutar preglednika. Sljedeća namjena je prenošenje informacija između korisnika i poslužitelja točnije podaci koje korisnik šalje prema poslužitelju mogu se poslati pomoću zahtjeva unutar *Javascript*-a bez da se stranica mora ponovno učitavati. Iako ima vrlo velike prednosti uz njih ima i vrlo velik nedostatak, *Javascript* je sigurnosno gledano loš kod.

Kako bi stranice bile lijepo oblikovane primjenjuje se (*CSS*) jezik. Sam po sebi to je vrlo jednostavan jezik po strukturi i vrlo sličan nekim uređenim tipovima podataka, pošto je i svrha ovog koda spremiti sve definicije uređenja o svakom elementu koji se nalazi na stranici.

Unutar ovog rada primijenjen je samo *Javascript* kako bi se mogli obraditi podaci i prikazati unutar grafova, pa će i to ukratko biti objašnjeno.

6.4.1. *Javascript* datoteka s funkcijama

Unutar ove datoteke nalaze se funkcije potrebne za slanje, učitavanje i prikazivanje podataka. Kao takva glavni je dio izvršnog koda unutar samog preglednika.

Prva funkcija koja će biti prikazana je dizajnirana da šalje podatke u kontroler koji ih prosljeđuje prema mikrokontroleru kako bi se pokrenuo eksperiment.

```
function sendData(){
        //Inicijaliziraj varijable
        var tme="[";
        var npn="[";

        //Spremi točke s ulaznog grafa
        for(var i = 0;i<chart.data[0].dataPoints.length-1;i++)
                {
                        tme=tme+String(chart.data[0].dataPoints[i].x)+",";
                        npn=npn+String(chart.data[0].dataPoints[i].y)+",";
                }
        tme=tme+String(chart.data[0].dataPoints[chart.data[0].dataPoints.length-1].x)+"]";
        npn=npn+String(chart.data[0].dataPoints[chart.data[0].dataPoints.length-1].y)+"]";

        //Spoji sve podatke u jednu JSON varijablu
        var send='{"kp":'+kp.value+',"ki":'+ki.value+',"dc":'+chart.data[0].dataPoints.length+',"tme":'

        //Pošalji podatke
        ajax('http://192.168.1.100/Zavrsni/sendData/' + '?value=' + String(send), [], '');
}
```

Kod 9. Slanje podataka

Nakon što je na grafu ucrtana referenca te nakon zadavanja parametara, gore navedenu funkciju pokreće pritisak na tipku ”Pokreni” . Izvršavanjem te funkcije sustav čeka podatke te ih učitava na graf kako je prikazano u sljedećem isječku.

```
//Izbriši stare podatke u databazi
$.ajax({url: 'http://192.168.1.100/Zavrsni/delete_db'})

//Čekaj 2 sekunde
setTimeout(function()
        {
                //Provjeri dali su stigli svi podaci
                var lst=$.ajax({url: "http://192.168.1.100/Zavrsni/last_data",
                                dataType:"text",
                                async: false}).responseText;

                //Ako nisu uđi u funkciju za ponovnu provjeru, ako jesu učitaj podatke
                if(lst=="False"){refresh()}
                else{get_data();}
        },2000);
```

Kod 10. Primanje podataka

Unutar funkcije *get_data()* ponavlja se naredba kao u funkciji iznad, uz jednu razliku da se nakon primitka podaci parsiraju u zasebne varijable.

Posljednji dio funkcionalnosti otpada na prikaz grafova i podataka na istima. Prethodno je bilo navedeno da grafovi primaju točno određeni tip podataka s točno određenim varijablama. Kada su unutar kontrolera zadovoljeni svi zahtjevi dolje navedena funkcija je standardna za izradu grafa.

```
chart1 = new CanvasJS.Chart("curr_data", {
        animationEnabled: true,
        zoomEnabled: true,
        title:{
                text: "Odziv struje"
        },
        axisX :{
                minimum:0
        },
        axisY :{
                includeZero:false
        },
        data: [
                {
                        type: "line",
                        showInLegend: true,
                        legendText: "Odziv",
                        dataPoints: struja
                },
                {
                        type: "line",
                        showInLegend: true,
                        legendText: "Naredba",
                        dataPoints: naredba
                }
        ]
});
```

Kod 11. Prikaz poziva grafa

Pošto je glavni kod izrade grafa sadržan unutar knjižnica sustava kojim se grafovi generiraju, unutar ovog koda definirane su samo glavne postavke grafa.

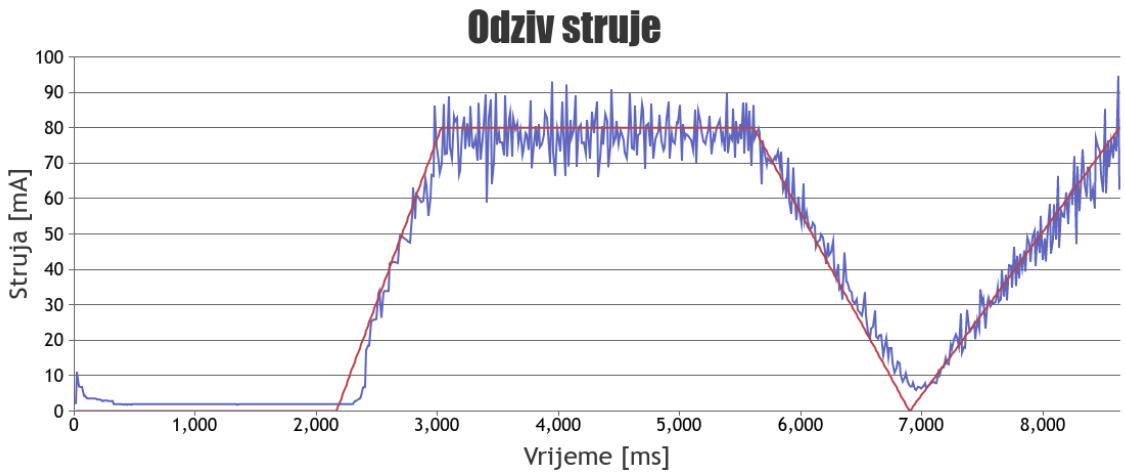
6.5. Izgled Web stranice

Iako je već objašnjena funkcionalnost same stranice u ovom dijelu bit će prikazana i potkrijepljena slikama izrađena web stranica. Prvotno će biti prikazana stranica koja služi za unos svih podataka.

Slika 22. Ulazna stranica

Iz slike je vidljivo da je sama stranica vrlo jednostavna s minimalnim funkcionalnostima kako bi sam naglasak bio na eksperimentu te na što jednostavnijem korištenju. Pri učitanju stranice otvara se generički graf koji je moguće izbrisati te unosom točku po točku iscrtati svoj vlastiti graf reference. Sljedeća polja nude unos parametara regulatora te je posljednja tipka ”Pokreni” kojom se pokreće eksperiment.

Nakon izvršenog eksperimenta dobiva se dijagram koji prikazuje odziv i referencu u istom koordinatnom sustavu. Tako će lakše biti doneseni zaključci o odzivu.



Slika 23. Rezultati eksperimenta

7. ZAKLJUČAK

Unutar ovog rada pomoću vrlo jednostavne makete istosmjernog motora obuhvaćeno je više područja. Primarna područja kojima se bavi su: elektronika, programiranje te regulacija. Svako od ovih područja moglo bi samo za sebe biti jedan rad primjeren ovoj razini. Stoga je kroz rad vidljivo da se temama nije pristupalo vrlo detaljno, nego na razini dovoljnoj za objašnjenje istog.

Motiv za izradu i pisanje ovog rada bio je predložiti vrlo interaktivan sustav kojim bi se na jednostavan način moglo izučavati područje upravljanja i regulacije. Važna napomena je da sustav nije spreman za korištenje u nastavi, ali je vrlo dobar početak prema izradi gotovog rješenja. Razlog tome je nedostatak razrade ekonomske isplativosti te samog umrežavanja većeg broja maketa u jedan sustava. Dodatno tema izlaganja nije obuhvaćala izradu sustava za izmjenu korisnika na virtualnim radnim mjestima.

Podloga ovog rada vrlo lako može biti primijenjena u ostalim područjima inženjerstva kao predložak za lakše razumijevanje i savladavanje naučenoga. Sljedeće sam praktični dio rada je vrlo dobar početak za izradu mnogo naprednijih sustava temeljenih na spoju interneta i realnih objekata.

Na kraju je vrlo važno izraziti želju da ovaj rad ne ostane samo u ovoj formi, nego da se razvija i dalje te da jednom pospješi sveukupni način izučavanja tema ključnih za moderno inženjerstvo.

8. LITERATURA

[1.]Mayr, Otto (1970), *The Origins of Feedback Control*, Clinton, MA USA: The Colonial Press, Inc.
[2.]*https://www.w3.org/Help/#webinternet*,What is the difference between the Web and the Internet?
[3.]*https://www.espressif.com/*, ESP32 Reference Manual
[4.]*http://www.web2py.com/book*,Web2Py Reference Manual
[5.]*https://www.arduino.cc/en/Guide/HomePage*,Getting started with Arduino
[6.]*https://www.mathworks.com/products/matlab.html*, Mathworks MATLAB
[7.]Majetić, Dubravko (2010), *Upravljanje i regulacija, Podloge za predavanja*, Zagreb, FSB

9. PRILOG

9.1. Web stranica

9.1.1. Datoteka *default.py*

```
# -*- coding: utf-8 -*-
# ---------------------------------------------------------------------------
# Leon Koren, 20.08.2018
# default controller
# ---------------------------------------------------------------------------

#Poziv korištenih knjižnica
import json
import requests

#Index stranica
def index():
        return dict()

#Funkcija za učitavanje brzine
def ucitaj_brzina():
        #Učitaj podatke i pretvori ih u riječnik
        podaci=db().select(db.podaci.brzina,db.podaci.vrijeme).as_json()
        podaci=json.loads(podaci)                #Standardiziraj podatke
        for i in podaci:
                i['x']=i.pop('vrijeme')
                i['y']=i.pop('brzina')
                #Izbaci zadnji podatak
        podaci=json.dumps(podaci[0:len(podaci)-1])
        #Dodaj zaglavlje u odgovor i pošalji podatke
        response.headers["Access-Control-Allow-Origin"] = '*'
        return podaci

#Učitavanje struje, jednak algoritam kao kod brzine
def ucitaj_struja():
        podaci=db().select(db.podaci.struja,db.podaci.vrijeme).as_json()
        podaci=json.loads(podaci)
        for i in podaci:
                i['x']=i.pop('vrijeme')
                i['y']=i.pop('struja')
        podaci=json.dumps(podaci[0:len(podaci)-1])
        response.headers["Access-Control-Allow-Origin"] = '*'
        return podaci
```

```
#Učitavanje naredbe, jednak algoritam kao kod brzine
def ucitaj_naredba():
        podaci=db().select(db.podaci.naredba,db.podaci.vrijeme).as_json()
        podaci=json.loads(podaci)
        for i in podaci:
                i['x']=i.pop('vrijeme')
                i['y']=i.pop('naredba')
        podaci=json.dumps(podaci[0:len(podaci)-1])
        response.headers["Access-Control-Allow-Origin"] = '*'
        return podaci

#Provjeri dali je u databazu došao zadnji podatak
def last_data():
        lstdt=db().select(db.podaci.vrijeme).last()
                if lstdt:
                        if lstdt['vrijeme']==-1:
                                return True
                        else:
                                return False
                else:
                        return False

#Zaprimi naredbu sa stranice i proslijedi na mikrokontroler
def sendData():
        param = {"napon":request.vars.value}
        url="http://192.168.1.15:8085/"
        r=requests.get(url=url,params=param)
        return r

#Isprazni databazu
def delete_db():
        db.podaci.truncate()
        response.headers["Access-Control-Allow-Origin"] = '*'

#Umetni podatke u databazu
def umetanje():
        data= request.body.read()
        db.podaci.bulk_insert(json_decode(data))

#Konvertiranje podataka u tip koji je moguće unesti
#pomoću bulk_insert() funkcije u databazu
def json_decode(data):
        jdata=json.loads(data)
        out1=[]
        for j in range(0,len(jdata.values()[0])):
                out={}
                for i in jdata.keys():
                        out[i]=jdata[i][j]
                        out1.append(out)
        return out1
```

9.1.2. Datoteka *functions.js*

```
//Datoteka functions.js
//Leon Koren
//20.08.2018

//Definicija globalnih varijabli
var chart,chart1,chart2;
var data = [{x:0,y:0},{x:0.2,y:0},{x:0.2,y:0.07},{x:0.8,y:0.07},{x:1.5,y:0}];
var kp;var ki;

//Pokreni nakon učitavanja cijele stranice
window.onload = function () {
        //Pronađi element imena output (izlazni grafovi) i sakri ih
        document.getElementById("output").setAttribute("hidden","");

        //Generiraj graf
        chart = new CanvasJS.Chart("input_data", {
                animationEnabled: true,
                zoomEnabled: true,
                //Naslov grafa
                title:{
                        text: "Ulazni podaci"
                },
                //Vrijeme ne može biti manje od nule
                axisX :{
                        minimum:0
                },
                axisY :{
                        includeZero:false
                },
                //Ulazni podaci
                data: [
                                {
                                type: "line",
                                dataPoints: data
                        }]
        });
//Iscrtaj graf
chart.render();
}

//Dodavanje novih točaka na ulazni graf
function addMore(){
//Učitaj x koordinatu zadnje točke na grafu
        if(chart.options.data[0].dataPoints.length>0)
        {
                var lx=chart.options.data[0].
                                dataPoints[chart.options.data[0].dataPoints.length-1].x
        }
        else
        {
                lx=0;
        }
```

```
//Učitaj podatke iz tekstualnog okvira
        var x=document.getElementById("tme");
        var y=document.getElementById("val");

//Provjera upisanih vrijednosti prije upisa na graf
        if(x.value!="" &&  parseFloat(x.value)>=lx)
        {
                x.style.backgroundColor="white"
                if(y.value!="")
                {
                        chart.options.data[0].dataPoints.push({x:parseFloat(x.value),
                                                                y:parseFloat(y.value)})
                        chart.render();
                        x.value="";
                        y.value="";
                        x.style.backgroundColor="white"
                        y.style.backgroundColor="white"
                }
                else
                {
                        y.style.backgroundColor="red"
                }
        }
        else
        {
                if(y.value==""){y.style.backgroundColor="red"}
                x.style.backgroundColor="red"
        }
}

//Funkcija zadužena za slanje i primanje podataka
var brzina="[]";var struja="[]";var naredba="[]";
function run_all(){
        kp=document.getElementById("kp");
        ki=document.getElementById("ki");
        //Provjera dali su upisani parametri
        if(kp.value=="" || ki.value==""){
                if(kp.value=="")
                {
                        kp.style.backgroundColor="red"
                }
                else
                {
                        ki.style.backgroundColor="red"
                }
        }
```

```
        else
        {
                kp.style.backgroundColor="white"
                ki.style.backgroundColor="white"
                //Pozovi funkciju za slanje podataka
                sendData();
                var i=0;
                var load_par=document.getElementById("loading")
                //Izbriši databazu
                $.ajax({url: 'http://192.168.1.100/Zavrsni/delete_db'})
                //Ponavljajuća provjera kompletnosti podataka
                setTimeout(function(){
                        var lst=$.ajax({url: "http://192.168.1.100/Zavrsni/last_data",
                                                dataType:"text",
                                                async: false}).responseText;
                        if(lst=="False"){refresh()}else{console.log(JSON.parse(brzina))}
                },2000);
                function refresh(){
                        i++;rpt='.';
                        var lst=$.ajax({url: "http://192.168.1.100/Zavrsni/last_data",
                                                dataType:"text",
                                                async: false}).responseText;
                        load_par.innerHTML="Loading"+rpt.repeat(i);
                        if(lst=="False"){setTimeout(refresh,2000)}
                        //Ako su podaci kompletni učitaj ih u databazu
                        else
                        {
                                brzina=$.ajax({
                                        url: "http://192.168.1.100/Zavrsni/ucitaj_brzina",
                                        dataType:"json",
                                        async: false
                                        }).responseText

                                struja=$.ajax({
                                        url: "http://192.168.1.100/Zavrsni/ucitaj_struja",
                                        dataType:"json",
                                        async: false
                                        }).responseText

                                naredba=$.ajax({
                                        url: "http://192.168.1.100/Zavrsni/ucitaj_naredba",
                                        dataType:"json",
                                        async: false
                                        }).responseText
                                brzina=JSON.parse(brzina);
                                struja=JSON.parse(struja);
                                naredba=JSON.parse(naredba);
                                //Iscrtaj grafove
                                output_graf();
                        }
                }
        }
}
```

```
//Pošalji podatke te pokreni eksperiment
function sendData(){
        var tme="[";
        var npn="[";
        for(var i = 0;i<chart.data[0].dataPoints.length-1;i++)
        {
                tme=tme+String(chart.data[0].dataPoints[i].x)+",";
                npn=npn+String(chart.data[0].dataPoints[i].y)+",";
        }
        tme=tme+String(chart.data[0].dataPoints[chart.data[0].dataPoints.length-1].x)+"]";
        npn=npn+String(chart.data[0].dataPoints[chart.data[0].dataPoints.length-1].y)+"]";
        var send='{"kp":'+kp.value+',"ki":'+ki.value+',"dc":'+chart.data[0].dataPoints.length+
                        ',"tme":'+tme+',"npn":'+npn+'}'
        ajax('http://192.168.1.100/Zavrsni/sendData/' + '?value=' + String(send), [], '');
}

//Funkcije za iscrtavanje izlaznih grafova
//Isto kao i kod ulaznog
function output_graf(){
        document.getElementById("input").setAttribute("hidden","");
        document.getElementById("output").removeAttribute("hidden");
        chart1 = new CanvasJS.Chart("curr_data", {
                animationEnabled: true,
                zoomEnabled: true,
                title:{
                        text: "Odziv struje"
                },
                axisX :{
                        minimum:0
                },
                axisY :{
                        includeZero:false
                },
                data: [
                                {
                                type: "line",
                                showInLegend: true,
                                legendText: "Odziv",
                                dataPoints: struja
                                },
                                {
                                type: "line",
                                showInLegend: true,
                                legendText: "Naredba",
                                dataPoints: naredba
                                }
                        ]
        });
```

```
        chart2 = new CanvasJS.Chart("sp_data", {
                animationEnabled: true,
                zoomEnabled: true,
                title:{
                        text: "Odziv brzine"
                },
                axisX :{
                        minimum:0
                },
                axisY :{
                includeZero:true
                },
                data: [
                                {
                                type: "line",
                                dataPoints: brzina
                                }
                          ]
                });
        chart1.render();
        chart2.render();
}
```

9.2. Arduino kod

```
/**********************************
* PI_reg.ino
* Leon Koren
* 25.08.2018
* ********************************/

//Poziv svih korištenih knjižnica
#include <Thread.h>
#include <ThreadController.h>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include <Adafruit_ADS1015.h>
#include "driver/mcpwm.h"
#include "soc/mcpwm_reg.h"
#include "soc/mcpwm_struct.h"
#include "driver/pcnt.h"
#include "driver/gpio.h"
#include "driver/timer.h"
#include <HTTPClient.h>
#include "soc/timer_group_struct.h"
#include "soc/timer_group_reg.h"

String nardb;
// Handle za procese zasebnih jezgri
TaskHandle_t xTask1;
TaskHandle_t xTask2;
#define NET_NAME "Hextech"  //SSID
#define PASS     "rkorenadmin12"    //PASSWORD
//Definicije konstanti korištenih u programu
//TIMER
// Djeljitelj hardwerskog clocka
#define TIMER_DIVIDER       16
// Konverzijski faktor
#define TIMER_SCALE         (TIMER_BASE_CLK / TIMER_DIVIDER)

//COUNTER
#define PCNT_UNIT_SPD       PCNT_UNIT_0      // Counter jedinica
#define PCNT_H_LIM_VAL      10000            // Gornji limit countera
#define PCNT_L_LIM_VAL      0                // Donji limit countera
#define PCNT_THRESH1_VAL    10000            // Gornja okidna vrijednost countera
#define PCNT_THRESH0_VAL    0                // Donja okidna vrijednost countera
#define PCNT_INPUT_SIG_IO   4                // Ulazni pin
#define PCNT_INPUT_CTRL_IO  HIGH             // Uvijek broji prema gore

//MCPWM
#define GPIO_PWM0A_OUT 12
#define GPIO_PWM1A_OUT 14
```

```
//WEB
#define SITE "http://192.168.1.100/Zavrsni/umetanje"
AsyncWebServer server(8085);

//ADS1115
Adafruit_ADS1115 ads;
int16_t count = 0;
double vrijeme;

//Inicijalizacija PWM signala
void init_mcpwm()
{
        mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0A, GPIO_PWM0A_OUT);
        mcpwm_gpio_init(MCPWM_UNIT_1, MCPWM1A, GPIO_PWM1A_OUT);
}

//Funkcija za mijenjnje širine PWM-a
void set_duty(float duty_a,float duty_b)
{
        mcpwm_set_duty(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_A, duty_a);
        mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_0, MCPWM_OPR_A, MCPWM_DUTY_MODE_0);

        mcpwm_set_duty(MCPWM_UNIT_1, MCPWM_TIMER_1, MCPWM_OPR_A, duty_b);
        mcpwm_set_duty_type(MCPWM_UNIT_1,MCPWM_TIMER_1, MCPWM_OPR_A, MCPWM_DUTY_MODE_0);
}

//Inicijalizacija timera
static void timer_init(timer_idx_t timer_idx, bool auto_reload)
{
        // Parametri
        timer_config_t config;
        config.divider = TIMER_DIVIDER;
        config.counter_dir = TIMER_COUNT_UP;
        config.counter_en = TIMER_PAUSE;
        config.alarm_en = TIMER_ALARM_DIS;
        config.intr_type = TIMER_INTR_LEVEL;
        config.auto_reload = auto_reload;
        timer_init(TIMER_GROUP_0, timer_idx, &config);

        //Definicija početne vrijednosti timera
        timer_set_counter_value(TIMER_GROUP_0, timer_idx, 0x00000000ULL);
}
```

```
//Inicijalizacija countera
static void pcnt_init(void)
{
        //Konfiguracija za brojač
        pcnt_config_t pcnt_config =
        {
                pulse_gpio_num : PCNT_INPUT_SIG_IO,  // Ulazni pin
                ctrl_gpio_num : PCNT_INPUT_CTRL_IO,  // Kontrolni pin (HIGH)
                lctrl_mode : PCNT_MODE_REVERSE,      // Definicije za kontrolni pin
                hctrl_mode : PCNT_MODE_KEEP,
                pos_mode : PCNT_COUNT_INC,           // +1 na pozitivni brid
                neg_mode : PCNT_COUNT_INC,           // +1 na negativni brid
                // Maksimalne i minimalne vrijednosti
                counter_h_lim : PCNT_H_LIM_VAL,
                counter_l_lim : PCNT_L_LIM_VAL,
                //Jedinica i kanal
                unit : PCNT_UNIT_SPD,
                channel : PCNT_CHANNEL_0
        };

        //Inicijalizacija
        Serial.println("cnt conf ok");
        pcnt_unit_config(&pcnt_config);
        //Ulazni filter signala 0-1024
        pcnt_set_filter_value(PCNT_UNIT_SPD, 100);
        pcnt_filter_enable(PCNT_UNIT_SPD);

        //Zaustavi brojač te izbriši
        pcnt_counter_pause(PCNT_UNIT_SPD);
        pcnt_counter_clear(PCNT_UNIT_SPD);

        //Pokreni brojač
        pcnt_counter_resume(PCNT_UNIT_SPD);
}

//Mjerenje brzine
double speedcount()
{
        double brzina = 0;
        pcnt_get_counter_value(PCNT_UNIT_SPD, &count);
        timer_pause(TIMER_GROUP_1, TIMER_1);
        timer_get_counter_time_sec(TIMER_GROUP_1, TIMER_1, &vrijeme);
        timer_set_counter_value(TIMER_GROUP_1, TIMER_1, 0);
        pcnt_counter_clear(PCNT_UNIT_SPD);
        timer_start(TIMER_GROUP_1, TIMER_1);
        brzina = (count / 512.0) * (60.0 / vrijeme);
        return brzina;
}
```

```
//Početno postavljanje
void setup() {
        //Inicijaliziraj PWM
        init_mcpwm();
        mcpwm_config_t pwm_config;
        pwm_config.frequency = 15000;
        pwm_config.counter_mode = MCPWM_UP_COUNTER;
        mcpwm_init(MCPWM_UNIT_0,MCPWM_TIMER_0,&pwm_config);
        mcpwm_init(MCPWM_UNIT_1,MCPWM_TIMER_1,&pwm_config);
        //Postavi PWM na nula
        set_duty(0,0);

        //Definicije pinova
        pinMode(27,OUTPUT);
        digitalWrite(27,HIGH);
        pinMode(4, INPUT_PULLUP);

        //Inicijaliziraj timer
        pcnt_init();
        timer_init(TIMER_0, 0);

        //Inicijaliziraj ADC
        ads.setGain(GAIN_ONE);

        //Pokreni serijsku komunikaciju
        Serial.begin(115200);

        //Pokreni ADC
        ads.begin();
//Inicijaliziraj i pokreni WiFi
WiFi.begin(NET_NAME, PASS);
//Spajanje na mrežu
Serial.print("Spajam se na ");
Serial.print(NET_NAME);
while (WiFi.status() != WL_CONNECTED)
{
        delay(100);
        Serial.print(".");
}
Serial.println("");
delay(100);
Serial.println(WiFi.localIP());     // Ispis IP-adrese
Serial.println(WiFi.macAddress());  // Ispis MAC-adrese
delay(250);
```

```
//Definiraj procese na jezgrama i pokreni iste
xTaskCreatePinnedToCore(
        runcore0,           // Funkcija na jezgri 0
        "runcore0",         // Ime funkcije na jezgri 0
        10000,              // Veličina stack-a dodjeljenog procesu
        NULL,               // Parametri procesa
        1,                  // Prioritet procesa
        &xTask1,            // Handle za proces 0
        0);                 // Broj jezgre na kojoj se izvršava proces

xTaskCreatePinnedToCore(
        runcore1,           // Funkcija na jezgri 1
        "runcore1",         // Ime funkcije na jezgri 1
        10000,              // Veličina stack-a dodjeljenog procesu
        NULL,               // Parametri procesa
        1,                  // Prioritet procesa
        &xTask2,            // Handle za proces 1
        1);                 // Broj jezgre na kojoj se izvršava proces
}
//Inicijalizacija svih varijabli
float integral_i=0,i_ref=0,ii_ref;
int coeff=1;
String prnt = "",struja_array="",brzina_array="",naredba_array="",vrijeme_array="";
String struja_array1="",brzina_array1="",naredba_array1="",vrijeme_array1="";
bool flag_sw=false,flag_send=false;
bool flag1,flag_end=false;

//Proces na jezgri jedan
void runcore1(void * parameter){
//Pokretanje Web servera za primanje naredbe
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
        {
                int paramsNr = request->params();
                for(int i=0;i<paramsNr;i++)
                        {
                                AsyncWebParameter* p = request->getParam(i);
        // Spremi podatak u argumentu "nrdb" te podigni flag da je stigao novi podatak
                                if(p->name()=="nrdb")
                                        {
                                                nardb=p->value();
                                                flag1=true;
                                                Serial.println(flag1);
                                        }
                        }
                request->send(200, "text/plain", "");
        });
server.begin();
```

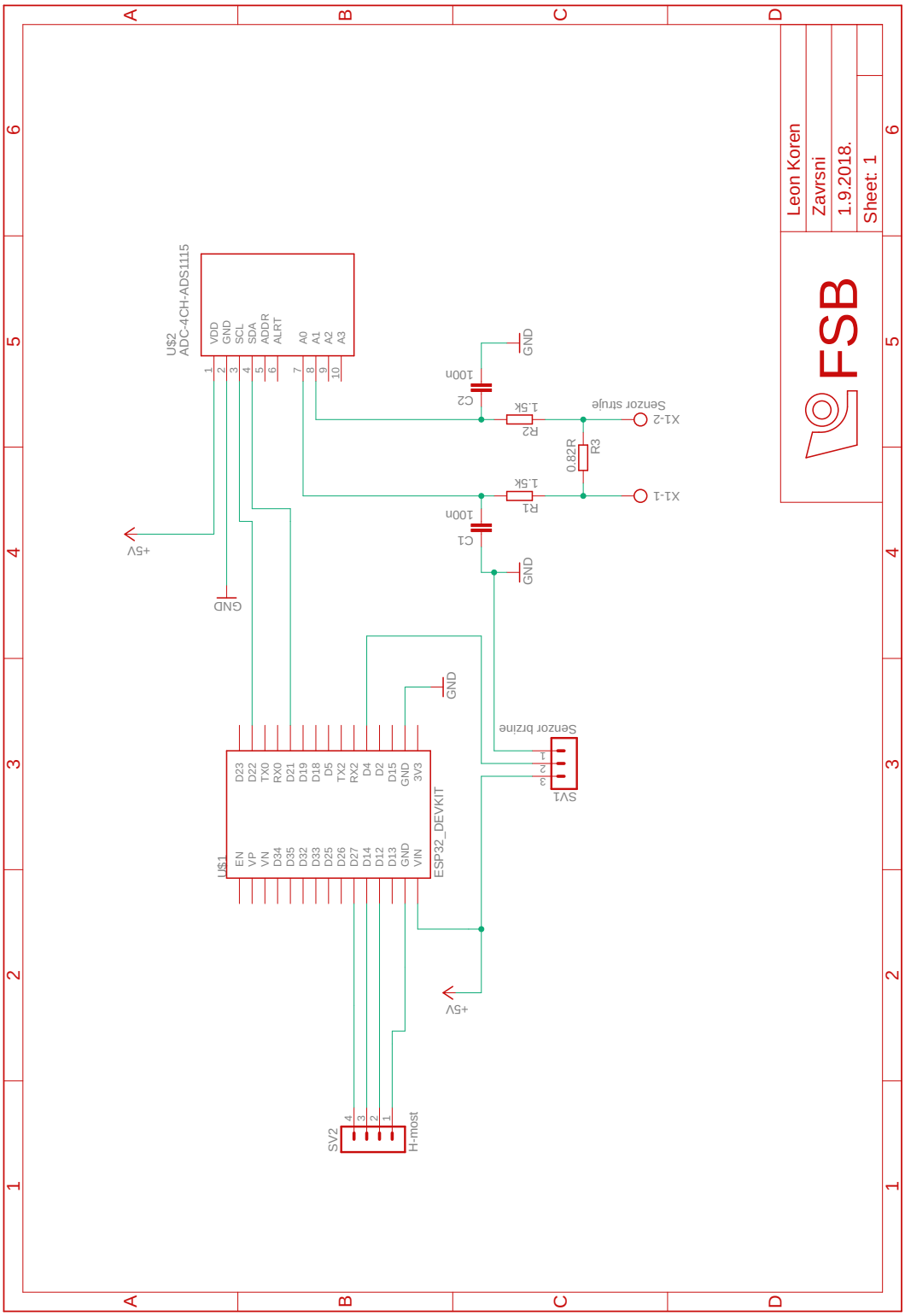
```
//Pokreni prvo mjerenje brzine i uđi u beskonačnu petlju
double brzina=speedcount();
while(1){
        vTaskDelay(10);

        //Ako je primljen novi podatak parsiraj i pokreni eksperiment
        if(flag1==1){
                flag1=false;
                JsonObject& data = jsonBuffer.parseObject(nardb);
                if(!data.success())
                {
                        Serial.println("parseObject() failed");
                }
                float ki=data["ki"];
                float kp=data["kp"];
                int dcount=data["dc"],k=1;
                double npn[dcount],tme[dcount],ttm=-1;
                //Ako su vremena ista (step) pomakni sljedeće za jedan period
                for(int i=0;i<dcount;i++){
                        tme[i]=data["tme"][i];
                        if(tme[i]==ttm)
                        {
                                tme[i]=tme[i]+0.0032;
                        }
                ttm=tme[i];
                npn[i]=data["npn"][i];
        }
        //Broj koraka
        int broj = ceil(tme[dcount-1]/0.0032);
        int j=-1;
        float vrm=0;
        int u_refa=0,u_refb=0;
        flag_end=false;
        int i=0,n=0;
        brzina = speedcount();

        //Izvršavanje eksperimenta
        for(i=0;i<=broj;i++)
        {
                //Promjeni varijablu za spremanje svakih 500 podataka
                if(500*k==i)
                {
                        k++;flag_send=true;
                        if(flag_sw){flag_sw=false;}
                        else{flag_sw=true;}
                }
                int t=micros();

                //Pronađi korak promjene signala
                if(ceil(tme[j+1]/0.0032)==i)
                {
                        j++;
                }
                //Funkcija pravca za naredbu u ubliku pravca
                i_ref=npn[j]+((npn[j+1]-npn[j])/float(tme[j+1]-tme[j]))*(i*0.0032-tme[j]);
```

```
                //Mjeri brzinu svaki 4. korak
                if(n==4)
                {
                        n=0;
                        brzina = speedcount();
                }
                n++;

                //PI regulator
                float struja = ads.readADC_Differential_2_3()*0.125*1.185;
                float i_err=i_ref-struja*coeff/1000.0;
                integral_i=integral_i+ki*i_err;
                float u_ref=kp*i_err+integral_i;

                //Anti-windup clamping
                if(u_ref>100){u_refa=100;u_refb=0;coeff=1;integral_i=100/ki;}
                if(u_ref>0 && u_ref<=100){u_refa=u_ref;u_refb=0;coeff=1;}
                if(u_ref==0){u_refa=0;u_refb=0;}
                if(u_ref<-100){u_refa=0;u_refb=100;coeff=-1;integral_i=-100/ki;}
                if(u_ref<0 && u_ref>=-100){u_refa=0;u_refb=-u_ref;coeff=-1;}
                vrm+=(micros()-t)/1000.0;

                //Izlaz regulatora
                set_duty(u_refa,u_refb);

                //Spremanje podataka
                if(flag_sw==true)
                {
                        brzina_array1+=String(brzina)+",";
                        struja_array1+=String(struja)+",";
                        naredba_array1+=String(i_ref*1000)+",";
                        vrijeme_array1+=String(vrm)+",";
                }
                else
                {
                        brzina_array+=String(brzina)+",";
                        struja_array+=String(struja)+",";
                        naredba_array+=String(i_ref*1000)+",";
                        vrijeme_array+=String(vrm)+",";
                }

        }
        //Kraj eksperimenta
        set_duty(0,0);flag_send=true;
        if(flag_sw){flag_sw=false;}
        else{flag_sw=true;}
        vTaskDelay(5000);
        flag_end=true;
        }
}
}
```

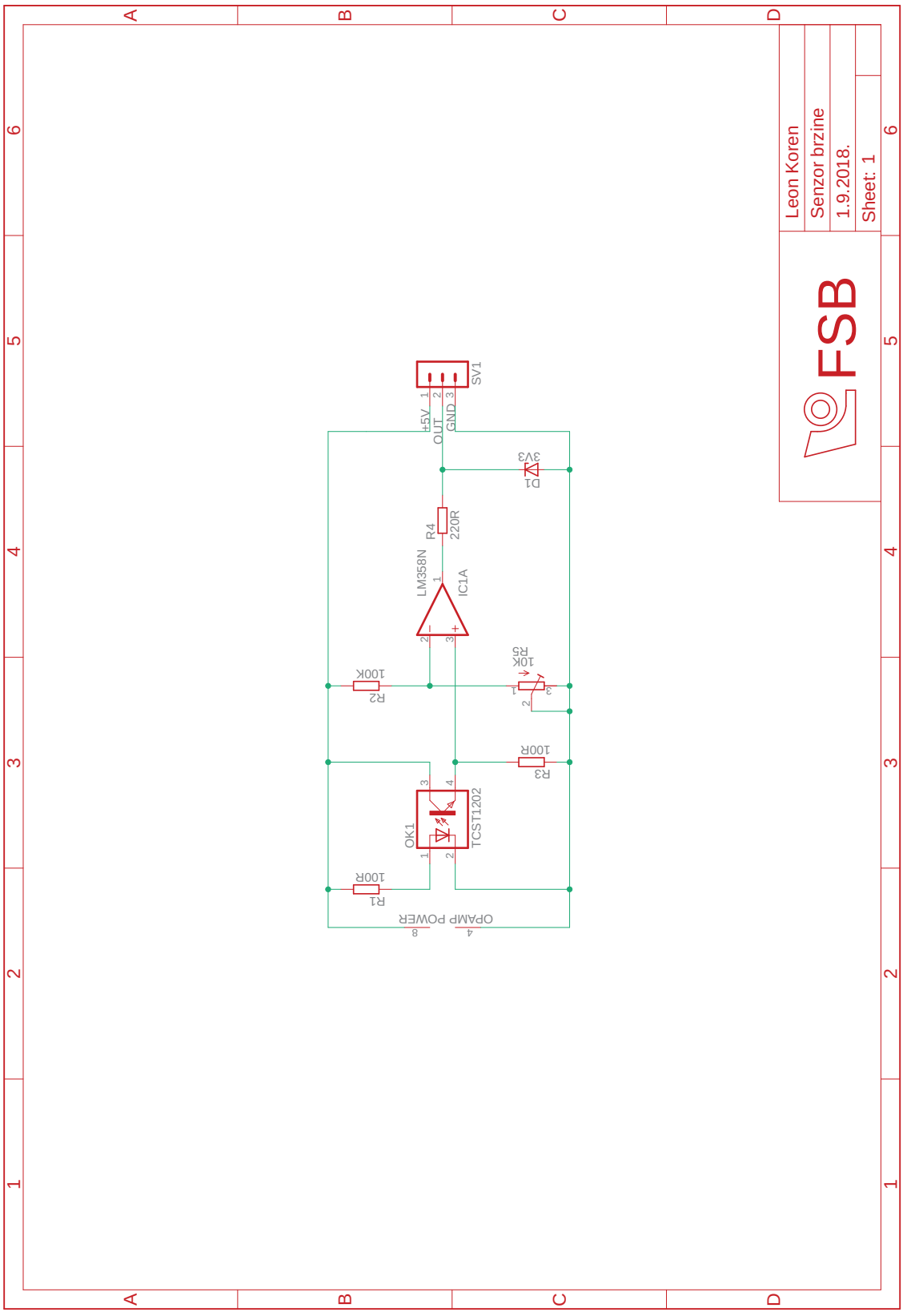
```
//Proces na jezgri nula
//Slanje podataka za vrijeme izvršavanja eksperimenta
HTTPClient http;
void runcore0(void * parameter){
String JSONmessageBuffer="";
while(1){
        vTaskDelay(10);

        //Ako je eksperiment gotov pošalji sve nule samo vrijeme -1(označi kraj)
        if(flag_end)
        {
                flag_end=false;
                http.begin(SITE);
                http.addHeader("Content-Type", "application/json");
                // Slanje podataka pomoću POST
                http.POST("{\"struja\":[0],\"brzina\":[0],\"vrijeme\":[-1],\"naredba\":[0]}");
                http.end();
        }

        //Odabir koje podatke šaljemo trenutno
        if(flag_send){
                flag_send=false;
                if(flag_sw){
                        struja_array.remove(struja_array.length()-1);
                        brzina_array.remove(brzina_array.length()-1);
                        naredba_array.remove(naredba_array.length()-1);
                        vrijeme_array.remove(vrijeme_array.length()-1);
                        JSONmessageBuffer="{\"struja\":["+struja_array+"],\"brzina\":["+
                                        brzina_array+"],\"vrijeme\":["+vrijeme_array+
                                        "],\"naredba\":["+naredba_array+"]}";
                        struja_array="",brzina_array="",naredba_array="",vrijeme_array="";
        }
        else
        {
                struja_array1.remove(struja_array1.length()-1);
                brzina_array1.remove(brzina_array1.length()-1);
                naredba_array1.remove(naredba_array1.length()-1);
                vrijeme_array1.remove(vrijeme_array1.length()-1);
                JSONmessageBuffer="{\"struja\":["+struja_array1+"],\"brzina\":["+
                                brzina_array1+"],\"vrijeme\":["+vrijeme_array1+
                                "],\"naredba\":["+naredba_array1+"]}";
                struja_array1="",brzina_array1="",naredba_array1="",vrijeme_array1="";
        }

        //Slanje podataka pomoću HTTP.post()
        vTaskDelay(15);
        http.begin(SITE);
        http.addHeader("Content-Type", "application/json");
        http.POST(JSONmessageBuffer); // Slanje podataka pomoću POST
        http.end();
        }
}
}
```

9.3. Sheme sklopa

9.3.1. Upravljački sklop



9.3.2. Senzor brzine



9.4. Prikaz postava