Upravljanje svesmjernim robotom u stvarnome vremenu preko pametnog telefona korištenjem http i SPI protokola

Veber, Hrvoje

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike i računarstva

Hrvoje Veber

Upravljanje svesmjernim robotom u stvarnome vremenu preko pametnog telefona korištenjem HTTP i SPI protokola

Završni rad

Osijek, 2021.

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike i računarstva

Hrvoje Veber

Upravljanje svesmjernim robotom u stvarnome vremenu preko pametnog telefona korištenjem HTTP i SPI protokola

Završni rad

Mentori: Jurica Maltar, izv. prof. dr. sc. Domagoj Matijević

Osijek, 2021.

Upravljanje svesmjernim robotom u stvarnome vremenu preko pametnog telefona korištenjem HTTP i SPI protokola

Sažetak

U ovom radu bavit ćemo se upravljanjem svesmjernog robota. Prvo ćemo se posvetiti teorijskim aspektima upravljanja svesmjernog robota. Prikazat ćemo robota u prostoru tako što ćemo razviti kinematički model robota i kotača. Nakon toga, obradit ćemo komunikaciju između dva mikrokontrolera putem SPI protokola. Na jedan mikrokontroler primat ćemo podatke sa Joysticka i slati ih na drugi mikrokontroler. Zatim ćemo reći nešto o upravljanju motora i PID regulatoru koji je zaslužan za glatku responzivnost motora. Na kraju ćemo prikazati praktični zadatak implementacije svesmjernog robota koristeći obrađene teme. Zanimljivo je kako smo s ovakvim jednim projektom povezali low-level i web programiranje kroz tri programska jezika (Python, C i JavaScript). Također, morali smo poznavati i hardware do neke razine, što znači da smo obuhvatili ne samo softverski dio već i hardverski. U konačnici dobili smo neuobičajenog svesmjernog robota kojeg sami možemo kontrolirati preko pametnog telefona. Činjenica je da će ovakvi roboti postati sve zastupljeniji i da će imati veliku ulogu u svijetu, a ovime smo samo zagrebali površinu onoga što sve roboti ovakvog tipa mogu postići.

Ključne riječi

Kinematika robota, SPI protokol, PID regulator, HTTP, Joystick, Web server

Controling omnidirectional robot in real time via mobile device using HTTP and SPI protocol

Summary

In this paper, we will deal with the control of an omnidirectional robot. We will first focus on the theoretical aspects of omnidirectional robot control. We will show the robot in space by developing a kinematic model of the robot and the wheel. After that, we will process the communication between the two microcontrollers via the SPI protocol. We will receive data from Joystick on one microcontroller and send it to another microcontroller. Then we will say something about motor control and PID controller which is responsible for the smooth responsiveness of the motor. Finally, we will present the practical task of implementing an omnidirectional robot using the processed topics. newline It is interesting how we connected low-level and web programming through three programming languages (Python, C and JavaScript) with one such project. Also, we had to know the hardware to some extent, which means that we included not only the software part but also the hardware part. Eventually we got an uncommon omnidirectional robot that we can control ourselves via a smartphone. The fact is that robots like this will become more and more prevalent and will play a big role in the world, and with this we have only scratched the surface of what all robots of this type can achieve.

Key words

Robot kinematics, SPI protocol, PID regulator, HTTP, Joystick, Web server

Sadržaj

1 Uvod

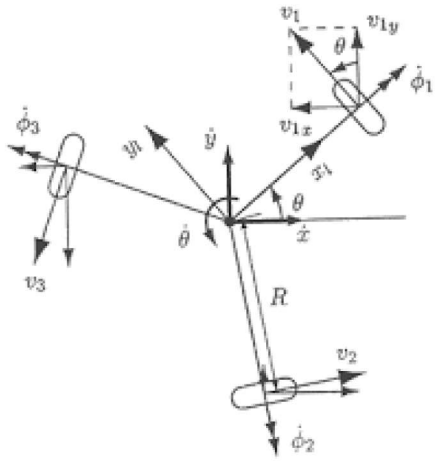
U području robotike važnost mobilnih robota neprestano raste. Zbog njihove slobode kretanja, mobilni roboti su fleksibilniji i mogu obavljati sve više zadataka. Trenutne primjene mobilnih robota vrlo su široke i uključuju:

- čišćenje u domaćinstvu i javnom prostoru
- prijevoz robe u bolnicama, tvornicama, lukama i skladištima
- istraživanje negostoljubivih terena kao što su svemir ili oceani
- rudarstvo
- uklanjanje eksploziva
- zabava i izvedbe inspekcije
- sigurnosne ophodnje

Posebna klasa mobilnih robota su svesmjerni roboti. Ovi roboti su dizajnirani za kretanje u 2D prostoru i mogu jednolikim gibanjem doći iz bilo koje točke do bilo koje druge točke u prostoru. Ideja ovog rada je upravljanje sa svesmjernim robotom koristeći Joystick na web/mobilnoj aplikaciji. Robot treba dohvatiti podatke sa Joysticka, obraditi te podatke, odnosno pretvoriti ih u kutne brzine te ih primijeniti na motorima(kotačima).

2 Kinematika

Ako želimo proučavati i utjecati na kretanje robota u okruženju, moramo znati kako se odnositi prema varijablama: kutni položaji i brzine osovina kotača. Prema tome, potrebno je razviti kinematički model robota.



Slika 1: Kinematika robota. Slika preuzeta iz "T.A. Baede, Motion control of an omnidirectional mobile robot"

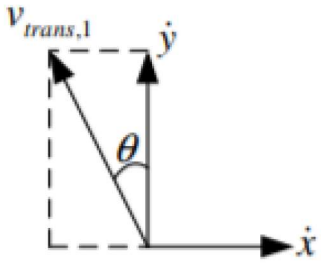
2.1 Kinematički model

Započnimo s definiranjem globalnog okvira [x, y] koji predstavlja okruženje robota, vidi sliku 1. Položaj i orijentacija robota u ovom globalnom okviru mogu se prikazati kao (x, y, θ). Globalna brzina robota može se zapisati kao ($\dot{x}$, $\dot{y}$, $\dot{\theta}$). Sada možemo definirati i lokalni okvir [x_L, y_L] koji je pričvršćen na samog robota. Središte ovog lokalnog okvira poklapa se s težištem robota. Tri švedska kotača su smještena pod kutom α_i (i = 1, 2, 3) u odnosu na lokalni okvir. Uzmemo li lokalnu os x_L kao početnu točku i brojimo stupnjeve u smjeru kazaljke na satu kao pozitivne, imamo $\alpha_1 = 0°$, $\alpha_2 = 120°$ i $\alpha_3 = 240°$.

Translacijske brzine kotača v_i koje djeluju na podu određuju globalnu brzinu robota u okruženju ($\dot{x}$, $\dot{y}$, $\dot{\theta}$) i obrnuto. Translacijska brzina osovine kotača v_i može biti podijeljena na dva dijela zbog same translacije i rotacije robota:

$$v_i = v_{trans.i} + v_{rot}. \tag{1.0}$$

Prvo, uzmimo u obzir translaciju. Na slici 1. prikazana je translacijska brzina $v_{trans.1}$ osovine kotača 1. Možemo mapirati jedinični vektor $v_{trans.1}$ na vektore $\dot{x}$ i $\dot{y}$ kako bismo dobili (1.1).



Slika 2: Translacijska brzina $v_{trans.1}$. Slika preuzeta iz "T.A. Baede, Motion control of an omnidirectional mobile robot"

$$v_{trans.1} = -sin(\theta)\dot{x} + cos(\theta)\dot{y}. \tag{1.1}$$

To vrijedi i za ostale kotače kada uzmemo u obzir da su vektori v_i pozicionirani pomakom $\theta + \alpha_i$, pa imamo:

$$v_{trans.i} = -sin(\theta + \alpha_i)\dot{x} + cos(\theta + \alpha_i)\dot{y}. \tag{1.2}$$

Međutim, ukoliko robot izvrši samo rotaciju, brzina osovine v_i mora zadovoljavati sljedeću jednadžbu:

$$v_{rot} = R\dot{\theta}. \tag{1.3}$$

Ovdje je R udaljenost od centra ravnoteže robota do kotača. Ako to sve sad uvrstimo u (1.0), imamo:

$$v_i = -sin(\theta + \alpha_i)\dot{x} + cos(\theta + \alpha_i)\dot{y} + R\dot{\theta}. \qquad (1.4)$$

Sada smo sparili translacijsku brzinu kotača s globalnom brzinom robota. Pogledajmo kako je translacijska brzina povezana s kutnom brzinom kotača $\dot{\varphi}_i$ preko izraza: $v_i = r\dot{\varphi}_i$. gdje je r radius kotača. Iz (3.6) i (3.5) nam slijedi:

$$\dot{\varphi}_i = \frac{1}{r}(-sin(\theta + \alpha_i)\dot{x} + cos(\theta + \alpha_i)\dot{y} + R\dot{\theta}).$$

Taj izraz možemo zapisati matično kao $\dot{\varphi}_i = J_{inv}\dot{u}$, gdje je J_inv inverzni Jakobian koji je direktna poveznica između kutnih brzina kotača $\dot{\varphi}_i$ i globalnih vektora brzine $\dot{u}$.

$$\begin{bmatrix} \dot{\varphi}_1 \\ \dot{\varphi}_2 \\ \dot{\varphi}_3 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} -sin(\theta) & cos(\theta) & R \\ -sin(\theta + \alpha_2) & cos(\theta + \alpha_2) & R \\ -sin(\theta + \alpha_3) & cos(\theta + \alpha_3) & R \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}. \qquad (1.5)$$

U većini slučajeva nije najzgodnije upravljati robotom u terminima globalnih koordinata. Puno je prirodnije i jednostavnije misliti i upravljati u terminima lokalnih koordinata. Stoga, pretvorimo globalne koordinate u lokalne na sljedeći način: Uzmimo da je

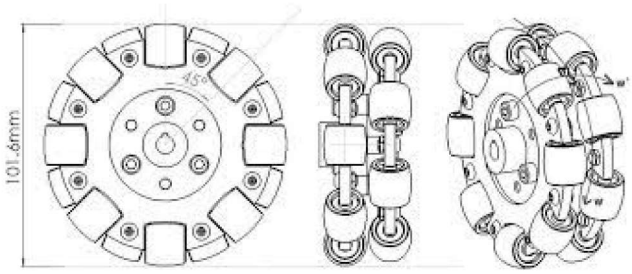
$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 & 0 \\ 0 & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x_L} \\ \dot{y_L} \\ \dot{\theta} \end{bmatrix}. \qquad (1.6)$$

Ako uvrstimo jednadžbu (1.6) u (1.5), dobijemo:

$$\begin{bmatrix} \dot{\varphi}_1 \\ \dot{\varphi}_2 \\ \dot{\varphi}_3 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} -sin(\theta) & cos(\theta) & R \\ -sin(\theta + \alpha_2) & cos(\theta + \alpha_2) & R \\ -sin(\theta + \alpha_3) & cos(\theta + \alpha_3) & R \end{bmatrix} \begin{bmatrix} cos(\theta) & 0 & 0 \\ 0 & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x_L} \\ \dot{y_L} \\ \dot{\theta} \end{bmatrix}.$$

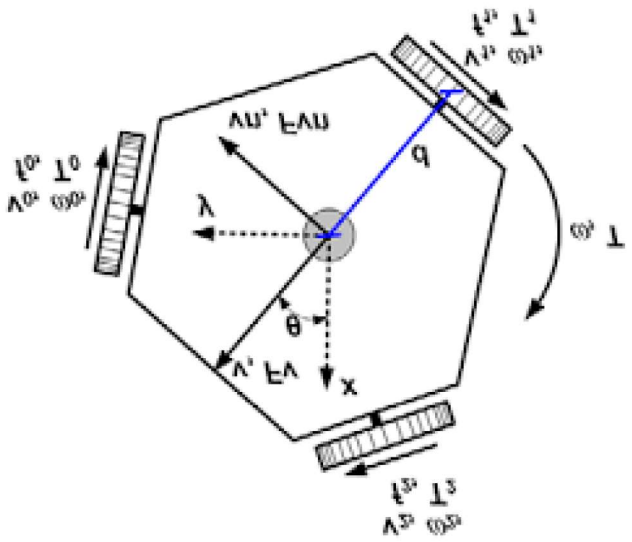
2.2 Kotači

Koristit ćemo švedske kotače. To su kotači s malim valjcima po obodu kotača koji su okomiti na smjer okretanja. Rezultat je takav da se kotač može voziti punom snagom, ali će i kliziti bočno s velikom lakoćom. Ti kotači imaju veliku ulogu u holonomskim pogonskim sustavima.



Slika 3: Švedski kotač. Slika preuzta iz "Article, Trajectory Tracking of an Omni-Directional Wheeled Mobile Robot Using a Model Predictive Control Strategy"

Često se koriste u izradi autonomnih robota koji mogu iskoristit prednost kratanja u svim smjerovima. Švedski kotači također se ponekad koriste kao kotači s pogonom za robote s diferencijalnim pogonom kako bi ubrzali okretanje. Platforma je opremljena s tri motora, a svaki motor služi za upravljanje jednim švedskim kotačem. Osi ova tri motora spajaju se na središte šasije i kutovi između svaka susjedna dva motora su jednaki. Na slici 4 prikazan je dijagrama platforme. Na temelju karakteristike švedskih kotača i analize modela sustava, platforma može provesti svesmjerno kretanje s prednostima snažne prilagodljivosti, visoke osjetljivosti, dobre stabilnosti i fleksibilna rotacija.

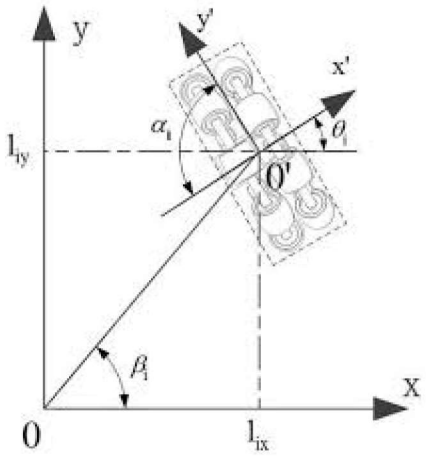


Slika 4: Dijagram platforme. Slika preuzeta iz "Bruno SicilianoOussama Khatib, Springer Handbook of Robotics"

Svaki kotač ima dva parametra za položaj: položaj u odnosu na centar platforme i pripadajući kut pozicioniranja. Koordinate švedskog kotača prikazane su na slici 5, gdje je xy koodrinatni okvir platforme sa središtem u centru, a $x^{'}y^{'}$ je koordinatni okvir kotača kojemu je središte također u centru. α_i je kut otklona svakog kotača, β_i je kut između pravaca $\overline{00'}$ i $\overline{0x}$ i

$(l_{ix}, l_{iy}, \theta_i)$ su položaj kotača i njegov kut u odnosu na koordinate xy.
l_{ix} i l_{iy} možemo izraziti kao

$$l_{ix} = l_i cos(\beta_i), l_{iy} = l_i sin(\beta_i).$$



Slika 5: Položaj kotača. Slika preuzeta iz

Neka su nadalje, r polumjer kotača, $V_i r$ vektor brzine kotača i ω_i kutna brzina kotača(motora). $[V_x V_y \omega]^T$ opisuje brzinu platforme u odnosu na pod, $[V_{ix} v_{iy} \omega_i]^T$ opisuje brzinu kotača u odnosu na xy koordinate i $[V_{ix}^{'}\ V_{iy}\ \omega_i^{'}]$ opisuje brzinu kotača u odnosu na koordinate $x^{'}y^{'}$. Kinematika kotača opisana je nasljedeči način:

$$\begin{bmatrix} V_{ix}^{'} \\ V_{iy}^{'} \end{bmatrix} = K_{i1} \begin{bmatrix} \omega_i \\ V_{ir} \end{bmatrix}, K_{i1} = \begin{bmatrix} 0 & \sin_{\alpha_i} \\ r & \cos_{\alpha_i} \end{bmatrix},$$

$$\begin{bmatrix} V_{ix} \\ V_{iy} \end{bmatrix} = K_{i2} \begin{bmatrix} V_{ix}^{'} \\ V_{iy}^{'} \end{bmatrix} = K_{i2} K_{i1} \begin{bmatrix} \omega_i \\ V_{ir} \end{bmatrix}, K_{i2} = \begin{bmatrix} \cos_{\theta_i} & -\sin_{\theta_i} \\ \sin_{\theta_i} & \cos_{\theta_i} \end{bmatrix},$$

$$\begin{bmatrix} V_{ix} \\ V_{iy} \end{bmatrix} = K_{i3} \begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix}, K_{i3} = \begin{bmatrix} 1 & 0 & -l_{iy} \\ 0 & 1 & l_{ix} \end{bmatrix},$$

gdje je inverz jednadžbe kinematike kotača dan sa:

$$K_{i2} K_{i1} \begin{bmatrix} V_{ix} \\ V_{iy} \end{bmatrix} = K_{i3} \begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix}, i = 1, 2, 3.$$

3 Komunikacija između mikrokontrolera

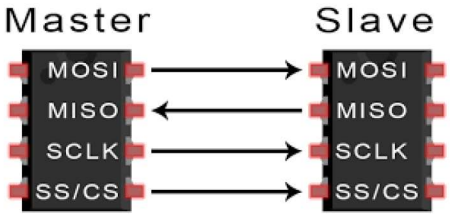
Postoji više različitih načina komunikacije između mikrokontrolera. U mogućnosti smo komunicirati bežičnim putem s:

- Bluetooth
- ZigBee
- WiFi

Ako ne tako, onda ubacite neke kabele u mješavinu i upotrijebite neograničeni broj utvrđenih protokola: I2C, SPI, UART, gdje poznavajući prednosti i mane od svakog možemo lako zaključiti koji nam najviše odgovara. U nastavku ćemo obraditi SPI protokol koji smo koristili za praktični zadatak.

3.1 SPI protokol

SPI je uobičajeni komunikacijski protokol koji koriste mnogi različiti uređaji. Na primjer, moduli SD kartica, moduli čitača RFID kartica i bežični odašiljač/prijamnik na 2,4 GHz koriste SPI za komunikaciju s mikrokontrolerima.
Jedna jedinstvena prednost SPI -a je činjenica da se podaci mogu prenositi bez prekida. Bilo koji broj bitova može se poslati ili primiti u kontinuiranom toku. S I2C i UART, podaci se šalju u paketima, ograničeni na određeni broj bitova. Uvjeti početka i zaustavljanja definiraju početak i kraj svakog paketa, pa se podaci prekidaju tijekom prijenosa.



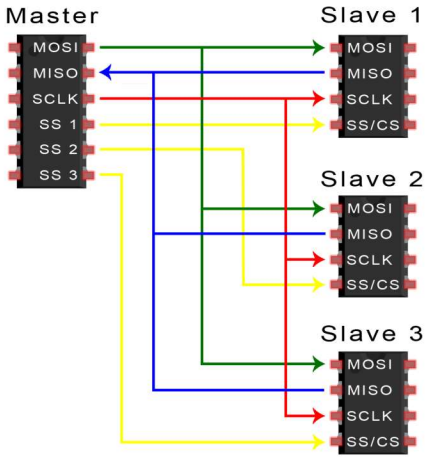
Slika 6: SPI komunikacija

Uređaji koji komuniciraju putem SPI-a su u odnosu master-slave. Master je upravljački uređaj (obično mikrokontroler), dok slave (obično senzor, zaslon ili memorijski čip) prima upute od nadređenog. Najjednostavnija konfiguracija SPI-a je jedan master, jedan slave sustav, ali jedan master može kontrolirati više od jednog slave-a.

- MOSI (Master Output/Slave Input) – Linija za slanje sa mastera na slave

- MISO (Master Input/Slave Output) – Linija za slanje sa slave-a na master.
- SCLK (Clock) – Linija za "clock" signal.
- SS/CS (Slave Select/Chip Select) – Linija za odabir slave-a na koji master šalje podatke.

SPI se može postaviti za rad s jednim master-om i jednim slave-om, a može se postaviti i s više slave-ova koje kontrolira jedan master. Postoje dva načina za povezivanje više slave-ova s glavnim uređajem. Ako master ima više pinova za odabir podređenog uređaja, slave uređaji mogu se spojiit kao na slici 7.



Slika 7: SPI komunikacija sa više slave-ova

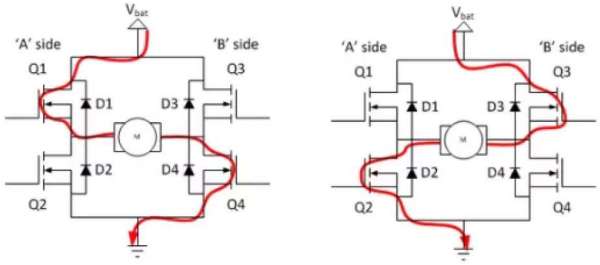
Master šalje podatke slave-u bit po bit, serijski kroz MOSI liniju. Slave prima podatke poslane s mastera na MOSI pinu. Podaci poslani s mastera na slave obično se prvo šalju s najznačajnijim bitom(msb). Slave također može serijski slati podatke natrag masteru putem MISO linije. Podaci poslani s slave-a natrag na master obično se prvo šalju s najmanje bitnim bitom(lsb).

4 Upravljanje motorima

4.1 H-Bridge i PWM

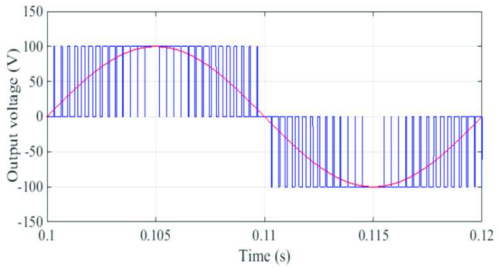
Motori robota bit će upravljani mikroprocesorom ATmega2560. Povratna petlja upravljanja kretanjem bit će zatvorena u središnjoj procesorskoj jedinici. Međutim, ova

jedinica ne može slati visoke napone i struje za isporuku snage na tri motora. Stoga je H-bridge(H-most) (vidi sliku 8.) ugrađen u sustav robota. H-Bridge se sastoji od četiri elektronička prekidača koji omogućuju okretanje motora unaprijed i unatrag te kočenje. Most ima tri ulaza: DIR za smjer vožnje, KOČNICA za kočenje i PWM za brzinu/radni ciklus. Za rad H-mosta potreban je PWM ulazni signal. Ovaj signal je jednostavan digitalni od 0 do 5V signal. Standardna DIO I/O kartica sama po sebi ne može proizvesti takav signal. Stoga je namjenska ploča PWM generatora uključena u elektroniku.



Slika 8: H-Bridge

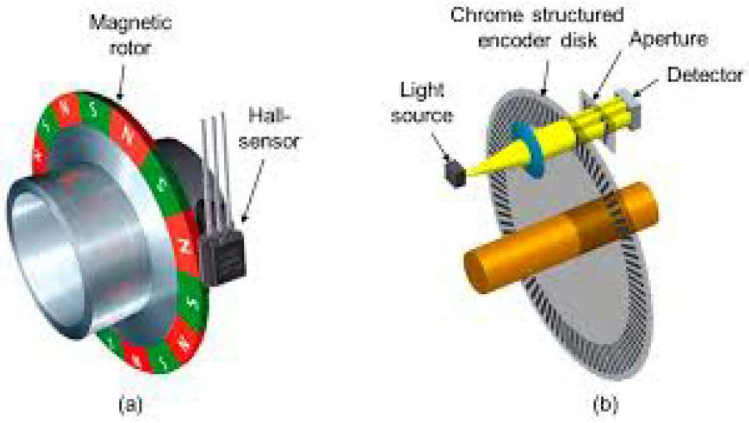
PWM ili modulacija širine impulsa znaka/magnitude (vidi sliku 9.) je tehnika u kojoj se radni ciklus (ili broj impulsa u jedinici vremena) signala modulira za podešavanje količine snage koja je poslati motorima. PWM signal kvadratni je val sa samo dvije vrijednosti napona koje odgovaraju brzini 0 i najvećoj brzini. Tehnika se sastoji od približavanja željenog kontinuiranog signala prebacivanjem između dvije vrijednosti napona (u biti uključivanje i isključivanje motora), tako da rezultirajuća prosječna vrijednost PWM signala odgovara željenoj kontinuiranoj vrijednosti (brzini).



Slika 9: Pulse Width Modulation

4.2 Enkoder

Pretvornik pomaka, davač pomaka, osjetilo pomaka ili enkoder (engl. encoder) je uređaj ili senzor (mjerno osjetilo) koji fizikalne veličine pretvara u električne signale preko pomaka (vidi slika 10.). Dijele se na ravne (linearne) i kutne (rotacijske). Kutni enkoder je elektromehanički uređaj koji pretvara kutni položaj ili kružno gibanje vratila ili osovine u analogni ili digitalni signal.



Slika 10: Encoder

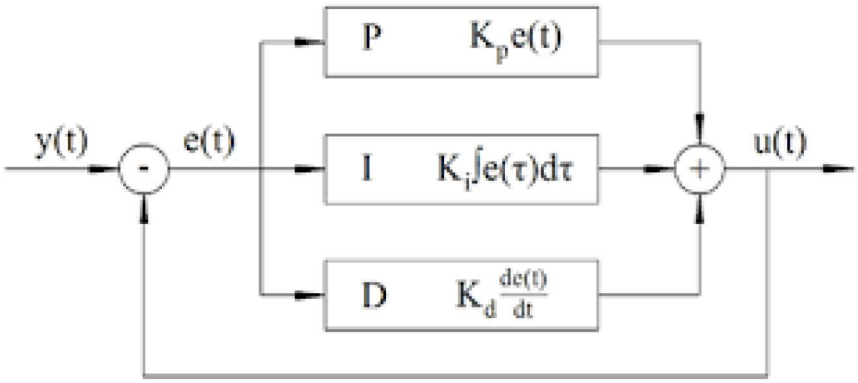
Jednostavno rečeno, enkoder je senzorski uređaj koji daje povratne informacije. Enkoderi pretvaraju kretanje u električni signal koji može očitati neka vrsta upravljačkog uređaja u sustavu za upravljanje kretanjem, poput brojača ili PLC -a. Enkoder šalje povratni signal koji se može koristit za određivanje položaja, brojača, brzine ili smjera. Upravljački uređaj može koristiti te podatke za slanje naredbe za određenu funkciju. Pomoću enkodera dobivamo informaciju o brzini kretanja naših motora. Rezolucija našeg enkodera izražena je u broju tickova. Jedan puni okretaj motara, u našem slučaju, iznosi 640 tickova. Mjenjanjem broja tickova možemo upravljati brzinom okretanja motora. Ako motor pomičemo za 640 tickova u sekundi, on če napraviti jedan puni okretaj po sekundi. Ako ga pak pomičemo za 1280 tickova po sekundi, on če napraviti dva puna okretaja po sekundi. Naravno svaki motor ima svoje maksimalne i minimalne brzine okretaja, a to lako saznamo iz dokumentacije motora.

4.3 PID regulator

Kako smo obradili upravljanje motora sa hardware-ske strane, imamo dobru podlogu za obradu software-skog dijela upravljanja sa motorima, stoga ćemo sada predstaviti PID regulator.
PID regulator mehanizam je upravljačke petlje koji koristi povratnu informaciju koja se široko koristi u industrijskim sustavima upravljanja i raznim drugim aplikacijama koje zahtijevaju kontinuirano modulirano upravljanje. PID regulator neprestano izračunava vrijednost pogreške kao razliku između željene zadane vrijednosti i izmjerene varijable procesa i primjenjuje korekciju na temelju proporcionalnih, integrabilnih i derivacijske pojmova (označeni s P, I i D), pa otuda i naziv.

$$u(t) = K_p e(t) + K_i \int e(t)\,dt$$

$u(t)$ = PID varijabla kontrole
K_p = proporcionalni doprinos
$e(t)$ = vrijednost pogreške
K_i = integralni doprinos
de = promjena pogreške
dt = promjena vremena

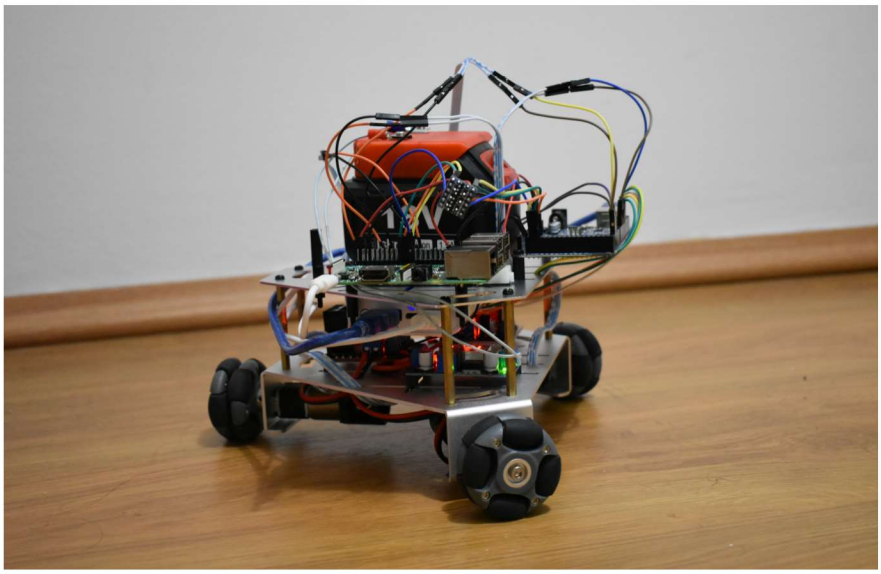


Slika 11: PID regulator

U praktičnom smislu, automatski primjenjuje točnu i osjetljivu korekciju na kontrolnu funkciju. Svakodnevni primjer je tempomat na automobilu, gdje bi se usponom na brdo smanjila brzina kada bi se primijenila samo konstantna snaga motora. PID algoritam regulatora vraća izmjerenu brzinu na željenu brzinu uz minimalno kašnjenje i prekoračenje povećavajući izlaznu snagu motora na kontroliran način.

5 Praktični zadatak - Svesmjerni robot

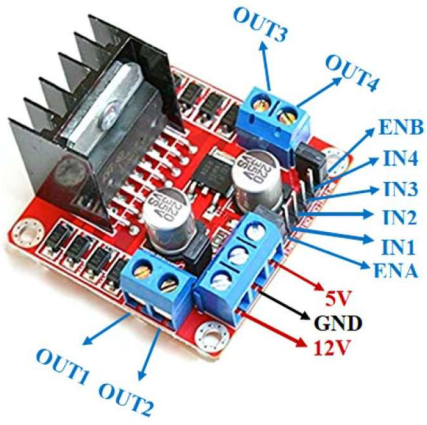
U ovom poglavlju opisat ćemo cijelu konstrukciju našeg svesmjernog robota. Objasniti ćemo kako su komponente povezane i koja im je uloga.



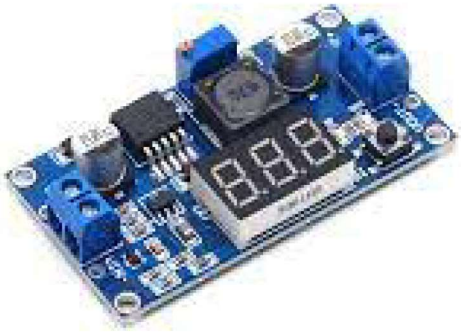
Slika 12 : Svesmjerni robot

5.1 Hardware

Robot je postavljen tako da je njegov okvir podijeljen na dvije razine i trokutastog oblika. Na prvoj razini nalaze se kotači, motori, dva H-bridge-a i step-down converter.

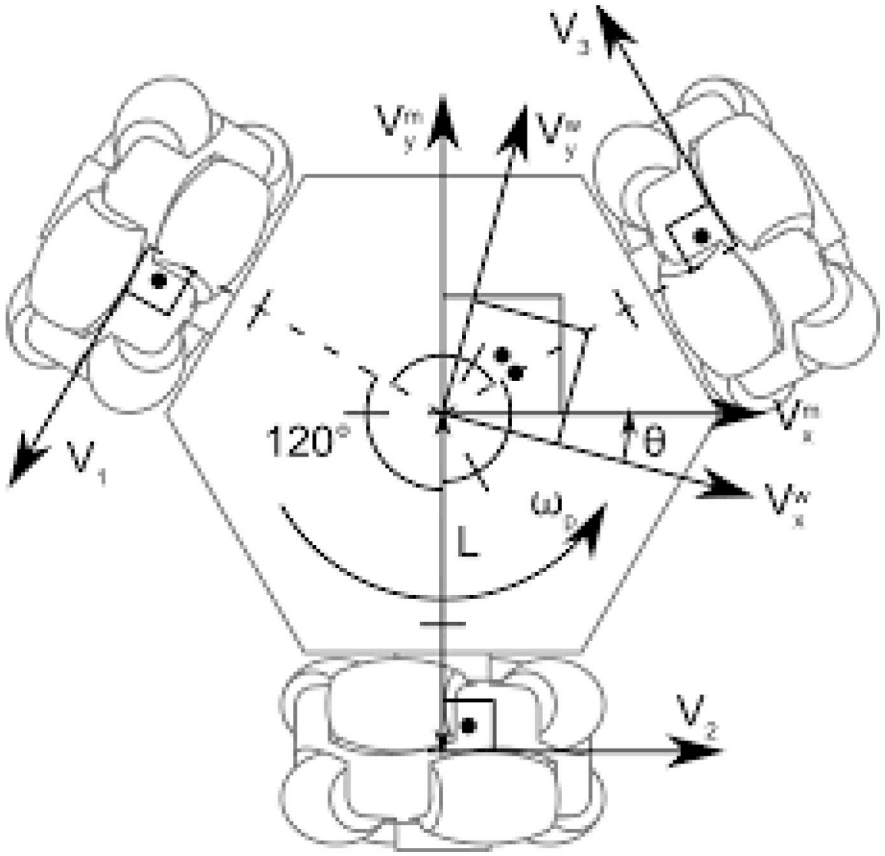


Slika 13 : H-bridge



Slika 14 : Step-down converter

Svaki kotač jednako je udaljen od susjedna dva i tangencijalno je postavljen s obzirom na pravac od središta do kotača. Kut između svaka dva kotača iznosi 120°. Ti kotači spojeni su na 12V DC motore sa encoderima koji nam služe za očitavanje brzine.
Motori su spojeni na H-bridge komponente kojima se jednostavno može obrnuti polaritet napona i tako mjenjati smjer okretanja motora. Na drugoj razini nalazi se baterija od 18V, Arduino Mega 2560 i Raspberry Pi 3.
Baterija napaja motore na koje je spojena preko step-down convertera, čija je uloga smanjivanje napona na optimalnih 12V. Svaki motor spojen je na Arduino Mega 256, koji je pak spojen sa Raspberry Pi 3.



Slika 15 : Pozicioniranje kotača

5.1.1 Raspberry Pi 3

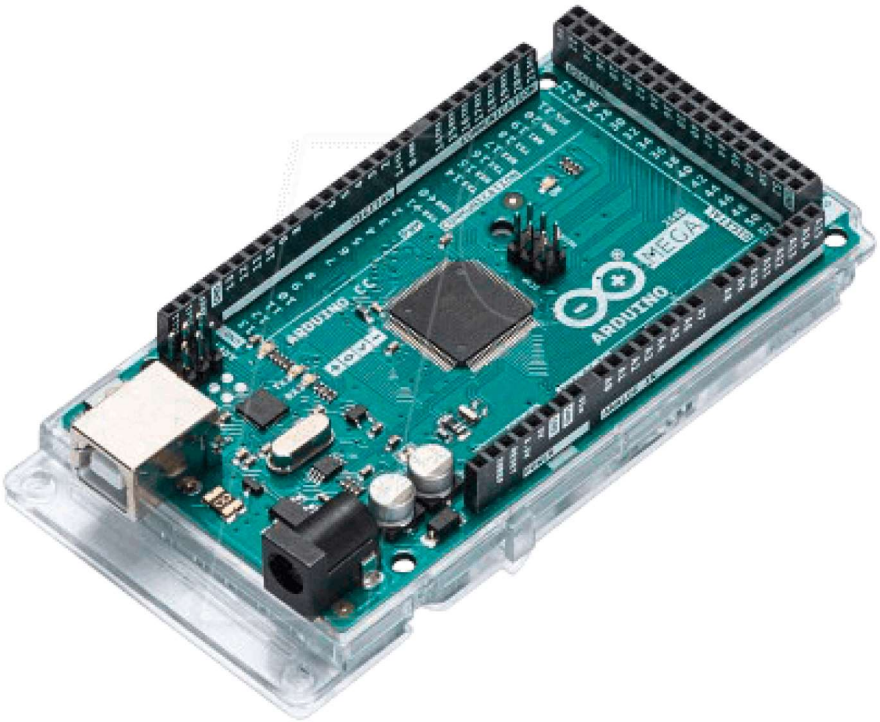
Raspberry Pi je jeftino računalo veličine kreditne kartice koje se priključuje na monitor računala ili TV i koristi standardnu tipkovnicu i miš. Sposoban je učiniti sve što biste očekivali od stolnog računala, od pregledavanja interneta i reprodukcije videa visoke razlučivosti, do obrade teksta i igranja igara. Štoviše, Raspberry Pi ima sposobnost interakcije s vanjskim svijetom i korišten je u širokom spektru projekata digitalnih proizvođača, od glazbenih strojeva i roditeljskih detektora do meteoroloških stanica.

Slika 16: Raspberry Pi 3

Model koji se koristi u ovom projektu je Rasspberry Pi 3. U našem slučaju on ima funkciju glavnog kontrolera koji povezuje funkcionalnosti Joysticka sa kotačima. Dohvaća podatke koje Joystick šalje, obrađuje ih te šalje putem SPI protokola na Arduino Mega 2560.

5.1.2 Arduino Mega 2560

Arduino Mega 2560 je jeftina, fleksibilna i jednostavna za korištenje programabilna ploča mikrokontrolera otvorenog koda koja se može integrirati u razne elektroničke projekte. Ova ploča može biti povezana s drugim Arduino pločama, Arduino štitovima, Raspberry Pi pločama i može upravljati relejima, LED diodama, servo motorima i izlazima. Ploča je zasnovana na ATmega2560. Ima 54 digitalna ulazno/izlazna pina (od kojih se 15 može koristit kao PWM izlaz), 16 analognih ulaza, 4 UART -a, kristalni oscilator od 16 MHz, USB veza, utičnica za napajanje, ICSP zaglavlje, i gumb za resetiranje. Sadrži sve potrebno za podršku mikrokontrolera.



Slika 17: Arduino Mega 2560

Na Arduinu uspostavljamo SPI komunikaciju sa Raspberry Pi 3, primljene vrijednosti obrađujemo PID regulatorm, upravljamo motorima te očitavamo njihovu brzinu pomoću encodera.

5.2 Software

Tehnologije

Implementacija sustava sastoji se od tri programska jezika, a to su Python, C i JavaScript. Python moduli:

- spidev - uspostavljanje SPI komunikacije
- flask - postavljanje web servera i komunikacija sa klijentom

Uređaji koji pokreću i čine sustav:

- Raspberry Pi 3 - pokretanje servera, dohvaćanje podataka sa klijenta i slanje kutnih brzina na Arduino Mega 2560
- Arduino Mega 2560 - dohvaćanje kutnih brzina, PID regulator, primjenjivanje brzina na motore(kotače)

Koristeći ove tehnologije uspostavljamo vezu između low-level i web programiranja.

Organizacija projekta

Ovaj projekt sastoji se od 3 skripte:

- client - JavaScript skripta koja postavlja web/mobilnu aplikaciju Joystick
  (pokreće se sa računala/mobilnog uređaja)
- server - Python skripta u kojoj postavljamo server, dohvaćamo podatke sa klijenta, obrađujemo ih te šaljemo na Arduino Mega 2560 preko SPI protokola
  (pokreće se na Raspberry Pi 3)
- mega - C skripta u kojoj primamo podatke putem SPI protokola sa Raspberry Pi-a, te koristeći PID računamo optimalni napon koji šaljemo na motore(kotače)
  (pokreće se na Arduino Mega 256)

Zadaća softvera je da poveže naš sustav u jednu cjelinu. Potrebno je uspostaviti i održavati komunikaciju među komponentama i server-klijent relaciji bez ikakvih smetnji kako bi robot pravilno funkcionirao.

Tijek koda izvršava se na sljedeći način:

1. Raspberry Pi i uređaj na kojem će se nalaziti Joystick spajamo na istu internetsku konekciju
2. Pokrećemo Python skriptu koja uspostavlja SPI komunikaciju sa Arduinom, otvara server na IP adresi Raspberry Pi-ja, te dohvaća podatke sa klijenta
3. Nakon što smo dohvatili podatke, obrađujemo ih i pretvaramo u odgovarajuće kutne brzine
4. Kada smo dobili kutne brzine šaljemo ih na Arudino putem SPI protokola
5. Primljene podatke na Arduinu dodatno obradimo koristeći PID regulator koji računa prikladni napon i šalje ga motorima

5.2.1 Http Server i Joystick

Na Raspberry-ju pokrećemo HTTP server koji pri zahtjevima HTTP POST metode očitava podatke sa Joysticka. Na slici br. 7 možemo vidjeti kako izgleda on izgleda i koje vrijednosti možemo očitati, a od kojih uzimamo distance, force, x i y.

Slika 18: Joystick

Joystick je implementiran u JavaScript okruženju koristeći nippleJs biblioteku (vidi Kod 1.).

```
var s = function(sel) {
    return document.querySelector(sel);
  };

  var joysticks = {
    zone: s('.zone.static'),
    mode: 'static',
    position: {
    left: '50%',
    top: '50%'
    },
    color: 'green'
  };
  var joystick;

  createNipple('static');

var isReady = true;
```

```
function bindNipple() {
  joystick.on('start end', function(evt, data) {
    if(isReady)
      debug(data);
  }).on('move dir', function(evt, data) {
    if(isReady)
      debug(data);
  })
}

function createNipple(evt) {
  joystick = nipplejs.create(joysticks);
  bindNipple();
}

function debug(obj) {
  const {position, force, pressure, distance, angle, direction} = obj;
  (async () => {
      isReady = false;
      const rawResponse = await fetch("http://192.168.0.107:8000", {
        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({position, force, pressure, distance, angle, dire
      });
      isReady = true;
    })();
}
```

Kod 1. Implementacija Joysticka

Unutar funkcije „debug()“ dohvaćamo sve podatke i spremamo u varijable. Nakon toga pozivamo asinkronu metodu koja se okida na svaki pomak našeg Joystick-a, a pri zahtjevima HTTP POST metode, server očitava poslane brzine. No, server se izvršava unutar posebne niti izvršavanja. Zbog toga ne stignemo odmah preračunati brzine i poslati na Arduino. Kako bismo spriječili gomilanje zahtjeva prema poslužitelju, na klijentu je implementirano blokiranje slanja novih zahtjeva dok prethodni zahtjev nije dobio odgovor od poslužitelja.

5.2.2 Obrada i slanje kutnih brzina

Kada smo dobili željene vrijdnosti sa Joysticka, krećemo ih obrađivati. Cilj nam je dobiti brzine koje su u takvom rasponu da ih motori mogu proizvesti (vidi Kod2.). U našem slučaju taj raspon će biti od 0 do 640 tickova. O tickovima ćemo reći nešto više u sljedećem potpoglavlju "Encoder".

```
import spidev, time, math
from flask import Flask, request
from flask_cors import CORS
import numpy as np

def mapx(x):
    return math.floor((x - 5) * 640 / (50 - 5))

def mapy(y):
    return math.floor((y - 5) * 640 / (250 - 5))

app = Flask(__name__)

CORS(app)
spi = spidev.SpiDev()
spi.open(0, 0)
spi.mode = 0b00
spi.max_speed_hz = 1000000
spi.bits_per_word = 8

@app.route('/', methods=['GET', 'POST'])
def getData():
    turn_transfer1 = 0
    turn_transfer2 = 0
    speed_transfer1 = 0
    speed_transfer2 = 0
    x_speed = 0
    y_speed = 0
    motor_values = [0, 0, 0, 0]

    data = request.get_json()

    force = data.get('force')
    distance = data.get('distance')
    x = data.get('position').get('x') - 206
    y = data.get('position').get('y') - 455

    if(force != None):
        if(force > 5):
            force = 5
        if(distance > 48) :
            y = y*force

    if(x > 5):
        x_speed = -1 * mapx(x)
    elif(x < -5):
        x_speed = mapx(abs(x))
    else:
        x_speed = 0

    if(y > 5):
        y_speed = -1 * mapy(y)
    elif(y < -5):
        y_speed = mapy(abs(y))
    else:
        y_speed = 0
```

Kod 2. Dohvaćanje i obrada podataka

Nakon što smo obradili podatke potrebno ih je poslati preko SPI protokola na Arduino. Za to koristimo spidev modul pomoću kojeg konfiguriramo SPI komunikaciju što također možemo vidjeti u gore priloženom kodu. Naime, SPI protokol radi tako da se podatci šalju byte po byte, pa ako želimo poslati neku vrijednost koja je veća od byte-a moramo rastaviti na više byte-ova i tako slati jedan po jedan. To radimo pomoću procedure zvane bit-masking.

5.2.3 Encoder

Enkoder spojimo na ulazne pinove Arduina i omogućimo interrupt vektore koji će reagirati na promjene stanja na tom pinu. Nakon svakog interrupta inkrementiramo broj tickova i tako dobijemo informaciju o brzini (vidi Kod 3.).

```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile uint8_t A, B, C, D, E, F;
volatile int32_t m1_ticks = 0;
volatile int32_t m2_ticks = 0;
volatile int32_t m3_ticks = 0;
volatile int32_t m1_ticks_old = 0;
volatile int32_t m2_ticks_old = 0;
volatile int32_t m3_ticks_old = 0;

ISR(INT0_vect) {
        A != B ? --m1_ticks : ++m1_ticks;
        A = PIND & (1 << PIND0) ? 1 : 0;
}

ISR(INT1_vect) {
        B = PIND & (1 << PIND1) ? 1 : 0;
        A != B ? --m1_ticks : ++m1_ticks;
}

ISR(INT2_vect) {
        C != D ? --m2_ticks : ++m2_ticks;
        C = PIND & (1 << PIND2) ? 1 : 0;
}

ISR(INT3_vect) {
        D = PIND & (1 << PIND3) ? 1 : 0;
        C != D ? --m2_ticks : ++m2_ticks;
}

ISR(INT4_vect) {
        E != F ? --m3_ticks : ++m3_ticks;
        E = PINE & (1 << PINE4) ? 1 : 0;
}

ISR(INT5_vect) {
        F = PINE & (1 << PINE5) ? 1 : 0;
        E != F ? --m3_ticks : ++m3_ticks;
}

void encoder_init() {
```

```
    DDRD &= ~(1 << DDD0);
    DDRD &= ~(1 << DDD1);
    DDRD &= ~(1 << DDD2);
    DDRD &= ~(1 << DDD3);
    DDRE &= ~(1 << DDE4);
    DDRE &= ~(1 << DDE5);

    EICRA |= (1 << ISC00) | (1 << ISC10) | (1 << ISC20) | (1 << ISC30);
    EICRB |= (1 << ISC40) | (1 << ISC50);
    EIMSK |= (1 << INT0) | (1 << INT1) | (1 << INT2) | (1 << INT3) | (1 << INT4) | (
    sei();
    uart_putstr("Encoder initialized!");
}
```

Kod 3. Implementacija enkodera

I sada kada imamo i tu povratnu informaciju o brzini naših motora, možemo implementirati PID regulator.

5.2.4 PID regulator

Kada smo primili podatke na Arduinu i sklopili ih u int16, prosljeđujemo ih PID regulatoru. U nastavku imamo priložen kod PID regulatora koji je implementiran formulom koja se nalazi na slici br. 8.

```
ISR(TIMER0_COMPA_vect) {
    ++ms;

    if(ms % 2 == 0) {
        sendToPlotter = 1;
    }

    if(ms % 10 == 0) {
        tps_m2 = (m2_ticks - m2_ticks_old) * 1000 / 19;
        tps_m3 = (m3_ticks - m3_ticks_old) * 1000 / 19;

        err2 = tps_w2 - tps_m2;
        err3 = tps_w3 - tps_m3;
        err_acc2 += err2;
        err_acc3 += err3;
        err_diff2 = err2 - err_old2;
        err_diff3 = err3 - err_old3;

        gain2 = P * err2 + I * err_acc2 + DI * err_diff2;
```

```
                gain3 = P * err3 + I * err_acc3 + DI * err_diff3;
                gain2 /= 10;
                gain3 /= 10;

                if(gain2 > 0) {
                        speed2 = gain2 > max ? max : gain2;
                        OCR5A = speed2;
                        OCR5B = 0;
                } else {
                        gain2 *= -1;
                        speed2 = gain2 > max ? max : gain2;
                        OCR5B = speed2;
                        OCR5A = 0;
                }

                if(gain3 > 0) {
                        speed3 = gain3 > max ? max : gain3;
                        OCR4A = speed3;
                        OCR4B = 0;
                } else {
                        gain3 *= -1;
                        speed3 = gain3 > max ? max : gain3;
                        OCR4B = speed3;
                        OCR4A = 0;
                }

                err_old2 = err2;
                err_old3 = err3;
                m2_ticks_old = m2_ticks;
                m3_ticks_old = m3_ticks;
        }
}
```

Kod 4. Implementacija PID regulatora

Time dobijemo optimalne vrijednosti koje šaljemo motorima kako bi oni radili što glađe, bez trzaja i naglih pokreta.

Literatura

[1] ATmega2560 datasheet, dostupno na:
(https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf)

[2] Bruno SicilianoOussama Khatib, Springer Handbook of Robotics, dostupno na:
(https://link.springer.com/referencework/10.1007%2F978-3-540-30301-5)

[3] Li-li Li, a, Jing-yu He, Yong-peng Zhao and Jian-hong Yang1, "Design of Microcontroller Standard SPI Interface", Institute of Microelectronics, School of Physical Science and Technology, Lanzhou University, Lanzhou 730000, P. R. China

[4] Introduction to Autonomous Mobile Robots, Second Edition, dostupno na:
(https://mitpress.mit.edu/books/introduction-autonomous-mobile-robots-second-editi

[5] Su Whan Sung, Jietae Lee, In-Beum Lee "Process Identification and PID Control"

[6] Josipa J. Strossmayera u Osijeku, Odijel za Matematiku, Ugrađeni sustavi - Predavanja i vježbe

[7] Josipa J. Strossmayera u Osijeku, Odijel za Matematiku, Web programiranje - Predavanja i vježbe