# Upravljanje mobilnim robotom s diferencijalnim pogonom

**Erceg, Roko**

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

Roko Erceg

# Upravljanje mobilnim robotom s diferencijalnim pogonom

Završni rad

Osijek, 2021.

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

Roko Erceg

# Upravljanje mobilnim robotom s diferencijalnim pogonom

Završni rad

Mentor: izv. prof. dr. sc. Domagoj Matijević

Komentor: Jurica Maltar

Osijek, 2021.

# Control of differential drive mobile robot


## Sažetak

U ovom radu bavimo se upravljanjem mobilnih robota s diferencijalnim pogonom. Najprije izučavamo kinematiku ovakvih robota kako bismo povezali okretanje kotača s promjenom robotove pozicije u prostoru. Analiziramo uvjete vrtnje i neproklizavanja standardnog fiksnog kotača i dobivene rezultate koristimo kako bismo opisali kretanje mobilnog robota. Sljedeći problem kojim se bavimo jest $I^2C$ komunikacija. Objasnit ćemo arhitekturu $I^2C$ uređaja i objasniti protok podataka kroz $I^2C$ sabirnicu. Nakon toga prelazimo na PID regulator. Na jednostavnom primjeru objasnit ćemo važnost pojedinih dijelova PID regulatora. Također, doći ćemo do jednadžbe koju ćemo koristiti kako bismo okretali motore željenim kutnim brzinama. Na kraju ćemo predstaviti jednu implementaciju mobilnog robota s diferencijalnim pogonom.



## Ključne riječi

mobilni robot, diferencijalni pogon, standardni fiksni kotač, direktna kinematika, inverzna kinematika, PID-regulator, mikrokontroler, Raspberry Pi, $I^2C$, upravljač, JavaScript, HTML, CSS, Node.js, C



## Abstract

In this paper we discuss control of differential drive mobile robots. Firstly we study kinematics of this kind of mobile robots so we could map wheel spin to a change in robot's position. We analyze rolling constraints and sliding constraints of fixed standard wheel and use the resulting expressions to describe mobile robot's movements. Next we talk about $I^2C$ communication. We will describe architecture of $I^2C$ devices and talk about how data flows through $I^2C$ bus. After that we move on to PID controller. We will use a simple example to show importance of every parameter of PID controller. Also, we will derive an equation that we will use to control angular velocity of our wheels. Lastly, we will show one implementation of a differential drive mobile robot.



## Key words

mobile robot, differential drive, fixed standard wheel, forward kinematics, inverse kinematics, PID controller, microcontroller, Raspberry Pi, $I^2C$, joystick, JavaScript, HTML, CSS, Node.js, C

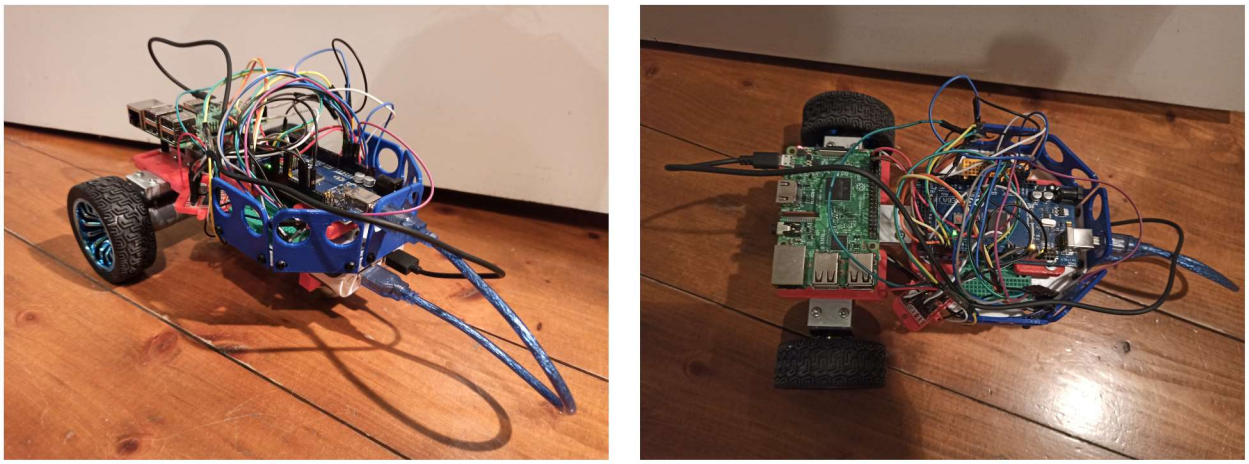
# Sadržaj

# 1 Uvod

Mobilni robot s diferencijalnim pogonom je robot koji za kretanje koristi dva nezavisno pogonjena kotača koji se nalaze na suprotnim stranama robotske šasije. Robot koji je izrađen uz ovaj završni rad možete vidjeti na slici 1. Njegovi glavni dijelovi su Raspberry Pi računalo, mikrokontroler Arduino Mega 2560 i dva elektromotora. Ovaj robot upravlja se pomoću mobilne aplikacije koja putem HTTP protokola šalje linearnu i kutnu brzinu prema poslužitelju na Raspberry Pi računalu. Poslužitelj tada dobivene brzine preračunava u kutne brzine lijevog i desnog motora. Kako su brzine realni brojevi, taj izračun odvija se na Raspberry Pi računalu jer Arduino Mega nema *floating point* jedinicu.

U prvom poglavlju bavimo se kinematikom robota s diferencijalnim pogonom. Upravo u ovom dijelu dolazimo do potrebnih formula za dobivanje kutnih brzina motora. Najprije objašnjavamo kako ćemo predstavljati poziciju i kretanje robota u prostoru. Zatim, dolazimo do potrebnih formula na dva različita načina. Prvi je promatranjem gibanja svakog kotača posebno i kombiniranjem njihovih doprinosa kretanju robota. Drugi način za doći do željenih formula jest analiziranje ograničenja specifičnih za tip kotača koji koristimo kod ovog mobilnog robota. Prvo ćemo iskazati ta ograničenja u obliku jednadžbi, a onda ćemo iskoristiti te jednadžbe kako bismo došli do željenih formula. Nakon što na poslužitelju pretvorimo dobivene vrijednosti u kutne brzine motora, trebamo ih poslati s Raspberry Pi računala na Arduino. Tu komunikaciju vršimo putem $I^2C$ protokola.

O $I^2C$ protokolu govorimo u drugom poglavlju ovog rada. Opisat ćemo arhitekturu potrebnu za $I^2C$ komunikaciju i objasnit ćemo na koji se način prenose podaci. Kada mikrokontroler dobije izračunate kutne brzine motora, onda on nekako treba reći motorima da se okreću željenom brzinom. Tako dolazimo do sljedećeg poglavlja koje se odnosi na PID regulator. U ovom poglavlju vidjet ćemo na primjeru tempomata za automobil na koji način radi PID regulator. Iskazat ćemo formulu prema kojoj ćemo na mikrokontroleru upravljati kutnim brzinama motora. Četvrto poglavlje sadrži moju implementaciju mobilnog robota s diferencijalnim pogonom uz objašnjenje svih dijelova programskog koda.



Slika 1: Robot s diferencijalnim pogonom.

# 2 Kinematika

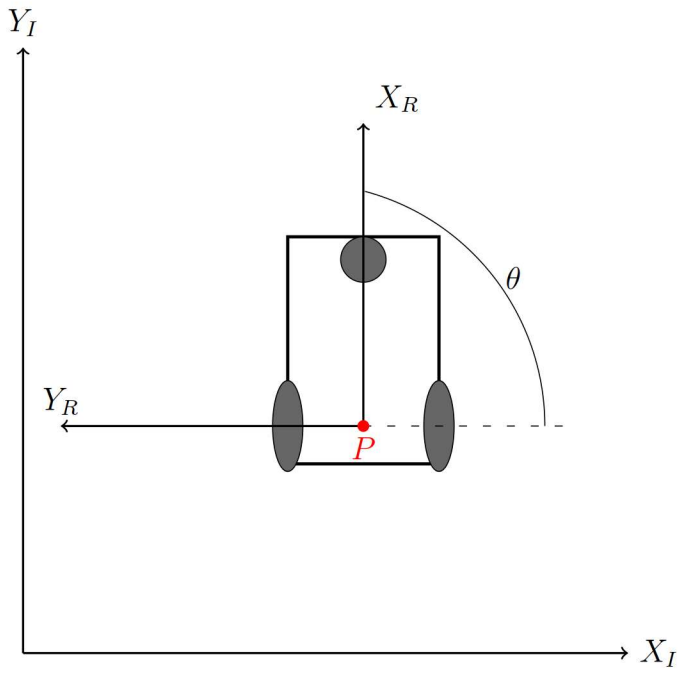
Kinematika je grana mehanike koja proučava gibanje tijela ili materijalne točke, bez obzira na njegove uzroke, masu i djelovanje sile. U ovom poglavlju bavit ćemo se kinematikom mobilnog robota s dva pokretna kotača. Poznavajući promjenu koordinata te promjenu orijentacije mobilnog robota, možemo doći do određenih zaključaka vezanih uz brzinu i putanju robota.

## 2.1 Pozicija mobilnog robota

Za precizno upravljanje mobilnim robotom, bitno je znati robotovu poziciju. Budući da mobilni roboti nisu fiksirani za jednu točku okoline, nego se slobodno njome kreću, za praćenje pozicije nije dovoljno samo očitati trenutno stanje pojedinog kotača. Kako bismo izračunali stanje $\xi$ u trenutku $t$, potrebno je, između ostalog, poznavati stanje robota u trenutku $t-1$. Radi jednostavnijeg računanja pozicije, uvodimo pojam materijalne točke.

**Definicija 21**
*Matrijalna točka jest idealizirano tijelo kojemu je ukupna masa koncentrirana u jednoj točki. Drugim riječima, to je zamišljena točka na promatranom tijelu koju, radi pojednostavljenja pri izračunima, promatramo kao to tijelo.*

U slučajevima kao što je naš, kada tijelo robota možemo promatrati kao jednu cijelinu, poziciju robota opisujemo pozicijom materijalne točke. Na slici 2, materijalna točka označena je točkom $P$.




Slika 2: Robotov lokalni koordinatni sustav u globalnom koordinatnom sustavu.

Kretanje mobilnog robota možemo pratiti u pravokutnom koordinatnom sustavu u ravnini. Budući da ćemo htjeti određivati promjenu položaja u robotovom lokalnom sustavu, gdje na riječi poput "naprijed", "natrag", "lijevo" i "desno" gledamo iz perspektive robota, imat

ćemo dva koordinatna sustava. Globalni koordinatni sustav određen je koordinatnim osima $X_I$ i $Y_I$. Robotov lokalni koordinatni sustav određen je osima $X_R$ i $Y_R$, te ima ishodište u materijalnoj točki $P$. Za reprezentaciju stanja robota u globalnom koordinatnom sustavu koristimo trodimenzionalni vektor:

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}.$$

Vrijednosti $x$ i $y$ određuju poziciju mobilnog robota u globalnom koordinatnom sustavu, dok vrijednost $\theta$ određuje orijentaciju robota (otklon osi $X_R$ s obzirom na os $X_I$). Kako bismo promjenu globalnog stanja robota $\dot{\xi}_I$ pretvorili u promjenu lokalnog stanja $\dot{\xi}_R$ u robotovom lokalnom koordinatnom sustavu koristimo ortogonalnu matricu rotacije:

$$R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Iz promjene globalnog stanja u promjenu lokalnog stanja dolazimo sljedećom formulom:

$$\dot{\xi}_R = R(\theta)\dot{\xi}_I. \tag{1}$$

Matrica rotacije $R(\theta)$ jest regularna matrica. Navedenu tvrdnju lako je dokazati. Pogledajmo determinantu matrice $R(\theta)$:

$$\det R(\theta) = \begin{vmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} = cos^2\theta + sin^2\theta = 1 \neq 0 \quad \forall\theta \in \mathbb{R}.$$

Kako je determinanta matrice rotacije različita od nule, to znači da je matrica rotacije regularna matrica. Inverz matrice rotacije jest njena transponirana matrica $R^T(\theta)$:

$$R^T(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Imajući ovo na umu, lako dolazimo do formule za računanje promjene globalnog stanja poznavajući promjenu lokalnog stanja robota. Množeći jednadžbu (1) slijeva matricom $R^T(\theta)$ dobivamo:

$$\dot{\xi}_I = R^T(\theta)\dot{\xi}_R. \tag{2}$$

Pogledajmo vrijednosti matrice rotacije te vrijednosti $\dot{\xi}_I$ i $\dot{\xi}_R$ na jednostavnom primjeru iz slike 2.

**Primjer 21**

*U ovom primjeru vrijednost $\theta$ iznosi $\frac{\pi}{2}$ pa matrica rotacije izgleda ovako:*

$$R(\frac{\pi}{2}) = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

*Uzmimo neka je promjena u lokalnom koordinatnom sustavu jednaka:*

$$\dot{\xi}_R = \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix}.$$

*Prema formuli (2):*

$$\dot{\xi}_I = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -\dot{y} \\ \dot{x} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}.$$

*Dobiveni rezultat ima smisla jer promjenu za pet jedinica u smjeru osi $X_R$ poistovjećujemo s promjenom za pet jedinica u smjeru osi $Y_I$, a za trenutnu orijentaciju robota te osi su paralelne i jednako orijentirane.*

## 2.2 Direktna kinematika

U ovom potpoglavlju bavimo se problemom određivanja pozicije robota u prostoru iz poznavanja kutnih brzina njegovih kotača. Parametri našeg mobilnog robota s diferencijalnim pogonom su radijusi kotača $r$ i duljina poluosovine $l$ (udaljenost sredine kotača do materijalne točke $P$). Preko parametara $r$ i $l$, te orijentacije robota $\theta$ i kutnih brzina lijevog i desnog kotača $\dot{\varphi}_1$ i $\dot{\varphi}_2$, želimo doći do funkcije $f$ koja daje translacijsku brzinu $v(t)$ i kutnu brzinu $\omega(t)$, tj.

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = f(l, r, \theta, \dot{\varphi}_1, \dot{\varphi}_2). \tag{3}$$

Translacijska brzina

$$v = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}, \tag{4}$$

nije ništa drugo nego promjena $x$ i $y$ koordinata u globalnom koordinatnom sustavu. Kutna brzina

$$w = \dot{\theta}, \tag{5}$$

jest promjena kuta $\theta$. Uvrštavajući (4) i (5) u (3) imamo:

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(l, r, \theta, \dot{\varphi}_1, \dot{\varphi}_2). \tag{6}$$

Kako pomoću formule (2) jednostavno možemo izračunati $\dot{\xi}_I$ iz $\dot{\xi}_R$, prvo ćemo promatrati utjecaj kutnih brzina kotača na promjenu lokalnog stanja robota. Veza između brzine $v$ i kutne brzine $\omega$ općenito se računa prema formuli:

$$v = r\omega, \tag{7}$$

gdje je $r$ udaljenost od osi vrtnje. U našem slučaju imamo dva kotača koja se kružno gibaju brzinama $\dot{\varphi}_1$ i $\dot{\varphi}_2$. Kako se materijalna točka $P$ nalazi na polovištu osovine koja nosi ova dva kotača, tada ona ima translacijsku brzinu u smjeru osi $X_R$ jednaku

$$\dot{x}_R = \frac{r\dot{\varphi}_1}{2} + \frac{r\dot{\varphi}_2}{2}. \tag{8}$$

Do gornje formule dolazimo tako da izračunamo posebno brzinu koju doprinosi pojedini kotač te dobiveni rezultat zbrojimo. Budući da se ovaj robot ne može kretati u smjeru osi $Y_R$ (u nastavku ćemo navesti uvjet neproklizavanja zbog kojega ovo smatramo istinitim), translacijska brzina u smjeru osi $Y_R$ iznosi:

$$\dot{y}_R = 0. \tag{9}$$

Za kutnu brzinu robota $\omega$, ponovno ćemo posebno pogledati slučajeve za svaki od kotača i onda zbrojiti dobivene rezultate. U slučaju kada je brzina lijevog kotača $\dot{\varphi}_2$ jednaka nula (slika 3), robot se rotira oko lijevog kotača. Duljina puta $z_1$ koju desni kotač prijeđe u jednoj sekundi jednaka je

$$z_1 = \frac{\dot{\varphi}_1}{2\pi rad/s} 2r\pi = \frac{r\dot{\varphi}_1}{rad/s}. \tag{10}$$

Kako bismo izračunali vrijednost kuta $\gamma_1$, potrebno je izračunati koliki dio kružnice opsega $4l\pi$[1] prijeđemo u toj jednoj sekundi:

$$\gamma_1 = \frac{\frac{r\dot{\varphi}_1}{rad/s}}{4l\pi} 2\pi = \frac{r\dot{\varphi}_1}{2l}\frac{s}{rad}. \tag{11}$$

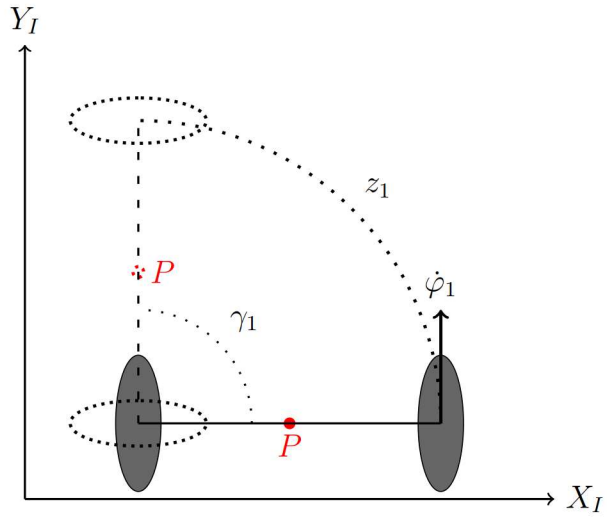
U ovom slučaju, kutna brzina materijalne točke $P$ iznosi:

$$\omega_1 = \frac{r\dot{\varphi}_1}{2l}. \tag{12}$$

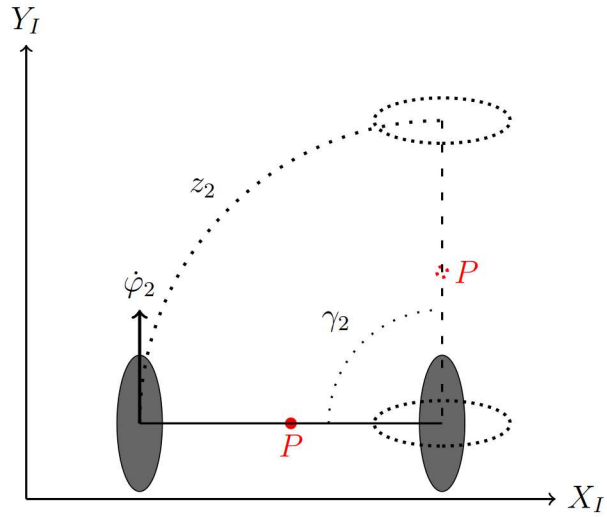
Slično, kada desni kotač miruje, a lijevi se giba (slika 4), dolazi do rotacije oko desnog kotača i imamo kutnu brzinu $\omega_2$:

$$\omega_2 = \frac{-r\dot{\varphi}_2}{2l}. \tag{13}$$

U ovom slučaju kutna brzina ima negativan predznak jer se robot okreće u smjeru kazaljke na satu.




Slika 3: Rotacija oko lijevog kotača.




Slika 4: Rotacija oko desnog kotača.

[1] Robot se rotira oko lijevog kotača i radi kružnicu polumjera $2l$

Sada kada smo izračunali $\dot{\xi}_R$, primjenom formule (2) dolazimo do promjene stanja robota u globalnom koordinatnom sustavu:

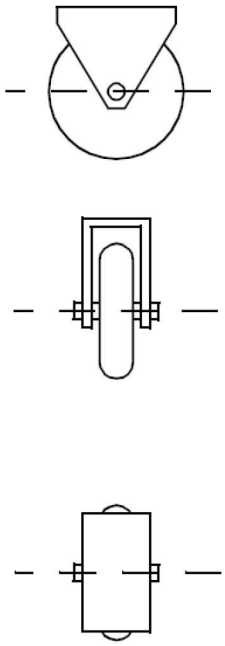
$$\dot{\xi}_I = R^T(\theta) \begin{bmatrix} \frac{r\dot{\varphi}_1}{2} + \frac{r\dot{\varphi}_2}{2} \\ 0 \\ \frac{r\dot{\varphi}_1}{2l} - \frac{r\dot{\varphi}_2}{2l} \end{bmatrix}. \tag{14}$$

Drugi način kojim rješavamo problem određivanja pozicije robota u prostoru svodi se na analizu ograničenja koja dolaze od svakog kotača mobilnog robota.

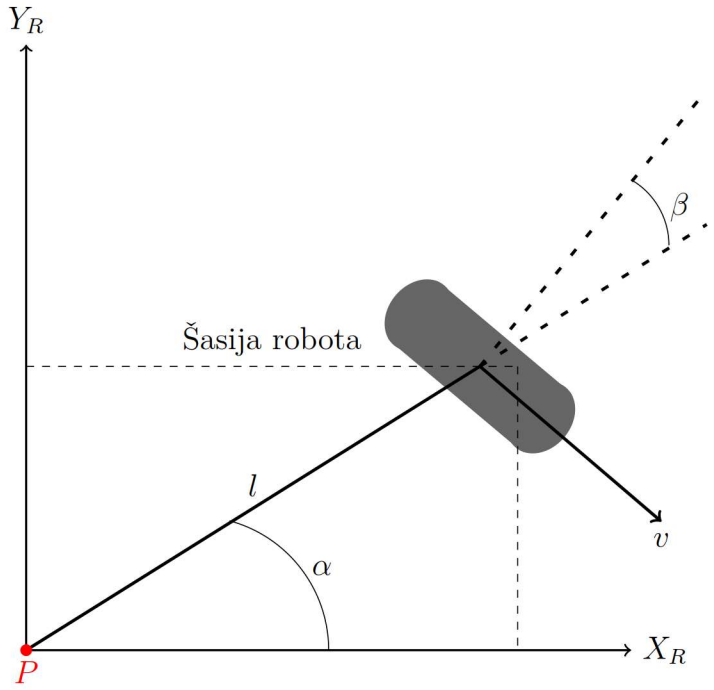
## 2.3 Standardni fiksni kotač

Postoje četiri osnovne vrsta kotača: standardni, okretni, švedski i sferni ili kuglasti. Svaki od ova četiri kotača ima određene prednosti i mane, te određena ograničenja. Pretpostavit ćemo da je kotač uvijek u dodiru s površinom po kojoj se kreće samo u jednoj točki. Također, pretpostavit ćemo da se kotač ne može kretati u smjeru paralelnom s osi rotacije (npr. na slici 2 kotači se ne smiju kretati u smjeru osi $Y_R$ ili $-Y_R$). Imajući na umu ove pretpostavke, iskazat ćemo uvjet vrtnje i uvjet neproklizavanja za standardni fiksni kotač.

U okviru ovog rada izrađen je mobilni robot koji za kretanje koristi dva standardna fiksna kotača. Standardni kotač vrti se oko nepomične osi rotacije. Kod takvog je kotača otklon od osi rotacije do osi skretanja jednak nula. Vrtnja je moguća u oba smjera, unaprijed i unazad.



Slika 5: Standardni kotač. Slika preuzeta iz [1].




Slika 6: Parametri standardnog fiksnog kotača.

Na slici 6 vidimo parametre jednog fiksnog standardnog kotača. Fiksni kut $\beta$ predstavlja otklon ravnine kotača do osovine robota. Vrijednost $l$ zajedno s kutom $\alpha$ jednoznačno određuje poziciju kotača u robotovom lokalnom koordinatnom sustavu s ishodištem u materijalnoj točki $P$. Još jedan bitan parametar, koji se ne vidi na slici 6, jest polumjer kotača $r$. Pozicija kotača s obzirom na njegovu horizontalnu os jest vrijednost iz intervala $[0, 2\pi)$ i označavamo ju s $\varphi$. Odnosno, kako ta pozicija ovisi o vremenu to je zapravo funkcija $\varphi(t)$.

Sljedećom jednadžbom iskazat ćemo uvjet vrtnje za standardni fiksni kotač:

$$\begin{bmatrix} sin(\alpha+\beta) & -cos(\alpha+\beta) & (-l)cos\beta \end{bmatrix} R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0. \tag{15}$$

Uvjet vrtnje nam govori da svo kretanje u smjeru ravnine kotača mora biti popraćeno odgovarajućom promjenom pozicije kotača $\dot{\varphi}(t)$.

Jednadžba za uvjet neproklizavanja za standardni fiksni kotač glasi:

$$\begin{bmatrix} cos(\alpha+\beta) & sin(\alpha+\beta) & lsin\beta \end{bmatrix} R(\theta)\dot{\xi}_I = 0. \tag{16}$$

Uvjet neproklizavanja nam govori da ne smije postojati kretanje kotača u smjeru ortogonalnom na ravninu kotača.

## 2.4 Inverzna kinematika

Inverzna kinematika bavi se problemom određivanja stanja kotača na osnovu poznavanja stanja robota. Do potrebnih formula doći ćemo koristeći uvjete vrtnje i uvjete neproklizavanja za fiksne standardne kotače, odnosno koristeći jednadžbe (15) i (16). Na kraju ćemo pokazati kako jednostavno, iz dobivenih formula za inverznu kinematiku, možemo doći do formula za direktnu kinematiku.

Kako bismo opisali kretanje robota, potrebno je razmotriti ograničenja koja svaki od kotača mobilnog robota postavlja na njegovo kretanje. U našem slučaju robota s diferencijalnim pogonom, imamo dva standardna fiksna kotača i jedan sferni kotač koji ne ograničava kretanje robota ni na koji način. Ograničenja koja dolaze od uvjeta vrtnje standardnih fiksnih kotača zapisat ćemo u matričnom zapisu. Prije nego što uvedemo matrični zapis, kuteve $\alpha$ i $\beta$ koji se odnose na desni kotač označavat ćemo indeksom 1, a kuteve koji se odnose na lijevi kotač indeksom 2. Definirajmo matrice $J_1$ i $J_2$ na sljedeći način:

$$J_1 = \begin{bmatrix} sin(\alpha_1 + \beta_1) & -cos(\alpha_1 + \beta_1) & (-l)cos\beta_1 \\ sin(\alpha_2 + \beta_2) & -cos(\alpha_2 + \beta_2) & (-l)cos\beta_2 \end{bmatrix} \tag{17}$$

$$J_2 = \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix}, \tag{18}$$

gdje je $r$ polumjer kotača (oba kotača imaju jednake polumjere). Pozicije kotača, s obzirom na njihove horizontalne osi, $\varphi_1(t)$ i $\varphi_2(t)$ stavit ćemo u matricu $\varphi(t)$:

$$\varphi(t) = \begin{bmatrix} \varphi_1(t) \\ \varphi_2(t) \end{bmatrix}. \tag{19}$$

Pomoću matrica (17), (18) i (19) možemo iskazati uvjet vrtnje (15) za oba kotača:

$$J_1 R(\theta)\dot{\xi}_I - J_2\dot{\varphi} = 0. \tag{20}$$

Na sličan način možemo matrično zapisati uvjet neproklizavanja za oba kotača. Definiramo matricu $C_1$:

$$C_1 = \begin{bmatrix} cos(\alpha_1 + \beta_1) & sin(\alpha_1 + \beta_1) & lsin\beta_1 \\ cos(\alpha_2 + \beta_2) & sin(\alpha_2 + \beta_2) & lsin\beta_2 \end{bmatrix}. \tag{21}$$

Pomoću matrice (21) možemo iskazati uvjet neproklizavanja (16) za oba kotača:

$$C_1 R(\theta)\dot{\xi}_I = 0. \tag{22}$$

Kada ujedinimo jednadžbe (20) i (22) u potpunosti smo opisali kretanje našeg robota s diferencijalnim pogonom:

$$\begin{bmatrix} J_1 \\ C_1 \end{bmatrix} R(\theta)\dot{\xi}_I = \begin{bmatrix} J_2\dot{\varphi} \\ \mathbf{0} \end{bmatrix}. \tag{23}$$

Prije nego što iskoristimo matrični zapis iz jednadžbe (23), potrebno je odrediti parametre $\alpha$ i $\beta$ za oba kotača. Pogledamo li sliku 2, lako vidimo da je vrijednost kuta $\alpha_1$ jednaka $\frac{-\pi}{2}$, dok je $\alpha_2 = \frac{\pi}{2}$. Vrijednosti kuteve $\beta$ u ovom su primjeru također lagane za izračunati. U slučaju kuta $\beta_2$, lijevi kotač nema otklona od osovine pa je $\beta_2 = 0$. Što se tiče kuta $\beta_1$, na prvi pogled možda bismo rekli kako i on iznosi 0, ali ako uzmemo u obzir smjer vrtnje motora, dolazimo do zaključka da je desni kotač okrenut za $\pi$ radijana u odnosu na osovinu. Uvrštavajući navedene vrijednosti u (23) dobivamo:

$$\begin{bmatrix} sin(\frac{-\pi}{2} + \pi) & -cos(\frac{-\pi}{2} + \pi) & (-l)cos\pi \\ sin(\frac{\pi}{2} + 0) & -cos(\frac{\pi}{2} + 0) & (-l)cos0 \\ cos(\frac{-\pi}{2} + \pi) & sin(\frac{-\pi}{2} + \pi) & lsin\pi \\ cos(\frac{\pi}{2} + 0) & sin(\frac{\pi}{2} + 0) & lsin0 \end{bmatrix} R(\theta)\dot{\xi}_I = \begin{bmatrix} \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix} \begin{bmatrix} \dot{\varphi}_1 \\ \dot{\varphi}_2 \end{bmatrix} \\ 0 \\ 0 \end{bmatrix}. \tag{24}$$

Kada pojednostavimo gornji izraz dobivamo:

$$\begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} R(\theta)\dot{\xi}_I = \begin{bmatrix} r\dot{\varphi}_1 \\ r\dot{\varphi}_2 \\ 0 \\ 0 \end{bmatrix}. \tag{25}$$

Budući da su lijevi i desni kotač našeg robota paralelni, uvjeti neproklizavanja za oba kotača su jednaki pa gornji izraz možemo dodatno pojednostaviti:

$$\begin{bmatrix} r\dot{\varphi}_1 \\ r\dot{\varphi}_2 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{bmatrix} R(\theta)\dot{\xi}_I. \tag{26}$$

Iz jednadžbe (26) dobivamo rješenje problema inverzne kinematike za našeg mobilnog robota:

$$\dot{\varphi}_1 = \frac{1}{r}\begin{bmatrix} 1 & 0 & l \end{bmatrix} R(\theta)\dot{\xi}_I, \tag{27}$$

$$\dot{\varphi}_2 = \frac{1}{r}\begin{bmatrix} 1 & 0 & -l \end{bmatrix} R(\theta)\dot{\xi}_I. \tag{28}$$

Iz jednadžbe (26) također dobivamo rješenje problema direktne kinematike, tj. dobivamo promjenu stanja robota iz promjene stanja kotača:

$$\dot{\xi}_I = R^T(\theta) \begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} r\dot{\varphi}_1 \\ r\dot{\varphi}_2 \\ 0 \end{bmatrix}. \tag{29}$$

Inverz matrice $M := \begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{bmatrix}$ možemo računati prema formuli: $M^{-1} = \frac{1}{\det M}(cofM)^T$:

$$\begin{vmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{vmatrix} = -1(-l-l) = 2l, \quad cofM = \begin{bmatrix} l & 0 & 1 \\ l & 0 & -1 \\ 0 & 2l & 0 \end{bmatrix}$$

$$M^{-1} = \frac{1}{2l}\begin{bmatrix} l & l & 0 \\ 0 & 0 & 2l \\ 1 & -1 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \\ \frac{1}{2l} & \frac{-1}{2l} & 0 \end{bmatrix}.$$

Dobiveni inverz uvrstit ćemo u jednadžbu (29):

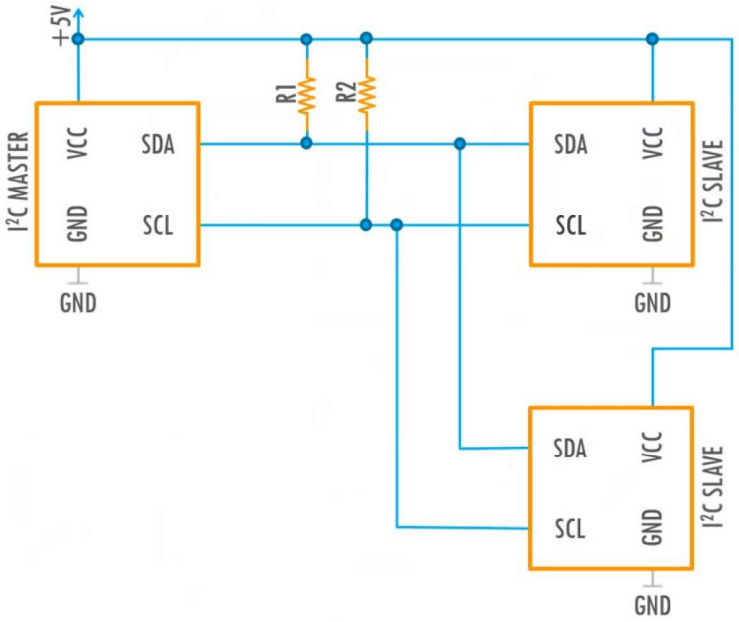
$$\dot{\xi}_I = R^T(\theta) \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \\ \frac{1}{2l} & \frac{-1}{2l} & 0 \end{bmatrix} \begin{bmatrix} r\dot{\varphi}_1 \\ r\dot{\varphi}_2 \\ 0 \end{bmatrix} = R^T(\theta) \begin{bmatrix} \frac{r\dot{\varphi}_1}{2} + \frac{r\dot{\varphi}_2}{2} \\ 0 \\ \frac{r\dot{\varphi}_1}{2l} - \frac{r\dot{\varphi}_2}{2l} \end{bmatrix}. \tag{30}$$

Ako dobiveni rezultat usporedimo s rezultatom (14) vidimo da su oni identični.

# 3 $I^2C$

$I^2C$ (Inter-Integrated Circuit) protokol izumljen je od strane tvrtke Philips osamdesetih godina prošlog stoljeća. Ovaj protokol osmislili su za potrebe komunikacije između različitih elektroničkih komponenata. U ovom završnom radu, mi ga koristimo za komunikaciju između Raspberry Pi računala i mikrokontrolera.

## 3.1 Arhitektura




Slika 7: $I^2C$ arhitektura. Slika preuzeta iz nastavnih materijala kolegija Ugrađeni sustavi.

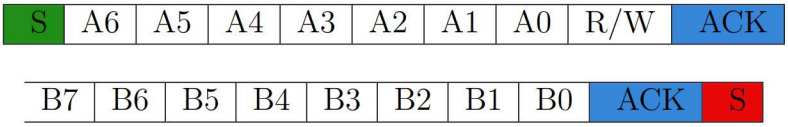
$I^2C$ za komunikaciju koristi dva kanala, SDA kanal i SCL kanal. SDA kanal je dvosmjeran i služi za prijenos podataka. Kada kažemo da je kanal dvosmjeran, to znači da komunikacija može teći u oba smjera. SCL kanal je jednosmjeran kanal i on služi za upravljanje frekvencijom prijenosa podataka. Half-duplex komunikacija omogućuje prijenos podataka u oba smjera, ali ne u isto vrijeme (prvo komunicira jedna strana pa onda druga).

Na slici 7 imamo jedan *master* uređaj i dva *slave* uređaja. U $I^2C$ komunikaciji općenito može biti više *master* i više *slave* uređaja. Svaki uređaj ima svoju jedinstvenu adresu. Najveći mogući broj uređaja u ovoj komunikaciji ograničen je na 128 uređaja (ukoliko se radi o 7-bitnoj adresi) ili 1024 uređaja (ukoliko se radi o 10-bitnoj adresi). *Master* uređaji upravljaju komunikacijom. U istom trenutku samo jedan *master* može upravljati komunikacijom na sabirnici (ostali se *master* uređaji tada ponašaju kao *slave* uređaji). *Slave* uređaji odgovaraju na zahtjeve *master* uređaja koji trenutno upravlja komunikacijom. *Master* uređaj može poslati ili zatražiti podatak od bilo kojeg *slave* uređaja.

Na gornjoj slici vidimo kako su svi uređaji spojeni na dvije iste sabirnice, SDA i SCL. Ovo je jedna od prednosti $I^2C$ protokola jer za komunikaciju možemo koristiti samo dvije žice, što je vrlo ekonomično. SDA i SCL kanali spojeni su preko pull-up otpornika $R1$ i $R2$ na 5V. Zbog takve konfiguracije, sve dok ne šaljemo nikakve podatke, napon SDA i SCL linije bit će 5V.

## 3.2 Prijenos podataka

Putem $I^2C$ protokola šaljemo poruke. Svaka poruka sadrži: početni uvjet, adresu, podatkovne okvire, R/W bitove, ACK bitove i krajnji uvjet.



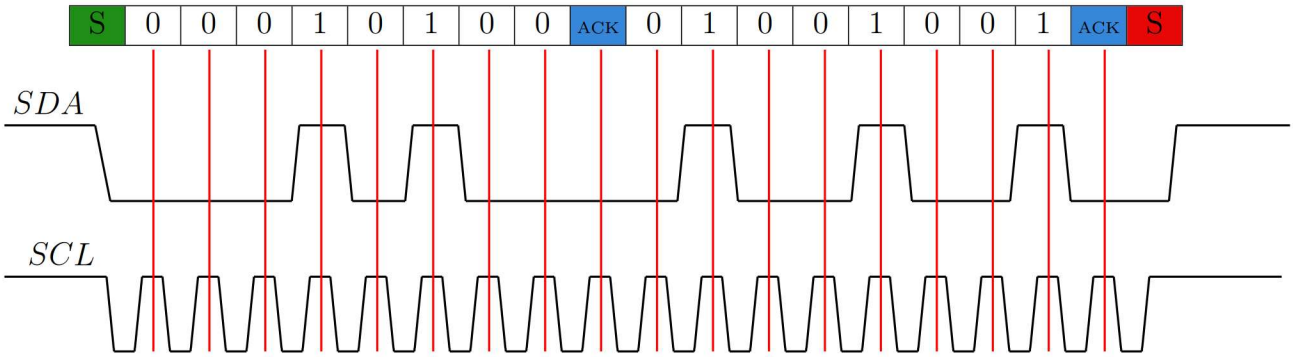

Slika 8: Poruka koja se šalje $I^2C$ protokolom.

Na slici 8 zelenom bojom označen je početni uvjet. Početni je uvjet prelazak SDA linije iz visokog u nisko stanje, prije nego što se SCL linija prebaci iz visokog u nisko stanje. *Slave* uređaji prepoznaju ovaj događaj. Nakon toga slijedi adresa onog *slave* uređaja s kojim *master* želi komunicirati. Sljedećih 7 bitova (A6-A0) čine adresu uređaja s kojim *master* inicira komunikaciju. *Slave* uređaji čija adresa ne odgovara prethodno emitiranoj adresi ignoriraju sljedeće bitove i čekaju krajnji uvjet. Iza adrese slijedi R/W bit koji uređaju na željenoj adresi govori želi li *master* da on čita ili piše po sabirnici (hoće li *slave* uređaj primiti ili slati određeni podatak). Ukoliko *master* želi da *slave* uređaj čita bitove sa sabirnice, R/W bit bit će nula, u suprotnom će R/W bit biti jedan.

Adresa i podaci šalju se u podatkovnim okvirima. Podatkovni okviri su duljine jednog bajta. ACK bitovi slijede nakon svakog uspješno, odnosno neuspješno primljenog podatkovnog okvira. ACK bit može slati *slave*, ali i *master*, što ovisi o R/W bitu. Ukoliko *master* piše podatke na sabirnicu, a *slave* ih čita (R/W bit je nula) tada ACK bitove šalje *slave* kako bi dao *master*-u informaciju je li uspješno primio podatkovni okvir. Ukoliko *master* traži podatak od *slave* uređaja (R/W bit je jednak 1), tada *master* postavlja vrijednost ACK bita nakon uspješno, odnosno neuspješno primljenog podatkovnog okvira. Kada je podatkovni okvir uspješno primljen ACK bit postavlja se na nulu, u suprotnom ACK bit postavlja se na jedinicu. Na ovaj način odmah znamo je li prijenos bio uspješan ili ga možda treba ponoviti ili na neki drugi primjereni način reagirati.

Krajnji uvjet je kada se SDA linija prebaci iz niskog u visoko stanje nakon što se SCL linija prebaci iz niskog u visoko stanje. Taj događaj *slave* uređajima naznačuje da je trenutna komunikacija na sabirnici završila.

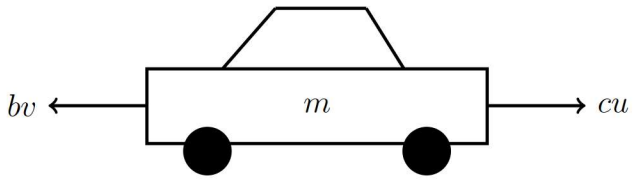
**Primjer 21**
*Master* šalje (R/W = 0) *slave* uređaju na adresi 10 (0001010) podatak 73 (01001001):



# 4 PID regulator

Nakon što iskoristimo kinematiku kako bismo izračunali potrebne kutne brzine kotača te nakon što ih pošaljemo prema mikrokontroleru putem $I^2C$ protokola, potrebno je nekako reći elektromotorima da se okreću izračunatim kutnim brzinama. PID regulator (Proportional Integral Derivative) je sustav regulacije koji se koristi za reguliranje različitih parametara poput temperature, tlaka, brzine itd. Ovaj regulator radi na način da kontinuirano računa pogrešku (razliku između stvarne i željene vrijednosti) i onda pokušava smanjiti tu pogrešku. U ovome radu, PID regulator koristimo kako bismo okretali elektromotore željenim kutnim brzinama. U sljedećem primjeru koristit ćemo PID regulator kako bismo "vozili" automobil željenom brzinom.

**Primjer 41**
*Imamo automobil mase* $\boldsymbol{m}$ *i želimo da se on kreće brzinom* $\boldsymbol{r}$*. Trenutna brzina automobila označena je s* $\boldsymbol{v}$*. Cilj nam je minimizirati izraz* $|r-v| \to min$*. Slovom* $\boldsymbol{u}$ *označena je kontrola (ubrzavanje ili kočenje). Koeficijent elektro-mehaničke transmisije označavamo slovom* $\boldsymbol{c}$*. Prilikom kretanja automobila nailazimo na otpor zraka. Otpor zraka je proporcionalan brzini i djeluje u smjeru obrnutom od smjera gibanja. Taj koeficijent proporcionalnosti označavamo s* $\boldsymbol{b}$*.*




*Prema II. Newtonowom zakonu, sila je jednaka umnošku mase i akceleracije (F=ma). Ukupnu silu dobivamo kao zbroj svih sila koje djeluju na automobil:*

$$m\dot{v} = cu - bv \tag{31}$$

*tj.*

$$\dot{v} = \frac{-b}{m}v + \frac{c}{m}u. \tag{32}$$

*Kako vrijeme prolazi, želimo da brzina* $\boldsymbol{v}$ *teži referentnoj brzini* $\boldsymbol{r}$*.*

*P slučaj:*
*Želimo dizajnirati P-regulator na način da je kontrola* $\boldsymbol{u}$ *proporcionalna pogrešci* $\boldsymbol{e=r-v}$*, tj. za neki* $k \in \mathbb{R}_+$ *želimo da vrijedi* $\boldsymbol{u=ke}$*. Ako pogledamo slučaj kada je postignuta referentna brzina* $\boldsymbol{r}$*, tada želimo da akceleracija* $\dot{v}$ *bude jednaka nuli. Uvrstimo taj rezultat u jednadžbu (32):*

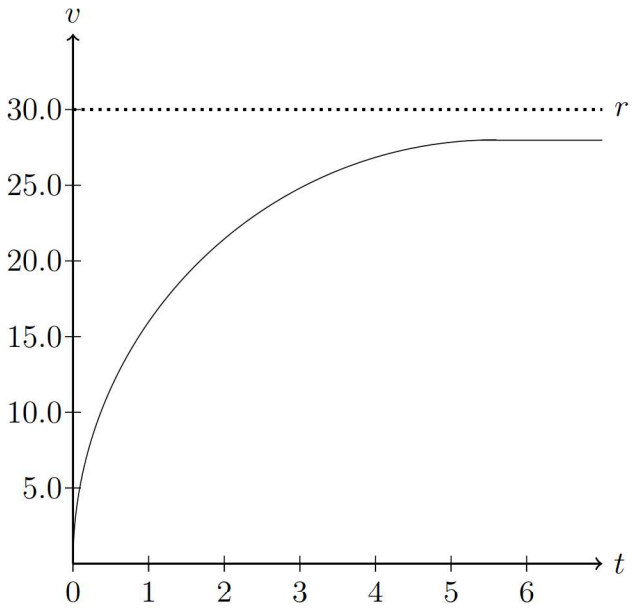
$$0 = \frac{-b}{m}v + \frac{c}{m}k(r-v) \Big/ * m$$

$$(b+ck)v = ckr \Big/ : (b+ck)$$

$$v = \frac{ck}{ck+b}r, \quad c,k,b > 0. \tag{33}$$

*Budući da je razlomak na desnoj strani jednadžbe (33) uvijek manji od jedan, referentna brzina se ne postiže.*

*PI slučaj:*




Slika 9: Graf promjene brzine uz korištenje P-regulatora.

*Ideja PI-regulatora jest akumulirati pogrešku koju predstavlja površina između pravca* $\boldsymbol{y=r}$ *i grafa funkcije koji opisuje kretanje brzine u ovisnosti o vremenu* $\boldsymbol{t}$*. Ako iskoristimo tu akumuliranu vrijednost na sljedeći način:*

$$u(t) = k_P e(t) + k_I \int_0^t e(\tau)\, d\tau, \tag{34}$$

*gdje* $k_P$ *predstavlja proporcionalnu konstantu, a* $k_I$ *integralnu konstantu, uspjet ćemo postići referentnu brzinu, ali uz moguće oscilacije oko referentne brzine* $\boldsymbol{r}$*. Spomenute oscilacije "izglađujemo" uzimajući u obzir derivaciju pogreške. Time dolazimo do PID regulatora.*

*PID slučaj:*
*Dodavanjem derivacije pogreške u izračun potrebnog upravljanja* $\boldsymbol{u}$ *u trenutku* $\boldsymbol{t}$ *postižemo visok utjecaj na responzivnost upravljanja. Sada imamo sljedeću jednadžbu:*

$$u(t) = k_P e(t) + k_I \int_0^t e(\tau)\, d\tau + k_D \frac{de(t)}{dt}. \tag{35}$$

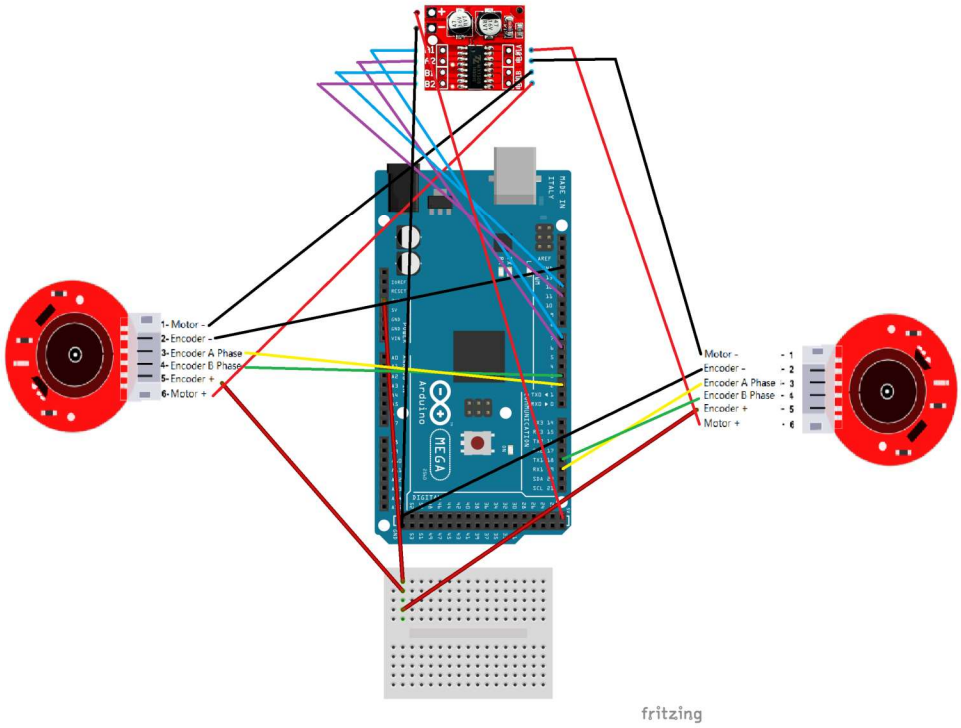
Jednadžbu (35) koristit ćemo na mikrokontroleru za postizanje željene kutne brzine motora. Određivanje idealnih konstanti $k_P$, $k_I$ i $k_D$ zaseban je problem koji rješavamo "Trial and error" metodom. U toj metodi uvrštavamo različite vrijednosti za spomenute konstante i pratimo ponašanje sustava. $k_P$ konstanta doprinosi stabilnosti sustava, dok $k_I$ konstanta doprinosi robusnosti susatava i što bržem postizanju referentne brzine. $k_D$ konstanta doprinosi smanjenju oscilacija oko referentne brzine. "D" dio PID-regulatora je dosta osjetljiv na šumove pa se u pravilu $k_D$ konstanta ne postavlja na preveliku vrijednost.

# 5 Implementacija

Izvorni programski kod može se vidjeti u cijelosti na GitLab repozitoriju[2].

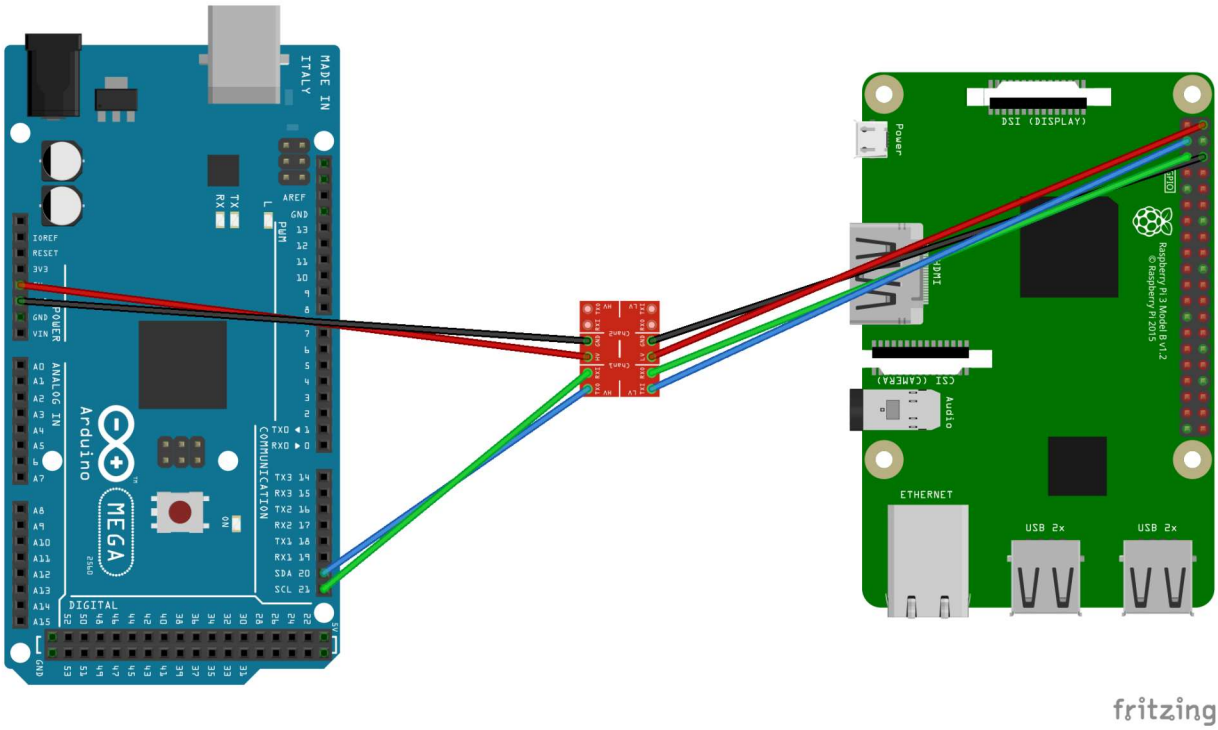
## 5.1 Spajanje dijelova robota

Kako bismo uspješno upravljali robotom potrebno je prvo točno spojiti sve njegove dijelove. Na sljedećim slikama uz pomoć alata Fritzing[3] napravljena je shema koja prikazuje kako pravilno spojiti elektromotore na Arduino Mega 2560 i kako spojiti Raspberry Pi računalo i Arduino Mega 2560 za uspješnu $I^2C$ komunikaciju. Kod spajanja motora, koristimo H-bridge kako bismo bez prekopčavanja motora mogli okretati oba motora u smjeru kazaljke na satu i u obrnutom smjeru od kazaljke na satu. Kod spajanja Raspberry Pi računala i Arduina koristimo Logic Level Converter budući da Raspberry Pi radi na 3.3V, a Arduino na 5V.




Slika 10: Spajanje motora na Arduino Mega 2560.

[2]https://gitlab.com/rerceg/zavrsni
[3]https://fritzing.org/




Slika 11: Master-Slave $I^2C$.

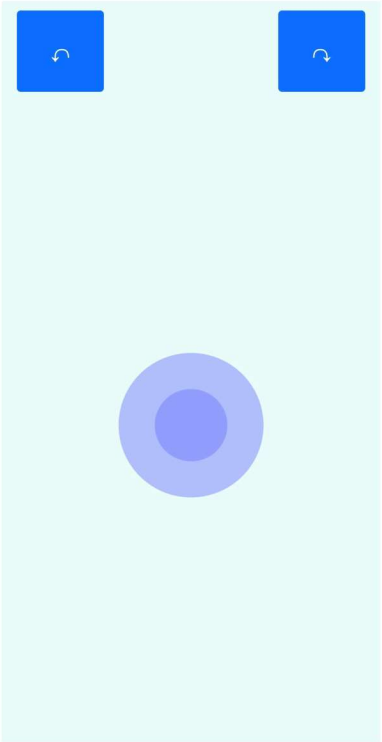
## 5.2 Klijent

Sljedeći kod (u kombinaciji s CSS datotekama koje smatram da nema potrebe posebno pojašnjavati) generira jednostavnu aplikaciju koju možete vidjeti na slici 12

### 5.2.1 index.html

```
<body>
  <div id="zone_joystick">
    <div class="rotateButtons d-flex p-3">
      <button class="btn btn-primary col-3" ontouchstart="inPlaceRotate(1)
    "
        ontouchend="inPlaceRotate(0)">
        <span style='font-size:30px;'>&#8630;</span>
      </button>
      <button class="btn btn-primary col-3 offset-6"
        ontouchstart="inPlaceRotate(2)" ontouchend="inPlaceRotate(0)">
        <span style='font-size:30px;'>&#8631;</span>
      </button>
    </div>
    <div class="zone"></div>
  </div>
</body>
```

Za upravljač vidljiv na slici 12 dovoljno je u HTML datoteci napraviti jedan  ID-a 'zone_joystick' koji sadrži prazan  klase 'zone', a ostatak posla obavlja skripta nipplejs.js i naša skripta main.js. Ostatak koda služi za generiranje dva gumba također vidljiva na istoj fotografiji. Oni služe za okretanje robota u mjestu (↺ - okretanje u smjeru kazaljke na satu i ↻ - okretanje obrnuto od smjera kazaljke na satu). Gumbi okreću robota sve dok su pritisnuti, a to postižemo korištenjem 'ontouchstart' i 'ontouchend'.



Slika 12: Upravljač robota.

Za izgled gumba (veličina, boja) koristimo Bootstrapove klase 'btn', 'btn-primary', dok za poziciju gumbova koristimo grid sustav pomoću klasa 'col-3' i 'offset-3'.

### 5.2.2 main.js

```
var joystickConf = {
      zone: document.querySelector('.zone'),
      mode: 'static',
      position: {
        left: '50%',
        top: '50%'
      },
      color: 'blue',
      size: 150
    };

var y_sredina = 0;
var joystick = nipplejs.create(joystickConf);

joystick.on('move end', function(evt, data) {
  move(data);
}).on('start', function(evt, data){
  if(y_sredina === 0) y_sredina = data.position.y;
});
```

Kako bismo kreirali naš upravljač potrebno je definirati željene specifikacije. Objekt *joystickConf* sadrži našu željenu konfiguraciju, poput željene pozicije, boje i veličine upravljača te definiramo način rada 'static'. Osim načina rada 'static', u kojem upravljač stoji

fiksiran na jednom mjestu i uvijek je vidljiv, postoje još i 'dynamic' i 'semi' načini rada.

Upravljač kreiramo naredbom *nipplejs.create(joystickConf)* te spremamo referencu na taj upravljač u varijablu *joystick*. Funkcionalnost upravljača određujemo tako da kažemo na koje događaje želimo da naš upravljač bude osjetljiv. Mogući događaji su:

- 'move' - upravljač je pomaknut
- 'start' - dotaknuli smo zonu upravljača
- 'end' - prestali smo dirati zonu upravljača

Kada pomaknemo upravljač ili ga pustimo (pa se on sam vrati na početnu poziciju) pozivamo funkciju *move()* i prosljeđujemo joj podatke o upravljaču. Podaci koje možemo dobiti od upravljača su: pozicija, sila, snaga pritiska, udaljenost, kut, pravac, trenutni položaj u ravnini. Prvi puta kada pomaknemo upravljač postavljamo vrijednost varijable *y_sredina*. Varijabla *y_sredina* nam služi kako bi naše y vrijednosti u koordinatnom sustavu bile u nekom segmentu [-A,A].

```
1  const address = "http://192.168.1.4:3000/";
2
3  function inPlaceRotate(id){
4    (async () => {
5        const rawResponse = await fetch(address + id, {
6          method: 'POST'
7        });
8
9        const content = await rawResponse.status;
10
11     })();
12 }
13
14 function move(obj) {
15   var {position, angle} = obj;
16   var y = -position.y + y_sredina;
17   var joystickData =  {"position" : {"x" : position.x, "y": y}, angle};
18
19   (async () => {
20     const rawResponse = await fetch(address, {
21       method: 'POST',
22       headers: {
23         'Accept': 'application/json',
24         'Content-Type': 'application/json'
25       },
26       body: JSON.stringify(joystickData)
27     });
28
29     const content = await rawResponse.status;
30
31   })();
32 }
```

Konstanta *address* sadrži lokalnu IP adresu Raspberry Pi računala. Ta adresa nam je potrebna kako bismo slali HTTP zahtjeve prema poslužitelju na Raspberry Pi računalu. Funkcija *inPlaceRotate(id)* poziva se pritiskom, odnosno otpuštanjem bilo kojeg od dva

gumba koja koristimo za rotaciju robota u mjestu. Ta funkcija šalje HTTP POST zahtjev prema poslužitelju. Vrijednost varijable *id* može biti 0,1 ili 2. Ova funkcija radi jednako neovisno o vrijednosti varijable *id*, ali na poslužitelju za različite vrijednosti varijable *id* radimo različite stvari. Za $id = 0$ zaustavljamo kretanje robota. Za $id = 1$ rotiramo robota oko svoje osi obrnuto od smjera kazaljke na satu. Za $id = 2$ rotiramo robota oko svoje osi u smjeru kazaljke na satu.

Funkcija *move(obj)* poziva se prilikom pomaka upravljača i prilikom otpuštanja upravljača. Od svih podataka koje dobijemo od upravljača u varijabli *obj* nama su potrebni jedino *position* i *angle*. Kako bismo za *y* vrijednost dobili brojeve iz segmenta [-A,A], umjesto *position.y* uzimamo vrijednost $(-position.y + y\_sredina)$ i stvaramo objekt *joystickData* koji ćemo poslati prema poslužitelju. Poslužitelju šaljemo HTTP POST zahtjev s objektom *joystickData* u JSON formatu.

## 5.3 Poslužitelj

```
1  const i2c = require('i2c-bus');
2  const address = 0xA;
3  const reset = Buffer.from([10]);
4
5  var send = 1;
6
7  function sendToMega(wbuf) {
8    i2c.openPromisified(1).
9    then(i2c1 => i2c1.i2cWrite(address, wbuf.length, wbuf).
10   then(_ => {i2c1.close(); send = 1;})).catch((error) => {
11     send = 0
12     setTimeout(sendToMega, 100, reset);
13   });
14 }
15
16 function znamenke(broj) {
17   let array = [1,1,1,1];
18   if(broj < 0) {
19     if(broj > -10) array[3] = -1;
20     else if(broj > -100) array[2] = -1;
21     else if(broj > -1000) array[1] = -1;
22     else array[0] = -1;
23     broj *= -1;
24   }
25   for (var i = 3; i >= 0; i--) {
26     array[i] *= broj%10;
27     broj = Math.floor(broj/10);
28   }
29   return array;
30 }
```

Kao što smo već spomenuli, $I^2C$ protokol koristimo za komunikaciju između Raspberry Pi računala i našeg Arduino Mega 2560 mikrokontrolera. U našem slučaju, adresa za Arduino Mega 2560 je postavljena na 10 (0xA - heksadecimalni zapis broja 10). Raspberry Pi računalo funkcionira kao *master*, a Arduino kao *slave*. To znači da Raspberry Pi računalo upravlja komunikacijom između ova dva uređaja. U našem slučaju imamo samo jedan *slave* uređaj, ali svejedno ga najprije moramo odabrati slanjem njegove adrese (0xA).

Izračunate brzine okretaja motora trebamo poslati prema mikrokontroleru. Te brzine, u našem slučaju, mogu biti najviše četveroznamenkasti cijeli brojevi. Odlučio sam ih slati znamenku po znamenku. Funkcija *znamenke*(*broj*) prima cijeli broj i vraća listu znamenki toga broja. Zbog programiranja mikrokontrolera, brzina svakog motora bit će reprezentirana s točno četiri znamenke (700 - [0,7,0,0], 0 - [0,0,0,0], -1500 - [-1,5,0,0]). Varijablu *send* koristimo kako bismo spriječili pokušaj slanja novih podataka prije nego što se pošalju trenutni podaci (na taj način izbjegavamo greške u slanju i primanju podataka).

*Buffer* je JavaScript klasa koja se koristi za reprezentaciju konačnog niza bajtova. Funkcija *sendToMega*(*wbuf*) prima JavaScript *Buffer* fiksne duljine te ga šalje putem $I^2C$ protokola na mikrokontroler adrese *address*, bajt po bajt. Nakon uspješno poslane poruke, zatvaramo komunikaciju i postavljamo vrijednost varijable *send* na 1 (kasnije ćemo vidjeti gdje smo tu vrijednost postavili na 0). Ukoliko se dogodila pogreška prilikom slanja tada, budući da ne znamo koliko bajtova (ukoliko i jedan) je uspješno preneseno na mikrokontroler, šaljemo *reset Buffer*. *reset Buffer* sadrži samo jedan bajt vrijednosti 10. Kasnije ćemo vidjeti da je mikrokontroler isprogramiran da kada primi vrijednost 10 zna da se dogodila pogreška i tada resetira vrijednosti odgovarajućih varijabli.

```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3
4  const app = express();
5  app.use(bodyParser.json());
6  app.use(function(req, res, next) {
7    res.header("Access-Control-Allow-Origin", "*");
8    res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With,
       Content-Type, Accept");
9    next();
10 });
11
12 const route = '/'
```

*Express* je jednostavni minimalistički web okvir (eng. web framework) koji koristimo za izradu našeg poslužitelja. *body* − *parser* koristimo kako bismo parsirali tijela dolazećih HTTP zahtjeva i na jednostavan način došli do primljenih podataka poslanih od strane upravljača na pametnom telefonu. U zaglavlju omogućujemo Cross-Origin Resource Sharing (CORS) kako bismo mogli primati HTTP zahtjeve s klijentske domene.

```
1  const stop = Buffer.from([0,0,0,0,0,0,0,0]);
2
3  app.post(route, (req, res) => {
4    const body = req.body;
5    if(send){
6      let lijevi = 0;
7      let desni = 0;
8      if(typeof(body.angle) !== 'undefined') {
9        var angle = 0;
10
11       if(body.angle.radian <= Math.PI) angle = body.angle.radian - Math.PI
       /2;
12       else angle = body.angle.radian - 3*Math.PI/2;
13
14       desni = Math.floor(1200*(body.position.y+angle*8.5)/(6.5*Math.PI));
15       lijevi = Math.floor(1200*(body.position.y-angle*8.5)/(6.5*Math.PI));
16     }
```

```
17      lijevi_znamenke = znamenke(lijevi);
18      desni_znamenke = znamenke(desni);
19      const wbuf = Buffer.from(desni_znamenke.concat(lijevi_znamenke));
20      send = 0;
21      sendToMega(wbuf)
22    } else {
23      if (typeof(body.angle) === 'undefined') setTimeout(sendToMega, 100,
       stop);
24    }
25    res.sendStatus(200);
26  });
```

Kada se pošalje HTTP POST zahtjev na adresu "http://192.168.1.4:3000/" poziva se gornja funkcija. Podatke dobivene od upravljača spremamo u konstantu *body*. Ukoliko trenutno ne šaljemo nikakve podatke prema mikrokontroleru ($send \neq 0$) potrebno je iz linearne i kutne brzine dobivene iz upravljača izračunati brzine okretaja lijevog i desnog motora.

Ako je $body.angle == "undefined"$ to znači da se upravljač trenutno nalazi u ishodištu i želimo da se robot ne kreće. U tom slučaju brzine okretaja lijevog i desnog motora ostaju 0. Ukoliko $body.angle \neq "undefined"$ tada je potrebno izračunati brzine motora. Upravljač sam odlučio promatrati kao gornju i donju polovicu pa su kutne brzine u rasponu $[-\pi/2, \pi/2]$ i spremamo ih u varijablu *angle*. Ako s $r$ označimo polumjer kotača našeg robota, a s $l$ polovicu udaljenosti između dva kotača robota tada brzinu okretaja dobivamo iz jednadžbi (27) i (28):

$$\dot{\varphi}_1 = (\dot{y} + l\dot{\theta})/r$$

$$\dot{\varphi}_2 = (\dot{y} - l\dot{\theta})/r,$$

gdje je $\dot{\varphi}_1$ brzina desnog motora, $\dot{\varphi}_2$ brzina lijevog motora, $\dot{y}$ linearna brzina, a $\dot{\theta}$ kutna brzina. Kako bismo dobili vrijednosti koje naš mikrokontroler koristi, potrebno je dobivene vrijednosti podijeliti s $2\pi$ te pomnožiti s 1200 (rezolucija enkodera motora). Brzine zaokružujemo na cijele brojeve jer naš mikrokontroler nema *floating point* jedinicu.

Dalje, dobivene brzine rastavljamo na znamenke već opisanom funkcijom *znamenke*(), te od njih stvaramo $Buffer\ wbuf$. Vrijednost varijable *send* postavljamo na 0, te $Buffer\ wbuf$ prosljeđujemo funkciji *sendToMega*(). Ukoliko je na početku izvršavanja gornje funkcije vrijednost varijable $send == 0$, tada provjeravamo je li upravljač u ishodištu ($body.angle == "undefined"$). Ukoliko je upravljač u ishodištu, tu vrijednost ne možemo zanemariti jer želimo da se robot ne kreće (ako ju zanemarimo robot će se nastaviti kretati iako smo mi željeli da se zaustavi). Iz tog razloga, sačekat ćemo 100 milisekundi kako bi se prijenos podataka na mikrokontroler završio i onda ćemo funkciji *sendToMega*() proslijediti $stop\ Buffer$ čija je vrijednost $[0, 0, 0, 0, 0, 0, 0, 0]$ kako bismo zaustavili robota. Ukoliko upravljač nije u ishodištu, nema potrebe da naknadno šaljemo željene brzine, budući da je zbog osjetljivosti upravljača razlika između dvije uzastopne poslane brzine zanemariva. Odnosno, upravljač toliko često šalje podatke, da ako nekoliko njih zanemarimo ništa se neće promijeniti (osim ako je to podatak za zaustavljanje robota).

```
1  app.post(route + ":id", (req, res) => {
2    if(send){
3      const id = req.params.id
4      var lijevi_znamenke;
```

```
5       var desni_znamenke;
6       if(id == 2){
7         lijevi_znamenke = [1,8,0,0];
8         desni_znamenke = [-1,8,0,0];
9       } else if(id == 1){
10        lijevi_znamenke = [-1,8,0,0];
11        desni_znamenke = [1,8,0,0];
12      } else if(id == 0){
13        lijevi_znamenke = [0,0,0,0];
14        desni_znamenke = [0,0,0,0];
15      }
16      const wbuf = Buffer.from(desni_znamenke.concat(lijevi_znamenke));
17      send = 0;
18      sendToMega(wbuf);
19    }
20    res.sendStatus(200);
21  });
22
23  app.listen(3000, () => console.log('Listening on port 3000'));
```

Kada se pošalje HTTP POST zahtjev na adresu "http://192.168.1.4:3000/:id" poziva se gornja funkcija. Vrijednost parametra *id* poslanog kao URI parametar HTTP POST zahtjeva spremamo u konstantu *id*. Ukoliko je $id == 2$, postavljamo brzinu lijevog motora na 1800, a desnog na -1800 i na taj način okrećemo robota u smjeru kazaljke na satu. Slično kada je $id == 1$, okrećemo robota obrnuto od smjera kazaljke na satu, a u slučaju $id == 0$ zaustavljamo okretanje robota. Pozivanjem metode *app.listen* pokrećemo naš poslužitelj.

## 5.4 Mikrokontroler

### 5.4.1 counter.h

```
1  #ifndef COUNTER_H_
2  #define COUNTER_H_
3
4  #include <avr/io.h>
5  #include <avr/interrupt.h>
6
7  void counter_init() {
8    TCCR0B |= (1 << CS01) | (1 << CS00);
9    TIMSK0 |= (1 << OCIE0A);
10
11   TCCR0A |= (1 << WGM01);
12   TCCR0A |= (1 << COM0A1);
13
14   OCR0A = 250;
15   sei();
16 }
17
18 #endif
```

Implementiramo brojač koji će pozivati prekid svaku milisekundu. Kako bismo dobili jednu milisekundu, s obzirom na baznu frekvenciju mikrokontrolera, potrebno je postaviti na jedinicu bitove CS01 i CS00 u registru TCCR0B (dijelimo baznu frekvenciju mikrokontrolera s 64) te postaviti vrijednost OCR0A registra na 250. Kako bismo omogućili prekide za Counter 0 postavljamo bit OCIE0A u registru TIMSK0 na jedinicu. Kada vrijednost brojača

bude jednaka vrijednosti u registru OCR0A želimo pozvati prekid pa postavljamo CTC način rada postavljanjem bitova WGM01 i COM0A1 na jedinicu u registru TCCR0A. Poziv funkcije *sei*() postavlja I bit u SREG registru na jedinicu i time globalno omogućuje prekide.

### 5.4.2 encoder.h

```
#ifndef ENCODER_H_
#define ENCODER_H_

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

volatile uint8_t A1, B1, A2, B2;
volatile int32_t ticks1 = 0;
volatile int32_t ticks2 = 0;
volatile int32_t ticks_old1 = 0;
volatile int32_t ticks_old2 = 0;

ISR(INT4_vect) {
  A1 != B1 ? --ticks1 : ++ticks1;
  A1 = PINE & (1 << PINE4) ? 1 : 0;
}

ISR(INT5_vect) {
  B1 = PINE & (1 << PINE5) ? 1 : 0;
  A1 != B1 ? --ticks1 : ++ticks1;
}

ISR(INT2_vect) {
  A2 != B2 ? ++ticks2 : --ticks2;
  A2 = PIND & (1 << PIND2) ? 1 : 0;
}

ISR(INT3_vect) {
  B2 = PIND & (1 << PIND3) ? 1 : 0;
  A2 != B2 ? ++ticks2 : --ticks2;
}

void encoder_init() {
  DDRE &= ~(1 << DDE4);
  DDRE &= ~(1 << DDE5);
  DDRD &= ~(1 << DDD2);
  DDRD &= ~(1 << DDD3);

  EICRA |= (1 << ISC20) | (1 << ISC30);
  EICRB |= (1 << ISC40) | (1 << ISC50);
  EIMSK |= (1 << INT4) | (1 << INT5) | (1 << INT2) | (1 << INT3);
  sei();
}

#endif
```

Enkoder nam služi kako bismo odredili koliko "otkucaja" (engleski: ticks) se motor okrenuo i u kojem smjeru. Naši motori u jednoj revoluciji naprave oko 1200 otkucaja. U datoteci encoder.h akumulirat ćemo broj otkucaja oba motora u varijablama *ticks*1 i *ticks*2. Varijable *ticks_old*1 i *ticks_old*2 služit će nam kako bismo odredili u kojem smjeru i za koliko

otkucaja se okrenuo pojedini motor.

Na pinovima E4, E5, D2 i D3 čitat ćemo napon na enkoderu pa te pinove postavljamo na 0. Pristupanjem EICRA registru i postavljanjem bitova ISC20 i ISC30 na jedinicu, te postavljanjem bitova ISC40 i ISC50 u EICRB registru na jedinicu omogućujemo prekide na pinovima INT2, INT3, INT4 i INT5 pri promijeni napona. Postavljanjem bitova INT2, INT3, INT4 i INT5 na jedinicu u registru EIMSK uključujemo prekide na pripadajućim pinovima. Poziv funkcije *sei*() postavlja I bit u SREG registru na jedinicu i time globalno omogućuje prekide.

*ISR*() funkcije pozivaju se prilikom promjene napona na pinovima E4, E5, D2 i D3 i, ovisno o naponu na odgovarajućim pinovima, povečavaju i smanjuju vrijednost varijabli *ticks*1 i *ticks*2. Varijable $A1$ i $B1$ predstavljaju stanje napona na pinovima A i B enkodera jednog motora, a $A2$ i $B2$ stanje napona na pinovima A i B enkodera drugog motora.

### 5.4.3 motor.h

```
1  #ifndef MOTOR_H_
2  #define MOTOR_H_
3
4  #include <avr/io.h>
5
6  void motor_init() {
7    DDRB |= (1 << DDB6) | (1 << DDB5);
8    DDRH |= (1 << DDH3) | (1 << DDH4);
9
10   TCCR1B |= (1 << CS10);
11   TCCR1A |= (1 << COM1A1) | (1 << COM1B1);
12
13   TCCR4B |= (1 << CS40);
14   TCCR4A |= (1 << COM4A1) | (1 << COM4B1);
15
16   TCCR1A |= (1 << WGM11) | (1 << WGM10);
17   TCCR4A |= (1 << WGM41) | (1 << WGM40);
18 }
19
20 #endif
```

Motorima ćemo upravljati promjenom napona na pinovima H3, H4, B5 i B6. Za PWM koristimo Timer 1 i Timer 4, budući da oni imaju 10-bitnu rezoluciju za analogni izlaz. To znači da ćemo napon iz segmenta $[0V, 5V]$ unutar mikrokontrolera zadati kao cjelobrojnu vrijednost između 0 i 1023.

U funkciji *motor_init*() postavljamo pinove H3, H4, B5 i B6 na jedinicu jer su oni vezani za Timer 1 i Timer 4. Postavljajući bitove CS10 i CS40 na jedinicu u registrima TCCR1B i TCCR4B postavljamo Timer 1 i Timer 4 na frekvenciju mikrokontrolera. Kako bismo postavili "Non-inverting mode" potrebno je postaviti bitove COM1A1 i COM1B1 na jedinicu u registru TCCR1A, te za Timer 4 postaviti bitove COM4A1 i COM4B1 u registru TCCR4A na jedinicu. Kako bismo omogućili 10-bitni "Phase Correct" PWM potrebno je postaviti WGM11 i WGM10 bitove na jedinicu te bitove WGM41 i WGM40 na jedinicu u registrima TCCR1A za Timer 1, ondosno TCCR4A za Timer 4.

### 5.4.4 main.c

```
#ifndef F_CPU
#define F_CPU 16000000UL
#endif

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/twi.h>
#include "encoder.h"
#include "motor.h"
#include "counter.h"

volatile uint64_t ms = 0;

volatile int32_t P, I, D;

volatile int64_t err1 = 0;
volatile int64_t err_old1 = 0;
volatile int64_t err_acc1 = 0;
volatile int64_t err_diff1 = 0;

volatile int64_t err2 = 0;
volatile int64_t err_old2 = 0;
volatile int64_t err_acc2 = 0;
volatile int64_t err_diff2 = 0;

volatile int64_t gain1 = 0;
volatile int16_t speed1 = 0;

volatile int64_t gain2 = 0;
volatile int16_t speed2 = 0;

volatile int64_t max = 1023;

volatile int16_t tps1 = 0;
volatile int16_t tps_m1 = 0;

volatile int16_t tps2 = 0;
volatile int16_t tps_m2 = 0;

void reset_errors() {
  err1 = 0;
  err_old1 = 0;
  err_acc1 = 0;
  err_diff1 = 0;
  err2 = 0;
  err_old2 = 0;
  err_acc2 = 0;
  err_diff2 = 0;
}
```

Na početku programa definiramo baznu frekvenciju mikrokontrolera: 16MHz. Zatim, nakon unošenja svih potrebnih datoteka, definiramo varijable koje su nam potrebne za PID regulator. Za konstrukciju PID regulatora, potrebne su nam proporcionalna konstanta P ($k_P$), integracijska konstanta I ($k_I$) te derivacijska konstanta D ($k_D$). Osim navedenih konstanti potrebno nam je i nekoliko promijenjivih varijabli. Budući da imamo dva motora, imat ćemo

i dva PID regulatora s različitim promijenjivim varijablama. Kako naš mikrokontroler nema *floating point* jedinicu, sve varijable su cijeli brojevi.

Upravljanje u trenutku $t$ koje je potrebno da brzinu okretaja motora dovedemo na željenu vrijednost računa se po jednadžbi (35). Konstante P, I i D je potrebno "naštimati" isprobavanjem različitih vrijednosti i promatranjem rezultata. Varijabla $err_i$ predstavlja trenutnu grešku. Varijabla $err\_old_i$ predstavlja grešku u trenutku $t-1$. Integral predstavljamo akumuliranom greškom koju spremamo u varijablu $err\_acc_i$. Derivaciju predstavljamo kao razliku trenutne greške i greške iz prošle iteracije te ju spremamo u varijablu $err\_diff_i$.

Dobivenu kontrolu $u_i(t)$ pohranjujemo u varijablu $gain_i$. Dobiveni $gain_i$ trebao bi biti u segmentu $[-1023, 1023]$, gdje za negativne vrijednosti okrećemo motor u jednom smjeru, a za pozitivne u drugom. Konstanta $max$ postavljena na vrijednost 1023 služi kako $gain_i$ nikada ne bi prešao vrijednost 1023, odnosno -1023. Kako bismo to postigli definiramo dodatnu pomoćnu varijablu $speed_i$ koja će biti vrijednost koju postavljamo na OCR registar.

Varijabla $tps_i$ predstavlja željenu kutnu brzinu motora, a $tps\_m_i$ predstavlja izmjerenu kutnu brzinu. U varijablu $ms$ spremamo protekli broj milisekundi. Funkcija $reset\_errors()$ postavlja varijable u koje spremamo pogreške na inicijalnu vrijednost.

```
ISR(TIMER0_COMPA_vect) {
  ++ms;
  if(ms % 5 == 0) {
    tps_m1 = (ticks1 - ticks_old1)*1000 / 5;

    err1 = tps1 - tps_m1;
    err_acc1 += err1;
    err_diff1 = err1 - err_old1;
    gain1 = P * err1 + I * err_acc1 + D * err_diff1;
    gain1 /= 50;

    if(gain1 > 0) {
      speed1 = gain1 > max ? max : gain1;
      OCR1A = speed1;
      OCR1B = 0;
    }
    else {
      gain1 *= -1;
      speed1 = gain1 > max ? max : gain1;
      OCR1B = speed1;
      OCR1A = 0;
    }

    tps_m2 = (ticks2 - ticks_old2)*1000 / 5;

    err2 = tps2 - tps_m2;
    err_acc2 += err2;
    err_diff2 = err2 - err_old2;
    gain2 = P * err2 + I * err_acc2 + D * err_diff2;
    gain2 /= 50;

    if(gain2 > 0) {
      speed2 = gain2 > max ? max : gain2;
      OCR4A = speed2;
      OCR4B = 0;
```

```
36      }
37      else {
38        gain2 *= -1;
39        speed2 = gain2 > max ? max : gain2;
40        OCR4B = speed2;
41        OCR4A = 0;
42      }
43
44      err_old1 = err1;
45      ticks_old1 = ticks1;
46      err_old2 = err2;
47      ticks_old2 = ticks2;
48    }
49  }
```

Funkcija $ISR(TIMER0\_COMPA\_vect)$ poziva se svake milisekunde. Unutar ove funkcije implementirali smo oba PID kontrolera. Funkcija se poziva svake milisekunde pa pri svakom pozivu povečavamo vrijednost varijable $ms$ za 1. Radi bolje rezolucije, svakih 5 milisekundi računamo novo upravljanje $u_i(t)$. Izmjerenu kutnu brzinu motora $tps\_m_i$ dobivamo kao razliku trenutnih otkucaja $ticks_i$ i otkucaja iz prošle iteracije $ticks\_old_i$ podijeljenu s 0.005 (zato što to radimo svakih 5 milisekundi).

Do vrijednosti ostalih varijabli dolazimo vrlo jednostavno. Trenutna pogreška $err_i$ jest razlika između željene brzine $tps_i$ i izmjerene brzine $tps\_m_i$. Akumulirana greška $err\_acc_i$ jest dosadašnja akumulirana greška plus trenutna greška $err_i$. Razliku trenutne greške $err_i$ i greške iz prošle iteracije $err\_old_i$ spremamo u varijablu $err\_diff_i$. $gain_i$ dobivamo prema navedenoj formuli, ali ga radi bolje upravljivosti dodatno dijelimo s 50. Ukoliko je $gain_i$ veći od 0, željenu brzinu postavljamo na OCRA registar, dok OCRB registar postavljamo na 0. U suprotnom OCRA registar postavljamo na 0, a OCRB registar na željenu brzinu. Prije toga provjeravamo je li željena brzina veća od 1023 , tj. manja od -1023 i u varijablu $speed_i$ spremamo odgovarajuću vrijednost koju onda stavljamo na odgovarajući OCR registar. Na kraju ažuriramo vrijednosti naših varijabli na način da $err\_old_i$ postaje $err_i$, a $ticks\_old_i$ postaje $ticks_i$.

```
1   uint8_t address = 10;
2
3   void i2c_init() {
4     TWAR = address << 1;
5     TWCR = (1 << TWIE) | (1 << TWEA) | (1 << TWEN) | (1 << TWINT);
6   }
7
8   volatile uint16_t r;
9   int cetveroznamenkasti = 0;
10  int flag = 0;
11  int flag2 = 0;
12  int negativan = 0;
13  int16_t tps1_temp = 0;
14
15  ISR(TWI_vect) {
16    switch(TW_STATUS) {
17      case TW_SR_DATA_ACK:
18      flag++;
19      r = TWDR;
20      if(r == 10){
21        flag = 0;
```

```
22        flag2 = 0;
23        negativan = 0;
24        cetveroznamenkasti = 0;
25        TWCR = 0;
26        TWCR = (1 << TWIE) | (1 << TWEA) | (1 << TWEN) | (1 << TWINT);
27        break;
28      }
29      if (r > 127) {
30        r = (256 - r)*-1;
31        negativan = 1;
32      }
33      if(negativan && cetveroznamenkasti != 0) cetveroznamenkasti =
      cetveroznamenkasti*10 -r;
34      else cetveroznamenkasti = cetveroznamenkasti*10 + r;
35      if(flag == 4){
36        negativan = 0;
37        flag = 0;
38        if (flag2) {
39          flag2 = 0;
40          tps2 = cetveroznamenkasti;
41          tps1 = tps1_temp;
42          tps1_temp = 0;
43          reset_errors();
44        }
45        else {
46          flag2 = 1;
47          tps1_temp = cetveroznamenkasti;
48        }
49        cetveroznamenkasti = 0;
50      }
51      TWCR = (1 << TWIE) | (1 << TWEA) | (1 << TWEN) | (1 << TWINT);
52      break;
53
54      case TW_BUS_ERROR:
55      TWCR = 0;
56      TWCR = (1 << TWIE) | (1 << TWEA) | (1 << TWEN) | (1 << TWINT);
57      break;
58
59      default:
60      TWCR = (1 << TWIE) | (1 << TWEA) | (1 << TWEN) | (1 << TWINT);
61      break;
62    }
63  }
```

Kako bismo inicijalizirali $I^2C$ na Arduinu, potrebno je u registar TWAR pohraniti adresu ovog *slave* uređaja. Kako smo već odredili na Raspberry Pi računalu, adresa mikrokontrolera bit će 10. Sedam najznačajnijih bitova adrese mikrokontrolera postavljamo na TWAR registar. Sljedeće trebamo omogućiti prekide za $I^2C$. To radimo tako da postavimo bit TWIE u registru TWCR na jedan. U TWCR registru također trebamo postaviti bitove TWEA, TWEN i TWINT na jedan kako bismo omogućili uparivanje adresa te omogućili $I^2C$.

Primljeni podatak spremat ćemo u varijablu $r$. Varijable *cetveroznamenkasti*, *flag*, *flag*2 i *negativan* služe nam kako bismo od primljenih znamenki sa Raspberry Pi računala složili točan broj. Varijabla *tps*1_*temp* pomaže nam da bismo mogli u isto vrijeme zadati brzine na oba motora.

U funkciji $ISR(TWI\_vect)$ imamo nekoliko slučajeva. Svi slučajevi nalaze se unutar registra TW_STATUS. U slučaju TW_BUS_ERROR i u zadanom slučaju samo resetiramo vrijednost TWCR registra. U slučaju TW_SR_DATA_ACK *master* šalje podatke i u ovom slučaju obavljamo sav posao. Prvo povečavamo vrijednost varijable $flag$. Kako smo ranije rekli, za brzinu jednog motora Raspberry Pi računalo poslat će četiri znamenke. Varijabla $flag$ služi nam kako bismo vodili računa o tome koliko znamenki smo primili. Primljena znamenka nalazi se u registru TWDR i tu vrijednost pohranjujemo u varijablu $r$.

Kao što smo spomenuli prilikom implementacije poslužitelja, ukoliko *master* pošalje broj 10 tada je poslužitelj registrirao pogrešku i želi da mikrokontroler resetira vrijednosti određenih varijabli. Varijable koje trebamo resetirati su $flag$, $flag2$, *negativan* i *cetveroznamenkasti*. Sada sljedeći broj koji primimo promatramo kao prvu znamenku brzine desnog motora. Prije nego što izađemo iz ovog slučaja, resetiramo vrijednosti u TWCR registru.

Ukoliko je primljeni broj (različit od 10) veći od 127, tada je to zapravo negativan broj. Kako bismo dobili stvarni poslani broj, dobiveni broj je potrebno oduzeti od 256 i pomnožiti rezultat s -1. U tom slučaju postavljamo vrijednost pomoćne varijable *negativan* na 1 kako bismo znali da sljedeće znamenke treba pridodati (oduzeti) negativnom broju. Na jednostavan način od dobivenih znamenki slažemo četveroznamenkasti broj (množimo s 10 akumulirani broj i zbrajamo, odnosno oduzimamo pristigle brojeve).

Ukoliko smo primili četiri znamenke tada pomoću varijable $flag2$ znamo jesmo li primili brzinu za desni ili lijevi motor. Ukoliko smo primili brzinu za desni motor ($flag2 == 0$) tu brzinu pohranjujemo u varijablu $tps1\_temp$, a vrijednost varijable $flag2$ postavljamo na 1. Ukoliko smo primili brzinu i za lijevi motor ($flag2 == 1$) tada vrijednosti varijabli $tps1$ i $tps2$ postavljamo na željene brzine te resetiramo odgovarajuće varijable. Pozivamo funkciju $reset\_errors()$ jer smo dobili nove kutne brzine motora i stare vrijednosti pogrešaka koje su vezane za prošle kutne brzine trebaju biti postavljene na 0. Na kraju svakog slučaja resetiramo vrijednosti TWCR registra.

```
1  int main() {
2    encoder_init();
3    motor_init();
4    counter_init();
5    i2c_init();
6
7    DDRD |= (1 << DDD0) | (1 << DDD1);
8    PORTD |= (1 << PORTD0) | (1 << PORTD1);
9
10   P = 30;
11   I = 6;
12   D = 10;
13   tps1 = 0;
14   tps2 = 0;
15
16   while(1){
17   }
18   return 0;
19 }
```

U main dijelu programa inicijaliziramo enkoder, motor, brojač i $I^2C$. Pinove D0 (SCL) i D1 (SDA) zajedno s pripadnim pull-up otpornicima postavljamo na jedan, određujemo vrijednosti PID konstanti te postavljamo željene kutne brzine na 0.

# 6 Zaključak

Koristeći naučeno iz prethodnih poglavlja, moguće je dizajnirati mobilnog robota s diferencijalnim pogonom kojim se može upravljati putem pametnog telefona u stvarnome vremenu. Pokazali smo kako željeno kretanje robota pretvoriti u potrebne okretaje motora. Također, objasnili smo arhitekturu $I^2C$ protokola i kako se šalju podaci ovim protokolom. Na kraju, prije konkretnog primjera implementacije ovog problema, objašnjen je PID regulator.

Implementacija predstavljena u petom poglavlju nalazi se na tri različita uređaja. Klijentski dio aplikacije (upravljač) nalazi se na pametnom telefonu i izrađen je pomoću HTML-a, CSS-a i JavaScript-a. Poslužitelj se nalazi na Raspberry Pi računalu i napravljen je u JavaScript runtime okruženju Node.js. U mojoj implementaciji, za komunikaciju između Raspberry Pi računala i mikrokontrolera, korišten je $I^2C$ protokol. Uz malene izmjene, gore objašnjena implementacija mogla bi se iskoristiti i kada bi se za komunikaciju koristio neki drugi protokol (npr. SPI protokol). PID kontroler implementiran je na mikrokontroleru Arduino Mega 2560.

# Literatura

[1] R. Y. Siegwart, I. R. Nourbakhsh, Introduction to autonomous mobile robots, MIT. 2004.

[2] B. Siciliano, O. Khatib, Springer Handbook of Robotics, Springer, 2008.

[3] http://www.hdip-data-analytics.com/_media/resources/pdf/exploring_raspberry_pi_pdfdrive.com_.pdf

[4] https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

[5] https://yoannmoi.net/nipplejs

[6] https://www.npmjs.com/package/express

[7] https://www.npmjs.com/package/body-parser

[8] https://www.npmjs.com/package/i2c-bus