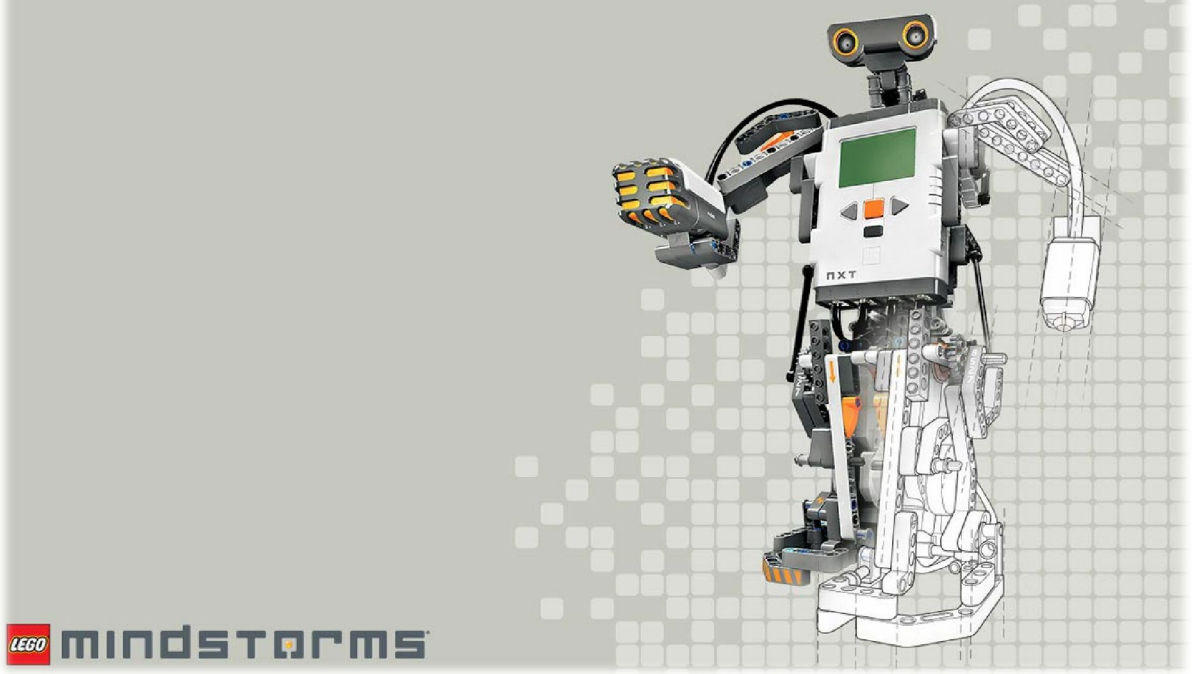
UNIVERZITET U KRAGUJEVCU – MAŠINSKI FAKULTET – KATEDRA ZA PRIMENJENU MEHANIKU I AUTOMATSKO UPRAVLJANJE

Primena lego robota u inženjerskoj edukaciji



Mentor:

Prof. dr Milan Matijević

Studenti:

Lekić Aleksandar 24/2001

Dejan Stevanović 220/92

Zoran Živanović 30/96

Kragujevac, 2010

Sadržaj:

Uvod

Ovaj rad se bavi opisom komponenti seta LEGO MINDSORMS 8547 kao i mogućnostima programiranja robota u NXT programu, kroz realizaciju tri osnovna praktična primera. To su: praćenje putanje, praćenje putanje sa zaobilaženjem prepreke i pozicioniranje u prostoru – nalaženje koordinata.

Praktični primeri su realizovani što je jednostavnije moguće, te se očekuje njihov dalji razvoj kroz druge studentske radove.

Struktura LEGO robota i komponente seta LEGO MINDSTORMS 8547

Lego MINDSTORMS NXT 8547 se sastoji iz sledećih komponenti (slika 1.1):

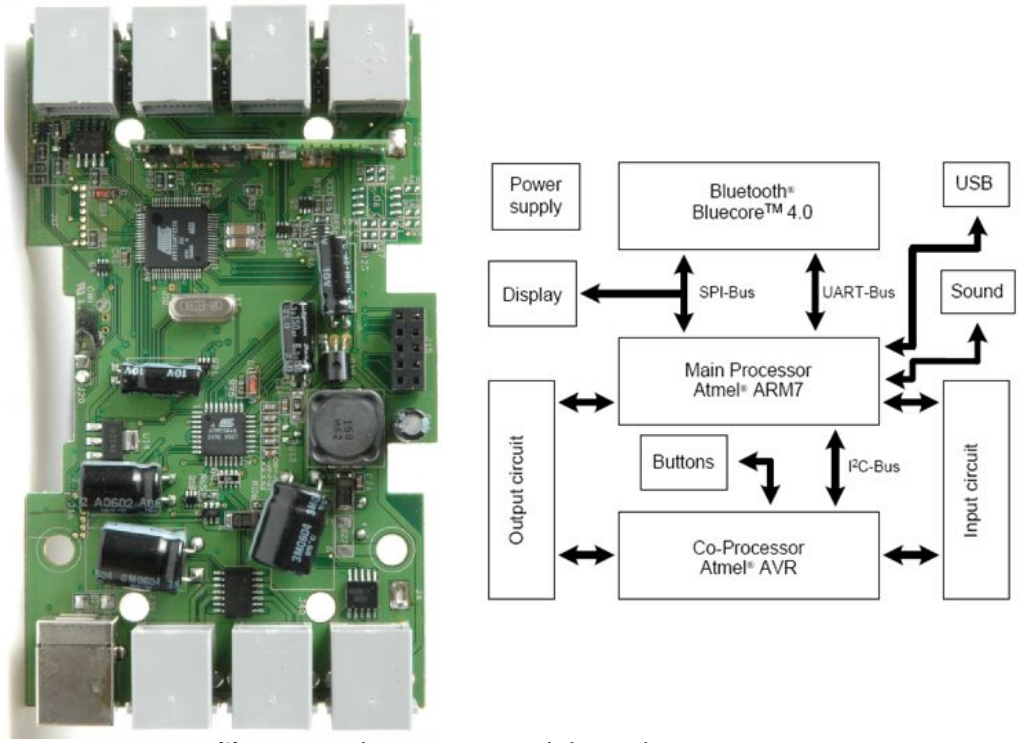
- Programabilne jedinice
- Tri servomotora
- Senzora (dva senzora za dodir, senzor za boju, ultrazvučni senzor)

Slika 1.1 Komponente seta LEGO MINDTORMS 8547

Progamabilna jedinica

Programabilna jedinica se sastoji iz (slika 1.2a i 1.2b):

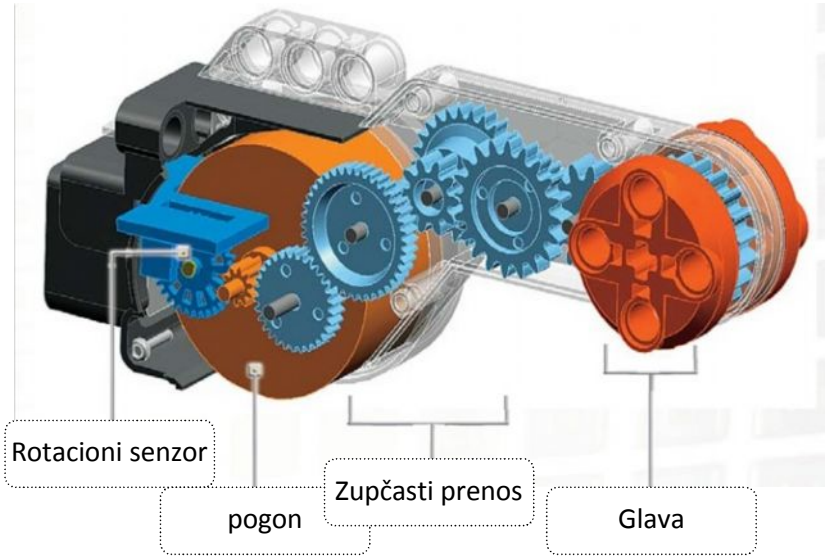
- Glavnog procesora - 32-bit ARM mikrokontrolera - Atmel AT91SAM7S256. Flash memorija/fajl sistem (256 kB), RAM (64 kB) i USB Device interfejs, radna frekvencija 48MHz
- Pomoćnog procesora - Atmel 8-bit AVR procesor, ATmega48 sa 4KB flash memorije i 512B RAMa, radne frekvencije od 8Mhz,
- 4 analogna inputa za senzore ,
- 3 PWM motor driver outputa sa ugrađenim enkoderima (1°rezolucije),
- Bluetooth-a (NXT ka NXT, NXT ka kompjuteru),
- LCD displeja rezolucije 100x64 piksela i front panela sa četiri tastera i zvučnika



Slika 1.2 Struktura programabilne jedinice

Servomotor

Servomotor se sastoji iz pogona – motora, osam zupčanika, rotacionog senzora i glave servomotora (slika 1.3)



Slika 1.3 Konstrukcija servomotora

Zbog prenosnog odnosa zupčanika, rotacioni senzor može da detektuje 1° rotacije glave servomotora. Detaljan izgled senzora rotacije je dat na slici 1.4.

Slika 1.4 Senzor rotacije servomotora

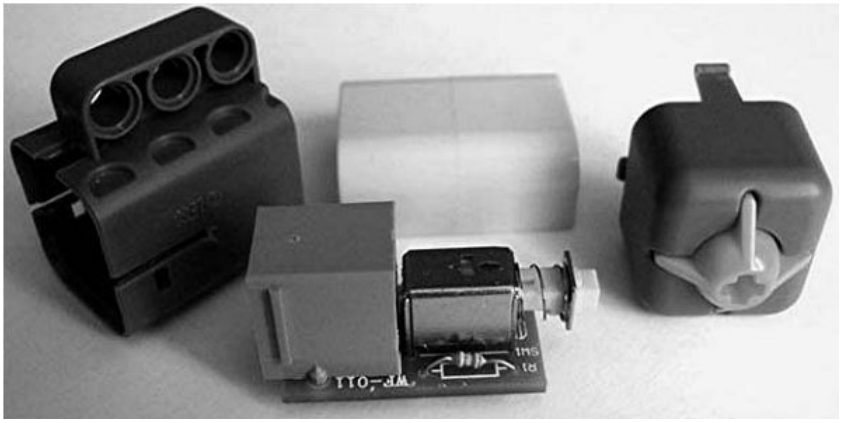
Tehnički parametri motora:

- Napajanje 9V
- Ugaona brzina 177 min^{-1}
- Obrtni moment 16.7 Ncm
- Snaga 2.03 W
- Efikasnost 41%

Senzor za dodir

NXT senzor za dodir je prikazan na slici 1.5. Dobra karakteristika je ta što ima krstasti otvor na sredini koji omogućava da se senzor direktno poveže sa ostalim delovima. Senzor za dodir se sastoji od štampanog kola na kojem se nalazi povratni prekidač i konektor (slika 1.6). Sa prekidačem je redno povezan otpornik, da se ne bi stvorio kratak spoj u slučaju ako se senzor za dodir slučajno poveže na izlazni port.

Slika 1.5 NXT Senzor za dodir



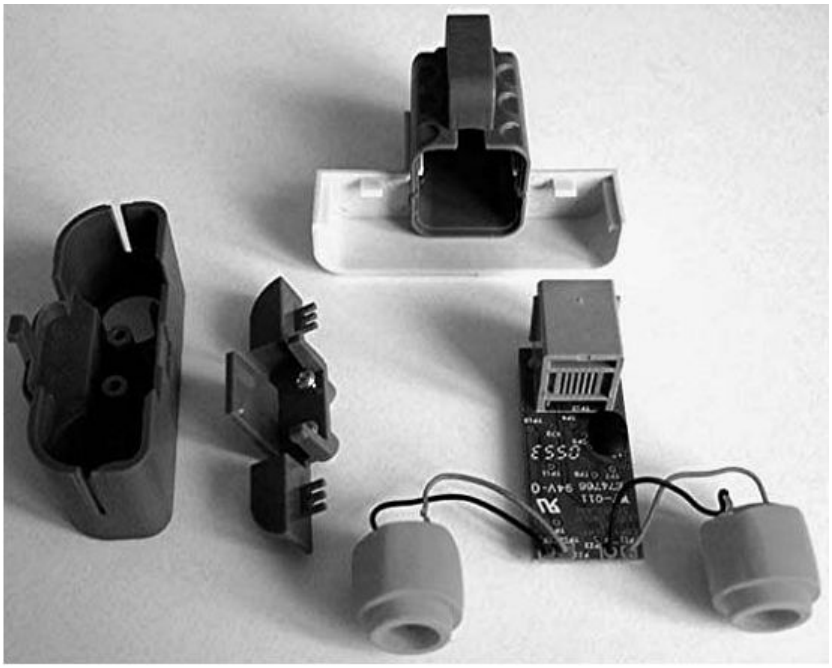
Slika 1.6 Unutrašnja struktura senzora za dodir

Ultrazvučni senzor

Ultrazvučni senzor je prikazan na slici 1.7. Zbog svoje kompleksnosti, ima svoj poseban mikroprocesor, zbog čega vraća izmerenu daljinu u apsolutnim jedinicama. Senzor radi po principu sonara, šaljući kratkotrajne signale od 40kHz, zatim meri vreme koje je potrebno da zvuk ode i odbije se od objekta. Očitavanja su bolja kada je objekat od kojeg se zvuk odbija veći, ako je okruženje robota koplikovanije, sa manjim objektima, merenje tad postaje manje pouzdano. Ne preporučuje se istovremeno korišćenje više ultrazvučnih senzora zbog moguće interference zvučnih talasa. Apsolutna greška pri merenju iznosi ±3cm.

Slika 1.7 Ultrazvučni senzor

Pored mikroprocesora, ultrazvučni senzor se sastoji i od mikrofona i zvučnika. Njegova unutrašnja strukra je prikazana na slici 1.8.



Slika 1.8 Unutrašnja struktra ultrazvučnog senzora

Senzor za boju

Izgled senzora za boju je prikazan na slici 1.9. NXT senzor za boju može da obavlja tri različite funkcije; može da radi kao senzor za boju, pri čemu može da razlikuje šest boja (crnu, plavu, zelenu, žutu, crvenu, belu); može da radi kao senzor za jačinu svetla (reflektovanog i ambijentalnog); i da radi kao lampa, pri čemu može emitovati crveno, zeleno ili plavo svetlo.

Slika 1.9 Senzor za boju

NXT portovi

Četiri ulazna porta se nalaze na dnu NXT programabilne jedinice, i numerisani su brojevima od 1 do 4. Tri izlazna porta se nalaze na vrhu progamabilne jedinice i označeni su slovima A, B i C. Ako se pogleda vrh konektora (NXT kabl) videće se šest pinova i žica koje povezuju senzore i aktuatore sa NXT upravljačkom jedinicom. Te žice su obojene sledećim bojama: bela, crna, crvena, zelena, žuta i plava. Njihova funkcija zavisi od toga da li se koriste na ulazu ili izlazu, i od toga na koji je senzor konektovan.

Senzorski pinovi

Tabela 1.1 daje pregled input pinova.

Tabela 1.1 Pregled input pinova

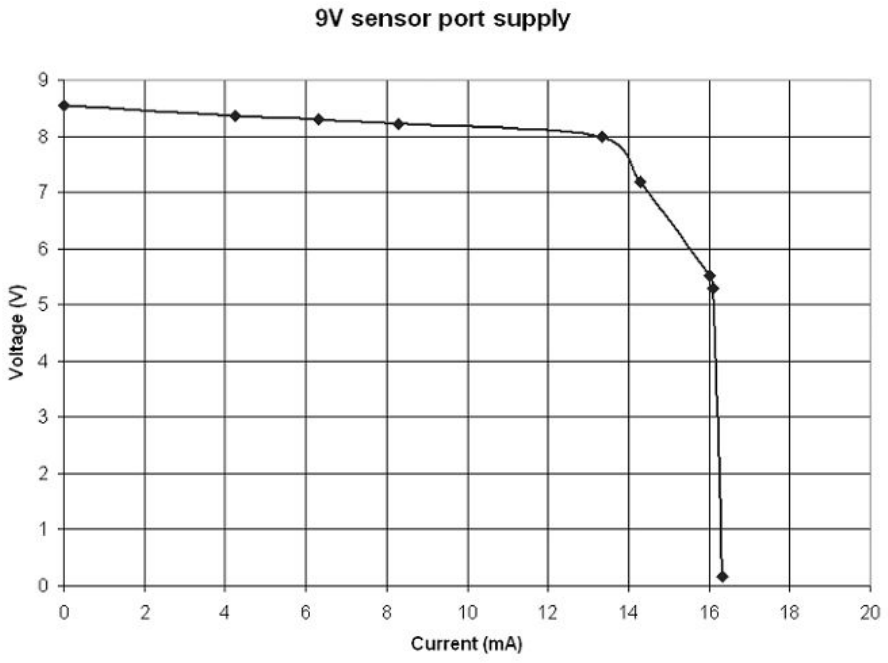
Broj pina	Boja	Ime
1	Bela	AN
2	Crna	GND
3	Crvena	GND
4	Zelena	Napajanje 4.3 V
5	Žuta	DIGI 0
6	Plava	DIGI 1

PIN 1 – Ovaj pin se koristi u dve svrhe: ili kao analogni input ili za napajanje od 9V pri korišćenju RCX senzora.

Kada se koristi kao analogni input, signal ide na 10-bitni AD konvertor. Ulazni signal može imati voltažu od 0 do 5V, i prevodi se u digitalnu vrednost između 0 i 1023. Vreme semplovanja je 3µs. Pin je direktno povezan na otpornik veličine 10kΩ.

Ovaj pin se takođe može koristiti za napajanje od 9V, što predstavlja maksimalno napajanje koje dolazi od baterija. Ako se koriste NiMH baterija, maksimalna voltaža je tad 7.2V. Na primer, NXT ultrazvučni senzor koristi ovo dodatno napajanje da bi imao više snage na svom transmiteru.

Za ove senzore, NXT vrši napajanje senzora u trajanju 3µs, a zatim čita vrednost u vremenskom intervalu od 0.1µs. Senzoru je potreban kondenzator kako bi imao napajanje za vreme čitanja vrednosti. Slika 1.10 pokazuje izlazni napon pri različitim jačinama struje, pri čemu baterije imaju punu snagu. Na slici se vidi da postoji granica od 14 mA, posle koje voltaža drastično opada.



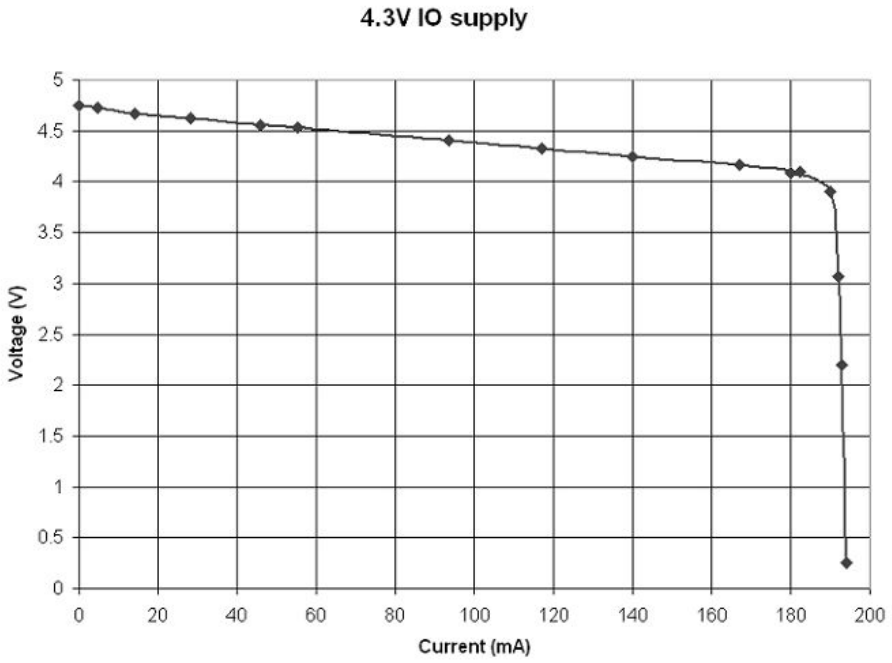
Slika 1.10 Naponsko – strujni dijagram

Pinovi 2 i 3 Drugi i treći pin su pinovi za uzemljenje. Ova dva pina su povezana zajedno i NXT senzorima. Svi signali se mere u odnosu na ove pinove. Senzor može koristiti ili samo jedan pin ili oba.

Pin 4 Ovo je glavni izvor napajanja za NXT senzore. Naspram pina 1, ovaj pin ima ograničen protok struje na 180 mA za sve ulazne i izlazne portove (slika 1.11). Svaki port može koristiti u proseku po 25 mA, ali je takođe moguće da neki port troši više struje ukoliko neki drugi troši manje. U tabeli 1.2 je prikazana potrošnja struje u odnosu na tip senzora.

Senzor	Jačina struje
Dodir	0
Zvuk	1.7 mA
Boja (lampa – on)	2.6 – 3 mA u zavisnosti od svetla
Boja (lampa – off)	16.3 mA
Ultrazvučni	4 mA
Rotacioni senzor	9 – 12 mA u zavisnosti od pozicije enkodera

Tabela 1.2 Potrošnja struje određenih senzora



Slika 1.11 Naponsko – strujna karakteristika PINa 4

Pinovi 5 i 6 Ovi pinovi su namenjeni za digitalne signale naponskog nivoa od 3.3V, i direktno su povezani sa NXT mikroprocesorom. Uglavnom se koriste za I^2C komunikaciju. Kada se koriste kao outputi, mora se viditi računa o granici od 3.3 V. Kada se koriste kao inputi, NXT ima zaštitno kolo koje sprečava oštećenja od većih napona. To zaštitno kolo ima otpornik od 4.7kΩ koje je redno povezano sa portom. Tako da ako je napon sa senzora prevelik, struja će biti manja, i elektronika neće biti oštećena.

Pored I^2C komunikacije, DIGI0 se koristi pri upravljanju NXT senzorom za svetlo, tačnije da upravlja stanjem LE diode (uklllučena – isključena).

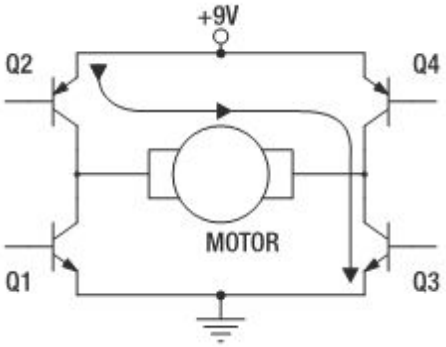
Output pinovi – pinovi servomotora

Tabela 1.3 daje pregled output pinova.

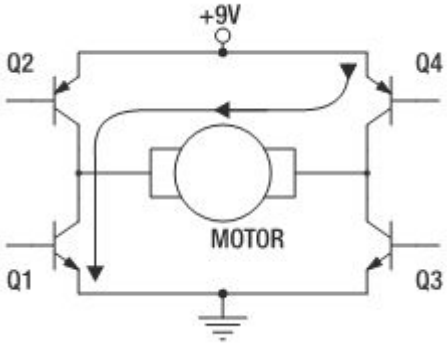
Broj pina	Boja	Ime
1	Bela	M1
2	Crna	M2
3	Crvena	GND
4	Zelena	Napajanje 4.3 V
5	Žuta	TACHO 0
6	Plava	TACHO 1

Tabela 1.3 Output pinovi

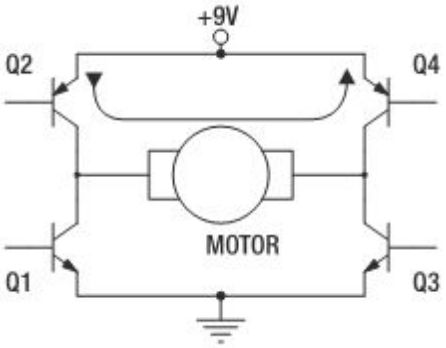
Pinovi 1 i 2 Ovi pinovi služe za napajanje motora. Maksimalna voltaža je voltaža na baterijama (9V za standardne, 7.2V za NiMH baterije). Motor se kontroliše pomoću kola koje se zove H-most (IC drivers LB1836M and LB1930M). H-most se sastoji iz četiri tranzistora, koji su označeni brojevima od 1 do 4. Kontrolno kolo je napravljeno tako da tranzistori 1 i 2 na jednoj strani, i tranzistori 3 i 4 na drugoj strani, ne sprovode struju istovremeno, kako bi se napajanje uštedelo. Slika 1.12 pokazuje stanje tranzistora pri hodu motora unapred, kada su uključeni tranzistori 1 i 4. Slika 1.13 pokazuje stanje motora pri hodu unazad, kada su uključeni tranzistori 2 i 3. Slike 1.14 i 1.15 prikazuju brake i floating stanja motora.



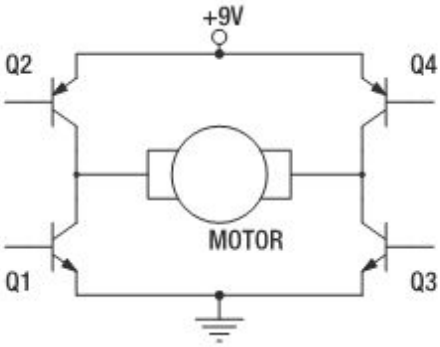
Slika 1.12 Hod unapred



Slika 1.13 Hod unazad

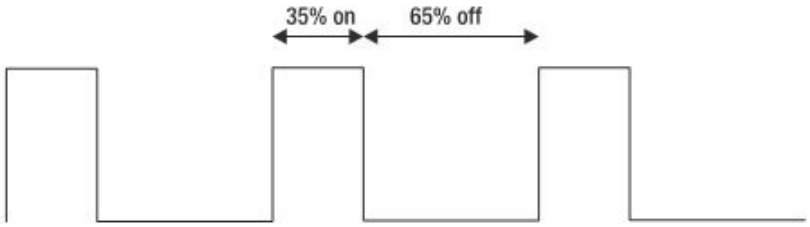


Slika 1.14 Brake stanje



Slika 1.15 Floating stanje

Brzinom motora se upravlja pomoću PWM signala (Slika 1.16).

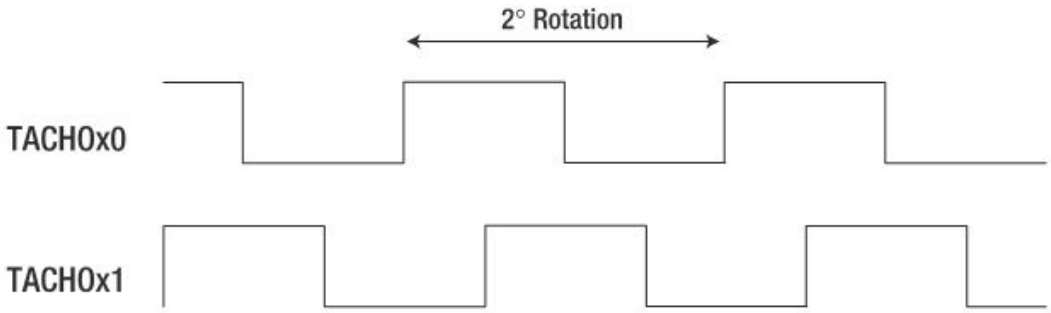


Slika 1.16 Upravljanje brzinom motora

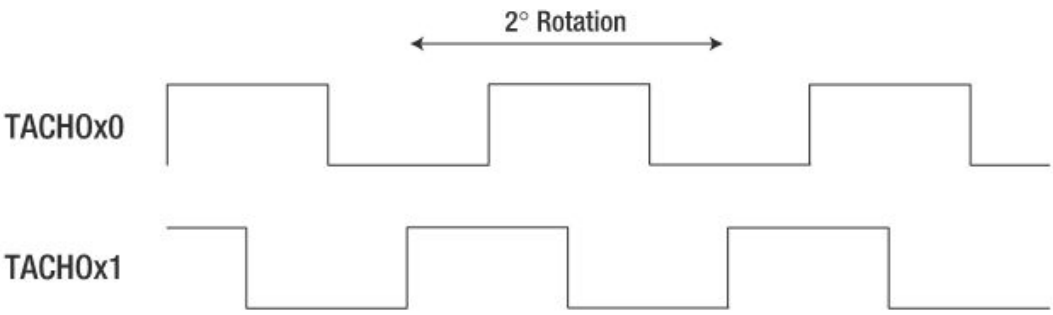
Dužina ciklusa je 128 µs. To odgovara frekvenciji od 7800 Hz, koja se može ponekad čuti. Ciklus dužine od 128 µs oogućava bolju linearnost (za NXT, odnos između brzine motora i voltaže je linearna). Jačina struje na output portovima iznosi približno 700 mA.

Pin 4 Ovaj pin je je konektovan na naponski izvor od 4.3 V, kojeg dele svi NXT portovi.

Pinovi 5 i 6 Ova dva pinova koristi optički enkoder koji je ugrađen u NXT motore. Enkoder generiše kvadratne signale, pomoću kojih NXT određuje smer i brzinu kretanja motora. Slika 1.17 prikazuje ove signale pri hodu motora unapred, a slika 1.18 pri hodu motora unazad. Frekvencija ovih signala odgovara rotacionoj brzini motora. Jedna polovina ciklusa signala odgovara rotaciji od 1°.



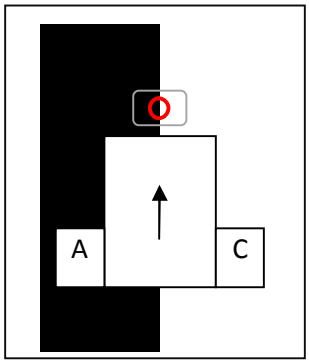
Slika 1.17 Izgled signala pri hodu motora unapred



Slika 1.18 Izgled signala pri hodu motora unazad

Upravljanje brzinom servo motora pomoću diskretnog PID kontrolera

EDGE FOLLOWER robot sa jednim senzorom



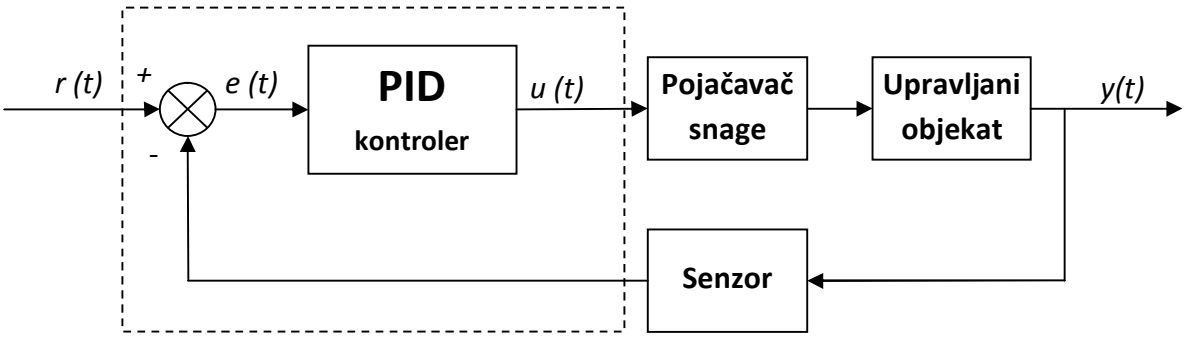
Slika 12.1 Edge follower robot u stacionarnom stanju

Na slici 12.1 je prikazana jednostavna skica Edge follower robota, koji se sastoji iz tela robota, servomotora A i C, i senzora za svetlo. Cilj je da robot prati crnu liniju, odnosno ivicu putanje.

Pretpostavimo da u stacionarnom stanju (kada se senzor za svetlo nalazi tačno između crne i bele površine) jačina odbijenog svetla iznosi 50%[1] (0% - crna, 100% - bela) i da želimo da je brzina na motorima A i C 50% od maksimalne. Iz ovoga se zaključuje da 50% odbijene svetlosti predstavlja ulaznu veličinu – veličinu koju pratimo.

Greška predstavlja razliku između trenutnog očitavanja senzora i željene veličine. Zadatak kontrolera je da ovu grešku eliminiše i/ili smanji na najmanju moguću meru najbrže moguće.

Prema gore navedenom, sistem sa slike 12.1 možemo prikazati blok dijagramom (1):



Slika 12.2 Blok dijagram robota *edge follower*

[1] U realnom stanju ovo nije slučaj, minimalna količina svetla će biti oko 10%, a maksimalna oko 50%, pa je srednja vrednost 30%; to je posledica refleksije površine, količine ambijentalnog svetla i dr. Pre izvođenja eksperimenta treba očitati ove vrednosti ručno i uneti ih u algoritam upravljanja.

PID kontroler je opisan sledećom diferencijalnom jednačinom (1):

$$u(t) = K_p \left[e(t) + \frac{1}{T_i} \int_0^t e(t)dt + T_d \frac{de(t)}{dt} \right] \quad (1)$$

Gde su:

K_p – proporcionalna konstanta

T_i – integraciona vremenska konstanta

T_d – diferencijalna vremenska konstanta

Za dovoljno male vremenske intervale *T* važi:

$$\frac{de(t)}{dt} \approx \frac{e(t) - e(t-T)}{T} \quad (2)$$

Na osnovu jednačine (2), jednačina (1) se može napisati u diskretnom obliku:

$$u(n) = K_p \left[e(n) + \frac{T}{T_i} \sum_{j=0}^{n} e(j) + \frac{T_d}{T} (e(n) - e(n-1)) \right] \quad (3)$$

Jednačina (3) se može napisati i kao:

$$u(n) = K_p e(n) + K_i \sum_{j=0}^{n} e(j) + K_d (e(n) - e(n-1)) \quad (4)$$

Jednačina (4) je poznatija u literaturi kao *Pozicioni algoritam*. Veličine K_p, K_i i K_d predstavljaju proporcionalnu, integralnu i diferencijalnu konstatnu; veličina *e(n)* je greška u trenutnom koraku, veličina *e(n-1)* predstavlja grešku u prethodnom koraku.

P regulator

Određivanje veličine K_p

Ako zakon upravljanja robotom formulišemo jednačinom:

$$u(n) = K_p e(n) \quad (5)$$

Odnosno da je kretanje robota u zavisnosti samo od greške, ponašanje robota biće takvo da se on pozicionira tačno iznad ivice putanje, međutim ono što želimo je da ponašanje robota bude takvo da se konstantno kreće i da prati ivicu putanje, pa ćemo jednačinu (5) koristiti u izmenjenom obliku:

$$\begin{aligned} u_l(n) &= u_0 + K_p e(n) \\ u_d(n) &= u_0 - K_p e(n) \end{aligned} \quad (5')$$

Gde su:

$u_l(n)$ – Brzina levog motora

$u_d(n)$ – Brzina desnog motora

u_0 – konstantna brzina; pri stacionarnom stanju ($e(n) = 0$) brzina kretanja robota je jednaka u_0

Rukovodeći se sledećim ograničenjima:

- $y(n)$ je ograničen maksimalnim i minimalnim očitavanjem senzora, koje se utvrđuju pre početka eksperimenta
- $r(n)$ je željena veličina i predstavlja srednju vrednost minimalnog i maksimalnog očitavanja senzora i utvrđuje se pre početka eksperimenta
- $u(n)$ je ograničen na vrednost 100 (maksimalna vrednost za NXT servo motore)
- veličinu u_0 smo izabrali da je jednaka vrednosti 50

Iz jednačine (5') dobijamo maksimalnu vrednost proporcionalne konstante K_p:

$$K_{pMAX} = \frac{u_{lMAX}(n) - u_0}{r_n - y_{nMAX}} \tag{6}$$

Znak „–" u jednačini (6) pokazuje da će pri skretanju robota na svetliju površinu brzina levog motora smanjiti, a brzina desnog točka povećati, što će prouzrokovati da robot skrene na manje svetlu površinu.

Izborom veličina u_{lMAX} i u_0 se može uticati na veličinu K_p.

Osim upravljanja brzinama svakog motora posebno, LEGO NXT Mindstorms ima opciju upravljanja pravcem kretanja preko veličine *SteerRate*, koja se kreće u opsegu -100 ÷ +100 (maksimalna negativna vrednost označava skretanje na levo tj. brzine motora biće maksimalne, stim što će im smerovi kretanja biti suprotni, i obrnuto). Ova opcija nam omogućava da zadamo željenu brzinu kretanja robota u stacionarnom stanju, i da preko veličine *SteerRate* zadamo željeno kretanje robota. Odnosno prema jednačini [5]:

$$Sr(n) = K_p e(n) \tag{7}$$

Iz jednačine (7) dobijamo:

$$K_{pMAX} = \frac{Sr(n)_{MAX}}{r_n - y_{nMAX}} \tag{8}$$

Znak „–" u jednačini (8) pokazuje da će pri skretanju robota na svetliju površinu, veličina *Sr* biti <0, što će prouzrokovati skretanje robota na manje svetlu površinu.

Ovaj zakon upravljanja daje brži odziv, obzirom da se motori kreću u oba smera, pa je ponašanje robota u oštrim krivinama brže i sistem je stabilniji.

Kako bi smo odredili pojčanje K_{pMAX} potrebno je prvo odrediti minimalnu i maksimalnu količinu odbijenog svetla, podaci su prikazani u tabeli 8.2.

PROGRAMSKI KOD za određivanje minimalne količine svetla

```
int SV;
int min = 1024;
task main()
{
   SetSensorType(IN_1, IN_TYPE_COLORRED); // u red modu radi kao
                                             lightsenzor
   SetSensorMode(IN_1, IN_MODE_RAW);  //rezultate daje od 0 do 1024
   Wait(SEC_1);
   while (true)
  {
    SV = Sensor(IN_1);
    if (SV < min)
    {
      min = SV;
    }

    NumOut(10,LCD_LINE3,min);
    NumOut(10, LCD_LINE4,SV);

  }
}
```

PROGRAMSKI KOD za određivanje maksimalne količine svetla

```
int SV;
int max = 0;
task main()
{
   SetSensorType(IN_1, IN_TYPE_COLORRED);
   SetSensorMode(IN_1, IN_MODE_RAW);
   Wait(SEC_1);
   while (true)
  {
    SV = Sensor(IN_1);
    if (SV > max)
    {
      max = SV;
    }
    NumOut(10,LCD_LINE3,max);
    NumOut(10, LCD_LINE4,SV);
  }
}
```

PROGRAMSKI KOD NXC programa edge follower – upravljanje brzinama motora

```
   #define max 582              // maximalna vrednost ocitavanja senzora
   #define min 153              // minimalna vrednost ocitavanja senzora
   #define V0 50                // konstantna brzina
```

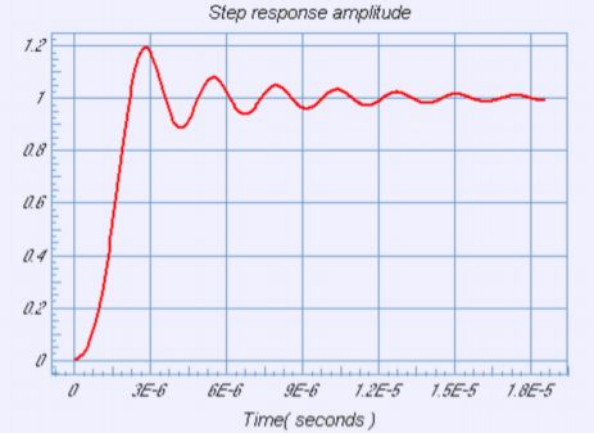
```
task main()
 {
 float mid = (max - min)/2;    // srednja vrednost - zeljena velicina
SetSensorType(IN_1, IN_TYPE_COLORRED);
SetSensorMode(IN_1, IN_MODE_RAW);
float Kp = 0.116;                      // Proporcionalna konstanta
while (true)                          // Beskonačna petlja
{
int y = SensorValue(S1);              // trenutno ocitavanje sa senzora
float e = mid - y;                    // funkcija greske
float Vl = V0 + Kp*e;                 // Brzina levog tocka
float Vd = V0 - Kp*e;                 // Brzina desnog tocka
OnFwd(OUT_A, Vl);
OnFwd(OUT_C, Vd);
}
}
```

PROGRAMSKI KOD NXC programa egde follower – upravljanje Steering rate

```
#define max 582 //maximalna vrednost ocitavanja senzora
#define min 153 //minimalna vrednost ocitavanja senzora

task main()
 {
 float mid = (max - min)/2;    //srednja vrednost - zeljena velicina
 SetSensorType(IN_1, IN_TYPE_COLORRED);
 SetSensorMode(IN_1, IN_MODE_RAW);
 float Kp = 0.233;             // Proporcionalna konstanta
 while (true)
 {
 int y = SensorValue(S1); // trenutno ocitavanje sa senzora
 float e = mid - y;            // funkcija greske
 float Sr = Kp*e/1000;         // Steering Rate
 OnFwdSync(OUT_AC, 50, Sr);    // Zakon upravljanja, motori A,C,
                               // brzina=50,
                               // Steering Rate = Sr
 }
 }
```

PI regulator

Polazeći od činjenice da nemamo matematički model sistema zaključujemo da je nemoguće odrediti kritično pojačanje i vreme smirenja koristeći se MatLabom, i drugim konvencionalnim metodama. Za dati sistem (u stacionarnom stanju brzina je 50% od maksimalne, odnosno V0 = 50, možemo pretpostaviti da je kritično pojačanje dato jednačinama (6) i (8), jer sve vrednosti pojačanja K_{pMAX} dovode motore u zasićenje pri maksimalnoj grešci.
Druga pretpostavka je da je odziv sistema na jediničnu odskočnu funkciju sličan onom sa slike 3.



Slika 12.3 Predpostavljeni odziv sistema

Iz gore navedenog zaključuje se da je moguće eksperimentalno odrediti vreme smirenja i vremensku konstanu *T* na sledeći način:

- za dati sistem proporcionalna konstanta $K_p = K_{pMAX}$; dok su konstante: $K_I = K_d = 0$
- vreme smirenja T_S je ono vreme koje je potrebno da protekne da bi greška bila u granicama od ±5%, pretpostavljamo da je $T_S < 10s$, merenje se vrši na pravoj liniji [6]
- vremenska konstanta *T* je jednaka vremenu izvršenja programa, određuje se na taj način što se program ograniči na 10000 ponavljanja, meri se vreme izvršenja programa, vremenska konstanta se određuje po obrascu $T = T_T/10000$, merenje se vrši na pravoj putanji

Prema navedenom konstante K_p, K_I, *i* K_d možemo odrediti prema Ziegler–Nichols metodi:

Vrsta kontrolera	K_p	K_I	K_d
P	0.5 K_{pMAX}		
PI	0.45 K_{pMAX}	1.2 K_pT/T_s	
PID	0.6 K_{pMAX}	2 K_pT/T_s	$K_pT_s/(8T)$

Tabela 8.1 Vrednosti konstati PID regulatora prema Ziegler–Nichols metodi

PROGRAMSKI KOD za određivanje vremena izvršenja 10000 petlji – upravljanje brzinama motora

```
#define max 582 // maximalna vrednost ocitavanja senzora
#define min 153 // minimalna vrednost ocitavanja senzora
#define V0 50

task main()
 {
 long t0 = CurrentTick();
 long t;
 float mid = (max + min)/2;          // srednja vrednost - zeljena velicina
 SetSensorType(IN_1, IN_TYPE_COLORRED); // podesavanje porta i tipa senzora
 SetSensorMode(IN_1, IN_MODE_RAW);
 float Kp = 0.233;                  // Proporcionalna konstanta
 repeat (10000)                    // ponavljanje petlje
 {
   int y = SensorValue(S1);       // trenutno ocitavanje sa senzora
   float e = mid - y;             // funkcija greske
   float Vl = V0 + Kp*e;          // Brzina levog tocka
```

```
  float Vd = V0 - Kp*e;          // Brzina desnog tocka
 OnFwd(OUT_A, Vl);
 OnFwd(OUT_C, Vd);
 t = CurrentTick() - t0;         // Ocitavanje vremena
 }
Off(OUT_AC);
NumOut(0, LCD_LINE4, t);         // prikazvanje vremena na displeju
Wait(SEC_15);                    // u trajanju od 15 sekundi
}
```

Tabela 8.2 Rezultati eksperimenta:

	Veličina upravljanja	
	Steering rate	**Brzine motora**
Min	153	153
Max	582	582
K_{pMAX} -prema jednačinama (6)(8)	0.466	0.233
Ts [6]	2 s	2 s
T	3.796 s 10000 petlji 18.992 s 50000 petlji 39.007 s 100000 petlji Srednja vrednost: 0.00038s	3.795 s 10000 petlji 26.170 s 50000 petlji 52.336 s 100000 petlji Srednja vrednost: 0.000475s

PROGRAMSKI KOD PI regulatora – upravljanje Steering rate

```
 #define max 582 // maximalna vrednost ocitavanja senzora
 #define min 153 // minimalna vrednost ocitavanja senzora

task main()
 {
 float mid = (max - min)/2;              // srednja vrednost - zeljena
                                         // velicina
 SetSensorType(IN_1, IN_TYPE_COLORRED);
 SetSensorMode(IN_1, IN_MODE_RAW);
 float Kpmax = 0.466;                    // Kriticno pojacanje
 float Ts = 0.8;                         // Vreme smirenja
 float dt = 0.00038;                     // Vreme izvrsenja jedne petlje
 float Kp = 0.45*Kpmax;                  // Proporcionalna konstanta
 float Ki = 1.2*Kp*dt/Ts;                // Integraciona konstanta
 float Sum = 0;                          // suma gresaka = 0 u pocetku
 while (true)
 {
 int y = SensorValue(S1);                // trenutno ocitavanje senzora
 int e = mid - y;                        // funkcija greske
 Sum += e;                               // suma gresaka
 int Sr = Kp*e + Ki*Sum;                 // Steering Rate
 OnFwdSync(OUT_AC, 50, Sr);              // Zakon upravljanja, motori
                                         // A,C, brzina=50,
                                         // Steering Rate = Sr

}
 }
```

PROGRAMSKI KOD PI regulatora – upravljanje brzinama motora

```
#define max 582 // maximalna vrednost ocitavanja senzora
#define min 153 // minimalna vrednost ocitavanja senzora
#define V0 50
task main()
{
float mid = (max - min)/2;            // srednja vrednost – zeljena
                                      // velicina
SetSensorType(IN_1, IN_TYPE_COLORRED);
SetSensorMode(IN_1, IN_MODE_RAW);
float Kpmax = 0.233;                  // Kriticno pojacanje
float Ts = 0.8;                       // Vreme smirenja
float dt = 0.000475;                  // Vreme izvrsenja jedne petlje
float Kp = 0.45*Kpmax;                // Proporcionalna konstanta
float Ki = 1.2*Kp*dt/Ts;              // Integraciona konstanta
float Sum = 0;                        // suma gresaka = 0 u pocetku
while (true)
{
 int y = SensorValue(S1);     // trenutno ocitavanje sa senzora
 float e = mid - y;           // funkcija greske
 Sum += e;                     // suma gresaka
 float u = Kp*e + Ki*Sum;     //
 float Vl = V0 + u;           // Brzina levog tocka
 float Vd = V0 - u;           // Brzina desnog tocka
OnFwd(OUT_A, Vl);
OnFwd(OUT_C, Vd);
}
}
```

PROGRAMSKI KOD PID regulatora – upravljanje Steering rate

```
#define max 582 // maximalna vrednost ocitavanja senzora
#define min 153 // minimalna vrednost ocitavanja senzora

task main()
{
int mid = (max - min)/2;              // srednja vrednost – zeljena
                                      //velicina
SetSensor(IN_1, SENSOR_TYPE_LIGHT_ACTIVE); // podesavanje porta i
                                           // tipa senzora
SetSensorMode(IN_1, SENSOR_MODE_RAW);
float Kpmax = 0.466;                    // Kriticno pojacanje
float Ts = 0.8;                         // Vreme smirenja
float dt = 0.00038;                     // Vreme izvrsenja jedne
                                           petlje
float Kp = 0.6*Kpmax;                 // Proporcionalna konstanta
float Ki = 2*Kp*dt/Ts;                  // Integraciona konstanta
float Kd = 0.125*Kp*Ts/dt;              // Diferencijalna konstanta
int Sum = 0;                            // suma gresaka = 0 u pocetku
int LastE = 0;                          // poslednja greska u t0 je 0
while (true)
{
int y = SensorValue(S1);                // trenutno ocitavanje senzora
float e = mid - y;                      // funkcija greske

float Dif = e - LastE;                  // razlika sadasnje i greske
                                        // u prethodnoj petlji
Sum += e;                               // suma gresaka
```

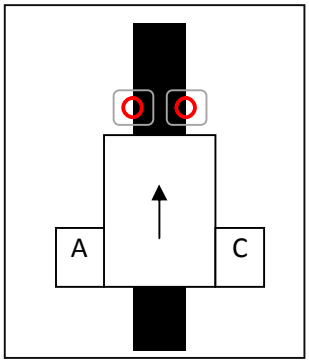
```
int Sr = Kp*e + Ki*Sum + Kd*Dif;        // Steering Rate
OnFwdSync(OUT_AC, 50, Sr);              // Zakon upravljanja, motori
                                        // A,C, brzina=50,
                                        // Steering Rate = Sr

LastE = e;
}}
```

Zaključak

Najbolje ponašanje robota, odnosno najbolji odziv sistema je dao algoritam za PI kontroler upravljanje veličinom *Steering rate.*
Pri ostalim algoritmima se ponašao nestablino, pravio je velika odstupanja od putanje. Kod algoritama koji su upravljali motorima ponašanje je bilo najnestabilnije, u nekom trenutku motori bi počeli da se okreću u suprotnom smeru (PI i PID). To je posledica pojave *Integral Windup*.
Kontroler koji poseduje integralni deo (PI ili PID kontroler), u toku njegovg rada signal greške se sabira, integralni deo kontrolera će se povećavati shodno zbiru prethodnih grešaka i veličini K_i. U nekom trenutku vremena ovo će dovesti do zasićenja, odnosno upravljana veličina će dostići maksimalnu ili minimalnu vrednost – ova pojava se naziva *Intregral Windup.*
Još jedna stvar koja može da utiče na ponašanje robota jeste i vreme oscilovanja, koje nismo uspeli da utvrdimo algoritamski pa smo usvojili vrednost na osnovu preporuke [6].

LINE FOLLOWER robot sa dva senzora

Na slici 12.4 je prikazan Line Follower robot sa dva senzora.



Slika 12.4 Line Follower robot u stacionarnom stanju

Imamo tri referentne veličine:

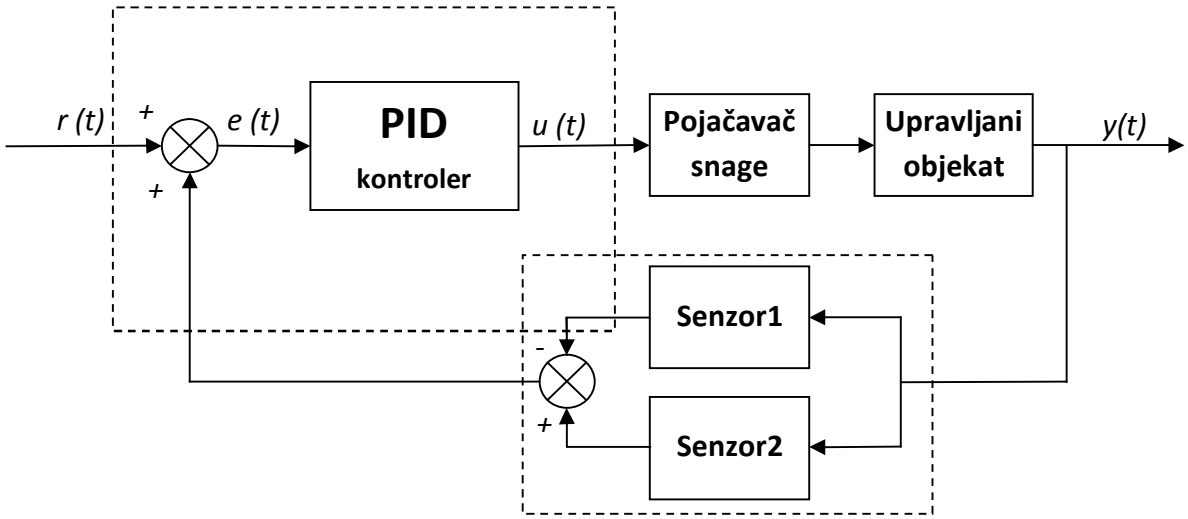
- Očitavanje desnog senzora
- Očitavanje levog senzora
- Razlika u očitavanju

Na osnovu referentnih veličina mozemo napraviti tri različita sistema upravljanja:

1) Upravlja se motorima zajedno, gde je Steering rate proporcionalan grešci, tj razlici očitavanja senzora.
2) Upravlja se svakim motorim posebno, gde je brzina odgovarajućeg motora proporcionalna razlici očitavanja senzora.
3) Upravlja se svakim motorom posebno, gde je brzna odgovarajućeg motora proporcionalna razlici očitavanja odgovarajućeg senzora i željene veličine.

Na osnovu prethodnog primera (slučaj Line follower robota sa jednim senzorom) odlučili smo se za prvi slučaj – upravljanje veličinom Steer rate, jer se u prethodnom primeru pokazalo, kod PI i PID regulatora a gde se upravljalo svakim motorom posebno, da se integralni deo konstantno povećava i ulazi u negativnu vrednost, te njegova veličina znatno premašuje P i D veličine pa se zbog toga javljalo da motori u jednom trenutku počinju da rade u suprotnom smeru od predviđenog.

Za ovaj sistem upravljanja motorima zajedno blok dijagram sistema je prikazan na slici 12.5. Sistem teži da razlika očitavanja senzora bude jednaka nuli, tj, *r(t)* = 0.



Slika 12.5 Blok dijagram Line Follower robota sa dva senzora

Eksperimentalno je utvrđeno da je maksimalna razlika u očitavanju: max = 450 pa je prema jednačini (7) K_{pMAX} = 0.222.

PROGRAMSKI KOD za utvrđivanje maksimalne razlike očitavanja senzora

```
// program izracunava maksimalnu razliku ocitavanja senzora
// i prikazuje je na displeju
int L;
int D;
int max = 0;
int dif;
task main()
{
   SetSensorType(IN_1, IN_TYPE_COLORRED);
   SetSensorType(IN_2, IN_TYPE_COLORRED);
   SetSensorMode(IN_1, IN_MODE_RAW);
   SetSensorMode(IN_2, IN_MODE_RAW);
   Wait(SEC_1);

   while (true)
  {
    D = Sensor(IN_1);
    L = Sensor(IN_2);
    dif = L - D;
    if (dif > max)
```

```
    {
      max = dif;
    }
    NumOut(10, LCD_LINE1, L);
    NumOut(10, LCD_LINE2, D);
    NumOut(10, LCD_LINE3,max);
    NumOut(10, LCD_LINE4,dif);
  }
}
```

Na osnovu preporuka [6] i ranije utvrđene vrednosti vremena izvršenja jednje petlje usvojili smo sledeće parametre po Ziegler-Nichols metodi:

Vrsta kontrolera	K_p	K_I	K_d
P	0.111		
PI	0.1	0.000015	
PID	0.133	0.0000337	131.44
PID*	0.113	0.0000337	1.3144

Tabela 8.3 Vrednosti konstati PID regulatora prema Ziegler–Nichols metodi

* Poslednje parametre PIDa smo utvrdili računski, osim parametra K_d koga smo umanjili za 100 puta od onog koji je dobijen računskim putem.

PROGRAMSKI KOD P regulatora

```
// upravlja se sa Steering opcijom
int L;
int D;
int dif = 0;
task main()
 {
   SetSensorType(IN_1, IN_TYPE_COLORRED);
   SetSensorType(IN_2, IN_TYPE_COLORRED);
   SetSensorMode(IN_1, IN_MODE_RAW);
   SetSensorMode(IN_2, IN_MODE_RAW);
   Wait(SEC_1);
   float Kp = 0.111;                  // Proporcionalna konstanta

 while (true)
 {
   L = SensorValue(S1);               // trenutno ocitavanje sa senzora
   D = SensorValue(S2);
   dif = L - D;                       // funkcija greske
   float Sr = Kp*dif;                 // Steering Rate
   OnFwdSync(OUT_AC, 50, Sr);         // Zakon upravljanja, motori A,C,
                                      // brzina=50,
                                      // Steering Rate = Sr
 }
 }
```

PROGRAMSKI KOD PI regulatora

```
// upravlja se sa Steering opcijom
int L;
int D;
int dif = 0;
int Sum = 0;
task main()
 {
   SetSensorType(IN_1, IN_TYPE_COLORRED);
   SetSensorType(IN_2, IN_TYPE_COLORRED);
   SetSensorMode(IN_1, IN_MODE_RAW);
   SetSensorMode(IN_2, IN_MODE_RAW);
   Wait(SEC_1);
   float Kp = 0.1;              // Proporcionalna konstanta
   float Ki = 0.000015;
 while (true)
 {
   L = SensorValue(S1);         // trenutno ocitavanje sa senzora
   D = SensorValue(S2);
   dif = L - D;                 // funkcija greske
   Sum += dif;
   float Sr = Kp*dif + Ki*Sum;  // Steering Rate
   OnFwdSync(OUT_AC, 50, Sr);   // Zakon upravljanja, motori A,C,
                                // brzina=50,
                                // Steering Rate = Sr
 }
 }
```

PROGRAMSKI KOD PID regulatora

```
// upravlja se sa Steering opcijom
int L;
int D;
int dif = 0;
int Sum = 0;
int LastE = 0;
int E;
task main()
 {
   SetSensorType(IN_1, IN_TYPE_COLORRED);
   SetSensorType(IN_2, IN_TYPE_COLORRED);
   SetSensorMode(IN_1, IN_MODE_RAW);
   SetSensorMode(IN_2, IN_MODE_RAW);
   Wait(SEC_1);
   float Kp = 0.1;                        // Proporcionalna konstanta
   float Ki = 0.00003374;
   float Kd = 1.3144;   // ovo je rucno podesena konstanta
                        // racunska je 100 puta veca
 while (true)
 {
   L = SensorValue(S1);              // trenutno ocitavanje sa senzora
   D = SensorValue(S2);
```

```
    dif = L - D;                        // funkcija greske
    E = dif - LastE;
    Sum += dif;
    float Sr = Kp*dif + Ki*Sum + Kd*E;  // Steering Rate
    OnFwdSync(OUT_AC, 50, Sr);          // Zakon upravljanja, motori A,C,
                                        //brzina=50,
                                        // Steering Rate = Sr
    LastE = dif;
  }
  }
```

Zaključak

Na osnovu gore predstavljenog možemo zaključiti da je ponašanje sistema u kome postoji samo jedan svetlosni senzor nestabilnije i više sklono ulasku u zasićenje odakle ga nije moguće vratiti u stabilnu zonu.

Na osnovu pozicionog algoritma:

$$u(n) = K_p e(n) + K_i \sum_{j=0}^{n} e(j) + K_d(e(n) - e(n-1))$$

vidi se uticaj veličina e(n) i e(n-1). Kod sistema sa jednim senzorom u veličinu e(n), funkcija greške ulazi kao razlika unapred zadate srednje vrednosti i očitavanje senzora pa u slučaju velikog zakrivljenja putanje sistem lako odlazi u stanje .

Za razliku od njega sistem sa dva senzora u veličinu e(n) vrednost funkcije greške ulazi kao razlika očitavanja sa oba senzora i na taj način sistem ima dobar odziv pri oštrim zakrivljenjima linije koja se prati. Pri delovima linije koja je bez zakrivljenja, pid regulacija vrši upravljanje tačno po zadatoj putanji. U slučaju promene širine putanje koja je zadata za praćenje, sistem sa dva senzora je indiferentan u odnosu na tu promenu, za razliku od sistema sa jednim senzorom.

Zaključak

Lego MINDSTORMS NXT predstavlja odličan početak u oblasti programiranja robota zbog svoje jednostavnosti, međutim nije podesan za složenije programiranje zbog ograničenih mogućnosti NXT bloka. Za složenije upravljanje predlaže se korišćenje kompjutera, na taj način što će NXT blok komunicirati sa računarom preko bluetooth-a. Takođe, ogromna prepreka u programiranju i izvršanjanju zadataka predstavlja velika potrošnja baterija, i različito ponašanje robota pri snazi na baterijama manjoj od 80%.

Lego NXT program ima jednostavan interfejs i lak je za korišćenje, ali ga treba izbegavati kada su u pitanju složeniji zadaci, te se u tu svrhu može koristiti NXC program. NXC program za razliku od NXT programa nije vizuelan, ima slične komande kaoprogramski jezik C++, i može da računa sa decimalnim brojevima.

Literatura

[1] Michael Gasperi, Philippe Hurbain, Isabelle Hurbain. Extreme NXT: Extending the LEGO MINDSTORMS NXT to the Next Level ©2007. ISBN-13 (pbk): 978-1-59059-818-4

[2] James Floyd Kelly. LEGO MINDSTORMS NXT-G Programming Guide ©2007. ISBN-13 (pbk): 978-1-59059-871-9

[3] Owen Bishop. Programming LEGO Mindstorms NXT ©2007

[4] Tomáš Belík. Využití robota Lego Mindstorms - Návrh a realizace speciálních úloh. CESKÉ VYSOKÉ UCENÍ TECHNICKÉ V PRAZE 2010.

[5] *FPGA Implementation of Closed-Loop Control.* Zhao, Wei, et al. Minneapolis : University of Minnesota, 2005.

[6] Sluka, J. A PID Controller For Lego Mindstorms Robots. [Online] 2009. http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html.