Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство

Ниво студија: Мастер академске студије

Модул: Примењена механика и аутоматско управљање

Предмет: Пројектовање система аутоматског управљања

Број индекса: 338/2013

Жељко С. Крстић

- Имплементација контролера LEGO Segway мобилног робота у програмском језику Python-

Мастер рад

Комисија за преглед и одбрану:

1. Проф др Милан Матијевић - ментор Датум одбране: 14.07.2017.

2. Прод др Гордана Богдановић Оцена: _____

3. Проф др Јованка Лукић

У оквиру овог дипломског - мастер рада кандидат треба да:

Опише синтезу контролера segway мобилног робота заснованог на моделу LEGO робота изабране структуре. Добијени математички модел треба софтверски имплементирати у програмском језику Python. Извршити експерименталну верификацију рада пројектованог система, објаснити добијене резултате и метрологију писања и имплементације управљачког софтвера.

Препоручена литература:

[1] М. Матијевић, Г. Јакуповић, Ј. Цар, Рачунарски подржано мерење и управљање, Машински факултет, Крагујевац, 2005

[2] Y.Yamamoto, "NXTway-GS (Self-Balancing Two-Wheeled Robot) Controller Design", Mathworks 2008. http://www.mathworks.com/matlabcentral/fileexchange/19147- nxtway-gs--self-balancing-two-wheeled-robot--controller-design

[3] M.Matijević, V.Cvjetković, V.Filipović, N.Jović, "Osnovni koncepti automatike i mehatronike sa LEGO programabilnimm robotom", Tehnika, vol. 4/2014, 2014.

[4] Yamamoto, Nottaway-GS Model Based Design – Control of self-balancing two-wheeled robot built with LEGO Mindstorms NXT, http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=13399&objectType=file, 2008.

САДРЖАЈ

РЕЗИМЕ

У овом раду приказано је пројектовање контролера Lego Segway робота, коришћењем Yamamoto – vog оргиналног кода у Матлабу. На почетку рада дат је преглед карактеристика робота (сервомотор, жироскоп, сензори за боју, додир...) а затим и физичке карактеристике робота. Математички модел робота дат је у облику модела у простору стања на основу чега су, помоћу повратне спреге по стању израчуната појачања серво контролера који управља балансирањем и кретањем робота. Управљачки програм написан је у Python програмском језику чије функције су објашњене и повезане са блоковима из симулинк модела. Након објашњења python програма следе објашњења повезивања робота са рачунаром, стартовања програма са робота и део који се односи на чување података и исцртавање дијаграма вредности добијених са сензора.

Кључне речи: Lego Segway, Повезивање робота, Чување података, Python

ABSTRACT

In this paper work is presented the design of the Lego Segway robot controller using Yamamoto`s code in Matlab. At the beginning of the work I present characteristics of the robot like servomotor, gyroscope, color sensor, touch sensor. After this I present the physical characteristic of system. The mathematical model of the robot is given in the form of a model in the state space and with feedback we calculate enhancement of the servo controller. Function of this controller is to control balancing and motion of robots. Program is written in a Python programming language which functions are explained and associated with blocks from the simulink model. After explaining the python program I show the explanations how to connect robot with computer and how we start the program from robot `cube`. Last part of work consists of diagrams which we plot with date collected from sensors.

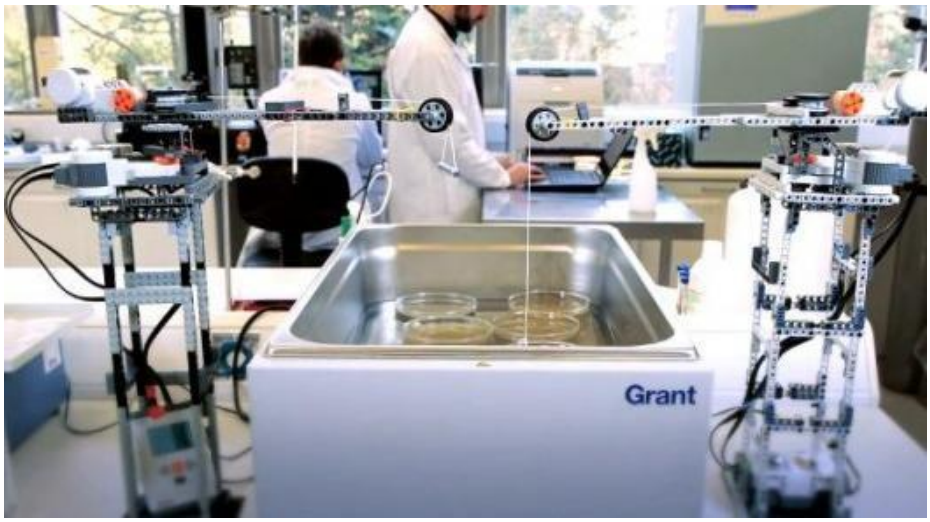
Key words: Lego Segway, Connecting robots, Data storage, Python

1. УВОД

Једна од карактеристика данашњице јесте да се сваким даном све више и више сусрећемо и суочавамо са напретком науке и технике. У последњих неколико година тај раст се повећавао застрашујућим темпом. Људи су постали зависници технологије и данас се не може замислити дан без могућности које технологија може да пружи. Брз развој науке и технике можемо назвати и фазом револуционарних промена. Повећање протока информација изазвао је брз развој рачунарских и комуникационих система док је индустрија због појаве аутоматизације човека ставила у положај надгледања производње. Тежња да се повећа продуктивност у индустријској производњи подразумевала је све већу аутоматизацију. Оваква аутоматизација се назива флексибилна аутоматизација која подразумева роботски систем. Када се каже робот, помисли се на научну фантастику међутим данас се на роботе гледа другачије. Њих посматрамо као уређаје који олаксавају рад са људима, првенствено на опасним и тешким пословима док се човеку остављају могућности и послови који траже знање, креативност и интелигенцију.

2. ПРИМЕНА ЛЕГО РОБОТА

Лего роботика је скуп малих делова који личе на лего коцке с тим што такви делови садрже машинска и електротехничка својства. Први лего комплет под називом RIS (Robotics Invention System) појавио се у фебруару 2000. године а лего контролер је носио назив RCX (Robotic Command eXplorers). Новија верзија лего робота која се појавила 2006.године носи ознаку NXT. Ова верзија је побољшана јер се поред основног дела робота налазе три серво мотора, четири сензора (сензор за додир, светлост, звук и растојање). Лего роботи који припадају трећој генерацији носе ознаку EV3. Трећа генерација лего робота се појавила септембра 2013.године. Свака од ових верзија садржи мозак или контролер робота. То је миркорачунар помоћу кога се контролише систем који командује роботу коју ће радњу извршавати. Роботи замењују човека на опасним, тешким и монотоним пословима. Роботи могу да реализују сложене задатке веома поузадно, без грешке, без замора и могу да раде онолико дуго колико је то потребно а може се и вршити промена задатака које треба да обавити. Лего роботи се могу користити за медицинска испитивања слика 1. Такође могуће је пројектовати лего робот који ће старије људе опомињати у одређеним тренуцима да узимају дневну дозу лекова. По боји гласа може се препознати више корисника и на основу тога да распореди дозу лекова за више особа. Велику примену су нашли и у индустрији јер је пренос материјала у неким производњама кључња радња. Пренос од једне машине до друге или са једне стране на другу страну покретне траке је битан да се уради прецизно и без застоја. Један од задатака које лего робот може да извршава је да разврстава предмете по боји. Лего роботи се користе за едукацију о аутоматизованим процесима који ће представљати будућност у роботици. Програмирање лего робота најчешће се може срести у основним и средњим школама као и на факултетима. Деца, ђаци и студенти кроз занимљиве теме и игре могу лако научити основне ствари роботике а и шире. Програмирање робота и контакт са њима омогућују квалитетније учење а и истовремено могу да осободе ђака или студента притиска који осећају приликом традиционалних учења.



Слика 1. Лего роботи у медицини

3. ЛЕГОПРОГРАМАБИЛНИ РОБОТ

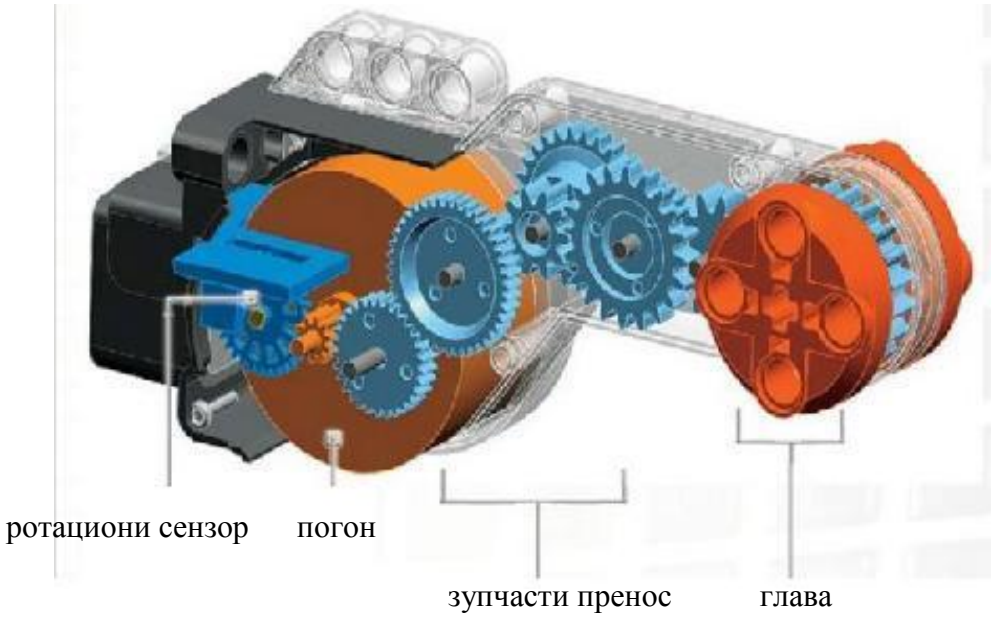
Основни LEGO Mindstorms NXT који се састоји од програмабилне јединице (цигле), два сензора за додир, сензора за боју/количину светлости, ултразвучног сензора растојања, три DC мотора са интегрисаним енкодером и редуктором приказан је на слици 2. Популарна „цигла“ се састоји из следећих компоненти:

- Главни процесор радне фрекфенције 48MHz, 32 – битни ARM микроконтролер Atmel AT91SAM – 7S256, флеш меморија 256 KB, RAM меморија 64KB и USB интерфејс;
- Помоћни процесор радне фреквенције 8MHz, 8 – битни Atmel AVR процесор Atmega48, флеш меморија 4KB и 512 B RAM меморија;
- 4 аналогна улаза за сензоре
- 3 PWM излаза за побуду LEGO DC мотора са каналом за читање енкодера интегрисаних у кућиште LEGO DC мотора;
- Бежични Bluetooth CSR BlueCoreTM 4 v2.0 +EDR систем који подржава серијски улаз, 47KB RAM меморије, 8MB флеш меморије, радни такт 26MHz;
- USB 2.0 за размену података максималном брзином од 12 Mbit/s;
- LCD монохроматски дисплеј резолуције 100x64 pi-ksela;
- Звучник 2-16KHz са 8-bitnom резолуцијом;
- Напајање, које чини 6 AA батерија (9V).

Слика 2. LEGO Mindstorms NXT 8547 комплет: програмабилна јединица, три DC мотора са интегрисаним енкодерима, два сензора за додир, сензор за боју или количину светлости, ултразвучни снзор растојања.

3.1. Сервомотор

Сервомотор се састоји из погона – мотора, осам зупчаника, ротационог сензора и главе сервомотора (слика 3.) Због преносног односа зупчаника, ротациони сензор може да детектује 1° ротације главе серво мотора.



Слика 3. DC мотор

Технички параметри мотора:

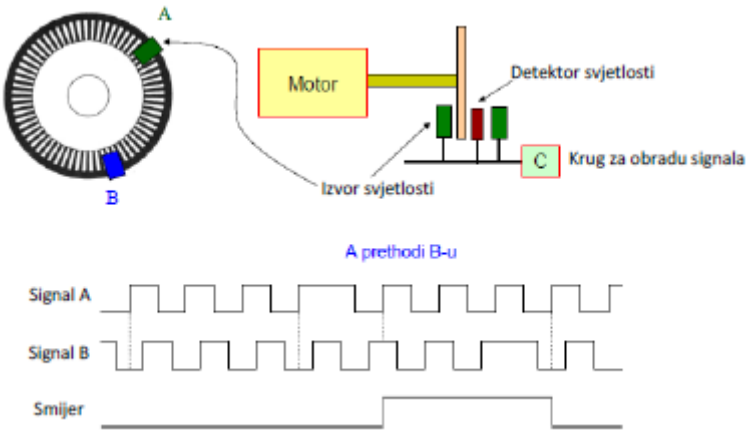
- Напајање 9V
- Угаона брзина 177 min^{-1}
- Обртни момент 16.7 Ncm
- Снага 2.03 W
- Ефикасност 41%

Серво мотори NXT 2.0 мобилног робота дају роботу могућност кретања и извршавања постављаних задатака претварајући електричну енергију из батерија у механичку енергију тј. ротацију осовине мотора.

Код мобилног робота LEGO NXT 2.0 користе се три серво мотора. Два погонска мотора директно везана на точкове робота, а трећи мотор служи за ротацију ултразвучног сензора удаљености што омогућује роботу да мери удаљеност од препреке у различитим смеровима. Мотори су преко 6 жичаног спојног кабла спојени на излазне прикључке. Ово је врста једносмерног мотора који су преко Lego NXT компоненте управљани по положају и по снази и могуће је им је задати смер окретања мотора. Регулација окретања је изведена преко регулацијске петље положаја с повратном спрегом и PID регулатором. Жељена снага се задаје у процентима максималне снаге а управљање снагом се врши PWM сигналом.

Директно управљање брзином обртања мотора није могуће преко NXT компоненте. Коначна средња брзина обртања мотора је резултат тренутног момента терета којим је мотор оптерећењ и тренутног стања напајања мотора тј. робота. Међутим, упркос недостатку директног управљања брзином окретања мотора на располагању је управљање положајем осовине мотора што омогућује прецизно програмирање увијања робота и обављање сложених задатака.

Инкрементални енкодери који су уграђени у кућиште серво мотора служе за мерење заокрета осовине мотора. Овај тип енкодера се састоји од извора светлости, ротирајућег диска с отворима за пролазак светла, детектора светлости и процесора који обрађује добијени сигнал слика 4.



Слика 4. Принцип рада отичког инкременталног енкодера

Како се ротирајући диск окреће тако се сноп светлости прекида или пропушта. Фото детектор детектује испрекидане светлосне сигнале и претвара их у електрични сигнал. Процесор обрађује добијене сигнале и даје информације о окретању осовине мотора.

Инкрементални енкодери код мобилног робота Lego NXT 2.0 користе два сигнала која су међусобно фазно померена како би се одредио смер и обртање мотора. Помоћу тих енкодера у сваком тренутку можемо добити информацију о релативном окретању осовине мотора с тачношћу од ± 1 цм.

3.2. Сензор за додир

NXT сензор за додир је приказан на слици 5. Добра карактеристика је та што има крстасти отвор на средини који омогућава да се сензор директно повеже са осталим деловима. Сензор за додир се састоји од штампаног кола на којем се налази повратни прекидач и конектор. Са прекидачем је редно повезан отпорник, да се не би створио кратак спој у случају ако се сензор повеже на излазни порт.

Слика 5. Сензор за додир

3.3. Ултразвучни сензор

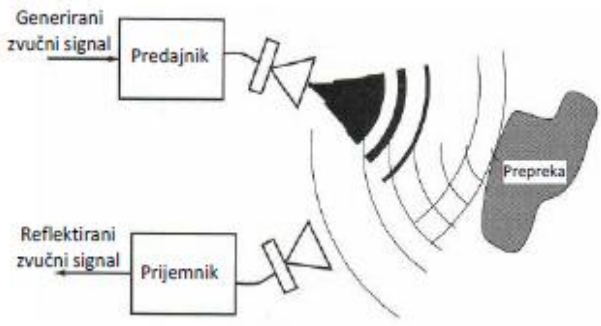
Ултразвучни сензор је приказан на слици 6. Због своје комплексности, има свој посебан микропроцесор, због чега враћа измерену даљину у апсолутним јединицама. Сензор ради по принципу сонара, шаљући краткотрајне сигнале око 40kHz, затим мери време које је потребно да звук оде и одбије се од објекта. Очитавања су боља када је објекат од којег се звук одбија већи, ако је окружење робота компликованије, са мањим објектима, мерење тада постаје мање поуздано. Не препоручује се истовремено коришћење више ултразвучних сензора због могуће интерференце звучних таласа. Апсолутна грешка при мерењу износи ± 3 цм.

Слика 6. Ултразвучни сензор

Главне карактеристике ултразвучних сензора су:

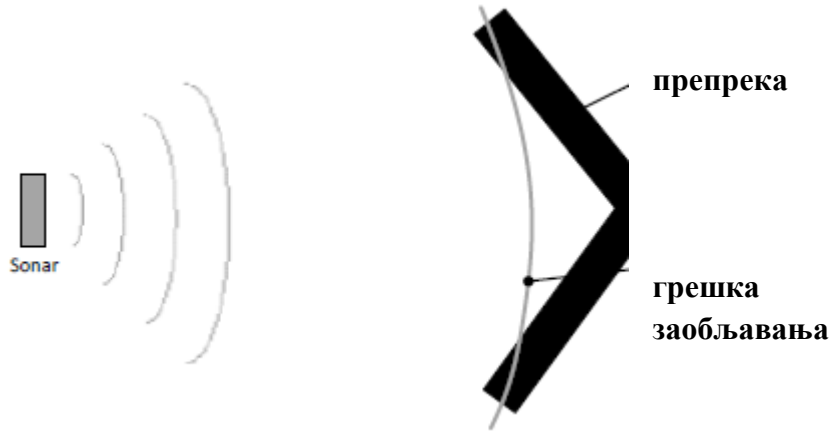
- Минимална мерена удаљеност
- Максимални домет
- Ширина зрака сензора
- Прецизност мерења удаљености

На слици 7. приказан је принцип рада ултразвучног сензора. Минимална мерена удаљеност зависи од трајања слања ултразвучног импулса. Максимални домет одређен је интезитетом ширења сигнала који опада са удаљеношћу и зависи од самог медија којим путују сигнали од температуре и влажности. Тачност мерења удаљености одређен је променама брзине звука услад различите температуре и влажности околине.



Слика 7. Принцип рада ултразвучног сензора

Код употребе ултразвучног сензора треба обратити пажњу на неке од проблема који се јављају у пракси. Један од њих је проблем заобљавања углова тј. немогућност сензора да детектује углове код препрека, а последица је интеграције интезитета зрака преко читавог простора рефлексије зрака слика 8.



Слика 8. Проблем заобљавања углова код ултразвучног сензора

3.4. Сензор за боју

NXT сензор за боју може да обавља три различите функције; може да ради као сензор за боју, при чему разликује шест боја (црну, плаву, зелену, жуту, црвену, белу); може да ради као сензор за јачину светла (рефлектованог и амбијенталног); и да ради као лампа, при чему може емитовати црвено, зелено или плаво светло. Плаво индикацијско светло означава да робот мирује, да је комуникација са рачунаром успостављена и да је робот спреман прихватити и извршити наредбу. Зелено значи да је робот у покрету а црвено да је робот ударио у препреку.

3.5. Жироскоп

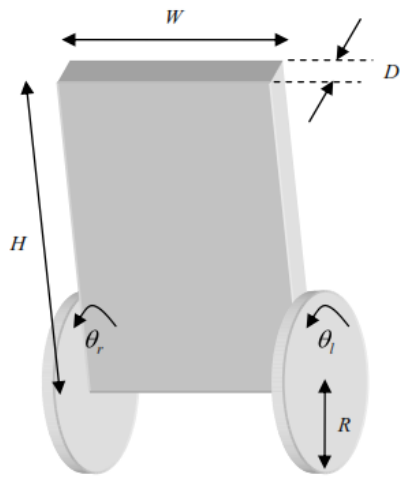
Жироскоп је уређај за мерење промене угла. Свој рад занива на ефекту интерференције светлости. Најчешће се користи у инерцијалним навигационим системима. NXT жироскоп може да мери ротацију угла +/- 360°. Конектује се са роботом стандардним nxt каблом. Више о жироскопу биће у даљем делу рада.

Слика 9. Жироскоп

4. ЛЕГОСАМОБАЛАНСИРАЈУЋИ РОБОТ – ФИЗИЧКЕ КАРАТЕРИСТИКЕ

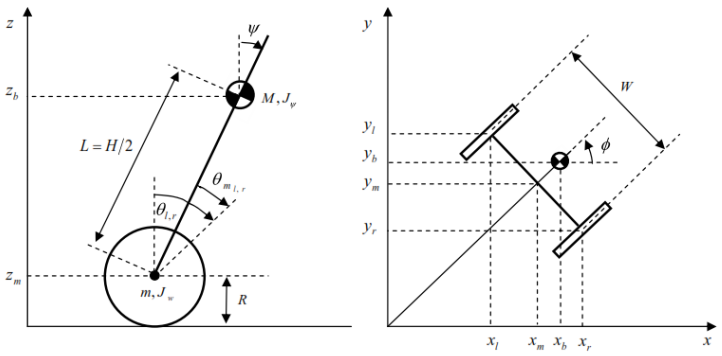
4.1. Модел система у простору стања

Физичке карактеристике мобилног робота знатно утичу на сложеност алгоритма. Мобилни робот према својим димензијама спада у групу малих робота. LEGO самобалансирајући робот се може посматрати као модел инверзног клатна на два точка слика 10.



Слика 10. Модел инверзног клатна на два точка

Слика 11. показује поглед са стране и одозго на модел инверзног клатна. Генералисане координате овог система су: ψ - вертикални нагиб клатна, $\theta_{l,r}$ - позициони угао левог и десног точка респективно, $\theta_{ml,r}$ - позициони угао левог и десног мотора респективно и φ - угао заокретања клатна у хоризонталној равни.



Слика 11. Модел инверзног клатна (поглед са стране и одозго)

Физички параметри самобалансирајућег робота који се користе су:

- $г = 9{,}81$ m/sec^2
- $м = 0{,}015$ kg – маса точка
- $P = 0{,}02$ m – полупречник точка
- $J_w = mR^2 / 2$ - момент инерције точка
- $M = 0{,}64$ kg – маса робота (без точкова)
- $W = 0{,}14$ m – ширина робота
- $Д = 0{,}04$ m – дебљина робота
- $X = 0{,}16$ m – висина робота
- $Л = H / 2$ - одстојање центра точка од тежишта
- $J_\psi = ML^2 / 3$ - момент инерције за вертикално нагињање
- $J_\phi = M\left(W^2 + D^2\right)/12$ - момент инерције за хоризонтално заокретање
- $J_m = 1\times10^{-5}$ kgm^2 - момент инерције ротора мотора
- $R_m = 6.69\Omega$ - статички електрични отпор мотора
- $K_b = 0,468$ - коефицијент повратне ЕМС
- $K_l = 0,317$ - коефицијент момента мотора
- $n = 1$ - однсос зупчаника
- $f_n = 0,0022$ коефицијент трења између мотора и тела робота
- $f_w = 0$ - коефицијент трења између точка и подлоге

Једначине кретања инверзног клатна се изводе коришћењем Лагранжеових једначина за основу координата са слике 11. Ако је смер инверзног клатна у позитивном смеру x осе у тренутку $m = 0$, координате се рачунају као:

$$(\theta, \quad \phi) = \left(\frac{1}{2}(\theta_l + \theta_r) \quad \frac{R}{W}(\theta_r - \theta_l) \right) \tag{1}$$

$$(x_m, \quad y_m, \quad z_m) = \left(\int \dot{x}_m dt, \quad \int \dot{y}_m dt, \quad R \right) \tag{2}$$

$$(\dot{x}_m, \quad \dot{y}_m) = \left(R\dot{\theta}\cos\phi \quad R\dot{\theta}\sin\phi \right) \tag{3}$$

$$(x_l, \quad y_l, \quad z_l) = \left(x_m - \frac{W}{2}\sin\phi, \quad y_m + \frac{W}{2}\cos\phi, \quad z_m \right) \tag{4}$$

$$(x_r, \quad y_r, \quad z_r) = \left(x_m + \frac{W}{2}\sin\phi, \quad y_m - \frac{W}{2}\cos\phi, \quad z_m \right) \tag{5}$$

$$(x_b, \quad y_b, \quad z_b) = (x_m + L\sin\psi\cos\phi, \quad y_m + L\sin\psi\sin\phi, \quad z_m + L\cos\psi) \tag{6}$$

Транслаторна кинетичка енергија T_1, кинетичка енергија ротације T_2 и потенцијална енергија $У$ су:

$$T_1 = \frac{1}{2}m\left(\dot{x}_l^2 + \dot{y}_l^{\,2} + \dot{z}_l^{\,2}\right) + \frac{1}{2}m\left(\dot{x}_r^{\,2} + \dot{y}_r^{\,2} + \dot{z}_r^{\,2}\right) + \frac{1}{2}M\left(\dot{x}_b^{\,2} + \dot{y}_b^{\,2} + \dot{z}_b^{\,2}\right) \tag{7}$$

$$T_2 = \frac{1}{2}J_w\dot{\theta}_l^{\,2} + \frac{1}{2}J_w\dot{\theta}_r^{\,2} + \frac{1}{2}J_\psi\dot{\psi}^2 + \frac{1}{2}J_\phi\dot{\phi}^2 + \frac{1}{2}n^2J_m\left(\dot{\theta}_l - \dot{\psi}\right)^2 + \frac{1}{2}n^2J_m\left(\dot{\theta}_r - \dot{\psi}\right)^2 \tag{8}$$

$$U = mgz_l + mgz_r + Mgz_b \tag{9}$$

Пети и шести сабирак у T_2 су кинетичка енергија ротације ротора левог и десног мотора. Лагранжијан $Л$ се добија следећом једначином:

$$L = T_1 + T_2 - U \tag{10}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = F_\theta \tag{11}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\psi}}\right) - \frac{\partial L}{\partial \psi} = F_\psi \tag{12}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\phi}}\right)-\frac{\partial L}{\partial \phi}=F_{\phi} \tag{13}$$

Једначине (11) до (13) дају:

$$\left[\left(2m+M\right)R^2+2J_w+2n^2J_m\right]\ddot{\theta}+\left(MLR\cos\psi-2n^2J_m\right)\ddot{\psi}-MLR\dot{\psi}^2\sin\psi=F_{\theta} \tag{14}$$

$$\begin{gathered}\left(MLR\cos\psi-2\mathrm{n}^2\,J_m\right)\ddot{\theta}+(ML^2+J_{\psi}+2n^2J_m)\ddot{\psi}-\\-MgL\sin\psi-ML^2\phi^2\sin\psi\cos\psi=F_{\psi}\end{gathered} \tag{15}$$

$$\begin{gathered}\left[\frac{1}{2}mW^2+J_{\phi}+\frac{W^2}{2R^2}\left(J_w+n^2J_m\right)+ML^2\sin^2\psi\right]\ddot{\phi}+\\+2ML^2\dot{\psi}\dot{\phi}\sin\psi\cos\psi=F_{\phi}\end{gathered} \tag{16}$$

Узимајући у обзир момент и вискозно трење мотора, генералисане силе се одређују као:

$$\left(F_{\theta},\quad F_{\psi},\quad F_{\phi}\right)=\left(F_l+F_r,\quad F_{\psi},\quad \frac{W}{2R}\left(F_r-F_l\right)\right) \tag{17}$$

$$F_l=nK_ti_l+f_m\left(\dot{\psi}-\dot{\theta}_l\right)-f_w\dot{\theta}_l \tag{18}$$

$$F_r=nK_ti_r+f_m(\dot{\psi}-\dot{\theta}_r)-f_w\dot{\theta}_r \tag{19}$$

$$F_{\psi}=-nK_ti_l-nK_ti_r-f_m\left(\dot{\psi}-\dot{\theta}_l\right)-f_m\left(\dot{\psi}-\dot{\theta}_r\right) \tag{20}$$

где је $u_{л,р}$ представља јачину струје кроз моторе.

Пошто је сигнал управљања напонски PWM сигнал, не можемо користити струју мотора директно. Стога је потребно одредити однос струје $u_{л,р}$ и напона $в_{л,р}$ коришћењем једначина мотора. Ако занемаримо трење у мотору, једначина мотора једносмерне струје постаје:

$$L_m\dot{i}_{l,r}=v_{l,r}+K_b\left(\dot{\psi}-\dot{\theta}_{l,r}\right)-R_mi_{l,r} \tag{21}$$

Сада, ако занемаримо и индуктивност мотора, струја мотора постаје:

$$i_{l,r}=\frac{v_{l,r}+K_b\left(\dot{\psi}-\dot{\theta}_{l,r}\right)}{R_m} \tag{22}$$

Из једначине (22), генералисана сила се може изразити преко напона на мотору:

$$F_\theta = \alpha\left(v_l + v_r\right) - 2\left(\beta + f_w\right)\dot{\theta} + 2\beta\dot{\psi} \tag{23}$$

$$F_\psi = -\alpha\left(v_l + v_r\right) + 2\beta\dot{\theta} - 2\beta\dot{\psi} \tag{24}$$

$$F_\phi = \frac{W}{2R}\alpha\left(v_l - v_r\right) - \frac{W^2}{2R^2}\left(\beta + f_w\right)\dot{\phi} \tag{25}$$

$$\alpha = \frac{nK_t}{R_m}, \quad \beta = \frac{nK_tK_b}{R_m} + f_m \tag{26}$$

4.2. Једначине стања инверзног клатна на два точка

На основу теорије управљања можемо извести једначине стања линеаризујући једначине кретања око равнотежне тачке. Ово значи да усвајамо ограничење $\psi \to 0$ $\left(\sin\psi \to \psi, \quad \cos\psi \to 1\right)$ и занемаримо елементе другог реда попут $\dot{\psi}^2$. Једначине кретања (14) до (16) постају:

$$\left[\left(2m + M\right)R^2 + 2J_w + 2n^2J_m\right]\ddot{\theta} + \left(MLR - 2n^2J_m\right)\ddot{\psi} = F_\theta \tag{27}$$

$$\left(MLR - 2n^2J_m\right)\ddot{\theta} + \left(ML^2 + J_\psi + 2n^2J_m\right)\ddot{\psi} - MgL\psi = F_\psi \tag{28}$$

$$\left[\frac{1}{2}mW^2 + J_\phi + \frac{W^2}{2R^2}\left(J_w + n^2J_m\right)\right]\ddot{\phi} = F_\phi \tag{29}$$

Једначине (27) и (29) садрже само θ, а једначина (28) садржи само ϕ и могу се матрично изразити као:

$$E\begin{bmatrix}\ddot{\theta}\\ \ddot{\psi}\end{bmatrix} + F\begin{bmatrix}\dot{\theta}\\ \dot{\psi}\end{bmatrix} + G\begin{bmatrix}\theta\\ \psi\end{bmatrix} = H\begin{bmatrix}v_l\\ v_r\end{bmatrix} \tag{30}$$

$$E = \begin{bmatrix}\left(2m + M\right)R^2 + 2J_w + 2n^2J_m & MLR - 2n^2J_m\\ MLR - 2n^2J_m & ML^2 + J_\psi + 2n^2J_m\end{bmatrix} \tag{31}$$

$$F = 2\begin{bmatrix}\beta + f_w & -\beta\\ -\beta & \beta\end{bmatrix} \tag{32}$$

$$G = \begin{bmatrix}0 & 0\\ 0 & -MgL\end{bmatrix} \tag{33}$$

$$H = \begin{bmatrix}\alpha & \alpha\\ -\alpha & -\alpha\end{bmatrix} \tag{34}$$

$$I\ddot{\phi} + J\dot{\phi} = K\left(v_r - v_l\right) \tag{35}$$

$$I = \frac{1}{2}mW^2 + J_\phi + \frac{W^2}{2R^2}\left(J_w + n^2 J_m\right) \tag{36}$$

$$J = \frac{W^2}{2R^2}\left(\beta + f_w\right) \tag{37}$$

$$K = \frac{W}{2R}\alpha \tag{38}$$

Уводимо $\mathbf{x_1}$, $\mathbf{x_2}$ као векторе стања, **y** као вектор улаза:

$$\mathbf{x}_1 = \left[\theta, \quad \psi, \quad \dot{\theta}, \quad \dot{\psi}\right]^T, \quad \mathbf{x}_2 = \left[\phi, \quad \dot{\phi}\right]^T, \quad \mathbf{u} = \left[v_l, \quad v_r\right]^T \tag{39}$$

Изводимо једначине стања на основу једначина (30) и (36)

$$\dot{\mathbf{x}}_1 = A_1\mathbf{x}_1 + B_1\mathbf{u} \tag{40}$$

$$\dot{\mathbf{x}}_2 = A_2\mathbf{x}_2 + B_2\mathbf{u} \tag{41}$$

$$A_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & A_1(3,2) & A_1(3.3) & A_1(3,4) \\ 0 & A_1(4,2) & A_1(4,3) & A_1(4,4) \end{bmatrix} \tag{42}$$

$$B_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ B_1(3) & B_1(3) \\ B_1(4) & B_1(4) \end{bmatrix} \tag{43}$$

$$A_2 = \begin{bmatrix} 0 & 1 \\ 0 & -J/I \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 & 0 \\ -K/I & K/I \end{bmatrix} \tag{44}$$

Елементи матрице су:

$$A_1(3,2) = -gMLE(1,2)/\det(E)$$
$$A_1(4,2) = gMLE(1,1)/\det(E)$$
$$A_1(3,3) = -2\left[(\beta + f_w)E(2,2) + \beta E(1,2)\right]/\det(E)$$
$$A_1(4,3) = 2\left[(\beta + f_w)E(1,2) + \beta E(1,1)\right]/\det(E)$$
$$A_1(3,4) = 2\beta\left[E(2,2) + E(1,2)\right]/\det(E)$$
$$A_1(4,4) = -2\beta\left[E(1,1) + E(1,2)\right]/\det(E)$$
$$B_1(3) = \alpha\left[E(2,2) + E(1,2)\right]/\det(E)$$
$$B_1(4) = -\alpha\left[E(1,1) + E(1,2)\right]/\det(E)$$
$$\det(E) = E(1,1)E(2,2) - E(1,2)^2$$

4.3. Формирање повратне спреге по стању

Улази актуатора (NXT мотора) су PWM сигнали на левом и десном мотору, иако се улаз **Y** изражава као напонски. Излази сензора су угао мотора θ_m, и брзина вертикалног нагињања $\dot{\psi}$. Одредити θ и ϕ преко θ_m. Постоје две методе за одређивање ψ преко $\dot{\psi}$:

1. Одређивање ψ нумеричком интеграцијом угаоне брзине,
2. Процена ψ помоћу опсерверва коришћењем на основу модерне теорије управљања

Овде ће се користити метод 1.

Стабилност - Дефинишемо систем као асимптотски стабилан када стање тежи нули независно од почетне вредности за улаз **Y** једнак нули.

$$\lim_{t\to\infty}\mathbf{x}(t) = 0 \tag{45}$$

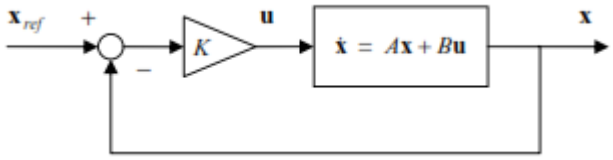
За једначину стања дату једначином:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) \tag{46}$$

Потребан и довољан услов за асимптотску стабилност је да реални део свих сопствених вредности матрице A буде негативан. Систем је нестабилан када има сопствене вредности са позитивним реалним делом.

4.4. Повратна спрега по стању

Повратна спрега по стању је управљачка стратегија која множи појачање повратне спреге *K* и разлику референтних стања $\mathbf{x}_{реф}$ и мерене вредности **x** и тако формира сигнал управљања. Ово је слично PD контроли у класичној теорији управљања. Слика 12. показује блок дијаграм повратне спреге по стању.



Слика 12. Блок дијаграм управљања повратном спорегом по стању

Једначине улаза и стања за систем на слици 12. су:

$$\mathbf{u}(t) = -K\left(\mathbf{x}(t) - \mathbf{x}_{ref}\right) \tag{47}$$

$$\dot{\mathbf{x}}(t) = \left(A - BK\right)\mathbf{x}(t) + BK\mathbf{x}_{ref} \tag{48}$$

Систем можемо учинити стабилним подешавањем појачања K јер тиме мењамо сопствене вредности матрице система $A - BK$.

Потребно је при коришћењу управљања повратном спрегом по стању да систем буде контролабилан. Потребан и довољан услов је да матрица контролабилности M_c буде пуног ранга ($rank\left(M_c\right) = n,$ n је ред вектора **x**)

$$M_C = \left[B, AB, \ldots A^{n-1}B\right] \tag{49}$$

Постоје две методе за одређивање појачања K.

1. Директно подешавање полова

Овим методом се израчунава појачање K тако да се половима (сопственим вредностима) матрице система $A - BK$ доделе жељене позиције. Параметри који се подешавају су полови и одговарајуће вредности појачања се одређују методом пробе и грешке, или коришћењем готових алгоритама (у MATLAB-u функција *place*).

2. Линеарно квадратно управљање

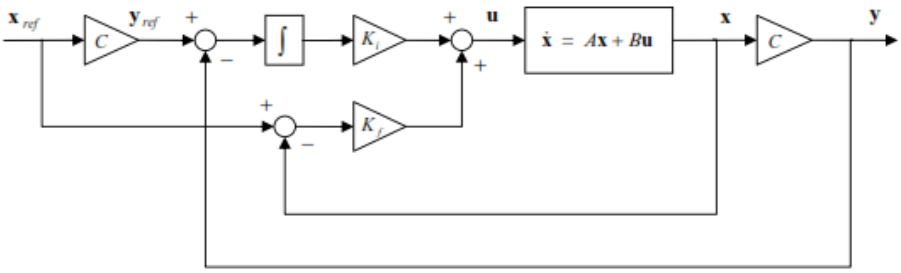
Овим методом се израчунавају појачања K тако да минимизују показатељ квалитета J који се може изразити као:

$$J = \int_0^\infty \left(\mathbf{x}(t)^T Q\mathbf{x}(t) + \mathbf{u}(t)^T R\mathbf{u}(t)\right)dt \tag{50}$$

Параметри који се подешавају су тежинске матрице стања Q и улаза P. Потребно је одредити одговарајуће елементе ових матрица преко пробе и грешке. У Matlabu функција *lqr* налази појачања за дати систем и тежинске матрице.

4.5. Серво контрола

Серво контрола је управљачка стратегија која омогућава да излаз система прати очекивано понашање. PID контролер у класичној теорији управљања представља врсту серво контролера. Да би систем пратио одскочне референтне улазе, потребно је додати интегратор у систем са затвореном повратном спрегом. Слика 13. приказује блок дијаграм серво PID контролера.



Слика 13. Блок дијаграм серво PID контролера

Појачања серво контролера се могу одредити на исти начин као у случају контролера са повратном спрегом по стању ако посматрамо проширени систем. Уводи се ново стање које представља разлику између излаза и референтне вредности. Следи опис одређивања појачања серво контролера.

Једначине стања и излаза су дате као:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) \tag{51}$$

$$\mathbf{y}(t) = C\mathbf{x}(t) \tag{52}$$

Користимо разлику $\mathbf{e}(t) = C\left(\mathbf{x}(t) - \mathbf{x}_{ref}\right)$, интеграл разлике $\mathbf{z}(t)$, стања проширеног система $\bar{\mathbf{x}}(t) = \left[\mathbf{x}(t), \mathbf{z}(t)\right]^T$, и редови вектора **x** и **y** су *н, м.* Једначине стања проширеног система су:

$$\begin{bmatrix} \dot{\mathbf{x}}(t) \\ \dot{\mathbf{z}}(t) \end{bmatrix} = \begin{bmatrix} A & 0_{n\times m} \\ C & 0_{n\times m} \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{z}(t) \end{bmatrix} + \begin{bmatrix} B \\ 0_{m\times m} \end{bmatrix} \mathbf{u}(t) - \begin{bmatrix} 0_{n\times m} \\ I_{m\times m} \end{bmatrix} C\mathbf{x}_{ref} \tag{53}$$

Ако се претпостави да је систем стабилан једначина (53) конвергира у:

$$\begin{bmatrix} \dot{\mathbf{x}}(\infty) \\ \dot{\mathbf{z}}(t) \end{bmatrix} = \begin{bmatrix} A & 0_{n\times m} \\ C & 0_{m\times m} \end{bmatrix} \begin{bmatrix} \mathbf{x}(\infty) \\ \mathbf{z}(\infty) \end{bmatrix} + \begin{bmatrix} B \\ 0_{m\times m} \end{bmatrix} \mathbf{u}(\infty) - \begin{bmatrix} 0_{n\times m} \\ I_{m\times m} \end{bmatrix} C\mathbf{x}_{ref} \tag{54}$$

Одузимајући једначину (54) од једначине (53) добијамо:

$$\begin{bmatrix} \dot{\mathbf{x}}_e(t) \\ \dot{\mathbf{z}}_e(t) \end{bmatrix} = \begin{bmatrix} A & 0_{n\times m} \\ C & 0_{m\times m} \end{bmatrix} \begin{bmatrix} \mathbf{x}_e(t) \\ \mathbf{z}_e(t) \end{bmatrix} + \begin{bmatrix} B \\ 0_{m\times m} \end{bmatrix} \mathbf{u}_e(t) \rightarrow \frac{d}{dt}\bar{\mathbf{x}}_e(t) = \bar{A}\bar{\mathbf{x}}_e(t) + \bar{B}\mathbf{u}_e(t) \tag{55}$$

Где је $\mathbf{x}_e = \mathbf{x}(t) - \mathbf{x}(\infty), \quad \mathbf{z}_e = \mathbf{z}(t) - \mathbf{z}(\infty), \quad \mathbf{u}_e = \mathbf{u}(t) - \mathbf{u}(\infty)$. Можемо користити управљање повратном спрегом да учинимо систем стабилним. Улаз је:

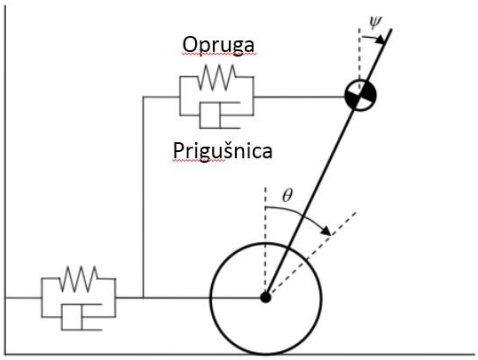
$$\mathbf{u}_e(t) = -K\bar{\mathbf{x}}_e(t) = -K_f \mathbf{x}_e(t) - K_i \mathbf{z}_e(t) \tag{56}$$

Ако претпоставимо да је могуће $\mathbf{x}(\infty) \to \mathbf{x}_{ref}, \quad \mathbf{z}(\infty) \to 0, \quad \mathbf{u}(\infty) \to 0$ тада добијамо улаз **y***(m)*.

$$\mathbf{u}(t) = -K_f \left(\mathbf{x}(t) - \mathbf{x}_{ref}\right) - K_i C \int \left(\mathbf{x}(t) - \mathbf{x}_{ref}\right) dt \tag{57}$$

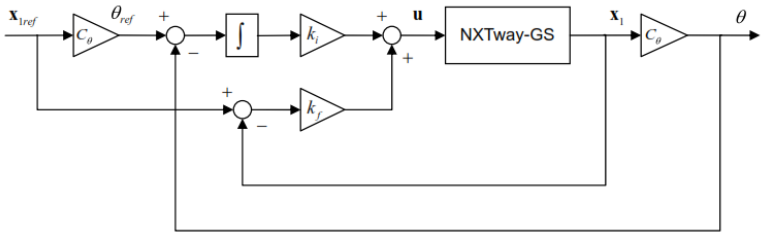
Ако би први члан једначине (57) био нула добили бисмо оно што модерна теорија управљања назива I-PD контролер. Међутим, због бољег неутралисања грешке стационарног стања користиће се PID управљање.

4.6. Пројектовање контролера самобалансирајућег робота



Слика 14. Модел инверзног клатна са два точка као систем масе на опрузи са пригушницом

Формирамо контролер повратне спреге по стању као на слици 15. Види се одавде да се за управљану величину узима средња угаона позиција точкова θ. Овде је κ_ϕ вектор појачања за повратну спрегу по стањима и може се посматрати да је овај део контролера налик PD контролеру. κ_u представља појачање додатно уведеног интегралног дејства. C_θ је матрица којом се вектор стања **X** преводи у вредност угаоне позиције точкова θ.



Слика 15. Контролер повратне спреге по стању за ЛЕГО робота

Вредности параметара κ_ϕ и κ_u се могу добити из жељених особина карактеристичне једначине целог система, односно подешавањем полова, или, као што је овде урађено, методом LQ оптимизације. У овом случају узимамо искуствено матрице Q и P:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 6\times10^5 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 4\times10^2 \end{bmatrix} \tag{58}$$

$$R = \begin{bmatrix} 1\times10^3 & 0 \\ 0 & 1\times10^3 \end{bmatrix} \tag{59}$$

Овде $Q(2,2)$ представља тежински фактор појачања за грешку вертикалног нагиба, док $Q(5,5)$ представља тежински фактор везан за интегрално појачање κ_u.

Решавање Рикатијевих једначина је спроведено помоћу уграђене MATLAB команде *lqr* и тиме су добијене следеће вредности:

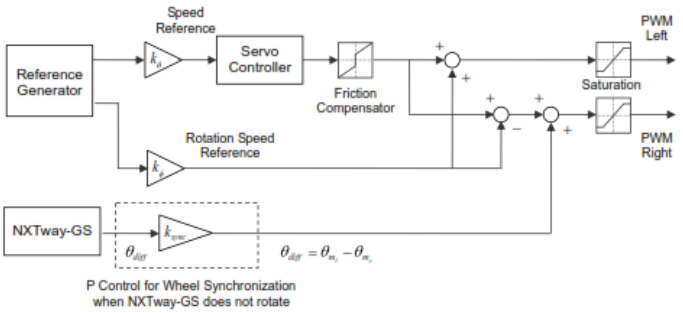
$$k_f = \begin{bmatrix} -0.8 & -66.3799 & -1.2293 & -7.0540 \end{bmatrix}, \tag{60}$$

$$k_i = -0.4472 \tag{61}$$

Потребно је умањити брзинско појачање $k_f[3]$ како би се умањиле осцилације робота. Даље потребно је увести додатна управљања:

- Ротација самобалансирајућег робота задавањем различитих брзина левом и десном точку
- Уводи се P контрола за синхронизацију точкова када се робот креће праволинијски јер се мотори не крећу истим брзинама, чак и када добијају идентичан PWM сигнал.

Последично, добијамо контролер чији је блок дијаграм приказан на слици 16. Овде су $k_{\dot\theta}$, $k_{\dot\phi}$, k_{sync} подесиви параметри.



Слика 16. *Блок дијаграм контролера самобалансирајућег робота*

4.7. Одређивање LQR коефицијената

$$\dot{x} = Ax + Bu \quad J = \int_0^\infty (x^T Q x + u^T R u + x^T S u) dt,$$

Члан уз S се увек изоставља.

Одређивање Q и R коефицијената:

1. Најједноставнији избор: $Q = I, R = \rho I \implies L = \|x\|^2 + \rho\|u\|^2$. Где варирамо ρ да добијемо добар одзив.
2. Дијагоналне тежинске вредности

   $$Q = \begin{bmatrix} q_1 & & \\ & \ddots & \\ & & q_n \end{bmatrix} \qquad R = \begin{bmatrix} r_1 & & \\ & \ddots & \\ & & r_n \end{bmatrix}$$

   Изабрати свако q_i тако да прираштај индексу преформансе за одговарајући члан буде једнак за максимално дозвољено одступање стања. Пример $x_1 =$ дистанца у метрима, $x_3 =$ угао у радијанима:

   $$1cm \text{ грешке } OK \implies q_1 = \left(\frac{1}{100}\right)^2 \quad q_1 x_1^2 = 1 \quad \text{када} \quad x_1 = 1cm$$

   $$\frac{1}{60} rad \text{ грешке } OK \Rightarrow q_3 = (60)^2 \quad q_3 x_3^2 = 1 \quad \text{када} \quad x_3 = \frac{1}{60} rad$$

Исто је и са r_i. Користимо ρ тако да балансирамо утицај улаза и стања система на индекс преформансе.

3. Излазне тежине: $y = Cx$ је излаз чија грешка желимо да буде што мања. Претпостављамо да је пар (A, C) обсервабилан.

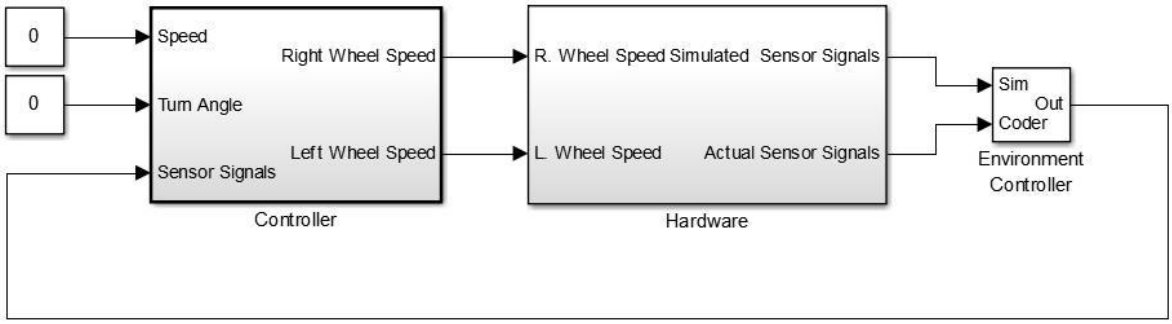
   $$Q = H^T H \qquad R = \rho I$$

   Овом методом већу пажњу придајемо излазној величини у односу на улаз.

4. Итеративним пробама за разне коефицијенте.

5. СИМУЛИНК МОДЕЛ - Матлаб

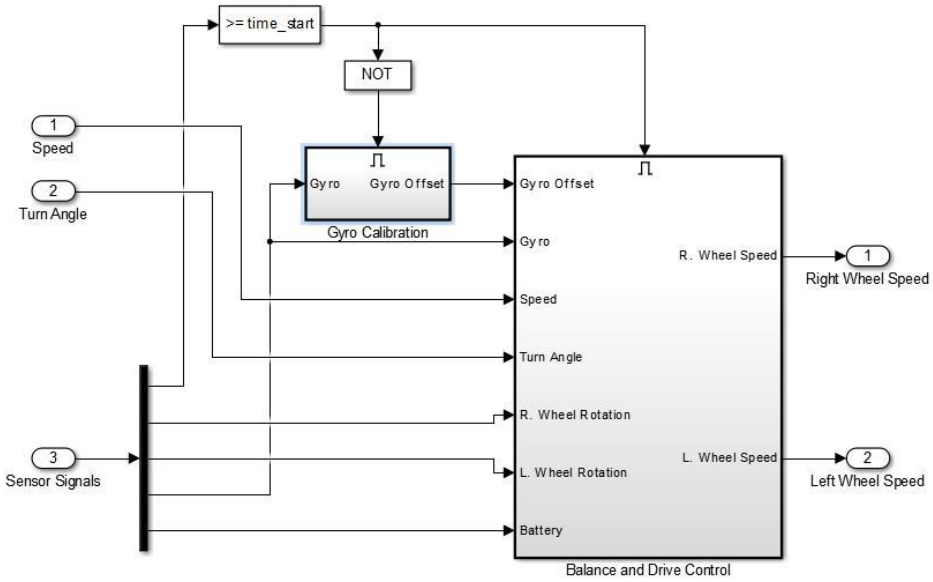
Сви основни подаци везани за кретање, стабилност, брзину налазе се и имплеметирају се у коцку баш из овог дела. Код који је исписан у Python програму произашао је из ове структуре модела. Целокупни изглед модела приказан је на слици 17. и састоји се од модела конролера, хардверског дела модела као блока тзв. избор окружења који одређује који ће од ових блокова модела бити коришћени а који не и то све у зависности од тога да ли је модел подешен на симулацију или изградњу C кода.



Слика 17. Целокупни изглед модела робота

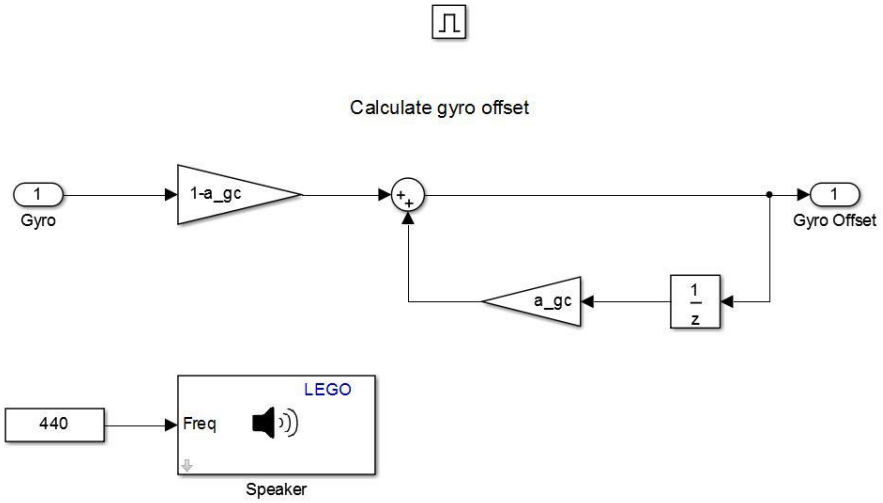
5.1. Контролер

Као улазне величине овог блока уводе се жељена угаона брзина точкова, жељени угао нагиба и вредности који се узимају са сензора. На слици 18. се могу видети подблокови блока контролера, и са слике се може уочити да и овај блок има своје подблокове.



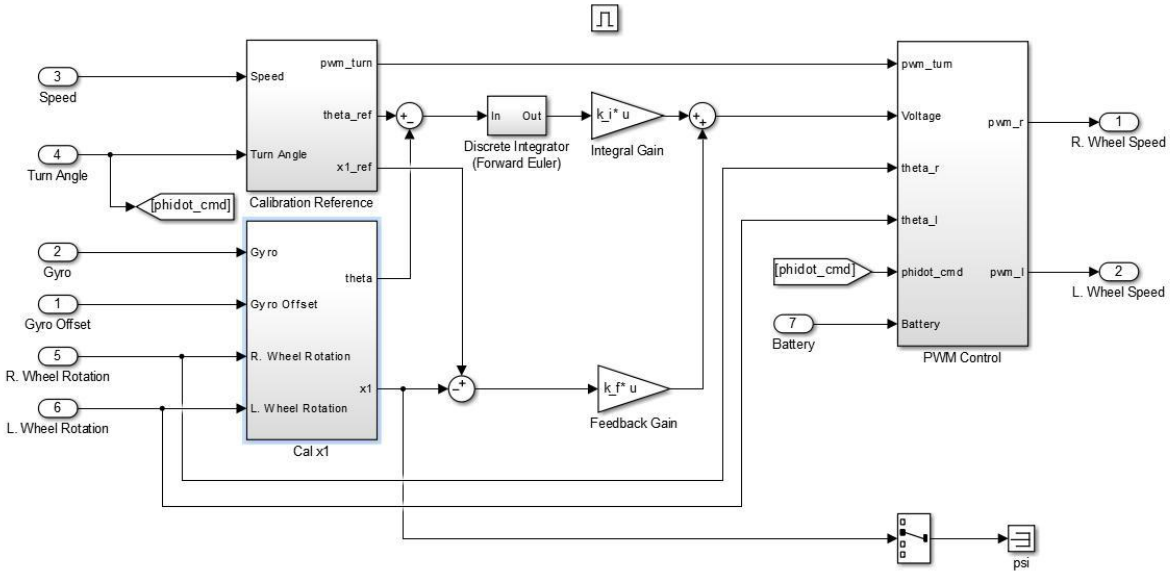
Слика 18. Подблокови контролера робот

Вредности са сензора које се узимају са робота су: вредност напона на батеријама, угао ротације левог и десног точка и врдност која се учитава са жироскопа. Као под блок овог блока јесте блок за калибрацију жироскопа, овај блок се користи како би се уклонио шум са сигнала који се учитава са жироскопа и то се врши помоћу дискретног филтера првог реда који је приказан на слици 19. На истој слици можемо видети један мали спикерфон који служи да упозори на то када се завршава калибрација сензора и почиње са радом други део контролера. Калибрација траје 4 сек. Уколико не држимо робота усправно и жироскоп сними погрешне податке може доћи до тога да нам робот падне или крене уназад.



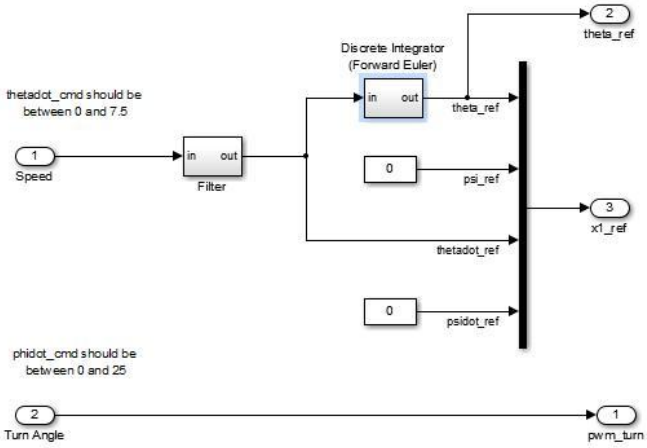
Слика 19. Изглед блока за калибрацију филтера

Када се заврши поступак калибрације долази до прорачунавања и одржавања робота односно до његовог кретања и одржавања стабилности. На слици 20. приказан је блок који одређује тренутна стања, врши интеграцију угла у точковим и одређује референтни угао у свакој периоди.



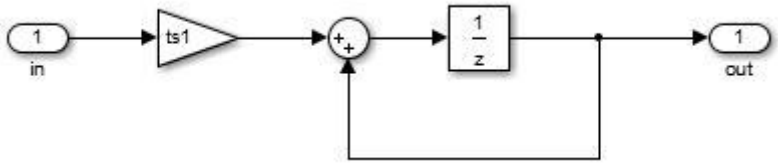
Слика 20. Блок за прорачунавање брзине на точковим

Блок за калибрацију референтне брзине и угла заокретања користи се за одређивање углове заокретање точкова и његово прослеђивање ка осталим блоковима.



Слика 21. Блок за израчунавање референтних вредности

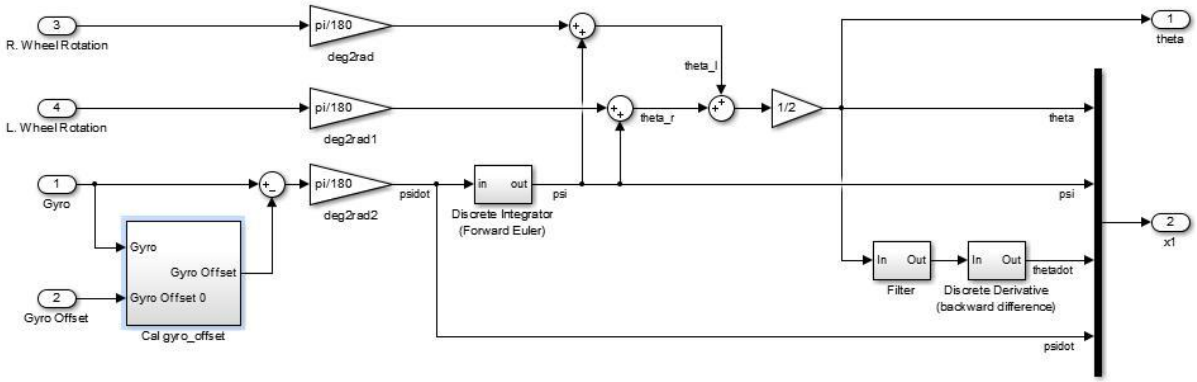
На слици 21. је приказан блок на основу кога се израчунава угао заокретања точкова, задаје референтни угао нагиба робота и угао заокретања робота се само прослеђује даље на блокове.



Слика 22. Дискретни интегратор

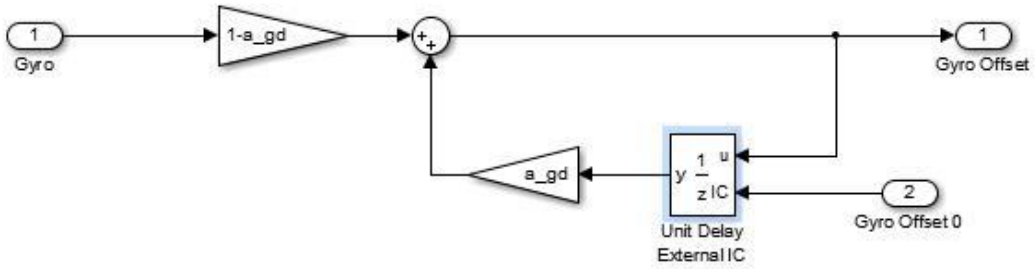
На слици 22. је приказан дискретни интегратор помоћу кога се интеграли брзина и добија угао заокретања. Овакав интегратор ће се користити и даље у програму. Као излаз из кола са слике 21. добијамо три величине: референтни угао заокретања точкова по периоди, угао заокретања укупног робота, и низ од четири референтне величине. Као чланови поменутог низа су: угаона брзина точкова, угао заокретања точкова, угаона брзина нагиба робота и угао нагиба робота.

У наредном блоку Cal x1врши се превођење углова окретања точкова из степена у радијане, подешавање офсета жироскопа и као резултат се добијају тренутне вредности угла и брзине нагиба и угла и брзине окретања точкова.



Слика 23. Одређивање тренутних вредности нагиба и окретања точкова

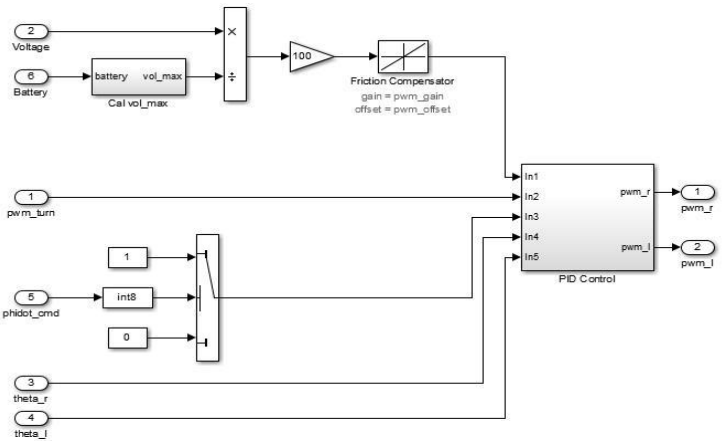
На слици 23. се види да је у овом блоку потребно направити нови блок који служи да одржи офсет на почетно прорачунатом нивоу а у следећем периоду га поново прерачунати.



Слика 24. Коло за одржавање офсета жироскопа на почетном ивоу у првој периоди

На слици 24. је приказано коло помоћу кога се у првој периоди задржава почетно прорачунат офсет жироскопа. Помоћу блока Unit Delay је дефинисано да се у првом тренутку офсет задржи на првом прорачунатом нивоу, а да се после сваког пута прорачунава и задржава на предходном нивоу. Појачање у повратној спрези је фактор заборављања, који се прорачунава у скрипти. Вредност са жироскопа представља величину брзине нагињања робота тако да је потребно интегралити ту вредност да би се добио угао нагињања робота. Вредност угла нагиба робота се сабира са оба угла заокретања точка јер је то стварни угао заокретања точка и дели са 2 како би добили средњу вредност заокретања точка. Да би смо добили брзину заокретања точка потребно је прво да се та вредност филтрира па тек онда диференцира како би се добила брзина на точковима. Као излаз из овог кола имамо угао тета и као и у предходном колу низ од четири величине, само у овом случају те величине су реалне.

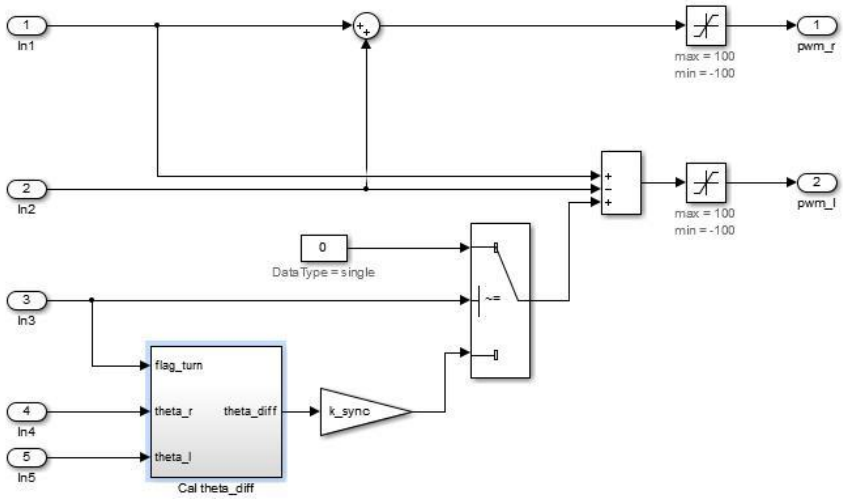
Из предходна два блока вредности тета и тета референто се одузимају и интеграле, а после тога појачавају интегралним појачањем. Исто тако вредности низа референтних и реалних вредности се одузимају и множе са појачањем. Вредност разлике и појачаних вредности углова тета умањује се за вредност појачане вредности разлике низа од четири елемента. Тај део се може посматрати као PID појачање из разога што имамо вредности углова и брзина и још сабирамо са интегрираним вредностима углова. Ова вредност се уводи у наредни блок где се користи за одређивање PWM сигнала. Наредни блок се користи за одређивање PWM сигнала који је потребно послати на моторе робота.



Слика 25. Коло за израчунавање жељеног PWM сигнала

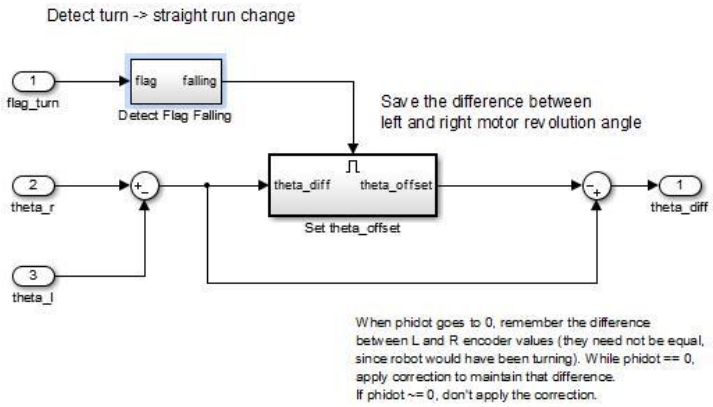
Са слике 25. се може видети да први део овог блока израчунава жељени PWM сигнал на основу стања батерије и тај сигнал се шаље у наредни блок где се одлучује да ли је потребно скретати робота или на исти сигнал слати на оба мотора. Овај сигнал се пре блока за распоређивање на моторе, првобитно уводи у сигнал у коло за компензовање трења електро мотора.

У следећем блоку се дефинише ограничење PWM сигнала и њихов међусобни однос, као и услови при којима је потребно да се један од њих убрза или успори како би се извршило заокретање.



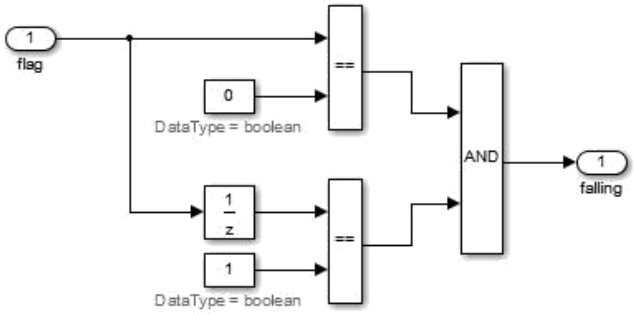
Слика 26. Изглед блока за одређивање PWM сигнала

Како је познато да за исти PWM сигнал два различита мотора не делују исто, потребно је да се изврши њихово синхронизовање, исто тако када је задато жељено хоризонтално заокретање потребно је да робот када достигне жељени угао настави праволинијско кретање. То се решава у два блока.



Слика 27. Израчунавање разлике углова заокретања

На слици 27. је приказано коло које израчунава разлику између углова заокретања точкова, и постоји услов да када је угао једнак задатом углу заокретања тада долази до задржавања те вредности угла између точкова.

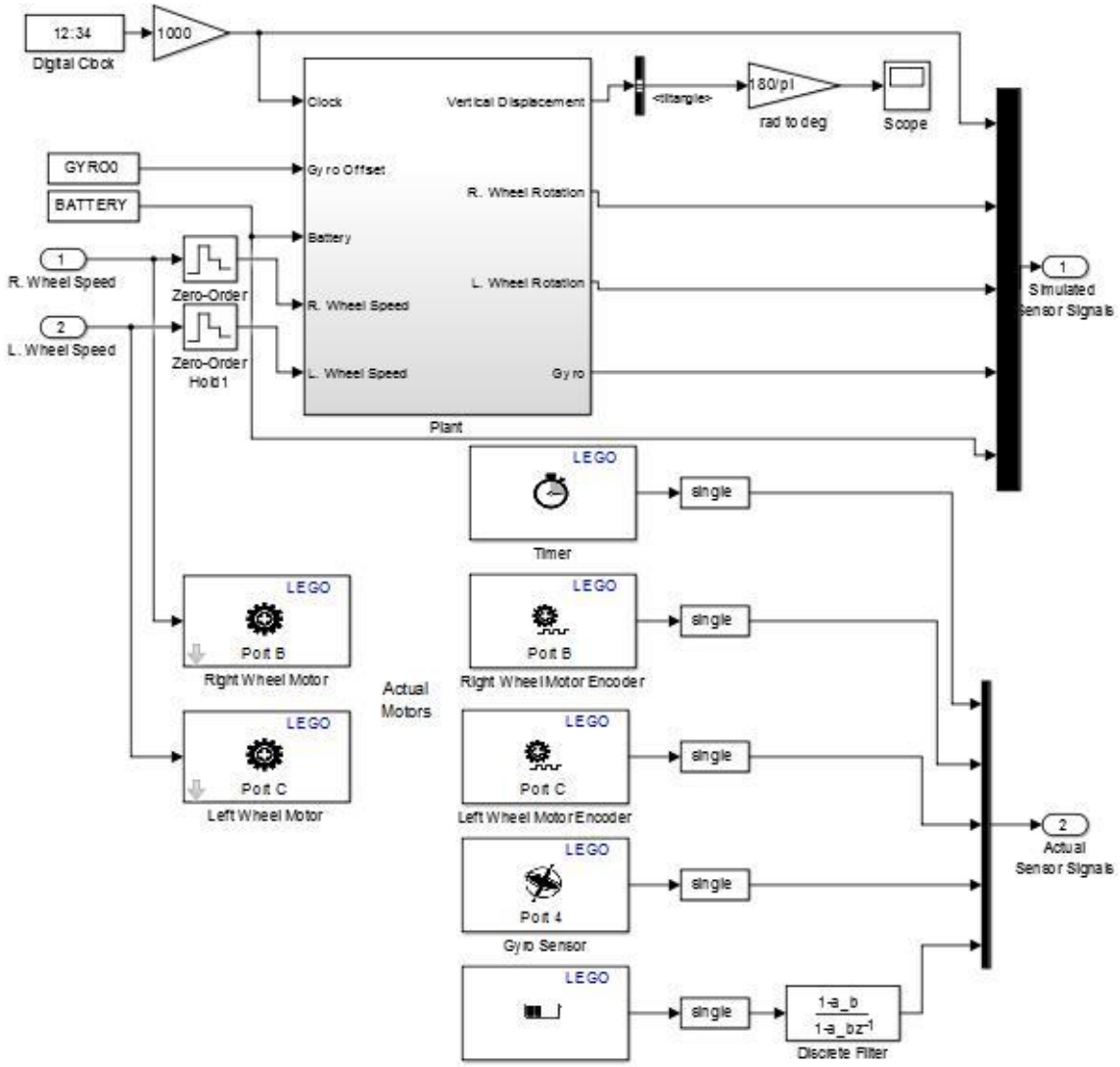


Слика 28. Блок за за одређивање услова за даље заокретање

На слици 28. је приказано логичко коло помоћу кога се одређује да ли је потребно наставити окретање робота како би достигао жељени угао или је потребно зауставити га у том тренутку и да се робот креће само право. Потпуно исто коло се активира и када је задато само праволинијско кретање, услед несавршености мотора долази до скретања робота на једну страну, па је потребно извршити корекцију путање.

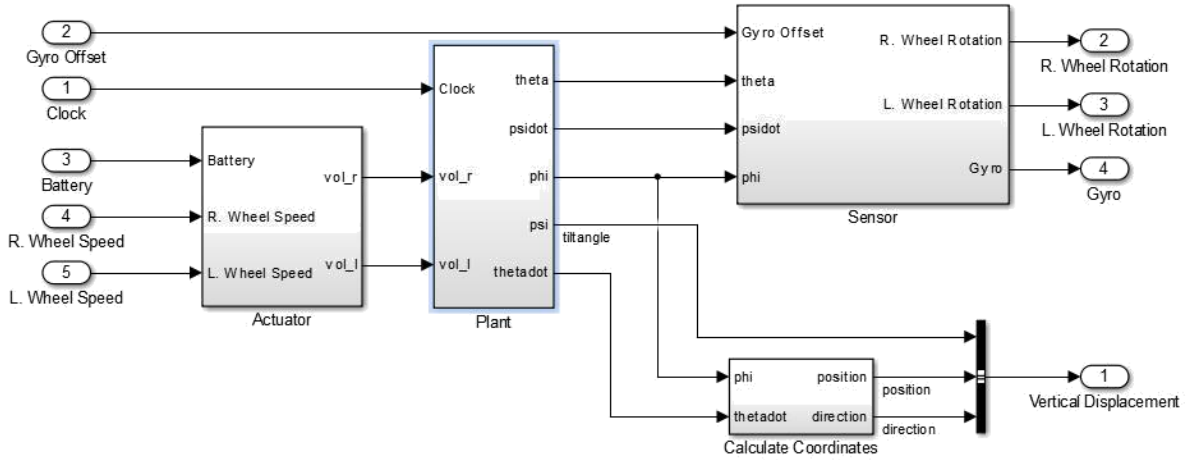
5.2. Хардвер

У предходном поглављу је описан модел контролера који се користи у оба мода овог робота и у симулацији и у стварном раду. Код овог модела није то случај доњи део се користи само за стварни мод, а горњи део само за симулацију.



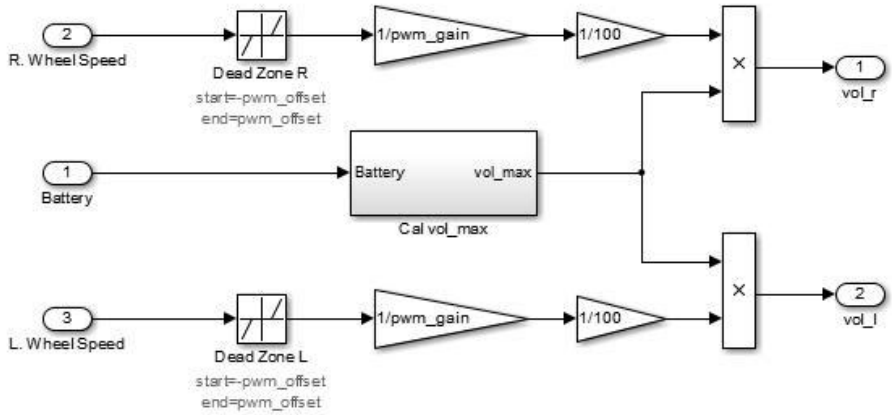
Слика 29. Модел хардвера

На слици 29. се може видети поменута подела на реални и на симулациони део, та разлика је лако уочљива јер се у доњем делу налазе блокови који представљају реалне делове овог робота. Овај блок је претрпео мале исправке, било је потребно дискретизовати сигнале који се прорачунавају као PWM сигнали у блоку контролера. Вредности напуњености батерија, сигнал жироскопа и сат, у горњем делу прозора су симулирани кроз код и кроз подблокове овог дела модела.



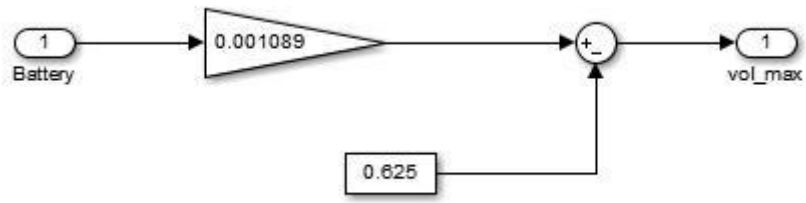
Слика 30. Изглед блока за симулацију рада робота

На слици 30. се може видети да блок за симулациују има све сегменте које треба да има прави робот и да се може у потпуности симулирати рад робора. Блок актуатор у коме се одређује потребан напон на моторима који зависи од жељене брзине окретања точкова.



Слика 31. Блок за симулацију мотора

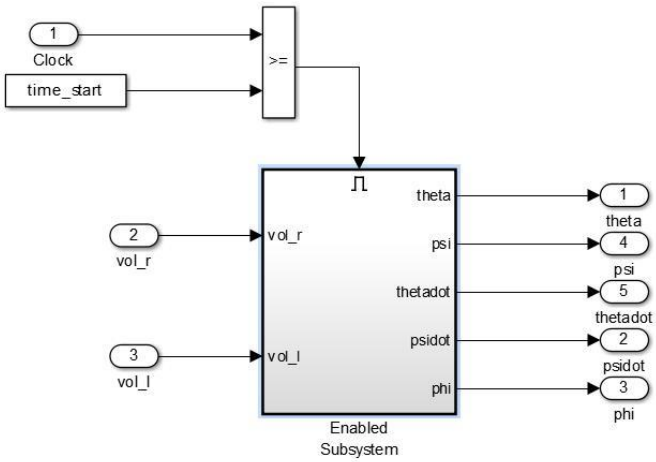
На слици 31. се може уочити да напон на крајевима мотора зависи и од степена напуњености батерија слика 32.



Слика 32. Симулација напуњености батерија

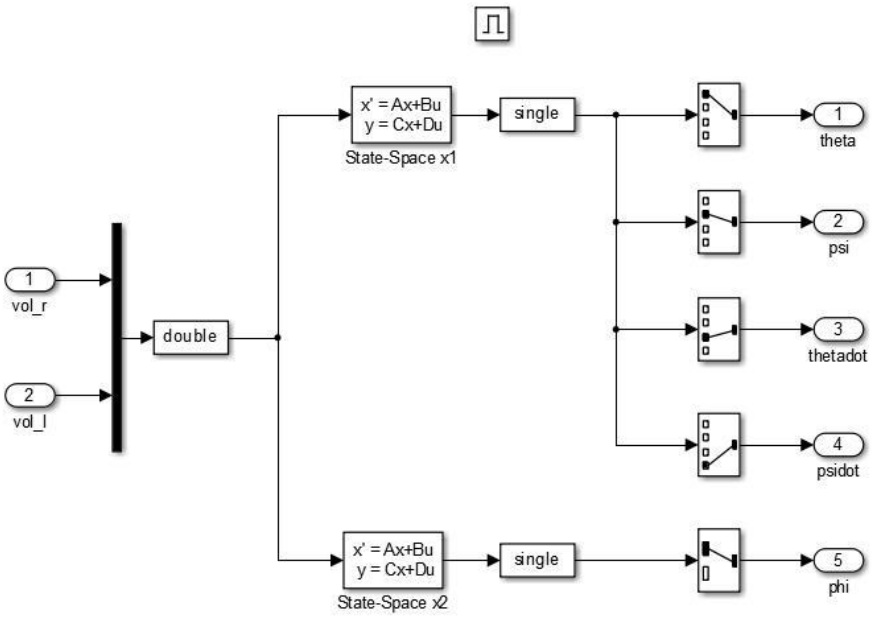
Поред тога симулирана је и мртва зона јер код стварног мотора, не остварује се окретање ротора при појави минималног напона.

Наредни блок прорачунава вредности углова заокретања и брзина заокретања робота.



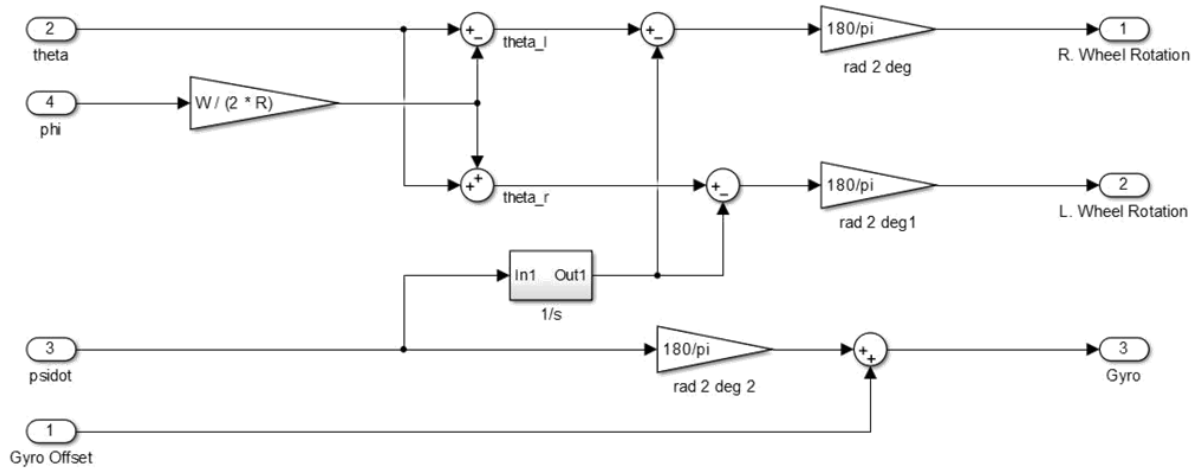
Слика 33. Блок за прерачунавање углова и брзина

Са слике се може видети да овај блок почиње да се извршава када је испуњен услов и то је да прође неко време за одређивање офсета жироскопа, time_start.



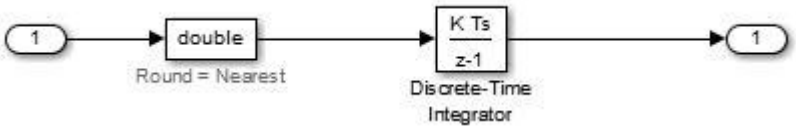
Слика 34. Рачунање помоћу једначина стања

На слици 34. се може уочити да се тренутне вредности заокретања и брзина заокретања рачунају помоћу једначина стања система. Из једног система једначина се израчунава нагиб и окретањ точкова, а из другог хоризонтално заокретање. Потребно је израчунати, на основу предходно израчунатих вредности углова и угаоних брзина, симулирану брзину точкова мотора. То се врши инверзним поступком, и на исти начин као у првом блоку који описује контролер. Отварањем блока који описује сензоре може се врло лако уочити и разумети инверзни поступак израчунавања кретања точкова.



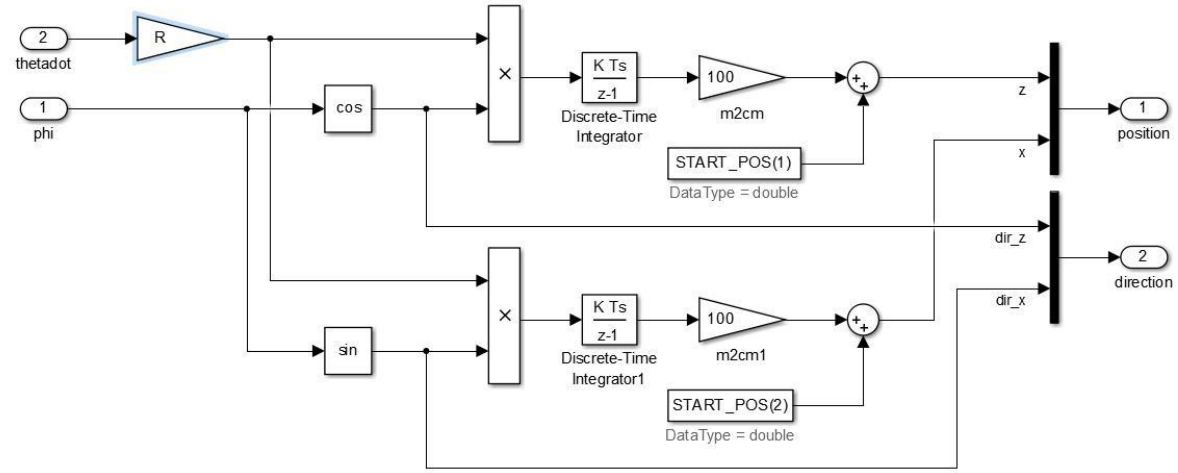
Слика 35. Одређивање окретања точкова и жироскопа

Са слике 35. се може видети инверзни поступак којим се од углова заокретања и брзина додавањем и одузимањем других углова долази до вредности окретања точкова.



Слика 36. изглед дискретног интегратора

На слици 36. се може видети дискретни интегратор који је потребан како би се одредио угао нагињања робора и ту вредност је потребно одузети од укупног заокретања точкова. Још један блок постоји у оргиналном фајлу, али је он за нас потпуно бескористан јер израчунава координате робота у простору које се могу исцртавати у 3D приказу.



Слика 37. Израчунавање просторних координата

На слици 37. је приказан блок који израчунава позицију и нагиб робота који се могу користити за визуелно исцртавање кретања робота.

6. ФОРМУЛИСАЊЕ PYTHON СКРИПТЕ НА ОСНОВУ МАТЛАБ СИМУЛИНК МОДЕЛА

Програмирање ЛЕГО робота коришћењем програма Python могуће је на два начина:

1. Програмски код написан у Python програмском језику можемо превести у nxc код који можемо `спустити` на коцку тако да је робот независан у односу на рачунар. Мана је што у овом случају не можемо користити Python као објектно орјентисани програм.

2. Програмски код написан у Python програмском језику извршавамо директно на рачунару и да се извршене наредбе директно шаљу роботу путем УСБ кабла или блутута. Овимо начином можемо искористити све предности објектно орјентисаног програмирања али је кретање робота ограничено дужином УСБ кабла или дометом блутута.

Пример који је објашњен у овом раду одрађен је на основу првог начина с тим што смо Python код компајлирали преко BricxCC програма како би робот директно могао да чита nxc фајл. Додатак nxc значи Not eXactly C тако да робот није способан да чита директно C код. Nxc нам омогућава уписивање управљачког програма робота у меморију робота тако да је робот самосталан. Поступак инсталирања Bricx Command Center – а можемо наћи на линку испод. На истом сајту можемо пронаћи објашњења за све таскове, променљиве и функције које користимо када компајлирамо Python код у nxc код.

http://bricxcc.sourceforge.net/; http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf

Када завршимо са инсталацијом потребног програма можемо да компајлирамо python скрипту и спустимо програм на робота. У наставку рада биће детаљније објашњена python скрипта повезана са дијаграмима из Матлаба као и неки проблеми са којима се можемо суочити при само пребацивању у nxc фајл.

6.1. Формулисање скрипте

6.1.1. Таскови

Pynxc програм се састоји из таскова. Таскови предсављају тредове система који се налазе у firmware – u NXT коцке. Захваљујући делу софтвера који се назива Scheduler свим тасковима се расподељују задаци и процесорско време, тако да се они практично извршавају истовремено. Како би програм адекватно радио мора да постоји бар један таск који се назива task main() и који ће бити извршен одмах по покретању програма. Овај таск је пре свега задужен за мерење времена и позивања осталих таскова тако да се оствари контрола у реалном времену. Захваљујући томе да се таскови извршавају истовремено, task main() истовремено прати време док се остали елементи програма извршавају.

Имајући у виду овакву архитектуру система, можемо тасковима да доделимо одговарајуће симулинк блокове и на тај начин формирамо Lego Segway контролер.

6.1.2. Калибрација и сигнализација

Као што је симулинк модел подељен по блоковима тако и тасковима додељујемо одређене блокове. Таску калибрација и сигнализација одговара блок приказан на слици 19. Дигитални филтер првог реда је имплементиран четвртој и петој линијо кода испод

```
def task_kalibracija():
   TextOut(1,LCD_LINE1,"kalibracija")
   ziroskop = SensorHTGyro(IN_4)
   ofsetZiroskopa = ziroskop*(1-a_gc)+ofsetZiroskopaPrethodni*a_gc
   ofsetZiroskopaPrethodni = ofsetZiroskopa
```

Трећа линија кода претсавља мерење са жироскопа

Део за сигнализацију приказан је у коду испод. Сигнализација служи да обавести да је калибрација у току при пуштању робота у рад. Овај блок значи да производимо тон фреквенције 440Hz у трајању од 100 милисекунди па онда да сачека (без прозвођења тона) наредних 200 милисекунди.

```
def task_signalizacija():
   svirka = 1
   PlayTone(440,100)
   Wait(200)
   svirka = 0
```

Променљива svirka служи као заставица да проверава да ли је произвођење тона у току. Task main() ће позивати сигнализацију само када је ова заставица једнака нули. Ако то не би било имплементирано на овај начин task signalizacija() би био позиван истовремено са task kalibracija() и резултат тога би било константо пиштање а не испрекидано какво желимо.

6.1.3. Контролер

Таск контролер је главни таск у коме се врши имплементација кода. Он је еквивалентан блоку у симулинку на слици 18. Balance and drive control, као и блока са слике 29. Где се врши мерења позиција мотора и жироскопа.

```
def task_kontroler():
   ugaoLeviD = 0.0 #float
   ugaoDesniD = 0.0 #float
   #uzimamo podatke sa senzora
   #prebaciti naknadno ove promenljive u long
   ugaoLevi = MotorRotationCount(OUT_C)
   ugaoDesni = MotorRotationCount(OUT_B)
   ziroskop = SensorHTGyro(IN_4)
```

Затим меримо ниво батерије и тај сигнал филтрирамо:

```
baterija = BatteryLevel() #int
baterijaFilt = baterijaPret*a_b+(1-a_b)*baterija
baterijaPret = baterijaFilt
```

Затим вршимо одређивање референтних вредности као на слици 21.

```
#kalibrisemo referencu
zeljeniUgao = zeljeniUgao+zeljenaBrzina*periodaOdabiranja #float
zeljenoPsi = 0 #float
zeljenoPsiTacka = 0 #float
```

Вршимо калибрацију жироскопа (одређивање дрифта) као на слици 24.

```
#kalibrisemo ziroskop
ofsetZiroskopa = ziroskop*(1-a_gd)+ofsetZiroskopaPrethodni*a_gd
ofsetZiroskopaPrethodni = ofsetZiroskopa
```

Радимо одређивање тренутних вредности нагиба, брзине нагињања, угла точкова и брзине точкова у степенима (односно степенима по секунди)

```
#prebacujemo uglove u radijane
ugaoLeviD = 3.14*ugaoLevi/180
ugaoDesniD = 3.14*ugaoDesni/180
psiTacka = (ziroskop-ofsetZiroskopa)*pi/180 #float

#integralimo psiTacka da dobijemo psi
psi = psiInteg #float
psiInteg = psiInteg + periodaOdabiranja*psiTacka #float

#formiramo srednji ugao teta
teta = (ugaoLeviD+ugaoDesniD)/2 + psi #float

#vrsimo numericku derivaciju da nadjemo tetaTacka
#prvo filter da uklonimo sum
tetaFilt = (1-a_d)*teta+a_d*tetaFiltPrethodno
#sada derivacija
tetaTacka = (tetaFilt-tetaFiltPrethodno)/periodaOdabiranja
#sada upisujemo memoriju za filter i derivaciju
tetaFiltPrethodno = tetaFilt
```

Сада имплементирамо контролер као на слици 20.

```
    #odredjujemo greske
    greskaTeta = zeljeniUgao-teta #float
    greskaPsi = zeljenoPsi-psi #float
    greskaPsiTacka = zeljenoPsiTacka-psiTacka #float
    greskaTetaTacka = zeljenaBrzina - tetaTacka #float
    #integral greske teta
    greskaTetaInteg = greskaTetaIntegPrethodno
    greskaTetaIntegPrethodno = greskaTetaIntegPrethodno + greskaTeta*periodaOdabiranja

    #racunamo dejstva
    dejstvoTeta = greskaTeta*pojacanjeTeta
    dejstvoPsi = greskaPsi*pojacanjePsi
    dejstvoTetaTacka = greskaTetaTacka*pojacanjeTetaTacka
    dejstvoPsiTacka = greskaPsiTacka*pojacanjePsiTacka
    dejstvoTetaInteg = greskaTetaInteg*pojacanjeTetaInteg
    #sabiramo i dobijamo upravljanje
    upravljanje = dejstvoTeta + dejstvoPsi + dejstvoTetaTacka + dejstvoPsiTacka +
dejstvoTetaInteg
```

На крају PWM сигнал се израчунава као на слици 25.

```
    maxVolti = 0.001089*baterijaFilt - 0.625 #float

    upravljanjeKorigovano = upravljanje/maxVolti*100 #float

    if (zeljenoZaokretanje == 0):
        if (zeljenoZaokretanjePrethodno != 0):
            ofsetTeta = ugaoDesniD - ugaoLeviD
    razlikaTeta = ugaoDesniD - ugaoLeviD - ofsetTeta #float

    if (zeljenoZaokretanje != 0):
        korekcijaTeta = 0.0
    else:
        korekcijaTeta = razlikaTeta*pojacanjeSinh
```

И добијени сигнал прослеђујемо моторима

```
    pwmLevi = max(-100,min(100,(upravljanjeKorigovano+zeljenoZaokretanje))) #int
kastovano
    pwmDesni = max(-100,min(100, (upravljanjeKorigovano-
zeljenoZaokretanje+korekcijaTeta)))
    OnFwd(OUT_B, pwmDesni)
    OnFwd(OUT_C,pwmLevi)
```

6.1.4. Task main

Task main као што сам напомену је онај таск који се први покреће. Он је задужен за праћење времена и позивања осталих таскова.

Прво се поставља жироскоп на PORT 4.

```
SetSensorHTGyro(IN_4)
```

Затим узимамо почетно време:

```
pocetnoVreme = CurrentTick()
```

Сада долазимо до бесконачне петље која ће се извршавати на сваких 4 милисекунде. Први упит је да ли је време мање или једнао 4000 ms. Ако јесте, тада је потребно спроводити калибрацију на сваке 4ms. Овде је имплементирана и логика за покретање игнализацијепреко провере вредности заставице svirka на сваких 4ms.

Када је време веће од 4000ms, тада се покреће таск контролер на сваких 4 ms. На тај начин имамо имплементирано управљање у реалном времену.

```
while True:
    if CurrentTick()-pocetnoVreme<=4000:
        StartTask(task_kalibracija)
        if svirka==0:
            StartTask(task_signalizacija)
        Wait(4)
    else:
        StopTask(task_kalibracija)
        StartTask(task_kontroler)
        Wait(4)
```

На овај начин је представљен python код који је компајлиран преко BrixCC а.

Проблем са којим ћете се уочити приликом пребацивања python кода у pynxc формат односи се на додељивању типа променљиве. Након пребацивања у pynxc BricxCC аутоматски додељује свакој променљивој тип int (integer) тако да је потребно доделити тип променљивој. Слика 38. и 39. приказују те разлике

```
int ofsetTeta = 0.0;
int kTeta = 1.0;
int ofsetZiroskopa = 0;
int periodaKontrolera = 4;
int tetaZeljeno = 0.0;
int zeljenaBrzina = 0.0;
int vremeKalibracije = 4000;
int kalibracijaZiroskopa = 0.8;
int stanjeBaterijeV = 0.0;
int kIntegralno = 1.0;
int kPsi = 1.0;
int filterTeta = 0.8;
int kRazlikaTeta = 0.35;
int psi = 0.0;
int pocetnoVremeMS = 0;
int tetaPrethodno = 0.0;
int tetaFiltrirano = 0.0;
int driftZiroskopa = 0.999;
int korekcijaBaterije = 0.8;
int kPsiTacka = 1.0;
int zeljenaRotacija = 0.0;
int kTetaTacka = 1.0;
void inicijalizacija()  {
    int stanjeBaterije;
    pocetnoVremeMS = CurrentTick();
    stanjeBaterije = BatteryLevel() / 1000.0;
    ofsetZiroskopa = SensorHTGyro(S4);
```

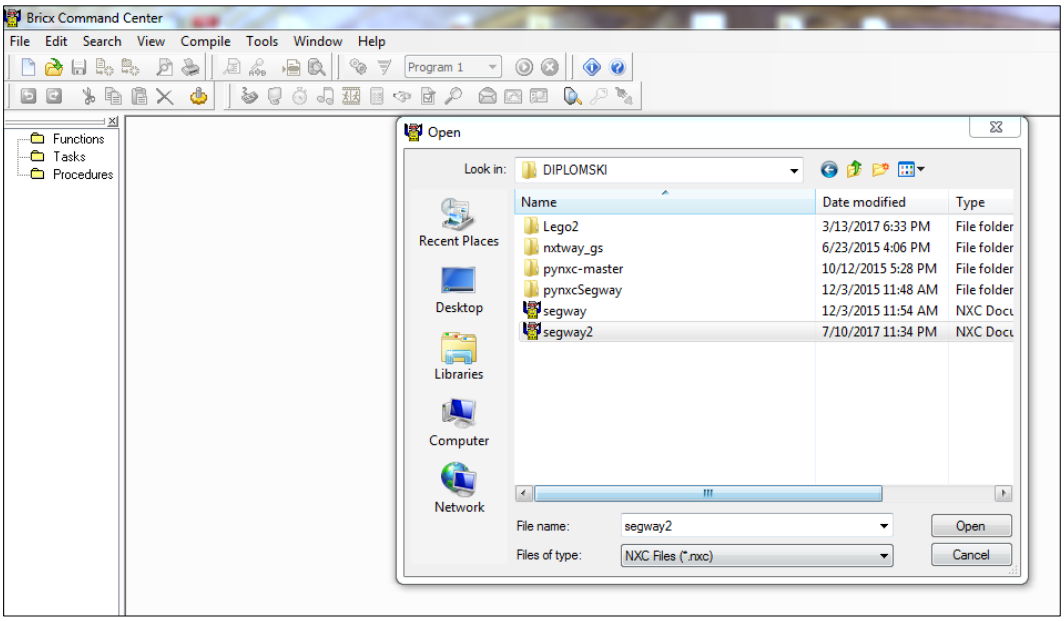
Слика 38. Промњњиве након компајлирања

```
float psiInteg = 0;
float pojacanjePsiTacka = -7.0540;
float ofsetTeta = 0;
float pojacanjeTetaInteg = -0.4472;
float ofsetZiroskopa = 0;
int zeljenoZaokretanjePrethodno = 0;
int zeljenaBrzina = 0;
float ofsetZiroskopaPrethodni = 0;
int zeljenoZaokretanje = 0;
float pi = 3.14;
float baterijaFilt = 0.0;
float tetaFilt = 0;
float periodaOdabiranja = 0.004;
float tetaFiltPrethodno = 0;
float a_b = 0.8;
float pojacanjeTeta = -0.8;
float a_gd = 0.999;
float a_gc = 0.8;
float pojacanjeSinh = 0.35;
float pojacanjePsi = -66.3799;
float greskaTetaIntegPrethodno = 0;
float a_d = 0.8;
int svirka = 0;
float baterijaPret = 0.0;
float pojacanjeTetaTacka = -1.2293*0.85;
task task_kalibracija()  {
    int ziroskop;
    TextOut(1, LCD_LINE1, "kalibracija");
    ziroskop = SensorHTGyro(IN_4);
    ofsetZiroskopa = (ziroskop * (1 - a_gc) + ofsetZiroskopaPrethodni * a_gc);
    ofsetZiroskopaPrethodni = ofsetZiroskopa;
```

Слика 39. Додељен тип променљивој (неопходна исправка)

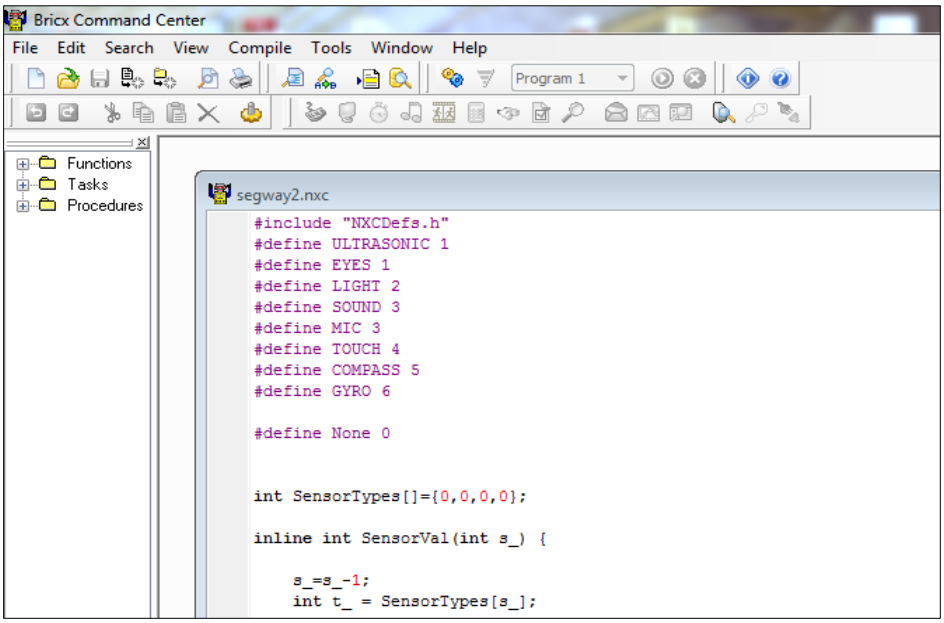
7. Туторијал – спуштање програма на коцку

Када конвертујемо python фајл у nxc потребно је да га компајлирамо сваки пут пре него што га спустимо на коцку. Свака измена програма захтева поновну обраду програма па тек онда пребацивање на коцку робота. Слика 40. је интерфејс самог BricxCC помоћу којег обрађујемо податке.

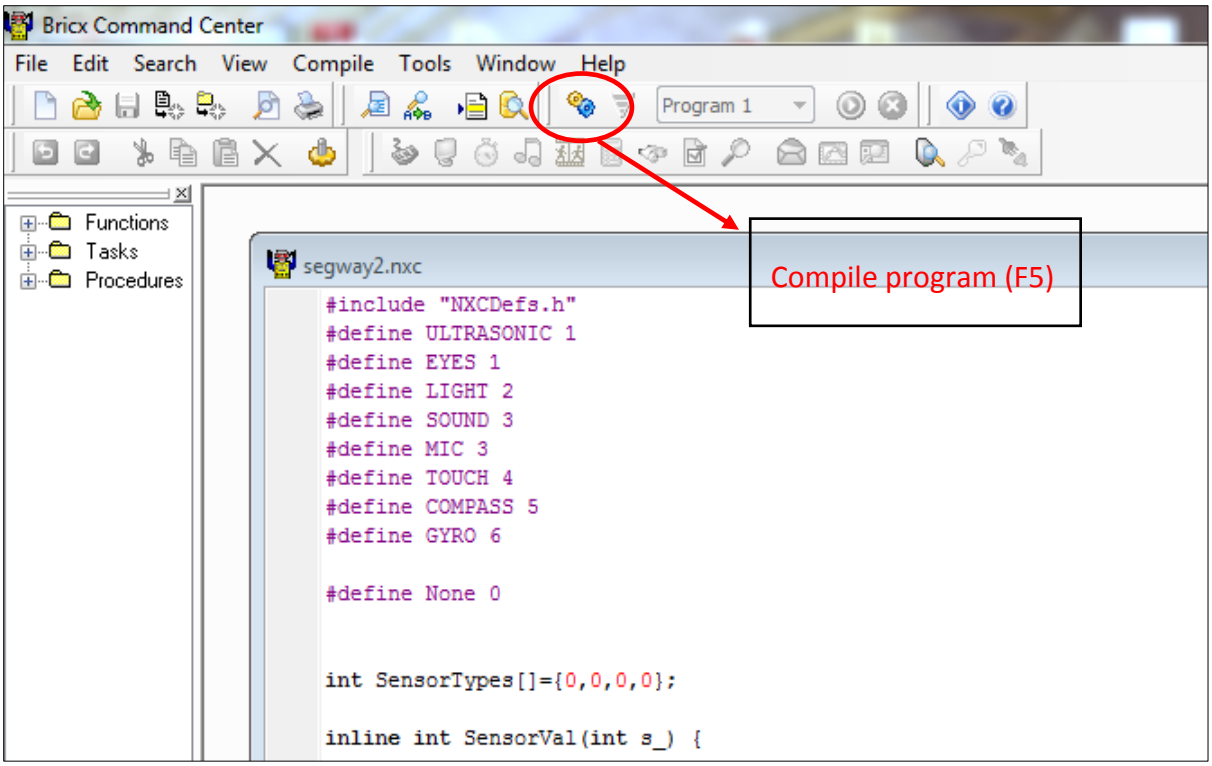


Слика 40. BricxCC

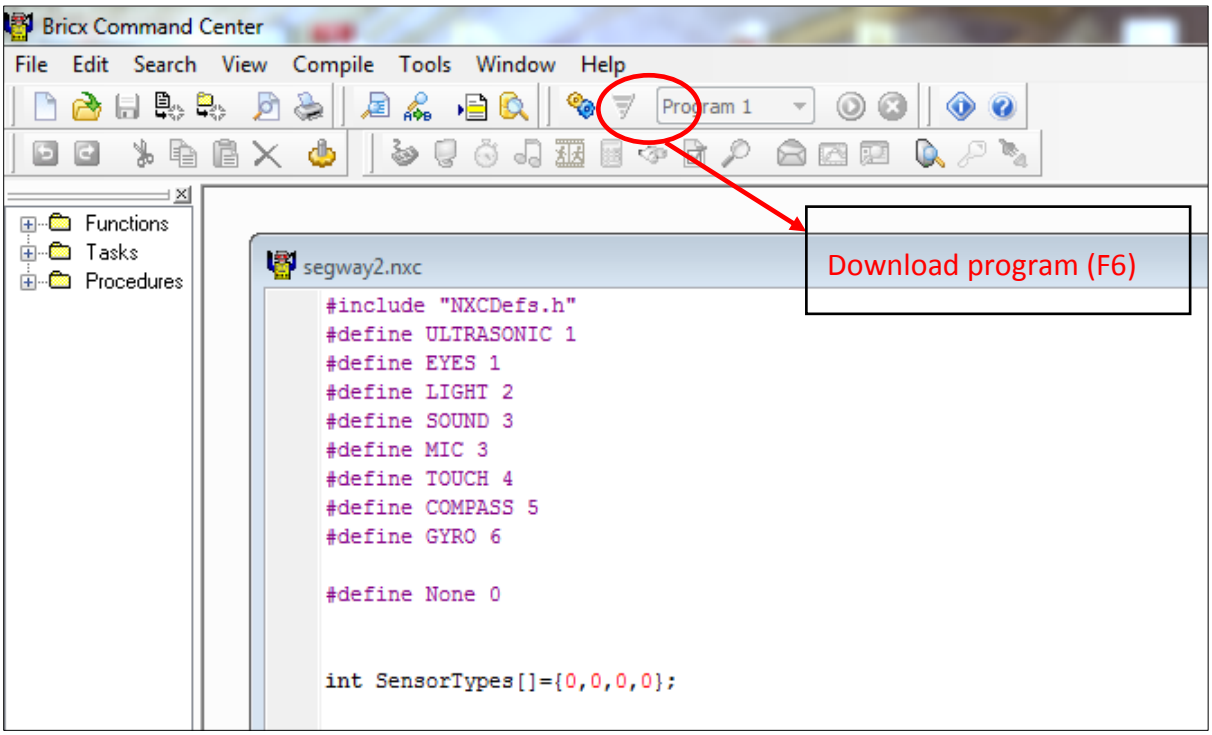
Kada otvorimo Bricx comander потребно је у директоријуму пронаћи nxc фајл који је обрађени python фајл. Одабиром на Segway 2 (у овом случају) када притиснемо open отвориће нам се програм који смо обрадили – Слика 41.



Слика 41. Nxc фајл пре спуштања на коцку



Слика 42. Припрема програма пре спуштања на коцку

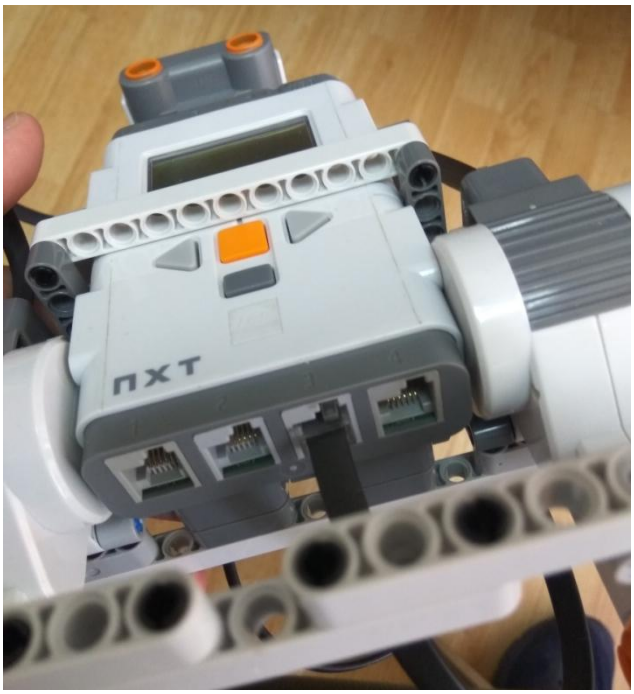


Слика 43. Спуштање програма на робота

На слици 43. приказано је опционо дугме које треба изабрати како би спустили програм на робота. Можемо видети да је иконица сива што значи да не можемо да је користимо. Разлог за то је што нисмо повезани са роботом.

Слика 44. USB конектор

Како би омогућили пренос програма на робота потребно је повезати робота и рачунар USB каблом. Када робот оствари конекцију можемо користити ицоницу за download и самим тим кренути са експериментима.



Пре пуштања програма на роботу неопходно је проверити да ли су сви улази добро повезани. На слици 45. можемо видети одређене портове које је потребно повезати са моторима робота. Портове повезати оним редом којим смо то написали у програму.

Након ове врсте провере можемо погледати и да ли је програм пребачен у меморију робота. На следећим сликама приказано је како контролишемо да ли је програм пребачен и шта је све потребно урадити да би се покренуо програм.

Слика 45. Портови Lego Segway робота

7.1. Покретање Lego робота

1. Притиснути наранџасто дугме за старт

2. Изабрати `my files` у менију робота

Свако кретање кроз мени робота врши се помоћу стрелица лево, десно. Улазак у мени програма робота врши се притиском на наранџасто дугме у средини. Наранџасто дугме нам служи и за покретање самог програм. Потребно је пронаћи датотеку коју смо спустили на робота уколико имамо више програма у меморији робота. Пример је дат са програмом под називом `moguće`.

3. Изабрати `software files`

4. Изабрати убачени програм `moguće`

5. Притиснути `moguće run`

6. Програм се покреће `kalibracija`

Када покренемо програм на екрану робота ће писати калибрација и почеће испрекидани звук о чему је било речи у претходном делу. Битна ствар коју треба напоменити је да морамо држати робота неколико секунди док траје калибрација и затим га пустити како би програм одрадио своје.

8. Експериментални део исцртавање дијаграма

У оквиру овог рада потребно је било и покренути робота и пратити његово понашање на промену параметара. Потребно је снимити видео којим ћемо показати његово кретање које ће да прате одређени дијаграми (брзине, управљања...)

Како би дошли до жељених резултата потребно је написати мали програм у самом програму чија би сврха била прикупљање података са сензора. Линија кода која је приказана испод предствља запис или чување података са сензора.

```
task main()  {
   long pocetnoVreme;
   DeleteFile("rezultat.csv");
   result = CreateFile("rezultat.csv", 32768, fH);
   SetSensorHTGyro(IN_4);
   pocetnoVreme = CurrentTick();
   while (true)  {
      if (((CurrentTick() - pocetnoVreme) <= 4000))  {
         StartTask(task_kalibracija);
         if ((svirka == 0))  {
            StartTask(task_signalizacija);
```

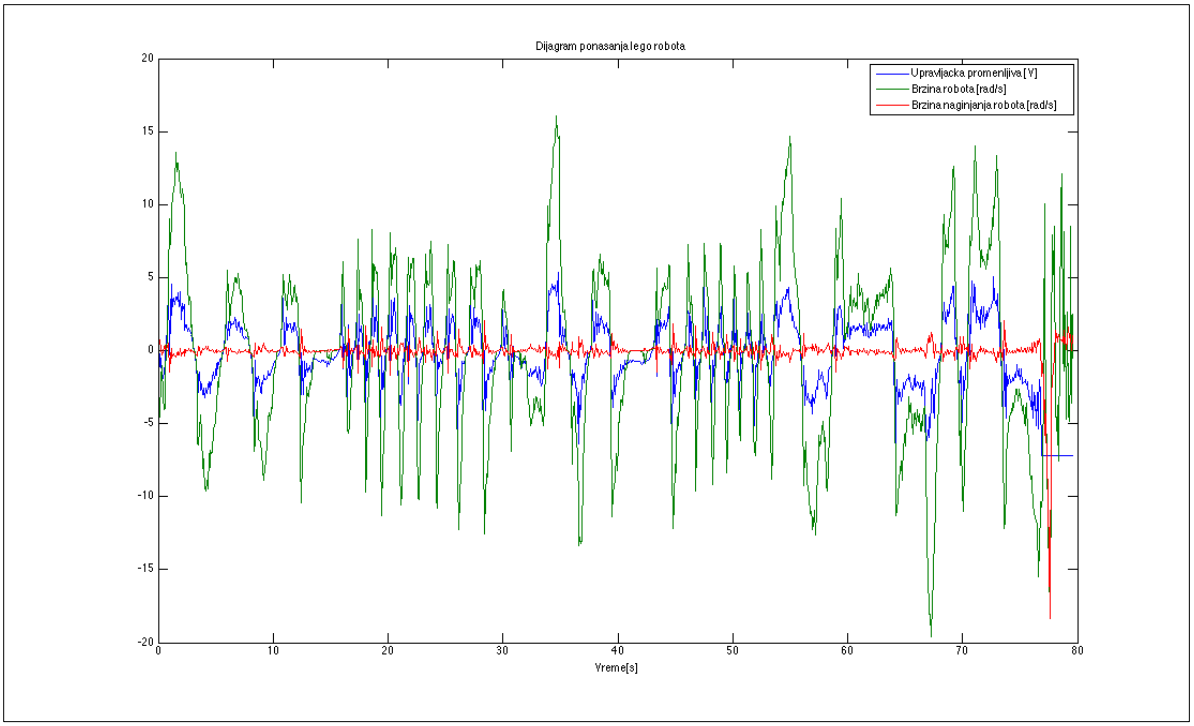
Линије кода назначене жутом бојом означавају да податке са жироскопа чувамо у фајл `rezultat.csv` на рачунару. Поновним пуштањем програм аутоматски брише и поново прави ову датотеку. Све вредности које добијемо са жироскопа добијамо у excel табели из које је лако правити дијаграме.

```
  }
   //razlikaTeta = ((ugaoDesniD - ugaoLeviD));
   razlikaTeta = (MotorTachoCount(OUT_B)-MotorTachoCount(OUT_C));
   korekcijaTeta = razlikaTeta * pojacanjeSinh;
   pwmLevi =  (upravljanjeKorigovano) - korekcijaTeta;
   pwmDesni =  (upravljanjeKorigovano) + korekcijaTeta;
   /* SetOutput(OUT_B, Power, pwmLevi, OutputMode,OUT_MODE_MOTORON,
                  RunState,OUT_RUNSTATE_RUNNING,
                  UpdateFlags,UF_UPDATE_MODE+UF_UPDATE_SPEED);
      SetOutput(OUT_C, Power,pwmDesni , OutputMode,OUT_MODE_MOTORON,
                  RunState,OUT_RUNSTATE_RUNNING,
                  UpdateFlags,UF_UPDATE_MODE+UF_UPDATE_SPEED);   */
   OnFwd(OUT_B, pwmDesni);
   OnFwd(OUT_C, pwmLevi);
   //NumOut(2, LCD_LINE2, tetaFilt);
   if (brojac++ == 10){
   string asd = StrCat(FormatNum("%d", kT++),
",",FormatNum("%3.3f",upravljanjeKorigovano),", ",FormatNum("%3.3f", tetaTacka), ", ",
FormatNum("%3.3f", psiTacka));
   WriteLnString(fH, asd,bv);
   brojac = 0;}
```

Овим делом кода смо доделили које вредности читамо са сензора и чувамо у датотеку. Снимамо податке за периоду одабирања, управљање, брзину (кретање како робот иде) и нагињање. Све ове вредности чувају се у `rezultat.csv`.

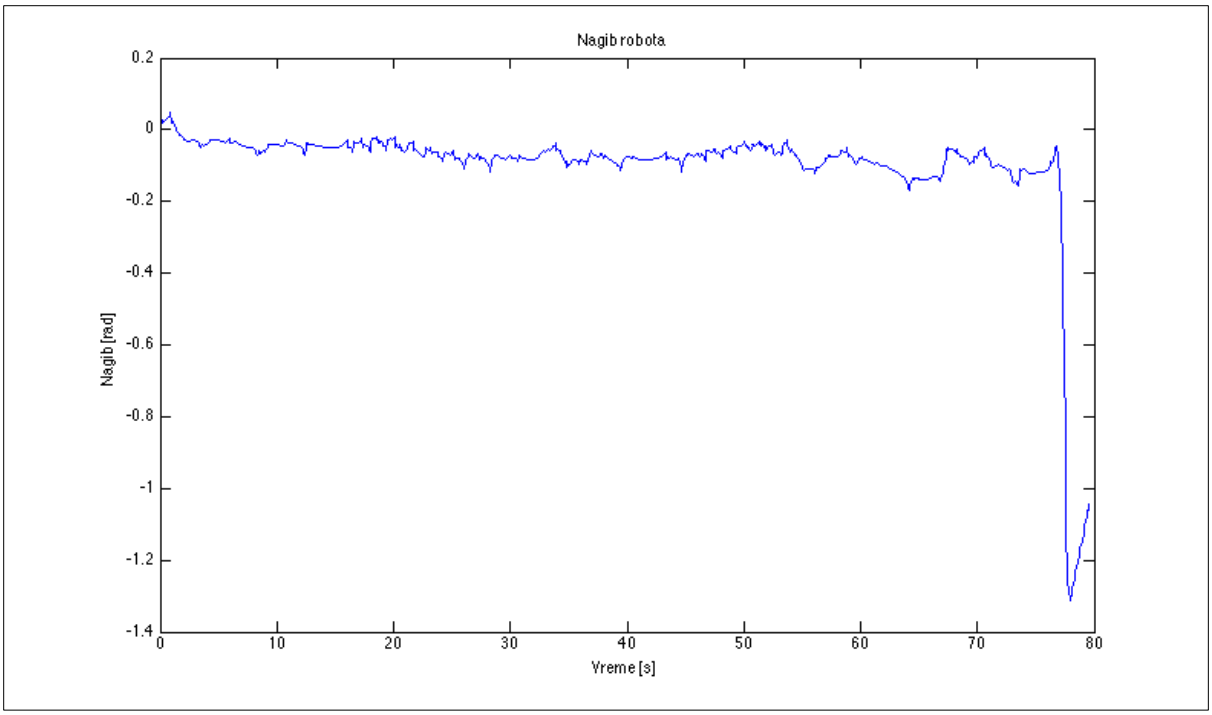
8.1. ДИЈАГРАМИ

Следећа два дијаграма приказују понашање робота приликом кретања и савлађивања препрека. На првом дијаграму је могуће видети вредности управљачке променљиве, угаоне брзине вратила мотора, и угаоне брзине нагињања робота. А на другом је приказано кретање робота до препреке и тренутак када је робот пао.



Слика 46. Дијаграм праволинијског кретања робота без препреке

Са слике 46. је могуће видети да постоји одређена линеарност између угаоне брзине мотора и вредности управљачке променљиве. Да би се ушло у увид колико се робот нагиње током рада, извршена је нумеричка интеграција података са жироскопског сензора и резултати су дати на следећем дијаграму.



Подаци су исказани у радијанима, и могуће је видети тренутак када је робот пао (77 секунда). Због саме природе нумеричке интеграције, могуће је видети на графику постојање малог дрифта.

9. ЗАКЉУЧАК

У оквиру овог мастер рада било је неопходно објаснити имплементацију контролера Lego Segway мобилног робота у програмском језику Python. Идеја самог рада је проистекла из кабинета за аутоматику где годинама у назад многи студенти имају пред собом добро организовани систем који омогућава стицање основних знања из света аутоматике. Програмирање Lego робота се учи из предмета ПСАУ и за студенте је представљао изазов који су генерације решавале и унапређивале. Овај тип учења је интересантан јер омогућава студентима да се сусретну са правим проблемима које је потребно решити кроз доста труда и залагања.

Циљ овог мастер рада је да објасни понашање једног система у смислу колико је тај систем способан да одговори изазовима. Како се не бих понављао коришћењем истих фраза око имплементације контролера поред тога резултат рада су видео записи који приказују кретање робота праволинијски у једном смеру, одржавање равнотеже и способност робота да се одупре побудама. Поред ових видео записа направљен је видео са роботом који покушава да пређе преко студентског индекса у циљу да се види колико је систем добро подешен. Видео клипове прате и дијаграми брзина, управљања које смо добили са сензора робота.

Поред циљева које сам навео мастер рад има и циљ да пружи што више информација генерацијама које долазе како би лакше и са више разумевања схватили суштину програмирања Lego робота и оставља простор за неке нове идеје и побољшања.

ЛИТЕРАТУРА

[1] М. Матијевић, Г. Јакуповић, Ј. Цар, Рачунарски подржано мерење и управљање, Машински факултет, Крагујевац, 2005

[2] Y.Yamamoto, "NXTway-GS (Self-Balancing Two-Wheeled Robot) Controller Design", Mathworks 2008. http://www.mathworks.com/matlabcentral/fileexchange/19147- nxtway-gs--self-balancing-two-wheeled-robot--controller-design

[3] M.Matijević, V.Cvjetković, V.Filipović, N.Jović, "Osnovni koncepti automatike i mehatronike sa LEGO programabilnimm robotom", Tehnika, vol. 4/2014, 2014.

[4] Yamamoto, Nottaway-GS Model Based Design – Control of self-balancing two-wheeled robot built with LEGO Mindstorms NXT, http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=13399&objectType=file, 2008.

[5] Т. Шуштершич, Семинарски рад, Управљање Lego самобалансирајућим роботом, Факултет Инжењерских наука, Крагујевац, 2017.

[6] А. Вељовић, Семинарски рад, Lego NXT, Факултет Инжењерских наука, Крагујевац 2017.

[7] A. Stanojević, R. Mitrović, N. Jović, T. Šušteršič, M. Matijević, "LEGO NXT motor u inženjerskom obrazovanju", Zbornik radova sa 59. Konferencije za elektroniku, telekomunikacije, računarstvo, automatiku i nuklearnu tehniku, ETRAN 2015, str. AU1.7.1-6 (1-6), ISBN 978-86-80509-71-6, Srebrno jezero, 8 – 11 Jun

[8] K.Ogata, "Discrete-Time Control Systems", Prentice-Hall, 1995.

ДОДАТАК

Robotic.py

```
psiInteg = 0
pojacanjePsiTacka = -7.0540
ofsetTeta = 0
pojacanjeTetaInteg = -0.4472
ofsetZiroskopa = 0
zeljenoZaokretanjePrethodno = 0
zeljenaBrzina = 0
ofsetZiroskopaPrethodni = 0
zeljenoZaokretanje = 0
pi = 3.14
baterijaFilt = 0.0
tetaFilt = 0
periodaOdabiranja = 0.004
tetaFiltPrethodno = 0
a_b = 0.8
pojacanjeTeta = -0.8
a_gd = 0.999
a_gc = 0.8
pojacanjeSinh = 0.35
pojacanjePsi = -66.3799
greskaTetaIntegPrethodno = 0
a_d = 0.8
baterijaPret = 0.0
pojacanjeTetaTacka = -1.2293*0.85
broj = 0
brzina = 0.5
#Kalibracija ziroskopa
#Srednja vrednost, 50 merenja
def kalibracija():
    while (CurrentTick < (tajmer + 5000)):
        Ziro = SensorHTGyro(S3)
        Wait(100)
        broj = broj + 1
        ofsetZiroskopa = ofsetZiroskopa + Ziro
        PlayTone(TONE_A5, 5)
    ofsetZiroskopa = ofsetZiroskopa / broj
def task_kontroler():
    dejstvoPsiTacka= 0
    dejstvoTeta= 0
    tetaTacka= 0
```

```
dejstvoPsi= 0
greskaTeta= 0
pwmDesni= 0
dejstvoTetaInteg= 0
greskaTetaTacka= 0
dejstvoTetaTacka= 0
upravljanje= 0
pwmLevi= 0
ugaoLevi= 0
psi= 0
greskaTetaInteg= 0
zeljenoPsi= 0
ziroskop= 0
psiTacka= 0
korekcijaTeta= 0
ugaoLeviD= 0
upravljanjeKorigovano= 0
greskaPsiTacka= 0
maxVolti= 0
zeljenoPsiTacka= 0
razlikaTeta= 0
ugaoDesniD= 0
greskaPsi= 0
ugaoDesni= 0
teta= 0
baterija= 0
zeljeniUgao= 0
ugaoLeviD = 0.0
ugaoDesniD = 0.0
ugaoLevi = MotorRotationCount(OUT_A)
ugaoDesni = MotorRotationCount(OUT_B)
ziroskop = SensorHTGyro(S3)
baterija = BatteryLevel()
baterijaFilt = (baterijaPret * a_b + (1 - a_b) * baterija)
baterijaPret = baterijaFilt
zeljeniUgao = (zeljeniUgao + zeljenaBrzina * periodaOdabiranja)
zeljenoPsi = 0
zeljenoPsiTacka = 0
ofsetZiroskopa = (ziroskop * (1 - a_gd) + ofsetZiroskopaPrethodni * a_gd)
ofsetZiroskopaPrethodni = ofsetZiroskopa
ugaoLeviD = 3.14 * ugaoLevi / 180
ugaoDesniD = 3.14 * ugaoDesni / 180
psiTacka = (ziroskop - ofsetZiroskopa) * pi / 180
psi = psiInteg
```

```
    psiInteg = (psiInteg + periodaOdabiranja * psiTacka)
    teta = ((ugaoLeviD + ugaoDesniD) / 2 + psi)
    tetaFilt = ((1 - a_d) * teta + a_d * tetaFiltPrethodno)
    tetaTacka = (tetaFilt - tetaFiltPrethodno) / periodaOdabiranja
    tetaFiltPrethodno = tetaFilt
    greskaTeta = (zeljeniUgao - teta)
    greskaPsi = (zeljenoPsi - psi)
    greskaPsiTacka = (zeljenoPsiTacka - psiTacka)
    greskaTetaTacka = (zeljenaBrzina - tetaTacka)
    greskaTetaInteg = greskaTetaIntegPrethodno
    greskaTetaIntegPrethodno = (greskaTetaIntegPrethodno + greskaTeta *
periodaOdabiranja)
    dejstvoTeta = greskaTeta * pojacanjeTeta
    dejstvoPsi = greskaPsi * pojacanjePsi
    dejstvoTetaTacka = greskaTetaTacka * pojacanjeTetaTacka
    dejstvoPsiTacka = greskaPsiTacka * pojacanjePsiTacka
    dejstvoTetaInteg = greskaTetaInteg * pojacanjeTetaInteg
    upravljanje = ((((dejstvoTeta + dejstvoPsi) + dejstvoTetaTacka) + dejstvoPsiTacka) +
dejstvoTetaInteg)
    maxVolti = (0.001089 * baterijaFilt - 0.625)
    upravljanjeKorigovano = upravljanje / maxVolti * 100
    pravo(upravljanjeKorigovano)
def pravo(pwm):
    #P kontroler, sinhronizovani motori
    pSinhh = (MotorTachoCount(OUT_A) - MotorTachoCount(OUT_B))*pojacanjeSinh
    SetOutput(OUT_A, Power, pwm - greskaSinh, OutputMode,
OUT_MODE_MOTORON,RunState, OUT_RUNSTATE_RUNNING, UpdateFlags,
UF_UPDATE_MODE+UF_UPDATE_SPEED)
    SetOutput(OUT_B, Power, pwm + greskaSinh, OutputMode,
OUT_MODE_MOTORON,RunState, OUT_RUNSTATE_RUNNING, UpdateFlags,
UF_UPDATE_MODE+UF_UPDATE_SPEED)

def task_main():
    kalibracija()
    while True:
        task_kontroler()
        Wait(4)
```

Segway.nxc

```
#include "NXCDefs.h"
#define ULTRASONIC 1
#define EYES 1
#define LIGHT 2
#define SOUND 3
#define MIC 3
#define TOUCH 4
#define COMPASS 5
#define GYRO 6

#define None 0

int SensorTypes[]={0,0,0,0};

inline int SensorVal(int s_) {

   s_=s_-1;
   int t_ = SensorTypes[s_];

   if (t_==None) {
      return -1;
   } else if (t_==ULTRASONIC) {
      Wait(100);
      return SensorUS(s_);
   } else if (t_ == COMPASS) {
      return SensorHTCompass(s_);
   } else if (t_ == GYRO) {
      return SensorHTGyro(s_);
   } else {
      return Sensor(s_);
   }

}

void DefineSensors(int s1,int s2,int s3,int s4) {

   int sa[];
   int i;

   ArrayInit(sa,0,4);
   sa[0]=s1;
   sa[1]=s2;
```

```
   sa[2]=s3;
   sa[3]=s4;

   for (i=0; i<4; i++) {

      switch (sa[i]) {
         case LIGHT:
            SetSensorLight(i);
            SensorTypes[i] = LIGHT;
            break;

         case ULTRASONIC:
            SetSensorLowspeed(i);
            SensorTypes[i] = ULTRASONIC;
            break;

         case SOUND:
            SetSensorSound(i);
            SensorTypes[i] = SOUND;
            break;

         case TOUCH:
            SetSensor(i,SENSOR_TOUCH);
            SensorTypes[i] = TOUCH;
            break;

         case GYRO:
            SetSensorHTGyro(i);
            SensorTypes[i] = GYRO;

         default:
            SensorTypes[i] = None;
            break;

      }
   }
}

void set_sensor(int port, int type)
{
   SensorTypes[port] = type;
   DefineSensors(SensorTypes[0], SensorTypes[1], SensorTypes[2], SensorTypes[3]);
}
```

```
#define Beep(duration) PlayTone(2600, duration)

#define set_sensors(s1, s2, s3, s4) DefineSensors(s1, s2, s3, s4)
#define sensor(port) SensorVal(port)

#define lcd_clear ClearScreen
#define lcd_reset ResetScreen
#define lcd_print_at(x, y, text) TextOut(x*6, 64 - y*8, text)

#define on_fwd OnFwd
#define on_rev OnRev
#define rotate_motor RotateMotor

#define random Random

#define wait Wait

// The value "6" means OUT_ABC which was not used for performance reasons.
inline void off(int port = 6)
{
   Off(port);
}


int _cur_lcd_line = 1;
int _line_position = 0;
void lcd_print(string text)
{
   string character;
   int char_pos = 0;

   for(int i = 0; i < StrLen(text); i++){
      character = SubStr(text, i, 1);

      if (SubStr(text, i, 1) == "\n"){
         _cur_lcd_line++;
         char_pos = 0;
         continue;
      }

      if(_cur_lcd_line > 7){
         ClearScreen();
         _cur_lcd_line = 1;
      }
```

```
      _line_position = 64 - _cur_lcd_line * 8;
      TextOut(char_pos*6, _line_position, character);
      char_pos++;
    }

    if (SubStr(text, StrLen(text)-1, 1) != "\n"){
      _cur_lcd_line++;
    }
}
int brojac = 0;
short bv;
unsigned int result;
byte fH;
long kT = 0;
float psiInteg = 0;
float pojacanjePsiTacka = -7.0540;
float ofsetTeta = 0;
float pojacanjeTetaInteg = -0.4472;
float ofsetZiroskopa = 0;
int zeljenoZaokretanjePrethodno = 0;
int zeljenaBrzina = 0;
float ofsetZiroskopaPrethodni = 0;
int zeljenoZaokretanje = 0;
float pi = 3.14;
float baterijaFilt = 0.0;
float tetaFilt = 0;
float periodaOdabiranja = 0.004;
float tetaFiltPrethodno = 0;
float a_b = 0.8;
float pojacanjeTeta = -0.8;
float a_gd = 0.999;
float a_gc = 0.8;
float pojacanjeSinh = 0.35;
float pojacanjePsi = -66.3799;
float greskaTetaIntegPrethodno = 0;
float a_d = 0.8;
int svirka = 0;
float baterijaPret = 0.0;
float pojacanjeTetaTacka = -1.2293*0.76;
task task_kalibracija()  {
    int ziroskop;
    TextOut(1, LCD_LINE1, "kalibracija");
    ziroskop = SensorHTGyro(IN_4);
```

```
    ofsetZiroskopa = (ziroskop * (1 - a_gc) + ofsetZiroskopaPrethodni * a_gc);
    ofsetZiroskopaPrethodni = ofsetZiroskopa;

}
task task_signalizacija()  {
    svirka = 1;
    PlayTone(440, 100);
    Wait(200);
    svirka = 0;

}
task task_kontroler()  {
    float dejstvoPsiTacka;
    float dejstvoTeta;
    float tetaTacka;
    float dejstvoPsi;
    float greskaTeta;
    int pwmDesni;
    float dejstvoTetaInteg;
    float greskaTetaTacka;
    float dejstvoTetaTacka;
    float upravljanje;
    int pwmLevi;
    long ugaoLevi;
    float psi;
    float greskaTetaInteg;
    int zeljenoPsi;
    int ziroskop;
    float psiTacka;
    float korekcijaTeta;
    float ugaoLeviD;
    float upravljanjeKorigovano;
    float greskaPsiTacka;
    float maxVolti;
    int zeljenoPsiTacka;
    float razlikaTeta;
    float ugaoDesniD;
    float greskaPsi;
    long ugaoDesni;
    float teta;
    int baterija;
    int zeljeniUgao;
    ugaoLeviD = 0.0;
    ugaoDesniD = 0.0;
```

```
    ugaoLevi = MotorRotationCount(OUT_C);
    ugaoDesni = MotorRotationCount(OUT_B);
    ziroskop = SensorHTGyro(IN_4);
    baterija = BatteryLevel();
    baterijaFilt = (baterijaPret * a_b + (1 - a_b) * baterija);
    baterijaPret = baterijaFilt;
    zeljeniUgao = (zeljeniUgao + zeljenaBrzina * periodaOdabiranja);
    zeljenoPsi = 0;
    zeljenoPsiTacka = 0;
    ofsetZiroskopa = (ziroskop * (1 - a_gd) + ofsetZiroskopaPrethodni * a_gd);
    ofsetZiroskopaPrethodni = ofsetZiroskopa;
    ugaoLeviD = 3.14 * ugaoLevi / 180;
    ugaoDesniD = 3.14 * ugaoDesni / 180;
    psiTacka = (ziroskop - ofsetZiroskopa) * pi / 180;
    psi = psiInteg;
    psiInteg = (psiInteg + periodaOdabiranja * psiTacka);
    teta = ((ugaoLeviD + ugaoDesniD) / 2 + psi);
    tetaFilt = ((1 - a_d) * teta + a_d * tetaFiltPrethodno);
    tetaTacka = (tetaFilt - tetaFiltPrethodno) / periodaOdabiranja;
    tetaFiltPrethodno = tetaFilt;
    greskaTeta = (zeljeniUgao - teta);
    greskaPsi = (zeljenoPsi - psi);
    greskaPsiTacka = (zeljenoPsiTacka - psiTacka);
    greskaTetaTacka = (3 - tetaTacka);
    greskaTetaInteg = greskaTetaIntegPrethodno;
    greskaTetaIntegPrethodno = (greskaTetaIntegPrethodno + greskaTeta *
periodaOdabiranja);
    dejstvoTeta = greskaTeta * pojacanjeTeta;
    dejstvoPsi = greskaPsi * pojacanjePsi;
    dejstvoTetaTacka = greskaTetaTacka * pojacanjeTetaTacka;
    dejstvoPsiTacka = greskaPsiTacka * pojacanjePsiTacka;
    dejstvoTetaInteg = greskaTetaInteg * pojacanjeTetaInteg;
    upravljanje = ((((dejstvoTeta + dejstvoPsi) + dejstvoTetaTacka) + dejstvoPsiTacka) +
dejstvoTetaInteg);
    maxVolti = (0.001089 * baterijaFilt - 0.625);
    upravljanjeKorigovano = upravljanje / maxVolti * 100;
    if ((zeljenoZaokretanje == 0))  {
        if ((zeljenoZaokretanjePrethodno != 0))  {
            ofsetTeta = (ugaoDesniD - ugaoLeviD);

        }
    }
    //razlikaTeta = ((ugaoDesniD - ugaoLeviD));
    razlikaTeta = (MotorTachoCount(OUT_B)-MotorTachoCount(OUT_C));
```

```
   korekcijaTeta = razlikaTeta * pojacanjeSinh;
   pwmLevi =  (upravljanjeKorigovano) - korekcijaTeta;
   pwmDesni =  (upravljanjeKorigovano) + korekcijaTeta;
   /* SetOutput(OUT_B, Power, pwmLevi, OutputMode,OUT_MODE_MOTORON,
                  RunState,OUT_RUNSTATE_RUNNING,
                  UpdateFlags,UF_UPDATE_MODE+UF_UPDATE_SPEED);
      SetOutput(OUT_C, Power,pwmDesni , OutputMode,OUT_MODE_MOTORON,
                  RunState,OUT_RUNSTATE_RUNNING,
                  UpdateFlags,UF_UPDATE_MODE+UF_UPDATE_SPEED);   */
   OnFwd(OUT_B, pwmDesni);
   OnFwd(OUT_C, pwmLevi);
   //NumOut(2, LCD_LINE2, tetaFilt);
   if (brojac++ == 10){
   string asd = StrCat(FormatNum("%d", kT++),
",",FormatNum("%3.3f",upravljanjeKorigovano),", ",FormatNum("%3.3f", tetaTacka), ", ",
FormatNum("%3.3f", psiTacka));
   WriteLnString(fH, asd,bv);
   brojac = 0;}
}
task main()  {
   long pocetnoVreme;
   DeleteFile("rezultat.csv");
   result = CreateFile("rezultat.csv", 32768, fH);
   SetSensorHTGyro(IN_4);
   pocetnoVreme = CurrentTick();
   while (true)  {
      if (((CurrentTick() - pocetnoVreme) <= 4000))  {
         StartTask(task_kalibracija);
         if ((svirka == 0))  {
            StartTask(task_signalizacija);

         }
         Wait(4);
      }
      else  {
         StopTask(task_kalibracija);
         StartTask(task_kontroler);
             Wait(4);

      }

   }

}
```