Univerza v Mariboru
*Fakulteta za elektrotehniko,*
*računalništvo in informatiko*

Mladen Trlep

# STABILIZACIJA VOZILA NA DVEH KOLESIH

Diplomsko delo

Maribor, november 2012

Univerza v Mariboru
*Fakulteta za elektrotehniko,*
*računalništvo in informatiko*

# STABILIZACIJA VOZILA NA DVEH KOLESIH

Študent: Mladen Trlep
Študijski program: Univerzitetni študijski program Elektrotehnika
Smer: Avtomatika in robotika
Mentor: izr. prof. dr. Rajko Svečko

Maribor, november 2012

Univerza v Mariboru
*Fakulteta za elektrotehniko,*
*računalništvo in informatiko*

Številka: E1017799
Datum in kraj: 16. 04. 2012, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 01/2010)
izdajam

**SKLEP O DIPLOMSKEM DELU**

1. **Mladenu Trlepu,** študentu univerzitetnega študijskega programa ELEKTROTEHNIKA, smer Avtomatika in robotika, se dovoljuje izdelati diplomsko delo pri predmetu Regulacije II.

2. **MENTOR: izr. prof. dr. Rajko Svečko**

3. **Naslov diplomskega dela:**
**STABILIZACIJA VOZILA NA DVEH KOLESIH**

4. **Naslov diplomskega dela v angleškem jeziku:**
**STABILITY OF TWO-WHEELED VEHICLE**

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije do 30. 09. 2012 v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:

Obvestiti:
- kandidata,
- mentorja,
- odložiti v arhiv.

## ZAHVALA

*Zahvaljujem se mentorju izr. prof. dr. Rajku Svečku za vso strokovno svetovanje v času nastajanja diplomskega dela in asistentu dr. Andreju Sarjašu, za nasvete in pomoč pri samem poteku diplomske naloge.*

*Posebna zahvala gre tudi staršema, ki sta mi omogočila študij in me spodbujala.*

# STABILIZACIJA VOZILA NA DVEH KOLESIH

**Ključne besede:** mikrokrmilnik, pospeškometer, žiroskop, stabilizacija, modeliranje, regulacija

**UDK: 621.86.033:681.51(043.2)**

**Povzetek**

Diplomsko delo se ukvarja s problematiko izdelave vozila na dveh kolesih. *Delo zajema modeliranje, opis potrebnih komponent ter izdelava vozila, na koncu pa še sintezo regulatorja in implementiranje programa na mikrokrmilniškem sistemu. Ključni deli celotnega sistema sta senzorja, ki zaznavata nagib vozila in regulator, ki je potreben za uspešno vodenje.*

# STABILITY OF TWO-WHEELED VEHICLE

**Key words:** microcontroller, accelerometer, gyroscope, stability, modeling, control

**UDK: 621.86.033:681.51(043.2)**

**Abstract**

*The thesis describes the construction of a two-wheeled vehicle. Work includes modeling, description of key components and vehicle construction, controller synthesis and implementation of the program on the microcontroller system. Key parts of the entire system are sensors, which can detect tilt of the vehicle and the conroller, which is required for successful control.*

# Kazalo vsebine

# Kazalo slik

# Uporabljeni simboli

$\theta$ - kot nagiba vozila

$M$ - vrtilni moment

$l$ - višina vozila

$m$ - masa vozila

$F$ - sila teže

$g$ - gravitacijski pospešek

$a$ - pospešek

$t$ - čas

$J$ - vztrajnostni moment

$\ddot{\theta}$ - kotni pospešek

$\dot{\theta}$ - kotna hitrost

$f$ - trenje

$C$ - regulator

$H$ - objekt vodenja

# Uporabljene kratice

USB - vodilo za zaporedni prenos podatkov (Universal Serial Bus)

DC - enosmerni tok (direct current)

PIC - programirljiv vmesniški krmilnik (Programmable Interface Controller)

UART - univerzalni asinhroni sprejemnik/oddajnik (Universal Asynchronous Receiver/Transmitter)

$I^2C$ - protokol za zaporedno komunikacijo ( Inter-Integrated Circuit)

SDA - priključek za zaporedno izmenjavo podatkov (Serial Data)

SCL- priključek za zaporedno uro (Serial Clock)

PWM - pulzno širinska modulacija (Pulse-Width Modulation)

# 1. Uvod

V industriji je veliko procesov, ki so po naravi nestabilni. Delovanje takšnih procesov brez kontrole oz. vodenja je nesmiselno, v nekaterih primerih celo nevarno. V teh situacijah je potreba po modeliranju in simulaciji nujno potrebna za uspešno vodenje procesov takšne vrste.

Tudi stabilizacija vozila na dveh kolesih je takšen proces, ki je po naravi nestabilen. Omenjeni proces si lahko predstavljamo tudi kot inverzno nihalo, ki ga poskušamo stabilizirati do te mere, da ga obdržimo v ravnovesni legi. Model inverznega nihala bomo tudi v nadaljevanju uporabili kot naš fizikalni opis problema.

Za uspešno vodenje vozila oz. fizikalnega procesa je potrebno sistem opisati z matematičnimi enačbami, kar imenujemo modeliranje. Izbrani fizikalni sistem ima lahko več matematičnih modelov, kar bomo spoznali v nadaljevanju. Šele z matematičnim modelom lahko izvedemo analizo problema, s kateri želimo določiti odzive sistema na izbrane vhodne signale ter začetna stanja.

Če z odzivom sistema nismo zadovoljni, moramo sistem korigirati. To izvedemo s sintezo in načrtovanjem regulatorja, ki bo reguliral naš sistem v želeni smeri. Ker je opisan proces nelinearen moramo za načrtovanje regulatorja celotni sistem linearizirati. Linearizacija je podrobneje opisana v poglavju o modeliranju.

Omenjeni postopki so osnova za izdelavo moje naloge, glavni cilj mojega dela pa je regulacija in vodenje sistema vozila na dveh kolesih. Vmesni cilji pri tem bodo, kot že prej omenjeno modeliranje nelinearnih in linearnih sistemov, načrtovanje in sinteza regulatorja, izdelava prototipa vozila in izdelava programa, ki bo skrbel za nadzor nad celotnim sistemom. Na koncu se bom posvetil še testiranju celotnega sistema na realnem primeru.

# 2. Modeliranje

Ena od prednosti modeliranja je, da ne potrebujemo fizičnega sistema, tako je možno načrtovanje novih sistemov brez izdelave prototipov. Modeliranje je postopek, ki omogoča, da s pomočjo matematičnega modela poljubnega fizikalnega sistema le-tega načrtujemo, v idealnem primeru, brez izdelave prototipa. V realnem primeru pa nam takšen pristop bistveno zmanjša potrebno število prototipov in skrajša čas razvoja novega sistema oz. izdelka. Prav tako je delo na modelu lažje kot na realnem sistemu. Samo simuliranje procesa je varno in ne predstavlja nevarnosti, kot bi se to lahko izkazalo na realnem procesu. Pomembna lastnost modeliranja je tudi, da nam pomaga spoznati in razumeti realne sisteme in njihove odzive na zunanje motnje.

V našem primeru je fizikalni problem znan, zato ga bomo zapisali z matematičnimi enačbami, da bomo dobili tako imenovani matematični model. Do njega lahko pridemo na dva načina, z uporabo fizikalnega modeliranja, kjer uporabimo osnovne fizikalne principe in naravne zakone za opis sistema ali pa z identifikacijo sistema, kjer z opazovanjem in eksperimenti določimo model. Slabost drugega načina je, da potrebujemo prototip realnega sistema, ki pa ga v našem primeru nimamo, zato bomo modelirali po prvem načinu – z fizikalnim modeliranjem.

Kot smo omenili že v uvodu, bomo za opis procesa uporabili inverzno nihalo, ki je odličen približek našega procesa. S pomočjo le tega bomo sestavili matematični model.

Preden pa začnemo z modeliranjem se je potrebno držati določene metodologije. Najprej strukturiramo primer, kar pomeni v našem primeru, da s premikanjem spodnjega dela nihala vplivamo na ravnotežje celotnega sistema – nihalo držimo v ravnotežni legi. Nato postavimo definicijo namena modela, kar pomeni, da bomo z našim modelom lahko simulirali fizikalni model, ter s pomočjo rezultatov in odzivov določili parametre našega regulatorja.

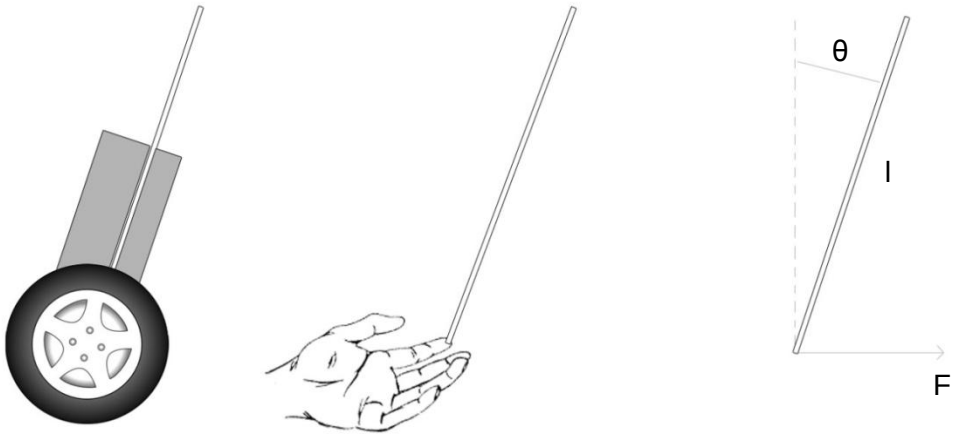
## 2.1 Metode

Modeliranje fizikalnega procesa lahko izvedemo na dva načina: z uporabo Newton-ovih fizikalnih zakonov ali po Lagrangeu. Modeliranje z Newton-ovimi enačbami temelji na ravnotežni enačbi vseh sil, medtem ko modeliranje s pomočjo Lagranga temelji na energijah, ki so prisotne v sistemu.

V našem primeru bom modeliranje izvedel z uporabo Newtonovih enačb.

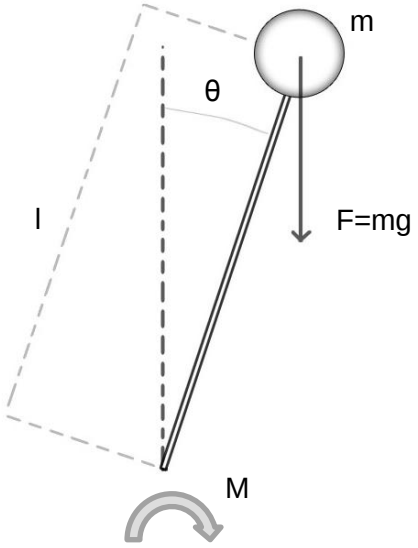
## 2.2 Inverzno nihalo

Koncept ohranjanja stabilne pozicije vozila na dveh kolesih je zelo podoben stabiliziranju palice v spodnji legi s konico prsta. Matematični objekt, ki ga opisuje to gibanje imenujemo inverzno nihalo.




*Slika 2.1: Vozilo na dveh kolesih, stabilizacija palice na konici prsta in inverzno nihalo*

Objekt, ki ga bomo stabilizirali, torej vozilo na dveh kolesih, bomo predstavil kot matematični model inverznega nihala. To nam bo služilo kot simulacijski objekt, na katerem bomo izvedli modeliranje, simulirali odziv na začetni pogoj, izpeljali prenosno funkcijo in z lineariziranim modelom nato tudi izvedli sintezo regulatorja.
Spodnja slika prikazuje model inverznega nihala, ko ga bomo uporabljal v naši nalogi.




*Slika 2.2: Inverzno nihalo*

Kot $\theta$ opisuje odmik od zgornje ničelne lege. Na nihalo lahko vplivamo z vrtilnim momentom $M$ v skrajni spodnji točki. Kot vidimo iz slike, je sila teže $F = m \cdot g$ tista, ki sili nihalo iz ravnovesne lege.

Opis sistema z matematičnimi enačbami:

Izhajamo iz 1. Newtonovega zakona $F = m \cdot a$. Ta nam pove, da sila *F* generira pospešek $a = F/m$. Če upoštevamo, da objekt stoji v začetni točki $x(0) = 0$ in nanj delujemo s silo *F*, potem bo naš objekt po času *t* v točki $x(t) = F \cdot t^2/m$ .

Ker je gibanje nihala rotacijsko, uvedemo naslednjo povezavo: $M = J \cdot \ddot{\theta}(t)$, kjer je *J* vztrajnostni moment, *M* pa predstavlja vrtilni moment na nihalo. Sedaj lahko zapišemo ravnotežno enačbo:

$$J \cdot \ddot{\theta}(t) = M(t) + m \cdot g \cdot l \cdot sin\theta(t) - f \cdot \dot{\theta}(t) \tag{2.1}$$

kjer:

- $M(t)$ predstavlja vrtilni moment na nihalo v spodnji točki
- $m \cdot g \cdot l \cdot sin\theta(t)$ predstavlja vpliv sile gravitacije in
- $f \cdot \dot{\theta}(t)$ predstavlja trenje, proporcionalno kotni hitrosti.

Vztrajnostni moment nihala je odvisen od tega, kako je porazdeljena masa po celotni dolžini nihala. V našem primeru bomo predpostavili, da je celotna masa skoncentrirana na vrhu nihala, iz tega sledi $J = m \cdot l^2$.

Sedaj lahko zapišemo končno enačbo, ki se glasi:

$$m \cdot l^2 \cdot \ddot{\theta}(t) = M(t) + m \cdot g \cdot l \cdot sin\theta(t) - f \cdot \dot{\theta}(t). \tag{2.2}$$

Rešitev te diferencialne enačbe opisuje gibanje nihala.

## 2.3 Linearizacija

Za potrebe izračuna regulacijskih algoritmov je potrebno naš model linearizirati. To izvedemo z metodo malih kotov, pri kateri predpostavimo, da je pri majhni spremembi kota *θ* izračunan sinus kota približno enak samemu kotu: $sin\theta \approx \theta$.

Tako ima naša linearizirana enačba naslednjo obliko:

$$m \cdot l^2 \cdot \ddot{\theta}(t) = M(t) + m \cdot g \cdot l \cdot \theta(t) - f \cdot \dot{\theta}(t). \tag{2.3}$$

Linearizirano enačbo lahko nato s pomočjo Laplaceove transformacije preslikamo v *s*-ravnino, tako da dobimo prenosno funkcijo našega sistema:

$$m \cdot l^2 \cdot \ddot{\theta}(t) = M(t) + m \cdot g \cdot l \cdot \theta(t) - f \cdot \dot{\theta}(t)$$

*Laplaceova transformacija*

$$m \cdot l^2 \cdot s^2 \cdot Y(s) = M(s) + m \cdot g \cdot l \cdot Y(s) - f \cdot s \cdot Y(s). \tag{2.4}$$

Po krajši preureditvi dobimo prenosno funkcijo:

$$H(s) = \frac{Y(s)}{M(s)} = \frac{1}{m \cdot l^2 \cdot s^2 + f \cdot s - m \cdot g \cdot l} \,. \tag{2.5}$$

Dobljeno prenosno funkcijo nato zapišemo v programski paket Matlab, kjer izvedemo nadaljnjo obdelavo sistema, preverimo stabilnost, primerjamo lahko linearizirani in nelinearizirani model.

## 2.4 Simuliranje v Matlab, Simulink programskem paketu

MATLAB je programsko okolje za razvoj algoritmov, analize podatkov, vizualizacijo in numerično računanje. Z uporabo MATLABA lahko rešimo tehnične računske probleme enostavneje kot z tradicionalnimi programskimi jeziki, kot so C, C++ in Fortran.

Program je napisan tako, da omogoča enostavno izvajanje matričnih operacij, reševanje diferencialnih enačb z numerično integracijo, grafični prikaz rezultatov vključno z animacijami. V tako imenovanih *m*-datotekah lahko v programskem jeziku, ki je zelo podoben programskemu jeziku Basic, zapišemo lasten program.

Poleg osnovne verzije MATLAB-a so na razpolago tudi orodja, ki jih lahko namestimo. Orodja vsebujejo množico funkcij, ki jih lahko uporabimo v svojih programih. Funkcije so običajno zapisane za vse najbolj pogoste operacije s področja, ki jih posamezno orodje pokriva.

Orodja obsegajo množico različnih področij, naštejmo jih samo nekaj:

- orodje za regulacijsko tehniko,
- orodje za numerično identifikacijo sistema,
- orodje za simbolično reševanje enačb,
- orodje za delo v realnem času in še vrsta drugih.

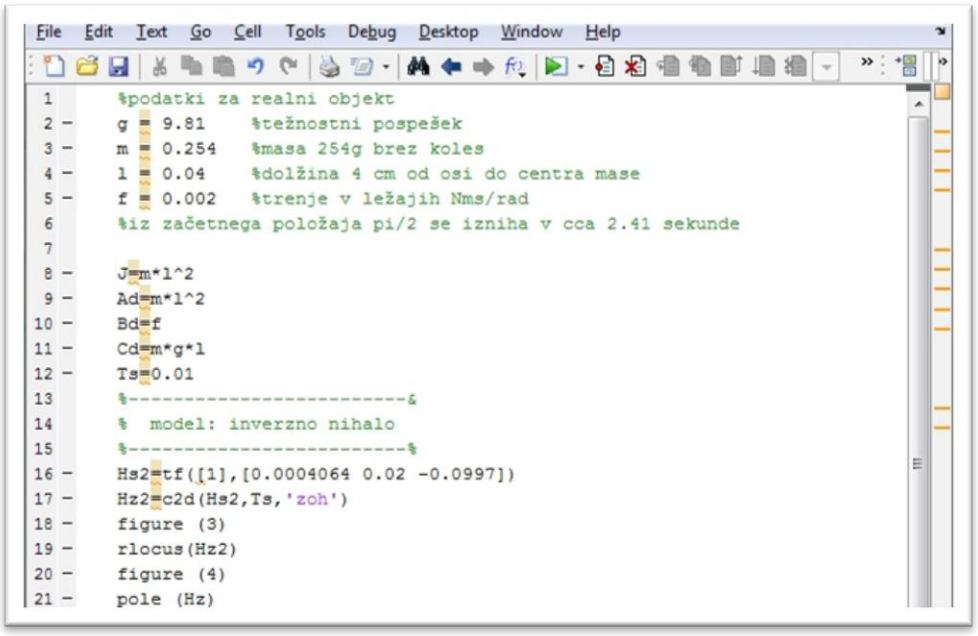
Na internetu lahko najdemo tudi orodja, ki so jih izdelali posamezniki in so prosto dostopna za uporabo.

V samem programu je odlično implementirana funkcija *HELP* – pomoč, ki jo lahko uporabimo kadarkoli, pomaga nam v primeru, ko ne vemo kakšna je sintaksa za določen ukaz ali pa nam na podlagi primera prikaže uporabo posameznega ukaza ali funkcije.

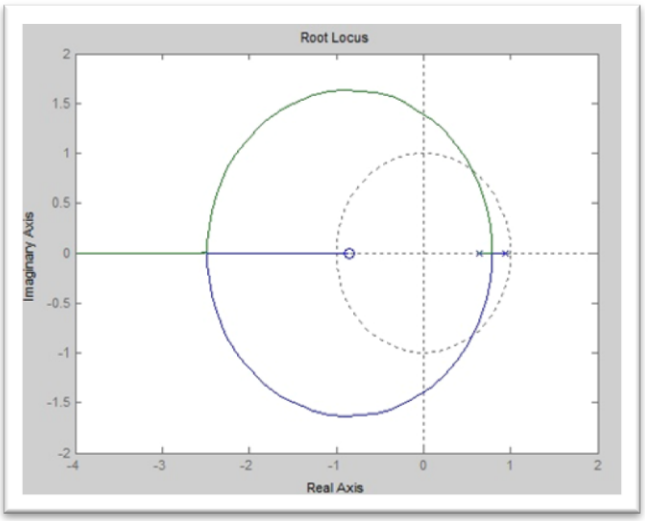
SIMULINK je okolje za simulacije in načrtovanje dinamičnih in implementiranih sistemov na podlagi modelov. Ponuja interaktivno grafično okolje in knjižnico z bloki, ki jih lahko prilagajamo. To nam omogoča načrtovanje, simulacijo, implementiranje in testiranje množice časovno odvisnih sistemov, kar vključuje telekomunikacije, vodenje, signalno procesiranje ter slikovno in video procesiranje.

SIMULINK je integriran s programom MATLAB, kar ponuja takojšen dostop do razširjenega področja orodij, ki nam omogočajo razvoj algoritmov, analize in vizualizacije simulacij, določanja parametrov okolja in definiranja signalov, parametrov in testnih podatkov.
Program Matlab je izdelek podjetja MathWorks, Inc [2].

Za potrebe simulacije smo najprej kreirali novo *m* datoteko, v katero smo vnesli parametre za simulacijo. Nato smo zapisali prenosno funkcijo (2.5), ki smo jo z ukazom *'c2d'* preslikali iz zvezne funkcije v diskretno. Z ukazom *'zoh'* smo nato določili, da se zadrži otipana vrednost in izvedli test stabilnosti, v našem primeru smo izrisali diagram lege korenov. Ugotovili smo, da je sistem stabilen.



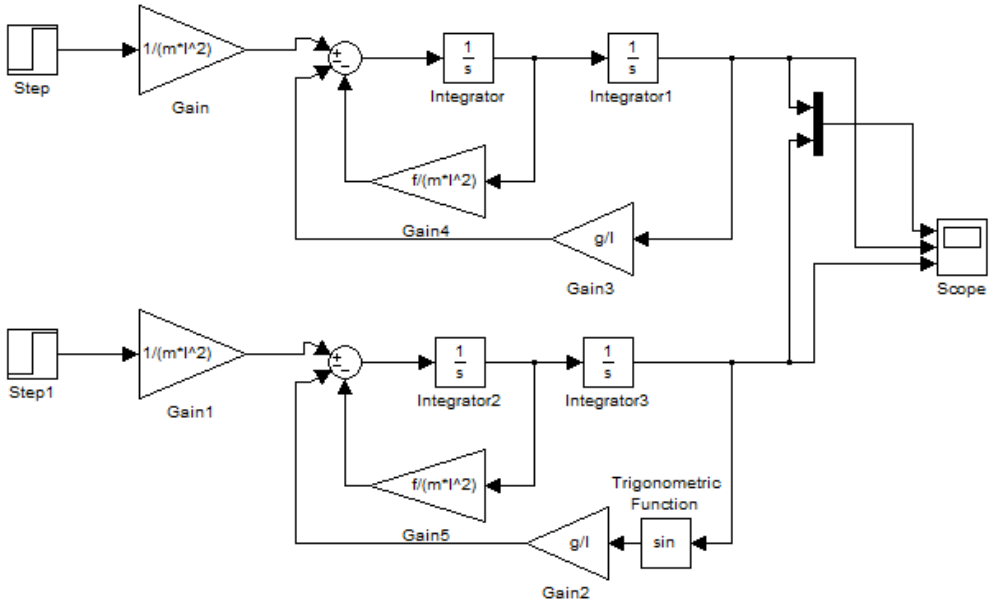

*Slika 2.3: m datoteka s parametri simulacije*



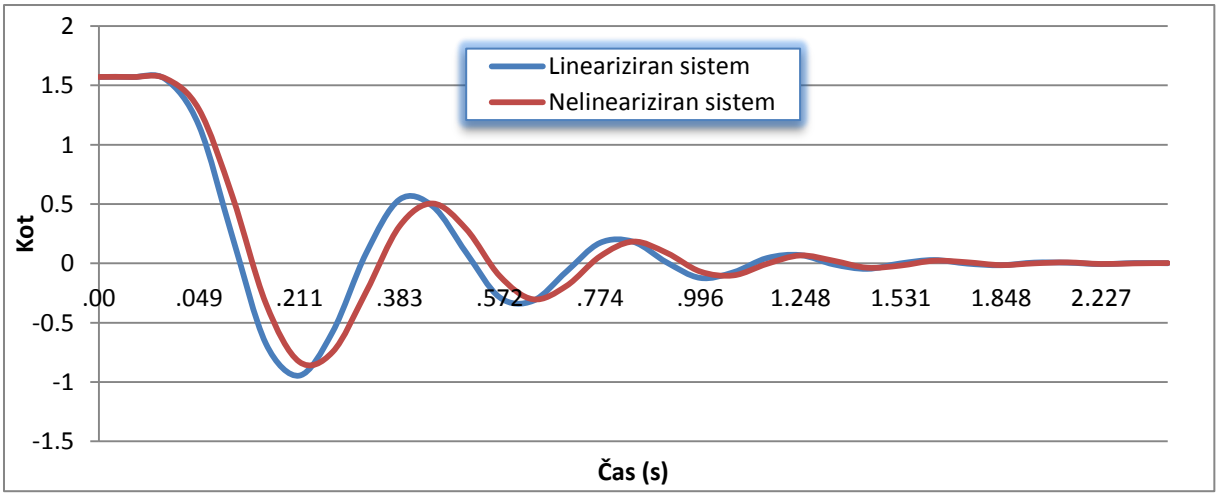

*Slika 2.4: Krivulja lege korenov, ničla ima vrednost -0.8488, pola pa 0.9453 in 0.6467*

Nato smo v programu Simulink sestavili blokovni diagram, ki opisuje naš sistem. Vzporedno smo naredili dva sistema, enega nelineariziranega in enega linearizirranega, da lahko vidimo, do kakšne napake pride pri linearizaciji.

Za samo simulacijo nismo uporabili modela inverznega nihala, ampak normalno nihalo, kateremu smo postavili začetni pogoj $\pi/2$, torej 90°. Sedaj lahko opazujemo, kako se sistem obnaša. To metodo smo uporabili tudi za to, da smo eksperimentalno določili kakšno je trenje v ležaji realnega objekta (podrobneje je to opisano v poglavju 5). Realni objekt smo postavil v začetno lego pri kotu 90° in merili čas v katerem se izniha. Nato smo v simulaciji tako dolgo spreminjali parameter dušenja, dokler nismo dobili sprejemljivega odziva.




*Slika 2.5: Simulacijska shema za primerjavo linearizirranega in nelineariziranega sistema*




*Slika 2.6: Rezultati simulacijske sheme in primerjava med obema odzivoma*

Iz rezultatov simulacije vidimo, da je razlika med odzivoma zelo majhna, tako da lahko linearizirran model uporabimo pri sintezi regulacijskega algoritma.

# 3. Sinteza regulatorja

Za uspešen regulacijski sistem je značilno, da je stabilen, ime sprejemljive odzive na vhodna vzbujanja, ima minimalno stacionarna odstopanje, odpravlja vplive motenj ter je manj občutljiv na spremembe parametrov. Zaradi vseh teh pogojev je težko pričakovati, da bo sistem imel optimalne lastnosti brez posebnih nastavitev. Lahko se celo zgodi, da so si določene zahteve pri regulaciji v nasprotju, kar privede do tega, da če izboljšamo eno s tem posredno poslabšamo drugo zahtevo. Zato je potrebno sprejeti nek kompromis in nastaviti zahteve tako, da jih je regulator zmožen izpolniti in ima še vedno sprejemljivo delovanje.

Želene lastnosti pogosto ni mogoče doseči samo s spreminjanjem parametrov, ampak je pogosto potrebno tudi najprej preoblikovati sistem z dodajanjem novih elementov, šele nato lahko dosežemo želeno obnašanje sistema. Ta postopek imenujemo sinteza regulacijskega sistema.

Sinteza regulacijskega sistema pa obsega več področij, potrebno je preučiti zahteve za regulacijo, načrtati strukturo samega sistema, izbrati primerne komponente in ustrezno nastaviti parametre. Če sistem spreminjamo oziroma v njega dodajamo nove komponente s ciljem, da dosežemo želeno zaprtozančno obnašanje imenujemo to kompenzacija. [1]

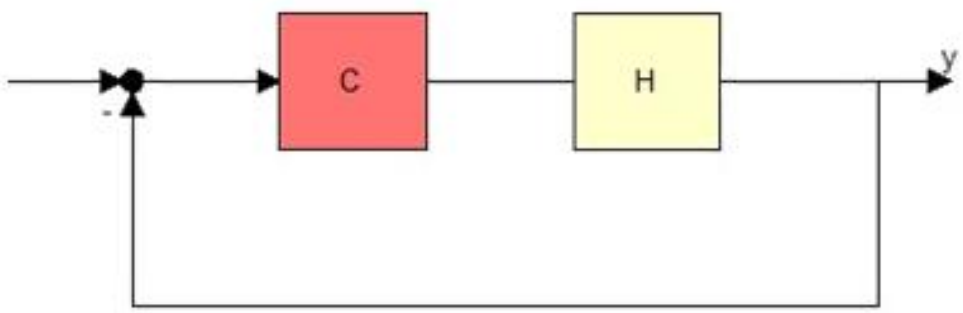
## 3.1 Metode

Regulator lahko načrtujemo na več načinov, pri tem pa lahko postopamo po različnih postopkih, tako poznamo naprimer polinomsko sintezo, sintezo v prostoru stanj. Drugi način za načrtovanje regulacijskega algoritma pa je s pomočjo temu namenjenih programov. V našem primeru bomo izdelali regulator s pomočjo programa znotraj Matlaba, ki se imenuje RLtool.

## 3.2 Sinteza regulatorja s pomočjo Matlaba

Čeprav so metode, pri katerih sami ročno izračunavamo regulator, dobre da spoznamo samo strukturo in načrtovanje, pa so slednje zelo zamudne in nam, pri načrtovanju realnega sistema vzamejo dosti časa. Zato smo se odločili, da bomo v našem primeru izvedli sintezo s programom Matlab, znotraj katerega imamo v ta namen pripravljen podprogram RLtool. Le ta nam je v veliko pomoč, saj lahko opazujemo obnašanje sistema v odprti in zaprti zanki, ali je ta stabilen in kaj povzročijo različne vrste regulatorja na naš celotni objekt vodenja.

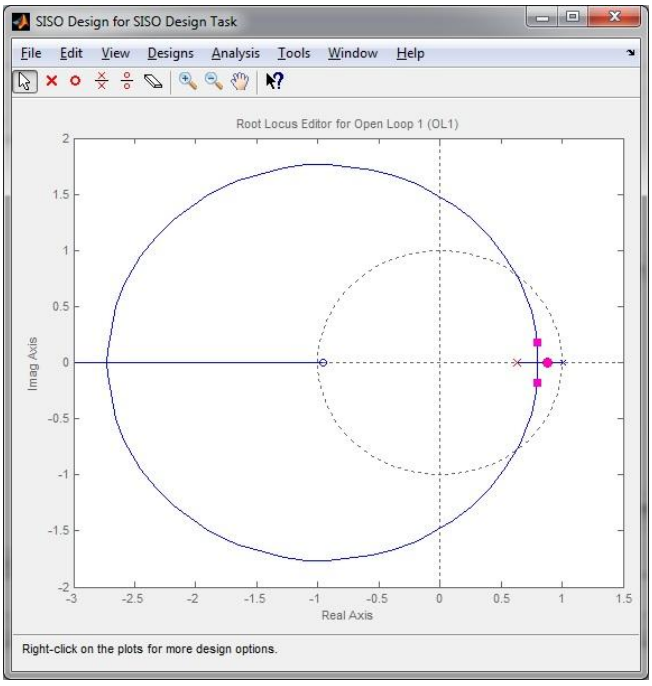
Ko poženemo program moramo najprej določiti, kakšno strukturo sistema želimo, v našem primeru smo izbrali regulator in objekt vodenja v isti veji, sistem pa ima negativno povratno vezavo. Blok »*C*« predstavlja regulator, blok »*H*« pa prenosno funkcijo našega modela.




*Slika 3.1: Shema sistema v zaprti zanki*

Nato lahko izbiramo med naslednjimi zavihki: urejevalnik regulatorja, grafično načrtovanje, izrisi za analizo podatkov in avtomatsko načrtovanje.

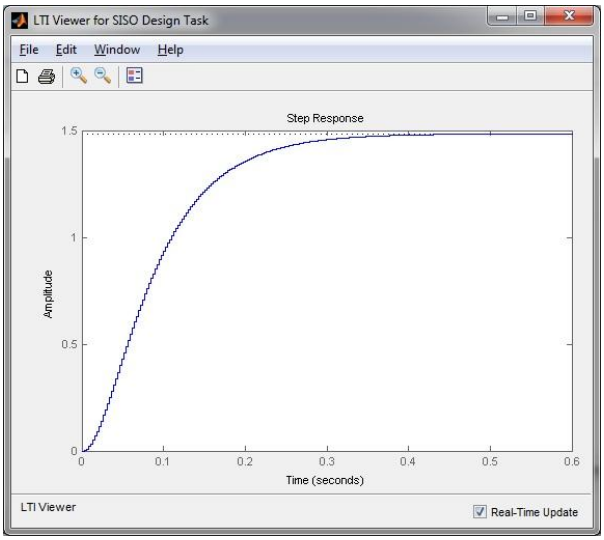
Odločili smo se za avtomatsko načrtovanje, znotraj katerega smo izbrali metodo načrtovanja PID, enačbo regulatorja smo dobili glede na naš objekt vodenja, izbrali smo metodo »robustni odzivni čas« ter določil tip regulatorja PD. Program je glede na vhodne parametre in naše nastavitve izrisal diagram lege korenov, iz katerega vidimo, da je sistem stabilen, ter še ima nekaj rezerve, ki jo lahko uporabimo za fino nastavitev obnašanja regulatorja.



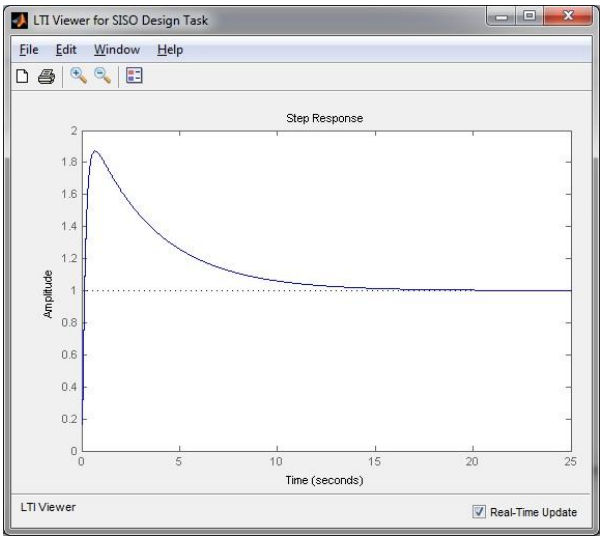

*Slika 3.2: Diagram lege korenov za obravnavani sistem*

Program nam tudi prikaže odziv sistema na stopnico. Pri načrtovanju regulatorja težimo k temu, da je odziv hiter, ima majhen prenihaj in se hitro izniha. Vsi ti pogoji pa so med seboj

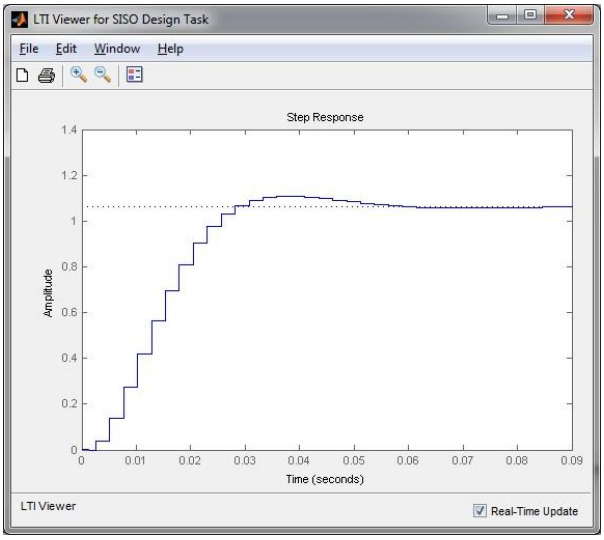
povezani in v realnosti ni mogoče, da bi imeli vse zahteve idealne, zato je potrebno najti nek kompromis, s katerim bo naš sistem deloval znotraj še sprejemljivih odzivov.
Na spodnjih slikah lahko vidimo, kakšne odzive dobimo, če spreminjamo tip regulatorja pri načrtovanju. Najboljši odziv je bil pri PD regulatorju, zato smo ga uporabili za naš sistem.



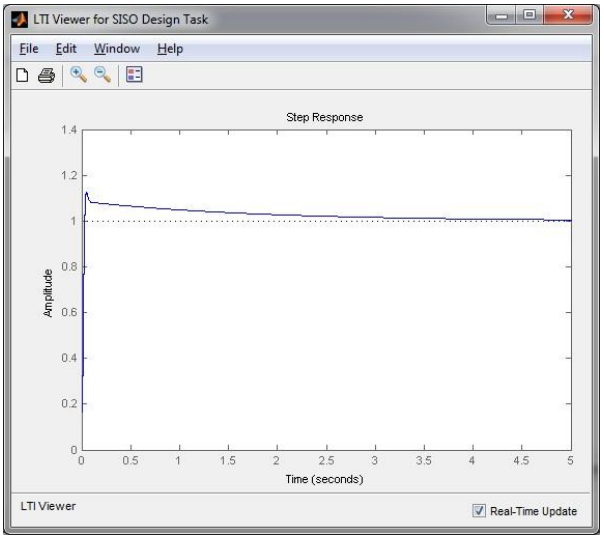

Slika 3.3: Odziv P regulatorja




Slika 3.4: Odziv PI regulatorja




Slika 3.5: Odziv PD regulatorja




Slika 3.6: Odziv PID regulatorja

Kot vidimo iz slike 3.3 je odziv P regulatorja brez prenihaja, ampak traja skoraj pol sekunde, da doseže želeno vrednost, to pa ni sprejemljivo v našem primeru, saj potrebujemo hitre odzive, če želimo uspešno stabilizirati vozilo.
V primeru PI regulatorja (slika 3.4) je odziv nesprejemljiv za naše potrebe, saj ima zelo velik prenihaj, ki nato potrebuje skoraj 15 sekund, da se ustali na želeno vrednost. Takšen regulator je naš sistem neuporaben.

Ko smo testirali odziv PID regulatorja je prišlo do podobnega rezultata, kot pri prejšnjem. Iz grafa na sliki 3.6 vidimo sicer zelo hiter odziv z manjšim prenihajem, ampak da se vrednost ustali na želeno, traja več kot tri sekunde. Tudi takšen regulator za naš sistem ni sprejemljiv.
V nadaljevanju smo nato preizkusili še delovanje PD regulatorja, katerega odziv prikazuje slika 3.5. Ta regulator se je po pričakovanjih izkazal za najboljšo izbiro. Odziv na stopnično spremembo je zelo hiter, prenihaj je majhen in se zelo hitro izniha. V manj kot 0.06 sekunde prehodni pojav izveni. Dinamika tega regulatorja je odlična za naš sistem in ga bomo uporabili v implementaciji znotraj programa.
Preden pa lahko to storimo, je potrebno izbrani regulator izvoziti iz RLtool orodja nazaj v Matlab.

$$C = 1.6779 \cdot \frac{(1-0.021 \cdot w)}{(1+0.0069 \cdot w)}, w = (z-1)/Ts \tag{3.1}$$

$$C = \frac{5.0523 \cdot (z-0.8773)}{(z-0.6305)} \tag{3.2}$$

Enačbo regulatorja (*C*) moramo nato še zapisati v diferenčni obliki, saj jo le tako lahko implementiramo v naš program. Spodaj lahko vidimo postopek:

$$C = \frac{izh(z)}{(e)} \tag{3.3}$$

$$\frac{izh(z)}{e(z)} = \frac{5.0523 \cdot (z-0.8773)}{(z-0.6305)} \tag{3.4}$$

$$z \cdot izh(z) - 0.6305 \cdot izh(z) = z \cdot 5.0523 \cdot e(z) - 4.4324 \cdot e(z) \tag{3.5}$$

$$izh(k+1) - 0.6305 \cdot izh(k) = 5.0523 \cdot e(k+1) - 4.4324 \cdot e(k) \tag{3.6}$$

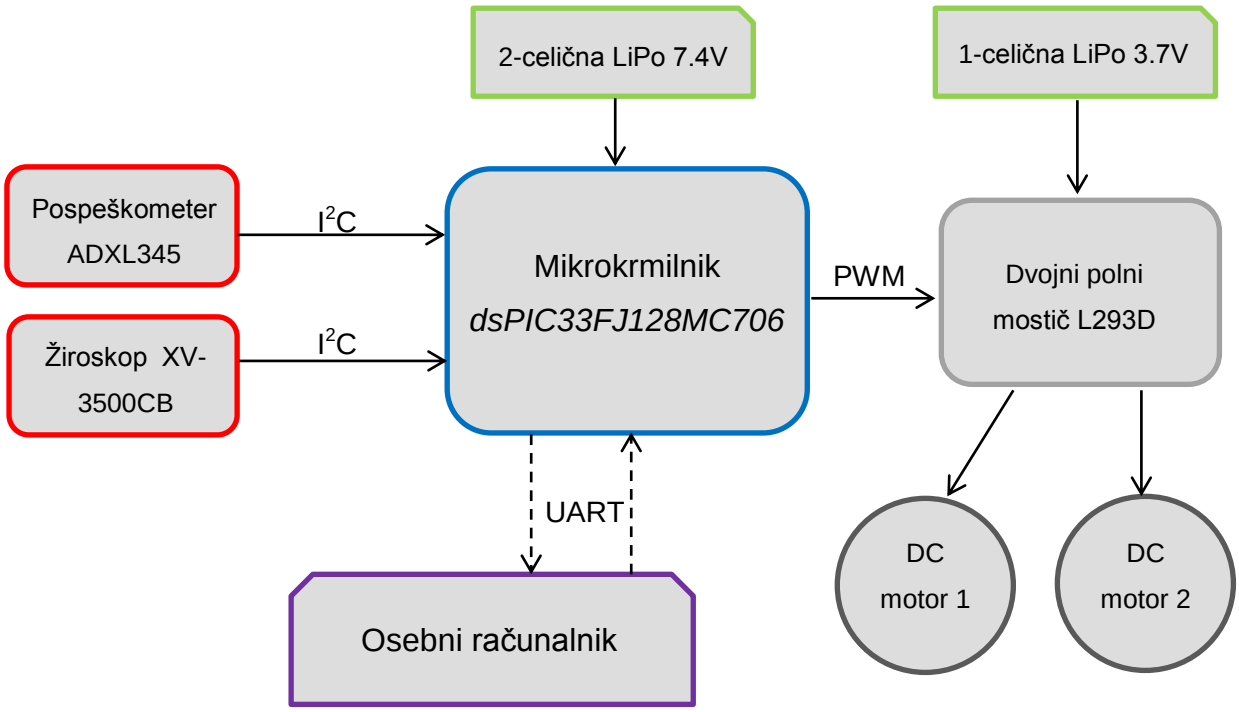
$$k+1 \rightarrow k$$

$$izh(k) = 0.6305 \cdot izh(k-1) + 5.0523 \cdot e(k) - 4.4324 \cdot e(k-1) \tag{3.7}$$

Zadnja enačba (3.7) predstavlja regulator v diferenčni obliki, ki smo ga nato implementirali znotraj programa.

# 4. Komponente sistema

Vodenje sistema bomo izvedli s pomočjo dsPIC33F mikrokrmilnika, dveh senzorjev, pospeškometra in žiroskopa, na izhodu sistema pa bomo imeli dvojni polni mostič L293D, na katerega bosta povezana dva DC motorja. Vsak motor poganja eno kolo. Za komunikacijo med senzorji in mikrokrmilnikom bomo uporabili $I^2C$ (ang. Inter-Integrated Circuit) vodilo, ki je tudi najpogostejši način komunikacije med integriranimi vezji. Vodilo je sestavljeno iz dveh aktivnih dvosmernih linij (SCL in SDA) ter skupne mase [7]. Za pogon motorjev preko H mostiča L293D pa bomo uporabil PWM signal. Celoten sistem bosta napajali dve LiPo bateriji. Mikrokrmilnik komunicira z računalnikom preko UART protokola. V našem primeru ga uporabljamo za nalaganje programa na mikrokrmilnik in za spremljanje ter kontrolo podatkov, ko se program izvaja. Na spodnji sliki lahko vidimo strukturo celotnega sistema.




*Slika 4.1: Zasnova sistema*

V nadaljevanju sledi predstavitev posameznih komponent sistema in njihove lastnosti.

## 4.1 dsPIC33 razvojna plošča


Na sliki 4.1 lahko vidimo razvojno ploščo, s pomočjo katere bomo sestavil naš sistem vodenja. Na njej se nahaja *dsPIC33FJ128MC706* mikrokrmilnik, to je visoko zmogljivi 16-bitni digitalni signalni krmilnik (DSC – digital signal controller). Plošča se lahko napaja preko USB vhoda ali pa preko zunanjena napajanja, ki je lahko od 7 pa do 9 V. Tukaj sta tudi dve tipki, ena za vklop in izklop, druga pa za reset mikrokrmilnika.

*Slika 4.2: dsPIC33F razvojna plošča*

Nekaj tehničnih podatkov:

-do 40 MIPS (miljonov operacij na sekundo)

-do 85 programirnih digitalnih V/I

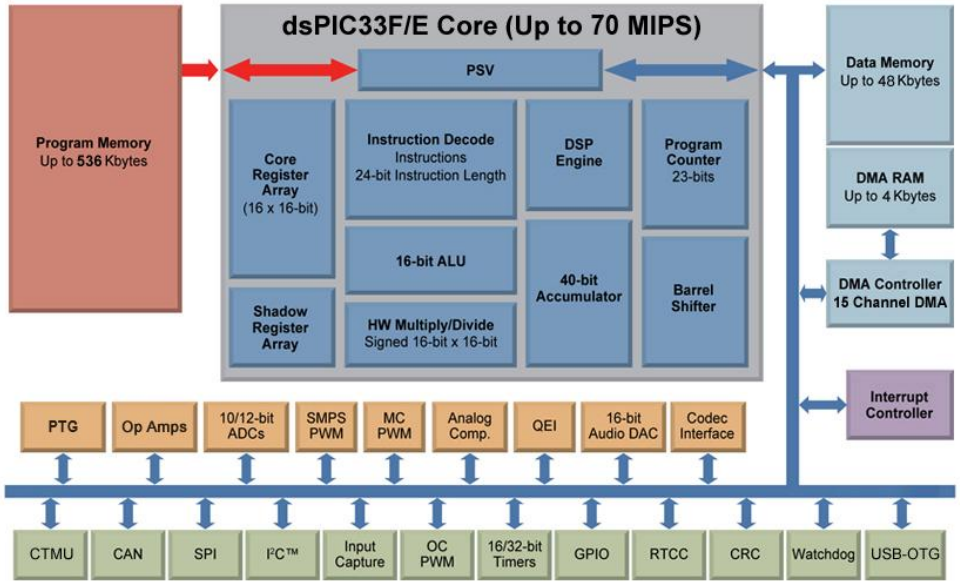
-do 256 flash pomnilnika

-podprta komunikacijska vodila: SPI, $I^2C$, UART, ECAN

-do 2 ADC modula

-do 8 kanalni PWM, več pa lahko najdete na internetni strani proizvajalca [4].

Kot zanimivost prilagam še blokovni diagram, kjer lahko vidimo celotno arhitekturo dsPIC33F mikrokrmilnika in kaj vse omogoča.




*Slika 4.3: Blokovni diagram arhitekture mikrokrmilnika [4]*

Podatke iz osebnega računalnika bomo na mikrokrmilnik prenašali preko USB UART kontrolerja, kot ga lahko vidimo na sliki 4.3. Ta adapter deluje tako, da simulira RS232 in tako pošilja podatke na priključeno napravo. Uporabljali ga bomo za nalaganje HEX datotek v mikrokrmilnik z temu namenjenim programom dsLoader ter za spremljanje podatkov med testiranjem delovanja programa in komponent sistema.

*Slika 4.4: USB UART adapter*

## 4.2 Dvojni polni mostič L293D

To vezje omogoča kontrolo smeri vrtenja DC motorjev in regulacijo hitrosti s PWM signalom. Sestavljeno je iz štirih logičnih vhodov, s katerimi določamo smer vrtenja dvema motorjema, potem sta tu še dva PWM vhoda, s katerima kontroliramo hitrost vrtenja vsakega motorja posamezno, glede na signale, ki jih dobimo iz našega mikrokrmilnika. Na vezju se nahaja tudi čip z komercialno oznako L293D [8], ki je v bistvu dvojni polni mostič ter se uporablja za pogon motorjev. Ta čip je zmožen poganjati dva DC motorja pri konstantnem toku 600 mA ali pa enega pri konstantnem toku 1200 mA. Za napajanje vezja smo uporabili 1-celično LiPo baterijo napetosti 3.7V. Pogosto imenujemo dvojni polni mostič tudi H mostič.


*Slika 4.5: Vezje dvojnega polnega mostiča L293D*

## 4.3 Senzorji, motorji in napajanje

Za pogon koles bomo uporabili dva DC motorja, ki jih najdemo v različnih verzijah. V našem primeru bomo uporabili standarden servo motor, ki se uporablja v radijsko vodenih modelih in v določenih robotskih aplikacijah. Ker imajo ti motorji v osnovi omejen kot zasuka, smo jih predelali tako, da smo odstranili varnostni zatič, posledično se lahko sedaj motor neomejeno vrti okoli svoje osi. Poleg tega je bilo potrebno odstraniti tudi analogni potenciometer, ki je skrbel za povratno informacijo, za koliko stopinj se je motor zavrtel.


*Slika 4.6: Standarden DC servo motor v ohišju*

Za zaznavanje naklona in premika bomo uporabili dva senzorja, en triosni pospeškometer in en žiroskopski senzor. Oba senzorja bosta z mikrokrmilnikom povezana preko $I^2C$ protokola.

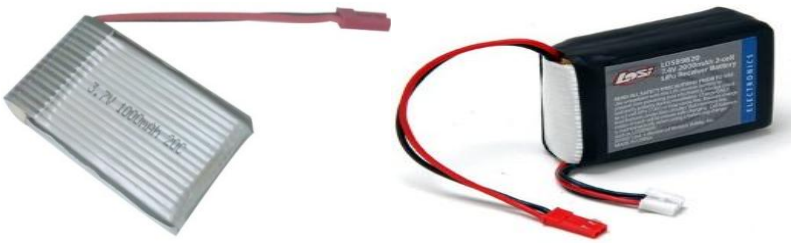
Triosni pospeškometer z oznako ADXL345 [5] je senzor, ki meri statični gravitacijski pospešek v aplikacijah, kjer se meri nagib, prav tako pa lahko merimo dinamične spremembe pospeška v primerih gibanja ali tresljajev. Ima visoko resolucijo z možnostjo merjenja pospeška do ±16 g. Sam senzor je zelo majhen, njegove mere so 3x5x1 mm, ima zelo nizko porabo energije in deluje na napetostih od 2.0 do 3.6V. Senzor zaznave pospeške v vse tri osi X, Y in Z. Tipična uporaba tega senzorja je v računalnikih in trdih diskih, kjer zaznava prosti pad, v mobilnih telefonih, v igralnih pripomočkih, v navigacijskih napravah in tudi v raznih aplikacijah v robotiki. Senzor lahko povežemo preko analognih vhodov na mikrokrmilnik, v našem primeru pa ga bom povezal preko $I^2C$ protokola, saj ta izvedba to omogoča.


*Slika 4.7: 3-osni pospeškometer ADXL345 na razvojni plošči*

Žiroskopski senzor oziroma senzor kotnih premikov ( Angular-Rate Sensor) nosi oznako XV-3500CB [6]. To je ultra miniaturni vibracijski žiroskopski senzor, katerega mere so 5x3.2x1.3 mm. Senzor je hermetično zaprt, uporablja visoko stabilni vibracijski kristal. Uporablja se v aplikacijah za stabilizacijo slike, za zaznavanje gibanja objektov in podobno. V našem primeru ga bomo uporabili za zaznavanje nagiba vozila v gibanju in v kombinaciji z pospeškometrom nam bo dajal dovolj informacij, da bomo lahko stabilizirali naš sistem. Ideja je bila, da se v inicializaciji prebere podatek iz senzorja pospeška ter se pretvori v kot oz. stopinje, za koliko je vozilo nagnjeno. V ravnovesni legi je kot vozila 0°. Ta podatek služi kot referenca za žiroskopski senzor, od te vrednosti se nato med gibanjem vozila prišteva ali odšteva trenutni kot, saj sam žiroskopski senzor pri zagonu ne ve, v kakšni poziciji se nahaja.

*Slika 4.8: Žiroskopski senzor XV-3500CB*

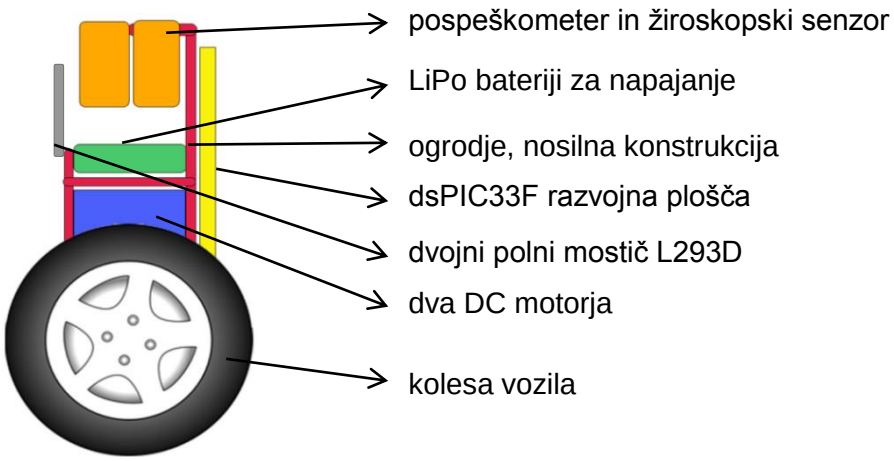
Na razvojni plošči mikrokrmilnika lahko izbiramo med napajanjem preko USB vodila, kar smo uporabljali pri programiranju, druga možnost napajanja pa je preko zunanjega vira, ki pa mora biti znotraj 7 do 9V napetosti. V ta namen smo za napajanje našega mikrokrmilnika uporabili dvo-celično LiPo (litijev polimer) baterijo, katera nam zagotavlja 7.4V enosmerne napetosti. LiPo baterije so v primerjavi z ostalimi tipi (starejše NiMh in NiCd ali alkalne baterije) lažje, imajo boljšo karakteristiko praznjenja in so okolju bolj prijazne.



*Slika 4.9: 1-celična 3.7V in 2-celična 7.4V LiPo baterija*

# 5. Izdelava vozila na dveh kolesih

Preden smo se lotili izdelave vozila naj omenimo, da nam je kot osnovna ideja, kako bi naj vozilo izgledalo služil večini znano prevozno sredstvo pod imenom Segway [9]. To je prevozno sredstvo, sestavljeno iz dveh vzporedno postavljenih koles, dveh motorjev, baterije za napajanje in elektronike, ki skrbi za nadzor vozila, da le to ne pade naprej ali nazaj.
Naš cilj je bil, da sestavimo vozilo, ki bo imelo podobno funkcijo kot Segway in načrtamo regulator, ki bo skrbel za stabilnost vozila na podlagi podatkov, ki jih bo prejel iz obeh senzorjev. Na sliki 5.1 lahko vidimo idejno sliko našega vozila ter postavitev posameznih komponent sistema.




*Slika 5.1: Zgradba vozila in razporeditev komponent*

Izhodišče za takšno namestitev komponent je bilo, da naj bo večina teže nad sredinsko osjo in čim bliže tlom, saj v tem primeru zmanjšamo mehanske učinke sil, ki bi lahko vplivale na ravnovesno lego vozila. Tako sem motorje v ohišju postavil v sredino, napajanje pa nad njih. Na osnovno konstrukcijo smo nato namestili še razvojno ploščo mikrokrmilnika in vezje dvojnega polnega mostiča. Na vrh smo namestili oba senzorja, vsakega na svojo stran vozila, pri čemer smo poskrbeli, da je sam čip, ki zaznava nagib oz. sile v primeru pospeškometra neposredno nad osjo kolesa.

Ko smo imeli elemente pritrjene na konstrukcijo, smo poskrbeli še za povezave med njimi. Oba senzorja se napajata iz izhodov, ki jih imamo na razvojni plošči mikrokrmilnika, komunikacija med njima in mikrokrmilnikom pa je izvedena z dvema žicama po $I^2C$ protokolu. Na razvojni plošči imamo tudi 4 digitalne izhode in 2 PWM izhoda, katere smo povezali z dvojnim polnim mostičem in služijo za določanje smeri ter kontrolo hitrosti obeh motorjev. Dvo-celična LiPo baterija napaja razvojno ploščo dsPIC33, medtem ko enocelična LiPo baterija skrbi za napajanje mostiča in motorjev.

Končno sestavljeno vozilo lahko vidimo na sliki 5.2. S samo izdelavo vozila nismo imeli posebnih težav, saj je vse potekalo po načrtu.

*Slika 5.2: Sestavljen prototip vozila na dveh kolesih*

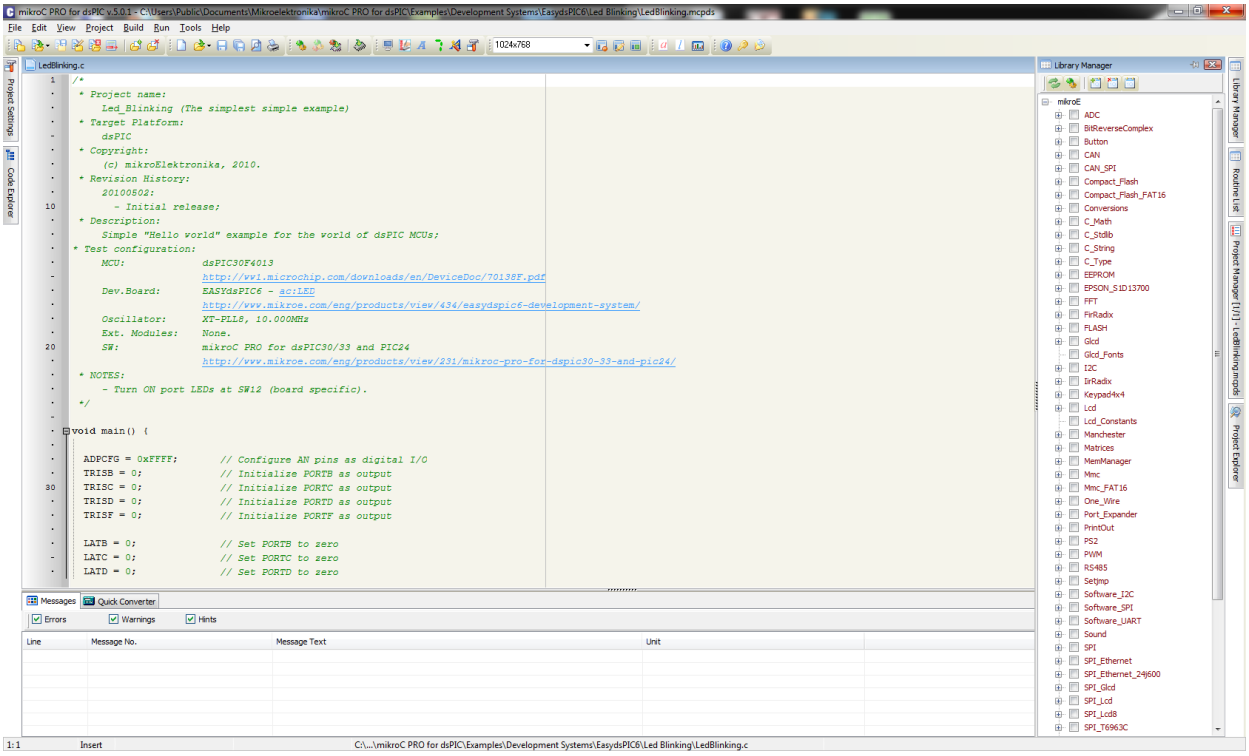
Ko smo zaključili z izgradnjo našega vozila smo se posvetili izdelavi programa v programskem jeziku C.

# 6. Programiranje

Za programiranje samega sistema oziroma za implementacijo regulacijskega algoritma v mikrokrmilnik smo uporabljali program MikroC za programiranje dsPIC mikrokrmilnikov od podjetja Mikroelektronika [3].

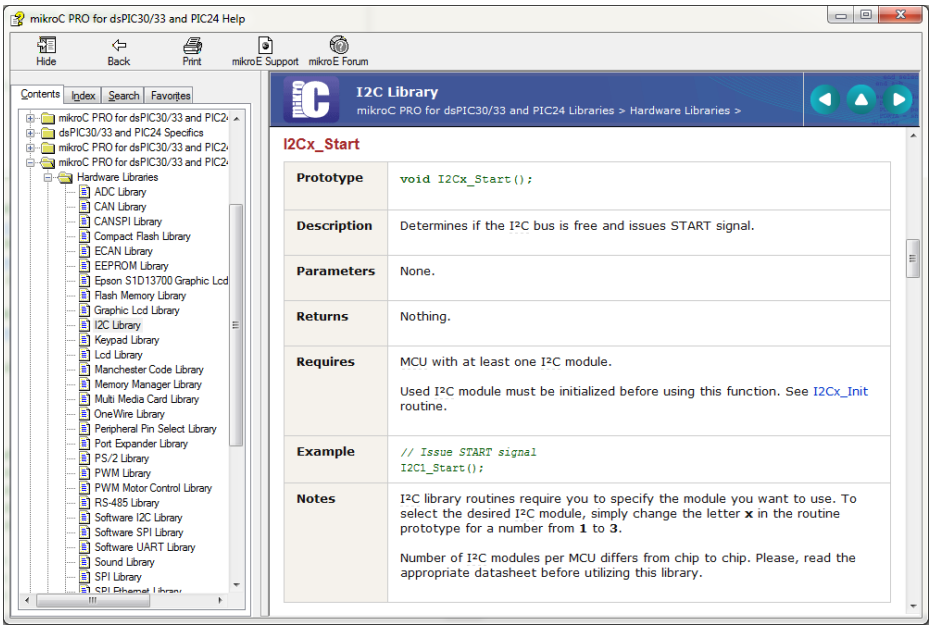
Program je zelo pregleden in nudi odlično podporo tako začetnikom, kot izkušenim programerjem, saj ima že pred pripravljene knjižnice, ki so obširno opisane in razložene s primeri. Programira se v C programskem jeziku, samo programsko okolje pa tako kot večina drugih ukaze obarva, tako da takoj vemo, če smo ukaz pravilno zapisali.

Ko prvič poženemo program se odpre primer programa, pri katerem utripa led dioda in simulira PWM signal na razvojni plošči, med izvajanjem programa.




*Slika 6.1: Program MikroC*

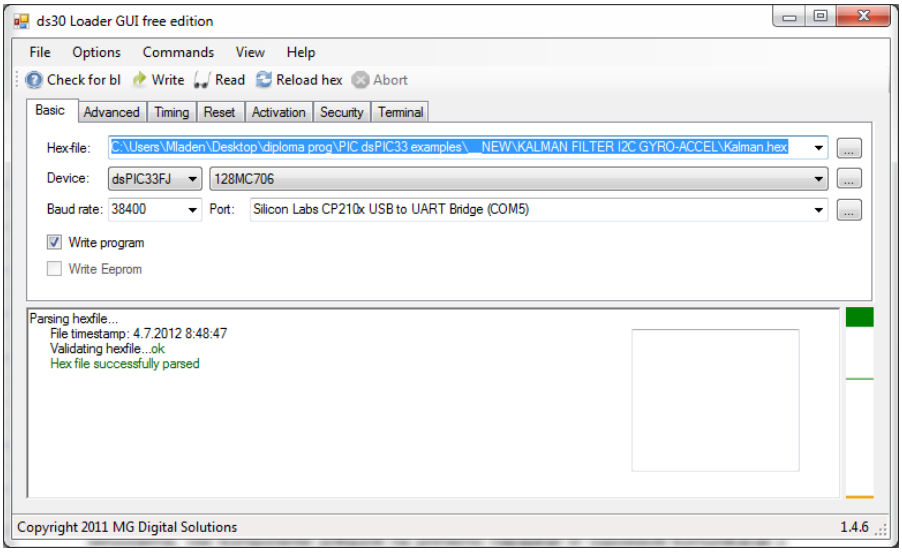
Spodnja slika prikazuje primer ukaza za vodilo $I^2C$ znotraj knjižnice v programu. Zapisano je, za kaj se uporablja ukaz, kaj potrebujemo za izvedbo in primer programske kode. Takšna vrsta pomoči znotraj samega programa je zelo dobrodošla, saj se uporabnik hitro in na praktičen način seznani z določenim ukazom in njegovo uporabo na primeru. To nam omogoča hitro implementacijo želenih funkcij.




*Slika 6.2: Primer pomoči za ukaz znotraj knjižnice v programu MikroC*

Preden pa smo se lahko lotili programiranja je bilo potrebno povezati mikrokrmilnik z obema senzorjema, vse komponente priključiti na primerno napajanje in vzpostaviti komunikacije z računalnikom, da smo lahko spremljali, kaj se dogaja znotraj mikrokrmilnika.

Mikrokrmilnik smo v začetku napajali preko USB-ja, saj je za fazo testiranja to povsem zadostovalo. Preko priključkov, ki so že integrirani na razvojni plošči smo to povezali z računalnikom preko USB vmesnika in uporabili UART protokol. Delovanje smo preverili s programom DSLoader, ki nam omogoča nalaganje *hex* datotek na mikrokrmilnik.




Slika 6.3: Program DSLoader

## 6.1 Komunikacijski protokol $I^2C$

Komunikacijski protokol $I^2C$ je bil razvit v začetku devetdesetih let s strani podjetja Philips. Namenjen je za komunikacijo med perifernimi napravami in mikrokrmilnikom.

Tako smo v našem primeru povezali mikrokrmilnik in oba senzorja preko tega vodila. Tako žiroskopski senzor kot senzor pospeška že imata na vezju integrirano podporo za komunikacijo preko $I^2C$ vodila. Osnovna zgradba je zelo enostavna saj celotna komunikacija poteka le po dveh vodnikih. Eden se uporablja za serijski pretok podatkov (SDA) drugi pa za serijsko uro (SCL). Gospodar (master), v našem primeru mikrokrmilnik, določa, kdaj se prenašajo podatki po liniji, sužnja (slave) – oba senzorja pa na zahtevo pošiljata podatke.

## 6.2 Komunikacijski protokol UART

Kratica UART (Universal asynchronous receiver/transmitter) je okrajšava za Univerzalni asinhroni sprejemnik/oddajnik. Običajno imamo v sistemu en oddajnik in sprejmnik, ki lahko delujeta simultano in neodvisno. Sam UART lahko deluje v pol-dvosmernem (half-duplex) načinu, torej samo sprejema ali samo oddaja, lahko pa deluje v popolnoma dvosmernem načinu (full-duplex), kar pomeni da istočasno sprejema in oddaja. Glavna naloga UART-a pa je, da izvaja paralelno-serijsko in serijsko-paralelno pretvorbo podatkov. V našem primeru uporabljamo UART v dveh situacijah. Prvič, ko nalagamo nov program na mikrokrmilnik in drugič, ko pošiljamo podatke iz mikrokrmilnika na računalnik. To smo uporabljali predvsem v fazi testiranja, saj smo le na ta način lahko videli, če program deluje v skladu s pričakovanji in kaj se dogaja s podatki, ki nam jih dajeta senzorja. Medtem, ko med samim delovanjem mirkokrmilnika oz. med stabilizacijo vozila UART-a ne uporabljamo.

## 6.3 Časovnik za prekinitev

Časovnik za prekinitev je osnova našega programa. V določenem časovnem intervalu sproži prekinitev izvajanja mikroprocesorja in izvede rutine, ki so sprogramirane. Ko se časovnik za prekinitev izvede, se preberejo novi podatki iz obeh senzorjev, izvedejo so konverzije podatkov v obliko, ki je primerna za uporabo v našem programu, nato sledi funkcija, v kateri se nahaja sprogramiran regulator, inicializacija smeri motorjev ter generira se PWM signal, s katerim krmilimo motorje.

## 6.4 Branje in obdelava podatkov iz senzorjev

Branje podatkov iz obeh senzorjev se izvrši znotraj prekinitvene funkcije. Takrat dobimo nove podatke iz senzorjev, ampak ti so v surovi obliki ter jih je potrebno obdelati v nam potrebno obliko. To se izvede znotraj programske sekvence, v kateri je sprogramirano, da podatki iz obeh senzorjev dajejo trenutni kot. To vrednost pa lahko nato uporabimo za zaznavanje pozicije oz. naklona našega vozila in temu primerno regulator odreagira in s PWM signalom krmili motorje, da naše vozilo ne pade.

Na začetku programa, oz. v prvem koraku se prebere podatek iz pospeškometra, saj nam ta, tudi ko je v mirovanju kaže določene sile, ki jih ustvarja sila gravitacije. S pomočjo tega podatka, ki nam pove, za koliko je vozilo nagnjeno naprej ali nazaj, nato postavimo referenčno vrednost za kot iz žiroskopskega senzorja. Ta senzor daje podatek samo takrat, ko je v gibanju. V samem programu pa imamo izvedeno to tako, da se kot prišteva ali odšteva od referenčne vrednosti začetnega kota, ko je naše vozilo v gibanju.

## 6.5 Regulator

Regulator smo implementirali znotraj programa v diferenčni obliki. Na spodnji sliki lahko vidimo samo strukturo in ukaze. Najprej določimo začetni pogrešek, ki ga dobimo tako, da od referenčnega kota odštejemo trenutni kot. Nato se izračuna izhod regulatorja po prej določeni formuli, temu koraku sledi pomik spremenljivk. Na koncu pa še določimo, da je vrednost na izhodu vedno pozitivna in zaokrožena na najbližje celo število, saj se le ta uporabi pri množenju za PWM izhod, ki mora biti vedno v obliki celega števila.

```
//PD regulator

        e_k=ref - kot_g;  //določitev trenutnega pogreška

        reg_out = (0.6305 * v_k_1) + (5.0523 * e_k) - (4.4324 * e_k_1);

        v_k_1=reg_out;  //pomik spremenljivk
        e_k_1=e_k;


        reg_abs = fabs (reg_out);    //izhod je vedno absolutna vrednost
        reg_abs = floor (reg_abs);   //zaokrožimo na najbližje celo število
```

*Slika 6.4: Struktura PD regulatorja v diferenčni obliki*

## 6.6 Generiranje PWM (pulznoširinsko moduliranega) signala

Naš mikrokrmilnik že vsebuje dva PWM izhoda, tako da PWM signalov nismo rabili posebej programirati. Glede na to da imamo na vezju 20 MHz oscilator, smo morali kvantizirati moč PWM signala tako, da programska vrednost »0« pomeni 0 V na izhodu H mostiča, medtem ko vrednost »20000« znotraj programa pomeni 3.7 V napetosti na motorjih.

PWM signal, ki gre iz mikrokrmilnika krmilimo z vrednostjo, ki jo da izhod regulatorja. To vrednost moramo primerno skalirati navzven, saj v primeru, da je premajhna motorji ne bodo imeli dovolj moči, da bi kontrolirali vozilo, če pa bo ta vrednost prevelika, pa lahko vozilo deluje preveč sunkovito, v skrajnem primeru celo pade.

Zato smo se odločili, da sprogramiram več razredov PWM signala, glede na trenutni kot oz. nagib vozila. PWM skalirni faktor navzven smo določili na osnovi poskusov, kako se vozilo obnaša pri različnih vrednostih in pod različnimi nagibi, tako smo se na podlagi teh zaključkov odločili, da bodo vrednosti med 1000 in 1300. Kriteriji za PWM signal se nanašajo na trenutni kot in so naslednji:

- $kot\_g \geq 1° \; in \; kot\_g \leq 15.5° \; ali \; kot\_g \geq -1° \; in \; kot\_g \leq -15.5°$
  V tem primeru je ojačanje izhoda regulatorja pomnoženo z konstanto 1000.

- $kot\_g \geq 15.5° \; in \; kot\_g \leq 25.5° \; ali \; kot\_g \geq -15.5° \; in \; kot\_g \leq -25.5°$
  Če se vozilo še vedno nagiba se izhod regulatorja pomnoži z 1200.

- $kot\_g \geq 25.5° \; ali \; kot\_g \leq -25.5°$
  Ko kot vozila preseže zgornja pogoja se izhod regulatorja pomnoži z 1300. Kot varnost pa je nadaljnje postavljen pogoj, če kot vozila preseže ± 55.5 stopinj se motorji izklopijo.

S tem pa še vodenje vozila ni končano, saj še nismo določili, pod katerimi pogoji se vrtijo motorji v katero smer. Ker se v našem primeru vrtita oba motorja v isti smeri, smo potrebovali za pravilno delovanje samo dva pogoja:

- $kot\_g \geq 1° \; in \; kot\_g < 55.5°$
  Če je izpolnjen zgornji pogoj, smo postavili dva vnaprej sprogramirana digitalna izhoda na mikrokrmilniku na logično 1, ter s tem aktivirali potrebne vhode na H mostiču, posledično sta se motorja vrtela naprej.

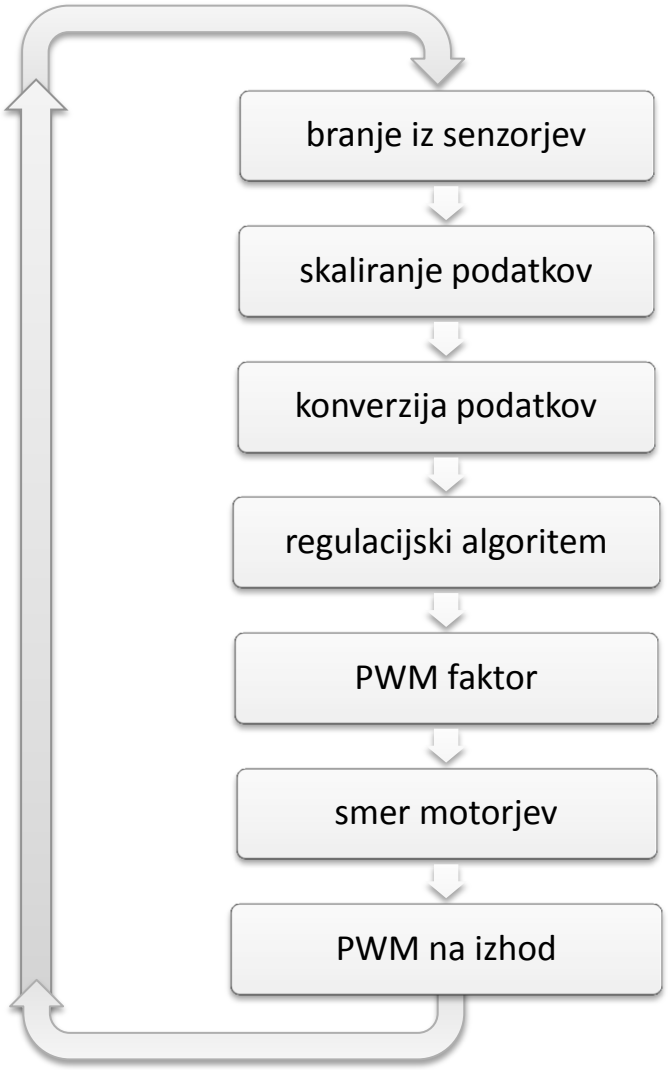
- $kot\_g \leq -1° \; in \; kot\_g > -55.5°$

  Ta pogoj je nasproten zgornjemu, če je vozilo nagnjeno nazaj, postavimo druga dva izhoda na logično 1 in tako aktiviramo vhode na H mostiču, da se motorja vrtita nazaj.

V primeru ko presežemo katero od vrednosti kota, ki so zajete znotraj opisanih pogojev, se vse vrednosti digitalni izhodov na mikrokrmilniku, za nadzor smeri vrtenja motorjev, postavijo na logično 0 in s tem preprečimo, da bi se motorji vrteli še po tem, ko vozilo pade na tla. Programsko smo to izvedli z »if - else if - else« stavkom.

## 6.7 Diagram poteka

Na spodnjem diagramu lahko vidimo potek ene iteracije programskega cikla, nato se cikel ponovi. Za to skrbi časovnik za prekinitev.




*Slika 6.5: Diagram poteka enega programskega cikla*

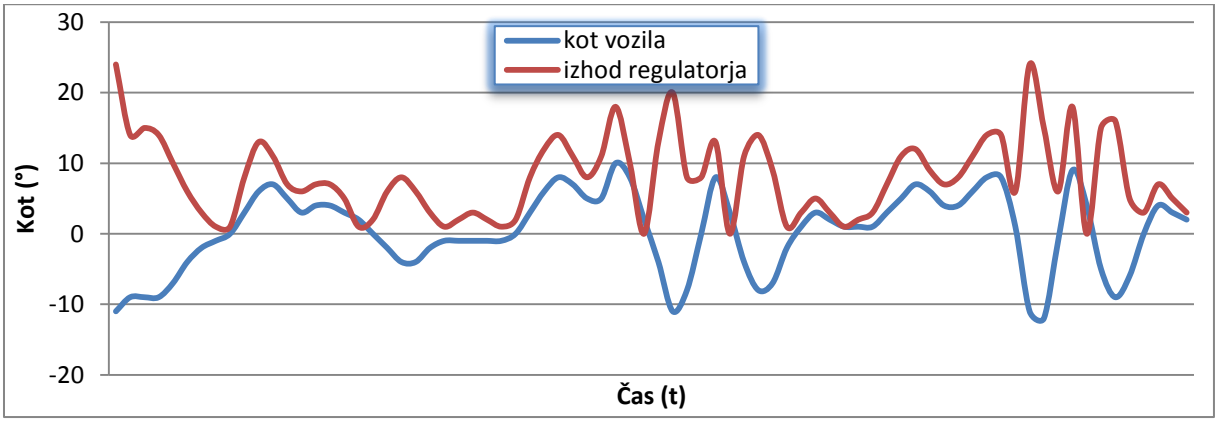
# 7. Testiranje celotnega sistema

V tej fazi smo testirali obnašanja sistema pod realnimi pogoji. S tem, ko sestavimo prototip vozila in napišemo program naše delo še ni zaključeno, saj če želimo da vse deluje tako kot smo si zamislili, moramo celotni sistem testirati na različne vplive in možne scenarije. Obstaja veliko dejavnikov, ki vplivajo na obnašanje celotnega objekta vodenja.

Sama regulacija objekta v določeni meri sledi že napisanim postopkom za sintezo regulatorja in implementacijo v sistem, nekatere stvari, kot npr. moč PWM signala, pa moramo določiti eksperimentalno. Pri tem smo izbirali različne konstante za ojačanje izhoda regulatorja in s tem definirali PWM izhode. Če je bila vrednost prevelika, je bil delovanje realnega objekta zelo sunkovito in nekontrolirano, v primeru premajhne vrednosti, pa motorji niso bili sposobni kontrolirati vozila in je le to padlo na tla, oz. se ni bilo sposobno obdržati v ravnovesju.

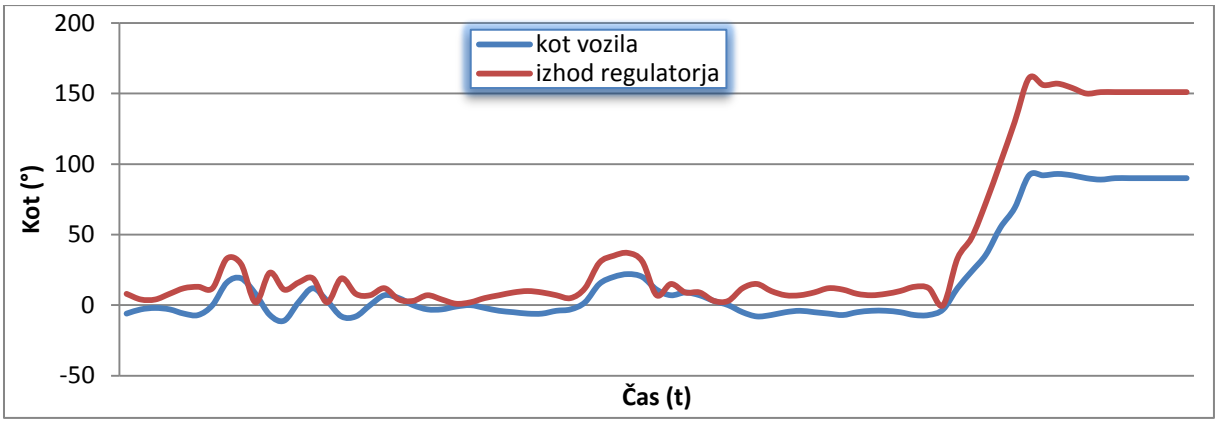
Med začetnim testiranjem prototipa in iskanju napak v programu smo za napajanje mikrokrmilnika uporabljali USB povezavo z računalnikom. Ko pa smo želeli, da vozilo postane avtonomno, smo morali preiti na napajanje preko baterije. Tukaj so se pojavile težave pri branju podatkov iz senzorjev. Napajanje smo izvedeli z eno dvo-celično baterijo za celoten sistem, ampak to se ni izkazalo za dobro odločitev, saj je prihajalo do napak pri branju iz senzorjev in s tem posledično do napačne interpretacije nagiba vozila znotraj programa. To smo nato rešili z ločenim napajanjem za pogonski del preko H mostiča in za ostalo elektroniko sistema. S tem smo odpravili težave s senzorji in sistem je nato deloval po pričakovanjih.

Na spodnji sliki lahko vidimo odvisnost izhoda regulatorja na kot oz. nagib samega vozila. Podatke smo zajemali preko UART povezave v podatkovno datoteko med samim delovanjem vozila. Pri tem vidimo, da večji kot je kot ali hitreje kot se sprememba pojavi, z večjo intenziteto regulator odreagira.




*Slika 7.1: Odvisnost izhoda regulatorja glede na kot vozila*

V primeru, da sistem ne bi bil stabilen ali pa da bi slabo izbrali parametre vodenja in faktor PWM signala, bi se to odražalo v sunkoviti vožnji vozila in tudi pogostih padcih. Do padca pride, če je nagib vozila prevelik, po naših testiranjih večji od 40°. Do tega lahko pride tudi, če se pred vozilom pojavi kakšna ovira in ne nujno zaradi napake v programu ali algoritmu.




*Slika 7.2: Odvisnost izhoda regulatorja glede na kot vozila pri padcu*

Na zgornji sliki lahko vidimo, kaj se zgodi, ko vozilo pade. Do tega lahko pride, kot že omenjeno, če se pred kolesi vozila pojavi ovira ter ima vozilo preveliko hitrost ali pa v primeru, da ko na vozilo delujemo z preveliko fizično silo in ga spravimo iz ravnovesja. Takrat vozilo povečuje izhod regulatorja, da bi s tem vzpostavilo ravnovesno lego, ampak nad določenim kotov to več ni mogoče. Takrat se v programu izvede varnostni mehanizem, ki nad kotom 55 stopinj v pozitivno ali negativno smer izklopi motorje in s tem prepreči nenadzorovano gibanje vozila.

# 8. Sklep

Cilj diplomske naloge je bila izdelava in vodenje vozila na dveh kolesih. Pri tem smo opisali ozadje problematike, potrebne sestavne dele in povezava le teh v celotni sistem. Za uspešno vodenje realnega objekta pa je bilo potrebno najprej zmodelirati objekt in s pomočjo Matlaba zasnovati primeren regulator, da bo naš sistem stabilen. Po testiranju in pogojih delovanja, ki smo jih imeli za naš sistem, smo izbral PD regulator. Še tako dober regulator ne more stabilizirati sistema, če nima za to potrebnih podatkov. V našem primeru sta za to poskrbela dva senzorja, senzor pospeška, ki je dal podatek o začetnem kotu vozila ter še žiroskopski senzor, ki je dajal podatke o nagibu vozila med gibanjem. Vse to nam je uspelo z dobrim načrtovanjem, izbiro primernih komponent in z uspešno sestavo in testiranjem celotnega sistema.
Vozilo se obnaša v skladu s pričakovanji, načrtani regulator, ki zagotavlja uspešno vodenje sistema pa je stabilen in se hitro odziva na spremembe v okolici, ki vplivajo na stabilnost vozila na dveh kolesih.

Zaradi same narave algoritma za nadzor vozila se po določenem času pojavi odstopanje vrednosti, ki nam kaže kot nagiba vozila. Zaradi integriranja se tudi napaka integrira in po določenem času moramo vozilo resetirati in ga pognati znova. To težavo bi lahko rešili s Kalmanovim filtrom [10]. Ostalih omembe vrednih težav pri izdelavi diplomske naloge ni bilo, tako da smo z delom in končnim rezultatom zadovoljni.

Za konec bi omenili še nekaj napotkov za morebitno nadaljnje delo na projektu. Smiselno bi bilo vgraditi brezžično komunikacijo med vozilom in računalnikom, saj bi lahko spremljali podatke med samim delovanjem vozila. Za vodenje vozila po prostoru pa bi potrebovali še kakšen dodaten senzor, npr. inkrementalni dajalnik na vsakem kolesu, s čimer bi vedeli, kolikšno pot opravi posamezno kolo. S tem bi nato lahko tudi kontrolirali zavoje vozila in spremljali prevoženo razdaljo. Glede na to, da je ena od težav vozila padec, ko se pojavi ovira pred kolesi bi bila ena od rešitev tudi ultrazvočni senzor razdalje, saj bi tako lahko varno ustavili vozilo ali obšli oviro ter se tako izognili morebitnemu padcu.

# 9. Viri

[1] - Svečko, R., *Diskretni regulacijski sistemi.* Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko, 2003.

[2] - http://www.mathworks.com

[3] - http://www.mikroe.com

[4] - http://www.microchip.com/pagehandler/en-us/family/16bit/architecture/dspic33f.html

[5] - http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf

[6] - http://www.epsontoyocom.co.jp/english/product/Sensor/set01/xv3500cb/index.html

[7] - http://en.wikipedia.org/wiki/I%C2%B2C

[8] - http://idmax.free.fr/Aide/Stepper/l293.pdf

[9] - http://www.segway.com/

[10] - http://en.wikipedia.org/wiki/Kalman_filter

Univerza v Mariboru
*Fakulteta za elektrotehniko, računalništvo in informatiko*

# IZJAVA O AVTORSTVU

## diplomskega dela

Spodaj podpisani/-a MLADEN TRLEP,

z vpisno številko E10177 99,

sem avtor/-ica diplomskega dela z naslovom:

STABILIZACIJA VOZILA NA DVEH KOLESIH

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  IZR. PROF. DR. RAJKO SVEČKO

  in somentorstvom (naziv, ime in priimek)

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v DKUM.

V Mariboru, dne 3.12.2012 Podpis avtorja/-ice: Trlep

Univerza v Mariboru
*Fakulteta za elektrotehniko, računalništvo in informatiko*

# IZJAVA O USTREZNOSTI DIPLOMSKEGA DELA

Podpisani mentor RAJKO SVEČKO izjavljam, da je
(ime in priimek mentorja)
študent MLADEN TRLEP izdelal diplomsko
(ime in priimek študenta-tke)

delo z naslovom: STABILIZACIJA VOZILA NA DVEH KOLESIH
(naslov diplomskega dela)

v skladu z odobreno temo diplomskega dela, Navodili o pripravi diplomskega dela in mojimi navodili.

Datum in kraj: MARIBOR, 3.12.2012

Podpis mentorja:

**UNIVERZA V MARIBORU**

FAKULTETA ZA ELEKTROTEHNIKO, RAČUNALNIŠTVO IN INFORMATIKO

**IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE ZAKLJUČNEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV**

**Ime in priimek avtorja-ice:** MLADEN TRLEP

**Vpisna številka:** E10177 99

**Študijski program:** ELEKTROTEHNIKA

**Naslov zaključnega dela:** STABILIZACIJA VOZILA NA DVEH KOLESIH

**Mentor:** IZR. PROF. DR. RAJKO ŠVETKO

**Somentor:**

**Podpisani-a** MLADEN TRLEP **izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Zaključno delo sem izdelal-a sam-a ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.**

**Tiskana verzija zaključnega dela je istovetna elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.**

**Zaključno delo zaradi zagotavljanja konkurenčne prednosti, varstva industrijske lastnine ali tajnosti podatkov naročnika:** /
**ne sme biti javno dostopno do** / **(datum odloga javne objave ne sme biti daljši kot 3 leta od zagovora dela).**

**Podpisani izjavljam, da dovoljujem objavo osebnih podatkov vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum zaključka študija, naslov zaključnega dela) na spletnih straneh in v publikacijah UM.**

**Datum in kraj:** MARIBOR, 3.12.2012

**Podpis avtorja-ice:** Mladen Trlep

**Podpis mentorja** /
**(samo v primeru, če delo ne sme biti javno dostopno):**

**Podpis odgovorne osebe naročnika in žig:** /
**(samo v primeru, če delo ne sme biti javno dostopno)**