Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Mohorčič

Mehki regulator za avtonomno vožnjo kolesa

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Uroš Lotrič

Ljubljana, 2013

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00096/2013
Datum: 09.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MIHA MOHORČIČ**

Naslov: **MEHKI REGULATOR ZA AVTONOMNO VOŽNJO KOLESA**
FUZZY CONTROLLER FOR AN AUTONOMOUS BICYCLE SYSTEM

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Na osnovi mehke logike izdelajte regulator, ki bo med vožnjo avtonomno držal kolo v stabilnem položaju. Kolo izdelajte iz gradnikov Lego Mindstorms NXT in predstavite ustrezni fizikalni model. Vožnjo kolesa z mehkim regulatorjem primerjate z vožnjo kolesa, ki uporablja proporcionalno-integralno-diferencialni regulator.

Mentor:

izr. prof. dr. Uroš Lotrič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Miha Mohorčič, z vpisno številko **63100315**, sem avtor diplomskega dela z naslovom:

Mehki regulator za avtonomno vožnjo kolesa

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Uroša Lotriča.
- 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 zbirki "Dela FRI".

V Ljubljani, dne 24. septembra 2013 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Urošu Lotriču za pomoč in vodenje pri izdelavi diplomskega dela. Zahvala gre tudi laboratoriju za adaptivne sisteme in paralelno procesiranje za posojo opreme.

Posebna zahvala gre moji družini, ki mi je tekom študija nudila oporo in spodbujala moje izobraževanje.

Kazalo

Slike

Povzetek

V diplomskem delu razviti sistem omogoča vodenje kolesa s pomočjo mehke logike. Predstavljen je koncept mehke logike in možnost uporabe. V nadaljevanju je fizikalno opredeljeno nagibanje kolesa in predstavljen Lego NXT model. S simuliranjem je poiskano področje hitrosti, kjer je model najbližje stabilni legi. V tem območju je realizirano vodenje z regulacijo PID ter vodenje s pomočjo mehke logike. Sledi primerjava implementaciji ter prednosti in slabosti vsakega načina vodenja in analiza med vožnjo pridobljenih podatkov. Omogočena je bluetooth povezava z računalnikom, preko katerega uporabnik krmili model med vožnjo. Ob uporabi je bilo ugotovljeno, da v konkretnem primeru vodenje z mehko logiko nudi minimalne prednosti pred vodenjem z regulatorjem PID. Ob implementaciji se je pokazalo tudi, da je uporaba mehke logike v sistemih z več vhodnimi parametri enostavnejša, potrebuje pa več računske moči. Ob koncu je podanih nekaj možnosti nadaljnjih izboljšav in izzivov pri nadaljnjem razvoju vodenja kolesa.

Ključne besede: primerjava, mehka logika, regulator PID, avtonomno kolo, giroskopski senzor

Abstract

In this thesis we implemented a fuzzy logic controller for stabilisation of an autonomous bicycle. We revised concept of fuzzy logic and researched possibilities of its usage. For our bicycle model, made from Lego NXT pieces, we defined its physical model and with simulation searched for velocity, at which model is most stable. We then implemented PID and fuzzy controller and compared their implementation and performance by analyzing data gathered from our test runs. We implemented a bluetooth link with computer, which can be used for steering the bicycle remotely by user. After analyzing our data we concluded that fuzzy logic in our case provides some advantages and flexibility that PID can not offer. Implementation and understanding of multiple input systems is greatly improved using fuzzy logic. At the end we have gathered some possible future additions to our system.

Keywords: comparison, fuzzy logic, PID controller, autonomous bicycle, gyroscopic sensor

Poglavje 1

Uvod

Avtomatizacija procesov je v današnjem času postala samoumevna na vsakem koraku. V zadnjih tridesetih letih je v avtomatizaciji procesov veliko veljave pridobila mehka logika, ki lahko iz nepopolnih ali zgolj opisnih podatkov poda trdne odgovore. Mehka logika se uporablja v širokem spektru področij. Z njeno pomočjo kamere ocenjujejo fokus in osvetlitev, industrija krmili naprave, široko je uporabljana v sistemih odločanja ter celo v vsakdanji beli tehniki.

Predvsem v področju avtomatizacije procesov je od sedemdesetih let naprej uveljavljena kot alternativa standardnim programerskim pristopom. Njena največja prednost je pristop k reševanju problemov, ki se približa načinu človeškega razmišljanja. Lastnost je uporaba mehkih množic, katerim elementi lahko pripadajo z neko določeno stopnjo pripadnosti. Na ta način lažje definiramo subjektivne pridevnike, na primer dolg-kratek ali visok-nizek. Kljub temu, da zaključke pridobivamo iz opisnih vhodov, je tako vodenje natančno, podaja lahko tako blage prehode med stanji kot tudi velike preskoke v odzivih.

V diplomskem delu bomo model kolesa, sestavljen iz delov Lego Mindstorms, poskušali med vožnjo uravnati v ravnovesno lego. Konstrukcijo modela bomo povzeli po delu Joepa Mutsaertsa iz univerze Delft [7], ki je uspešno balansiral podoben model z uporabo proporcionalnega krmilnika.

V delu bo predstavljen koncept mehke logike, njen razvoj in uporaba. Pojasnili bomo njene poglavitne prednosti in slabosti ter načine uporabe.

Sledi predstavitev uporabljenega modela in njegove lastnosti. Pojasnili bomo glavne sestavne dele in njihovo vlogo v modelu. Naš problem, uravnoteženje kolesa, bomo poskusili fizikalno opredeliti, ter vnaprej poiskati najboljše parametre za naše razmere.

V nadaljevanju bomo prikazali način vodenja brez in z mehko logiko ter prikazali rezultate. Primerjali bomo mehko logiko s proporcionalnim krmilnikom in opisali opažanja. Na podlagi teh rezultatov bomo tako lahko ocenili uspešnost vodenja in nadaljnje možnosti razvoja modela.

Poglavje 2

Mehka logika

V računalništvu smo navajeni trdnih odločitev. Zaradi zasnove računalniških sistemov je sposobnost računalnika omejena na odločanje med trdimi množicami. Element tako neki množici pripada ali ne. V večini primerov je tako delovanje zadostno, toda tak način dela je neustrezen v primeru subjektivno ocenjenih ali nepopolnih podatkov, težja pa je tudi implementacija linearnih odzivov na vhodne podatke. Mehka logika za razliko od tega omogoča določeno stopnjo nejasnosti, ob tem pa ustvarja jasne odzive [1].

2.1 Zgodovina

V šestdesetih letih je področje mehke logike začel razvijati L. Zadeh [2]. Svetovna znanost v tem času je takšno razmišljanje zasmehovala, saj je poenostavljanje problemov in pomanjkanje natančnosti veljalo kot zmotno.

Prvi večji zagon je ideja dobila z razvojem japonskega hitrega vlaka, kjer je uporaba mehkih množic omogočila izboljšanje kakovosti vožnje in ekonomike potovanja [3]. Na vzhodu, predvsem na Japonskem, se je kmalu razširila na mnoga uporabna področja. Tehnološka podjetja so princip začela uporabljati v industrijskem vodenju, prepoznavi pisav, napovedovanju potresov in njihovem modeliranju.

Leta 1985 sta Takagi in Sugeno predlagala modeliranje z mehko identifi-

kacijo sistema. Model razširi Mamdanijevo sklepanje in predstavlja sklepanje preko funkcije [4]. Opis problema tako poteka z uporabo pravil če-potem, ki jih poda strokovnjak, hkrati pa omogoča računsko interpretacijo in lahko adaptacijo parametrov. S tem načinom je problem lažje opisati v laiku razumljivem jeziku. Vstopni podatki so tako lahko podani v ostri (ang. crisp) ali mehki (ang. fuzzy) obliki, sistem pa jih v nadaljnji obdelavi ustrezno obdela. Pristop olajša uporabo različnih virov in vrst podatkov, ob primerni zasnovi pa z uporabo mehkih podatkov ne izgubimo natančnosti. Pristop k reševanju problemov je pridobil na popularnosti, saj se približa človeškemu pristopu k reševanju problemov. Zaradi relativno enostavnega razumevanja in pristopa je najbolj razširjena oblika implementacije mehke logike.

2.2 Prednosti mehke logike

Z mehko logiko v množice vnesemo delno pripadnost. Element tako lahko množici pripada popolnoma ali ne, obstaja pa tudi možnost delne pripadnosti. Mehka logika lahko v obzir zajame več virov podatkov in njihove vrednosti ovrednoti, z oblikovanimi pripadnostnimi funkcijami pa jih logično predstavi in pridobiva odzive na vhode.

Mehka logika stremi k približanju delovanja človeških možganov. Predstavljajmo si, da povprašamo člana populacije o mnenju, ali je desetmetrska palica dolga ali kratka. Odgovor bi bil najverjetneje dolga. Odgovor na vprašanje o palici dolžine sto metrov bi bil najverjetneje enak. S standardnim pristopom tako obe palici uvrstimo v isti razred, kljub razliki med njima. Mehka logika pa nam uspe podati odgovor, da je desetmetrska palica dolga, vendar pripada razredu z manjšo stopnjo pripadnosti kot stometrska. Tako je prehod med dolgo in kratko zabrisan in lahko odvisen tudi od drugih dejavnikov, ne vpliva pa na pripadnost razredu, zgolj na njeno stopnjo.

Največja prednost mehke logike je predstavitev informacij v obliki, ki je uporabniku razumljiva, uporabnik pa lahko v sistem vnese tudi lastne izkušnje in pričakovanja. S tem se poenostavi začetni prenos znanja, ki ga je človek že osvojil in se ga sistemu ni potrebno učiti ponovno. Pojavlja se vse več hibridnih implementacij, ki uporabljajo mehko logiko, v ozadju pa na primer nevronska mreža spremlja in prilagaja množice, ki jih le-ta uporablja.

Potrebno je poudariti tudi razliko med mehko logiko in verjetnostjo, ki se poraja ob razumevanja delovanja.

- Mehka logika nam s svojo mero pripadnosti poda mero nejasnosti, lahko tudi delne resnice neke trditve.
- Verjetnost nam poda mero verjetnosti vključenosti elementa v množico.

Razlika na prvi pogled ni očitna, a pojasni povsem različen pristop. Verjetnost se ukvarja z verjetnostjo dogodka, medtem ko se mehka logika ukvarja z dejstvi, ki pripadajo dogodku.

2.3 Ozadje delovanja

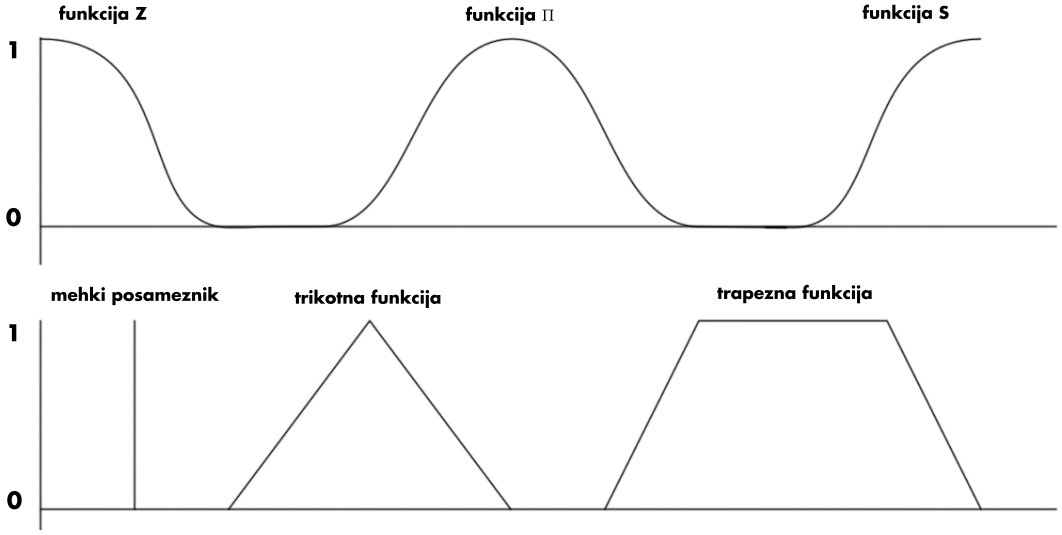
V standardnem obravnavanju podatkov v računalništvu podatke razvrščamo v dve skupini. Element v tako lahko množici A pripada ali ne:

$$\chi_A(v) = \begin{cases} 1 & \text{če } v \in A \\ 0 & \text{sicer} \end{cases} \quad . \tag{2.1}$$

V teoriji mehke logike je namesto karakteristične funkcije definirana pripadnostna funkcija (ang. membership function):

$$\mu_A(v) : U \rightarrow [0, 1] \quad . \tag{2.2}$$

Pripadnostna funkcija nam poda mero pripadnosti elementa določeni množici. Oblika pripadnostne funkcije je prosta. Lahko je zvezna, trikotna, kvadratična, Gaussova, njeno obliko in operacije med funkcijami pa določi strokovnjak na področju uporabe.



Slika 2.1: Različne pripadnostne funkcije.

Mehka logika je izredno močno orodje ob hkratni uporabi več vhodnih spremenljivk. Vhode in izhode povezujejo stavki če-potem, katerih pogoji in akcije so plod intuicije ali matematične interpretacije. Njihovo povezovanje je tako enostavno, dodajanje ali odstranjevanje pravil pa ne zahteva večjega posega v kodo.

2.3.1 Osnovne operacije

V sklopu mehke logike poznamo večino klasičnih operacij nad množicami. Unija, presek, komplement, algebraičen produkt in vsota, relacije enakosti, podmnožic, nadmnožic so orodja, ki jih lahko uporabljamo pri ocenjevanju mehkih množic. Logično gledano so klasične množice poseben primer mehkih množic, katerih pripadnost je označena zgolj z 1 ali 0. Iz tega pa sledi, da veljavne operacije v klasičnih množicah veljajo tudi v mehkih množicah:

- unija množic A ∪ B
$$\mu_{A\cup B}(v) = \max(\mu_A(v), \mu_B(v)) \quad , \quad \text{za vse } v \in U \quad ,$$

- presek množic A ∩ B
$$\mu_{A\cap B}(v) = \min(\mu_A(v), \mu_B(v)) \quad , \quad \text{za vse } v \in U \quad ,$$

- komplement ¬ A
$$\mu_{\neg A)} = 1 - \mu_A(v) \quad , \quad \text{za vse } v \in U \quad ,$$

- algebraični produkt množic
$$\mu_{AB}(v) = \mu_A(v) \cdot \mu_B(v) \quad , \quad \text{za vse } v \in U \quad ,$$

- algebraična vsota množic
$$\mu_{A+B}(v) = \mu_A(v) + \mu_B(v) \quad , \quad \text{za vse } v \in U \quad ,$$

- enakost množic A in B
$$\mu_A(v) = \mu_B(v) \quad , \quad \text{za vse } v \in U \quad .$$

Potrebno je omeniti, da za razliko od klasičnih množic ne velja vedno

$$A \cup \neg A \neq U \quad ,$$
$$A \cap \neg A \neq \{\} \quad .$$

Unija množice in njenega komplementa nam torej ne poda nujno univerzalne množice, presek pa ni nujno prazna množica.

Mehke množice prav tako obdržijo vse lastnosti asociativnosti, komutativnosti in distributivnosti:

- asociativnost
$$(a \circ b) \circ c = a \circ (b \circ c) = a \circ b \circ c \quad ,$$

- komutativnost
$$a \circ b = b \circ a \quad ,$$

- distributivnost
$$a \circ (b \bullet c) = (a \circ b) \bullet (a \circ c) \quad ,$$

kjer ∘ in • predstavljata poljubno operacijo nad množicami.

2.3.2 Mehčanje vhodnih spremenljivk

V sistem mehke logike lahko vstopa poljubno mnogo spremenljivk v različnih oblikah. V primeru, da v sistem že dobimo opisne spremenljivke in njihovo pripadnost, mehčanje vhoda ni potrebno. Ob prihodu numeričnih podatkov v sistem, pa zaradi narave nadaljnje obdelave, podatke najprej mehčamo. Mehčanje vhoda je računanje pripadnosti vhoda izhodni funkciji. Primer pripadnostnih funkcij za vhod si lahko pogledamo na sliki 2.1. V običajnih razmerah vhodu določimo tri pripadnostne funkcije - na primer nizko, normalno in visoko stanje, vendar pa s številom funkcij nismo omejeni [5]. Za različne potrebe in implementacije pa je razvitih več različnih oblik pripadnostnih funkcij.

V praktični implementaciji se največkrat uporabljajo preprostejše funkcije (trikotne, trapezne in funkcije mehkih posameznikov), saj je njihovo obdelovanje hitrejše, ob njihovem pravilnem razvrščanju pa dosežemo dovolj dobro natančnost.

V primeru, da na isto izhodno funkcijo B vpliva n vhodnih funkcij, v obzir zajamemo vse. V praksi se vrednost izhodne funkcije najpogosteje izračuna s sledečima enačbama:

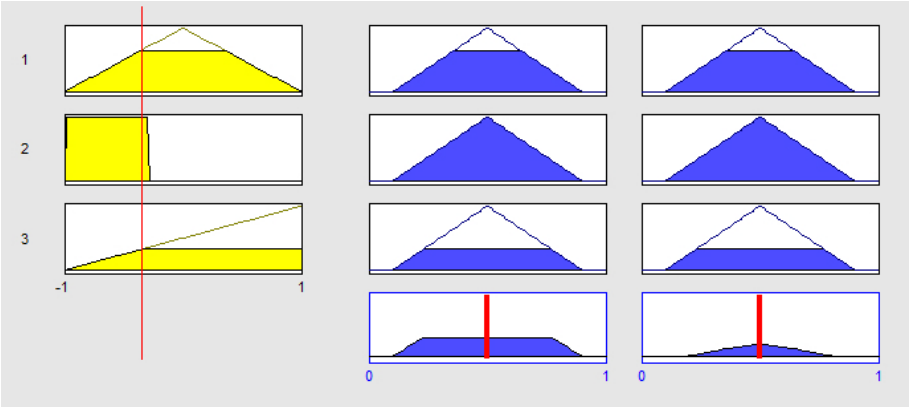
- sklepanje z minimalno pripadnostjo funkcijam

$$B' = B \cdot \min(\mu_{A_1}(x'_1), \mu_{A_2}(x'_2), ..., \mu_n(x'_n)) \quad , \tag{2.3}$$

- sklepanje s produktom funkcij

$$B' = B \cdot \mu_{A_1}(x'_1) \cdot \mu_{A_2}(x'_2) \cdot ... \cdot \mu_n(x'_n) \quad . \tag{2.4}$$

Spremenljivke $\mu_{A_1}(x_1)$, $\mu_{A_2}(x_2)$ do $\mu_{A_n}(x_n)$ predstavljajo vse vrednosti vhodnih funkcij, ki vplivajo na izhodno funkcijo B ob vrednosti x, končni rezultat sklepanja pa predstavlja B'.



Slika 2.2: Prikaz sklepanja z minimalno pripadnostjo in produktom funkcij.

V prvem stolpcu slike 2.2 so prikazane tri različne vhodne funkcije, ki vplivajo na isto izhodno funkcijo. Ob istem vhodu, ki ga prikazuje rdeča vertikalna črta v levem stolpcu, podajo različno stopnjo pripadnosti. Drugi in tretji stolpec prikazujeta pripadnost izhodne funkcije ob vhodni funkciji v isti vrsti levega stolpca. Četrta vrstica drugega in tretjega stolpca prikazuje končni rezultat sklepanj. V drugem stolpcu je prikazano sklepanje z minimalno pripadnostjo funkcijam, tretji stolpec pa prikazuje sklepanje s produktom funkcij. Po končanem sklepanju sta lika različnih oblik, sistem pa po iskanju centra gravitacije poda isti odziv. Podan je z rdečo vertikalno črto na posameznem liku.

V tem delu že lahko opazimo velike razlike v časovni zahtevnosti različnih funkcij in načinov sklepanja. Računalniški sistemi podatke veliko hitreje primerjajo, kot množijo. Ob primerni implementaciji je sklepanje z minimalno pripadnostjo izvedeno hitreje kot sklepanje s produktom funkcij. V primeru omejenih sistemskih sredstev, se funkcije, ki zahtevajo več računanja pripadnosti, pogosto nadomeščajo s preprostejšimi približki. Pogosto je predvsem nadomeščanje funkcij sinusoidnih oblik s trapezno funkcijo, ki omogoča lažje računanje pripadnosti. Trapezno funkcijo lahko predstavimo kot množico preprostih likov (kvadratov in trikotnikov), nad katerimi lažje izvajamo iskanje težišča in računanje površine.

2.3.3 Ostrenje izhoda

Rezultat mehčanja je pred izhodom iz sistema potrebno še izkazati v numerični vrednosti. To predstavlja zadnji korak pred izhodom iz sistema. Za to operacijo mehke vhodne vrednosti preslikamo na izhodno funkcijo pripadnosti. Obstaja več možnosti ostrenja, opisali pa bomo najpogostejše [6].

Najbolj pogosto uporabljena metoda je iskanje centra gravitacije (ang. Center of gravity). Ob vhodu x algoritem vrača težišče lika na osi x, ki ga opiše pripadnostna funkcija $\mu_B(x)$ in os x po celotni dolžini. Osnovna definicija algoritma je sledeča:

$$COG(x) = \frac{\int x\mu_B(x)dx}{\int \mu_B(x)dx} \quad , \tag{2.5}$$

običajno pa vrednost računamo s približkom

$$COG(x) = \frac{\sum_x x\mu_B(x)dx}{\sum_x \mu_B(x)dx} \quad . \tag{2.6}$$

Središče vrhov (ang. mean of maxima) vzame srednjo vrednost lokalnih maksimumov v izhodni funkciji B, pri čemer $core(B)$ predstavlja množico lokalnih maksimumov pripadnostne funkcije B, $|core(B)|$ pa predstavlja število vseh različnih lokalnih maksimumov:

$$MOM(x) = \frac{\sum_{x \in core(B)} x}{|core(B)|} \quad . \tag{2.7}$$

Največji in najmanjši vrh (ang. first of maxima, last of maxima) sta definirana kot:

$$FOM(B) = \min core(B) \quad , \tag{2.8}$$

$$LOM(B) = \max core(B) \quad .$$

Središčni vrh, $MiOM(B)$ (ang. middle of maxima), je definiran kot izbira srednjega izmed maksimumov v $core(B)$, če je število $|core(B)|$ liho, ali enega izmed srednjih vrhov $core(B)$, če je $|core(B)|$ sodo število. Izbira katerega (večjega ali manjšega) je prepuščena programerju.

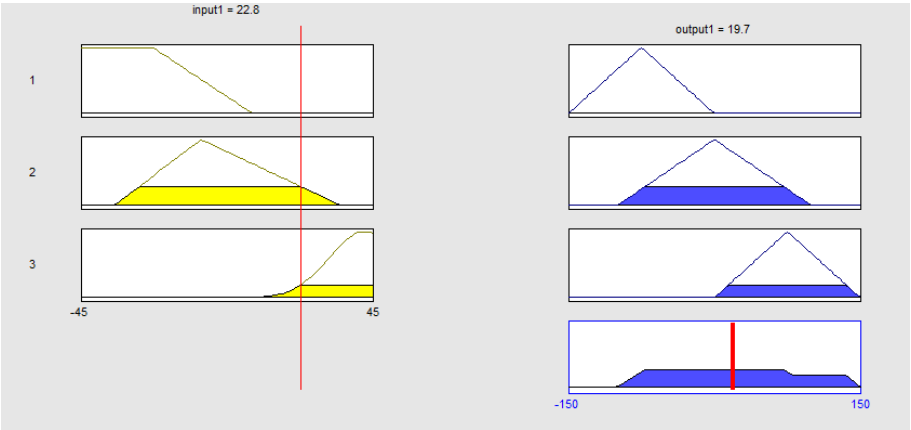
Vsak izmed načinov ima svoje prednosti in slabosti. Odvisno od razpoložljivih sistemskih sredstev, želje po diskretnem ali zveznem odzivu, programerjevega znanja in vrste pripadnostnih funkcij, se izbere metodo, ki problem rešuje v želenem časovnem okvirju z želeno natančnostjo. V praksi se tako največkrat uporablja center gravitacije, ki je uporabniku najbolj razumljiv. Proces izračuna težišče lika po osi x, ki ga ustvarijo izhodne pripadnostne funkcije. Oddaja zvezen izhod, z manipulacijo izhodnih pripadnostnih funkcij pa je odziv lahko manipulirati.

Vse zgoraj predstavljene metode ostrenja vračajo vrednost. Manj uporabljene so tudi metode ostrenja, ki vračajo funkcijo. Uporabimo jo na primer lahko v naslednjem sistemu mehke logike.

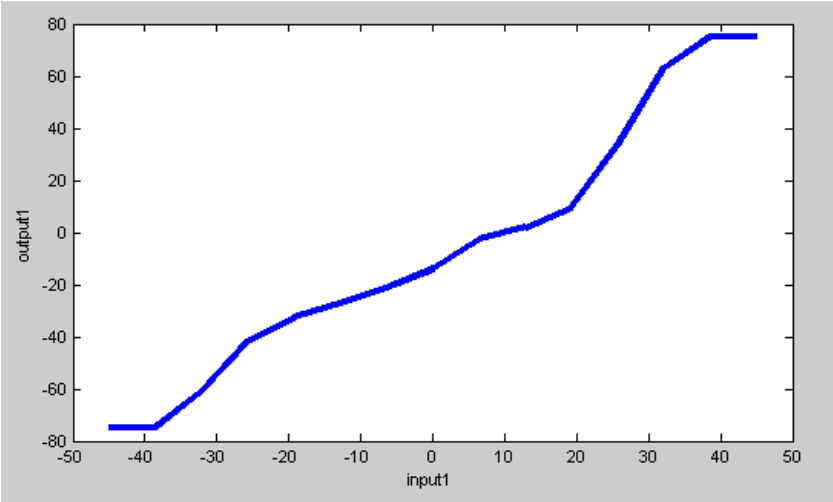
Slika 2.3 nam prikazuje možne pripadnostne vhodne in izhodne funkcije v sistem. Na levem delu slike so z rumeno barvo označene vhodne pripadnostne funkcije, z modro na desni pa izhodne pripadnostne funkcije. Rdeča črta na levi strani nam podaja vhod v sistem. Na sliki je razvidno, kako se izračuna pripadnost določeni funkciji in kakšno vrednost zajame. Na desni strani je z modro barvo označen del pripadnostnih funkcij ki ustreza izhodu. Pripadnost vhodni funkciji predstavlja višino lika, ki ga opiše izhodna funkcija. Unija pripadnostnih izhodnih funkcij nam poda lik, ki se mu izračuna težišče po osi x. Težišče, označeno z rdečo črto na desni strani, je naša izhodna vrednost sistema, izračunana po metodi centra gravitacije.

Paket nam omogoča tudi enostavno vizualizacijo odziva skozi vse možne vhodne vrednosti. Na sliki 2.4 je odziv na sistem, ki uporablja pravila s slike 2.3. Kot opazimo, smo z relativno enostavnimi funkcijami ustvarili kompleksen zvezen odziv.

V nadaljevanju lahko sistemu dodamo še en vhod. Posamezni vhodi so drug od drugega neodvisni in imajo svoje pripadnostne funkcije. Na sliki 2.5 je viden primer, kjer funkciji input1 in input2 delujeta na svojo izhodno pripadnostno funkcijo, v peti vrstici pravil pa funkciji uporabljata presek (minimum) vhodnih funkcij, ki je izražen v izhodni pripadnostni funkciji.

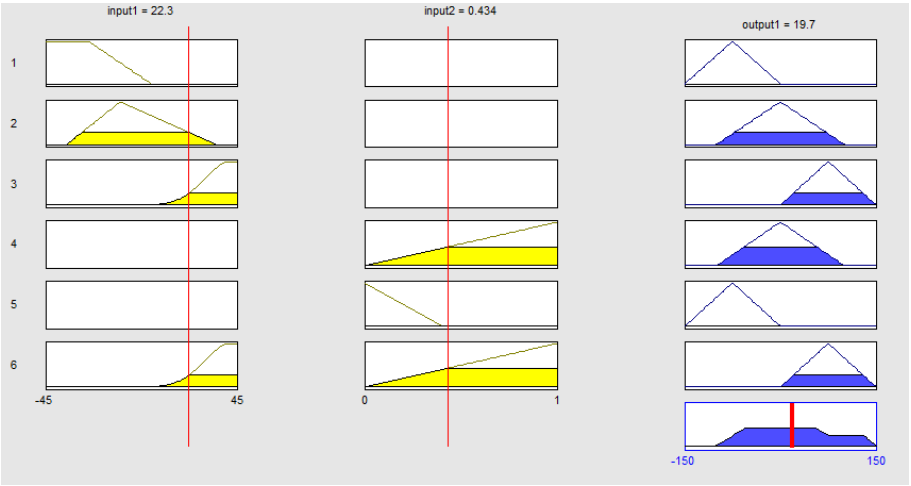


Slika 2.3: Primer mehčanja in ostrenja ene spremenljivke.

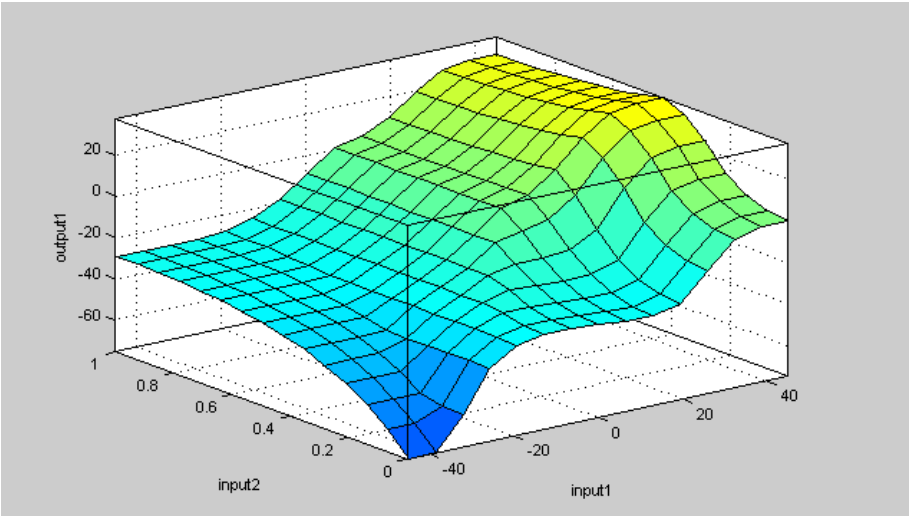


Slika 2.4: Odziv sistema z uporabo mehke logike ob eni vhodni spremenljivki.

Slika 2.6 predstavlja odziv sistema, ki uporablja pravila na sliki 2.5. Tokrat je odziv predstavljen v tridimenzionalni obliki, kjer vrednosti potujejo po vseh kombinacijah obeh spremenljivk, odziv pa je prikazan kot višina. Opazimo lahko izrazito nesimetričnost odziva, ki jo je v primerjavi z drugimi načini vodenja (na primer regulacija PID) lahko doseči.



Slika 2.5: Primer mehčanja in ostrenja dveh spremenljivk.



Slika 2.6: Odziv sistema z uporabo mehke logike ob dveh vhodnih spremenljivkah.

Poglavje 3

Gradnja NXT modela

Za praktično preizkušanje vodenja smo izdelali model kolesa. Zgledovali smo se po modelu, ki je bil uspešno voden z regulacijo P [7]. Model smo v začetku prilagodili, ker smo menili, da bomo izboljšali lastnosti, vendar smo se skozi več iteracij približali referenčnemu modelu, tako po razporeditvi težišča kot geometriji okvirja.

3.1 Predstavitev Lego NXT

Kompleti Lego Mindstorms so namenjeni predvsem mladim, ki se spoznavajo z robotiko in gradnjo kompleksnih sistemov. Kljub temu omogočajo sestavo relativno zapletenih sistemov. Uporabljen komplet Lego 8527 tako vsebuje veliko sestavnih delov, senzorjev in programirljivo kocko NXT. Z nadgradnjo programske opreme na kocki, komplet postane primeren tudi za vodenje zapletenih sistemov.

3.1.1 Programirljiva kocka NXT

Glavni del kompleta je osnovna enota, ki ima v sebi procesor, štiri vhode za senzorje in tri izhode za elektromotorje. Na sprednji strani prikazuje informacije na monokromatskem zaslonu in omogoča vodenje preko štirih

tipk, na zadnji strani pa je prostor za 6 baterij tipa AA ali akumulatorsko baterijo.

Slika 3.1: Programirljiva kocka Lego NXT.

V programirljivo kocko je vgrajen procesor Atmel ARM7 SAM7S256, ki deluje s frekvenco 49 MHz, skupaj z 8 MHz koprocesorjem Atmel AVR ATmega48, ki skrbi za vhodno izhodne enote. Čeprav so frekvence relativno nizke, je to dovolj za obdelavo podatkov iz senzorjev in ukazovanje motorjem.

Programska oprema, ki je ob prvi uporabi naložena na programirljivo kocko, je neprimerna za naše delo, saj je programiranje omejeno na povezovanje vnaprej podanih modulov. Na kocko smo naložili sistemsko programsko opremo RobotC, ki omogoča programiranje osnovne enote v jeziku C. Programska oprema RobotC, razvita na univerzi Carnegie Mellon, kodo stisne in ustvari močno optimizirano strojno kodo [8]. S tem pridobimo dostop do vseh sistemskih sredstev in možnost izvajanja daljših programov. Jezik se izvaja hitreje kot druge implementacije, predvsem pa omogoča zahtevnejše, uporabniško definirane konstrukte in operacije.

Kocko programiramo z računalnikom preko vmesnika USB ali vmesnika bluetooth. Povezava nam omogoča nadzor kocke, spreminjanje uporabljenih parametrov, največja prednost pa je realno časovni razhroščevalnik, ki teče v razvojnem okolju RobotC. Preko vmesnika bluetooth lahko pridobi-

vamo podatke o poteku programa tudi brez fizične povezave med kocko in računalnikom, kar se izkaže kot izredno uporabno orodje pri premikajočih modelih.

3.1.2 Giroskop HiTechnic

Merjenje hitrosti nagibanja bomo opravljali z giroskopom HiTechnic, ki ne spada v osnovni komplet modela, temveč je samostojna zunanja enota. Giroskop podaja zgolj trenutno kotno hitrost ki jo zaznava, uporabnik pa nato programsko računa nagib. Ker vrednosti senzorja vseskozi drsijo, je potrebno z logiko popravljati začetni odmik.

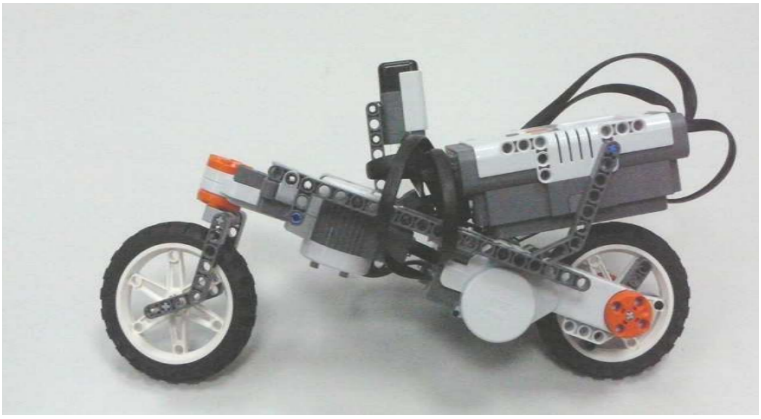
Giroskop je sposoben zaznati obrate do 360° na sekundo, branje z njega pa je mogoče do 300-krat na sekundo. Omogoča dovolj dobro ločljivost, da je primeren za sledenje odmika od začetne pozicije z minimalno napako. Giroskop deluje na zaznavanju kotne hitrosti okoli ene osi. Glede na njegovo pozicijo tako lahko sledimo orientaciji modela ali njegovemu nagibu.

Kar smo spoznali ob nadaljnjem delu je, da senzor vrača zgolj trenutno vrednost in sam ne gladi odziva. To je sicer pravilno obnašanje merilnih naprav, vendar pa lahko programerju predstavlja nov set težav. Program vrednosti bere periodično, zato hitro pridobimo napake, senzor pa je izredno občutljiv na neželene tresljaje, ki se pojavljajo ob uporabi. Rešitev predstavlja izredno gosto branje podatkov s senzorja ali programsko korekcijo (glajenje) vrednosti.

3.2 Sestavljenje modela

Glavni dizajn kolesa je povzet po [7]. Podan primer nam je služil kot referenca za začetno delo. Pri sestavi zadnjega ogrodja smo sledili referenčnemu dizajnu, zaradi manjših koles in različnih sestavnih delov pa smo dizajn sprednjega dela močno prilagodili.

Po prvi izgradnji se je pokazalo, da naše spremembe sicer omogočajo da je kolo odzivnejše, vendar pa postane ogrodje premalo togo in tako vnese v



Slika 3.2: Referenčni model [7].

vodenje preveč napak. Giroskop, pritrjen na sprednji del ogrodja, je tako zaradi torzijsko premalo togega prednjega dela prejemal vibracije motorja in dodatne premike zvijanja, kar je povzročalo trzanja v odzivu. Zaradi krajše medosne razdalje je bil model bolj odziven, vendar manj stabilen. Hitrost elektromotorjev je bila zaradi hitrega nihanja prepočasna, sunki navora motorjev pa preveliki, vodenje je bilo nestabilno.

Model smo večkrat razstavili, spreminjali geometrijo in premikali težišče. Model z večjimi kolesi, daljšo medosno razdaljo in bolj togim ogrodjem tako razvije počasnejši odziv na vodenje, vendar je bolj stabilen in manj občutljiv na zvijanje okvirja.

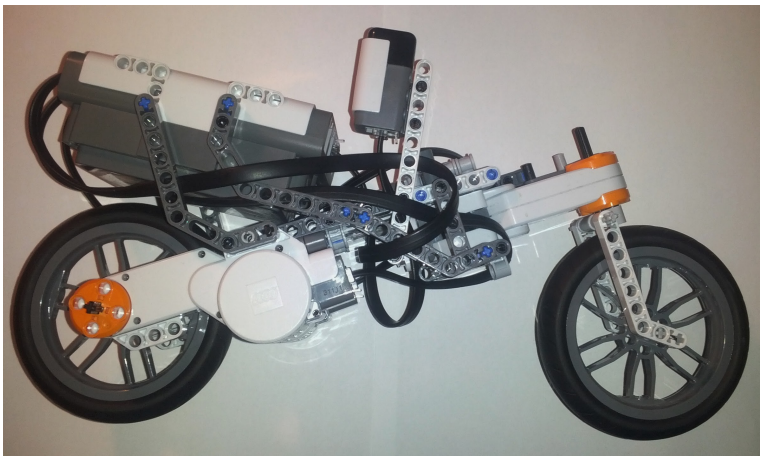
Model uporablja dva motorja za pogonski del, ki skrbita za premikanje modela. Nameščena sta vzporedno na vsaki strani pogonskega kolesa. Z razporeditvijo težjih komponent enakomerno ob osi v smeri gibanja, smo zagotovili, da je kolo čim bolj stabilno in nam ni potrebno programsko reševati problema neuravnovešenosti.

Nad zadnjim kolesom, pomaknjeno proti koncu modela, smo namestili najtežjo komponento - programirljivo kocko. Pozicioniranje le-te nam zviša težišče, vendar je zaradi prostorskih omejitev namestitev drugje praktično nemogoča. Zaradi zvišanja težišča model tako postane manj stabilen, hkrati pa nam zaradi večje vrtilne količine model bolj predvidljivo, manj sunkovito

spreminja naklon.

Na sprednjem delu je nameščen krmilni motor, na katerega je direktno vezano krmilo. Motor ima v pogonskem delu zračnost, ki omogoča, da se krmilo prosto odmika od svoje podane lege. Kljub temu se je v nadaljevanju izkazalo, da zračnost prenosa nima večjega vpliva na samo vodenje modela.

Težišče modela je postavljeno 8 cm pred zadnjo osjo, na višini 7 cm, pri naklonu 45° pa model na ravni podlagi nasede z ohišjem motorja.



Slika 3.3: Končni dizajn našega modela.

Poglavje 4

Fizikalne zakonitosti modela kolesa

Preverili smo osnovne zakonitosti obnašanja kolesa ob vožnji. Ob normalni vožnji s kolesom je največji faktor v obnašanju kolesa kolesar, ki z gibanjem svojega telesa in premiki krmila spreminja težišče kolesa ali smer potovanja. Ob opazovanju ugotovimo, da kolo tudi ob odstranitvi kolesarja ob potisku samo vzdržuje ravnotežno lego in se ne prevrne, dokler obdrži zadostno hitrost. V zadnjih raziskavah je ovržena teorija, da kolo ostane pokonci zgolj zaradi giroskopskega učinka, čeprav le-ta pripomore k večji stabilnosti [9].

Po mnenju raziskovalcev ima največ vpliva pri taki vožnji razporeditev mase na kolesu. Zaradi lažjega sprednjega dela kolo s prvim koncem prej začne padati v zavoj, s tem kolesu premakne težišče in obrne krmilo. Posledica je sprememba smeri, hkrati pa povrnitev kolesa v bolj stabilno lego. V našem primeru kolo ne začne zavijati samo, saj je trenje v ohišju motorja preveliko in samo sprememba težišča ni dovolj za premik krmila.

4.1 Osnovne fizikalne enačbe

[10] lahko opišemo z naslednjo enačbo:

$$\ddot{\chi} + \frac{bv}{ha}\dot{\psi} + \frac{v^2}{ha}\psi - \frac{g}{h}\chi = 0 \quad , \tag{4.1}$$

kjer je ψ zasuk krmila, v hitrost kolesa, a medosna razdalja kolesa, χ nagib kolesa, h višina težišča, b horizontalna razdalja od težišča do zanje osi, g gravitacijski pospešek.

V primeru, da predpostavimo, da je zasuk krmila ψ vedno za faktor k različen od nagiba, se enačba poenostavi:

$$\psi = k \cdot \chi \quad ,$$

$$\ddot{\chi} + \frac{bv}{ha}k\dot{\chi} + [\frac{v^2 k}{ha} - \frac{g}{h}]\chi = 0 \quad . \tag{4.2}$$

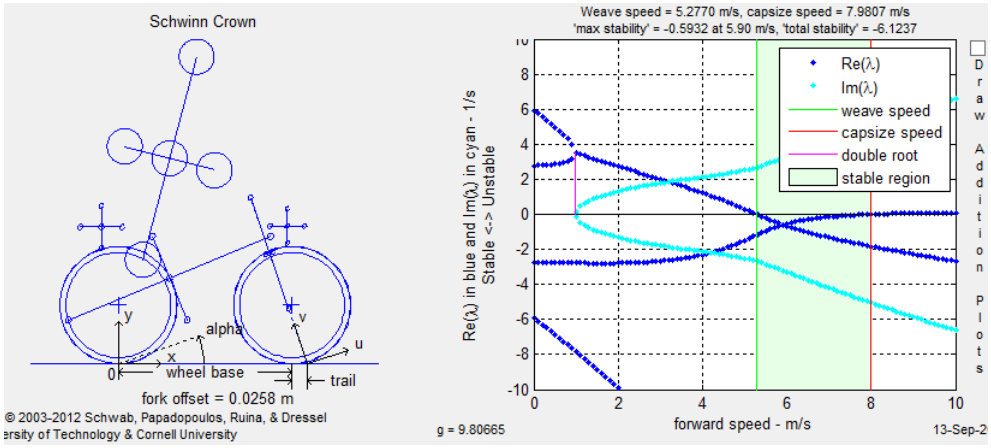
Enačba predpostavlja, da je hitrost kolesa enakomerna. Z njo se lahko izračuna vpliv nagiba na zasuk krmila ali obratno, predstavi pa tudi vpliv razporeditve mase na modelu. Enačba opisuje nagibanje izmišljenega kolesa. Zaradi njene splošnosti ne pridobimo novih informacij za vodenje našega modela. Za natančne izračune je potrebno upoštevati geometrijo kolesa in vplive sil, ki nastajajo ob vožnji.

[10, 11, 13] so s simulacijami pokazali, da se kolo ob enakomernem gibanju lahko vodi s sukanjem krmila v smer padca. Simuliran je bil tudi mehki krmilnik, ki je za vhod uporabljal nagib kolesa in zasuk krmila. V simulacijah so raziskovalci potrdili možnost uravnoteženja modela s pomočjo mehke logike.

S pomočjo enačb smo dobili potrditev o možnosti vodenja kolesa z mehko logiko, vendar so enačbe preveč splošne za trdne zaključke glede parametrov vodenja ali simulacije našega modela. Razumevanje tega dela enačb nam pripomore pri fizični gradnji modela. Z razumevanjem fizikalnih lastnosti in ob opazovanju obnašanja našega modela se lažje odločimo za fizične spremembe na modelu, razporeditvi teže in dimenzijah.

4.2 Model JBike6

Za realnejšo predstavitev fizikalnih lastnosti smo uporabili orodje JBike6. Orodje pri podanih parametrih izračuna hitrosti, pri katerih kolo samostojno vzdržuje ravnovesno lego [15]. Uporabljen fizikalni model je veljaven zgolj, če predpostavljamo, da premikajoči deli kolesa med seboj ne ustvarjajo trenja. Zaključki nam tako podajo močno idealizirano simulacijo gibanja modela. Uporabljene enačbe so nadaljevanje enačbe (4.1), z dodatnim upoštevanjem parametrov, na primer obsega koles, giroskopskega učinka koles, razporeditve teže in geometrije.



Slika 4.1: Simulacija realnega kolesa [15].

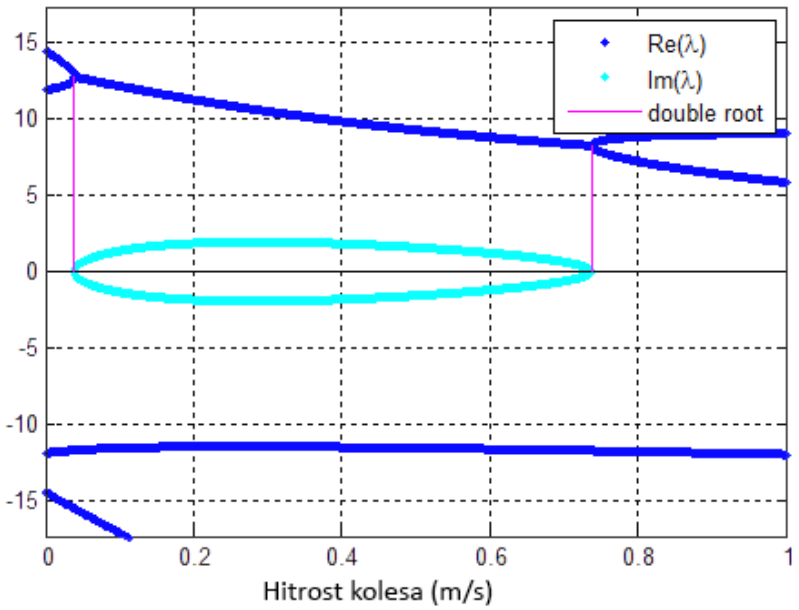
Na sliki 4.1 je prikaz simulacije realnega kolesa in njegovega nihanja z orodjem JBike6. Iz grafa lahko razberemo hitrosti, pri katerih je kolo v stabilni legi (zeleno področje), in skrajne lege stabilnega področja (rdeča in zelena vertikalna črta). V tem področju hitrosti kolo zaniha okoli ravnovesne lege vsaj enkrat, preden izgubi dovolj hitrosti in pade na stran.

Program nam grafično prikaže rezultate in območja, v katerih je vožnja kolesa stabilna. S temno modro barvo je predstavljen realni del lastnih vrednosti računskih matrik. Ta del nam prikaže odziv kolesa na spremembo kota krmila. Bližje kot je nič, hitreje se bo kolo ustalilo v ravnovesni legi.

Svetlo modra črta nam prikazuje hitrost nihanja okoli ravnovesne lege. V primeru, da vrednosti ni, je kolo nestabilno v taki meri, da do nihanja ne pride, oziroma krmilo s svojim zasukom v padec pospešuje.

Naš model kolesa smo premerili in stehtali komponente, nato pa parametre prenesli v program. Slika 4.2 prikazuje rezultat simulacije jBike6 s sledečimi parametri:

$a = 0{,}22$ m,	medosna razdalja,
$d = 0{,}094$ m,	premer platišča z plaščem,
$b = 0{,}08$ m,	razdalja težišča od zadnje osi,
$h = 0{,}07$ m,	višina težišča,
$\lambda = 70°$,	naklon sprednjih vilic,
$m = 0{,}7$ kg,	masa celotnega kolesa,
$m_p = 0{,}035$ kg,	masa platišča in plašča.



Slika 4.2: Rezultati simulacije jBike6.

Razvidno iz rezultatov simulacije na sliki 4.2 je, da kolo ne poseduje ravnovesne lege, najbližje stabilnosti pa smo v območju okoli 0,7 m/s. Naš model ob polni moči motorjev na pogonskem kolesu razvije hitrost vrtenja okoli 120 obratov/min. Ob polmeru kolesa 4,7 cm, je končna hitrost modela okoli 0,6 m/s. Model bomo tako poganjali ob maksimalni moči motorjev. Ob idealnih pogojih bi se s tem najbolj približali hitrosti, kjer bomo model najlaže uravnali.

Poglavje 5

Vodenje modela

Sestavljenemu modelu smo razvili programsko kodo za vodenje in interakcijo. Program je napisan v jeziku C z dopolnitvami, ki jih ponuja razvojno okolje RobotC. Okolje poenostavi interakcijo z vhodi programirljive kocke in upravljanje z motorji, olajša nam inicializacijo senzorjev in komunikacijo preko vmesnika bluetooth. RobotC nam omogoči prejemanje vrednosti senzorjev in nastavljanje moči motorjem ter spremljanje njihove pozicije z enostavnimi ukazi.

5.1 Osnovno ogrodje programa

Glavnino našega dela predstavlja vodenje robota, v ozadju pa je potrebno nekaj režije, da model uspešno prejema in obdeluje podatke. V ozadju model neprestano osvežuje podatke senzorjev, skrbi za premikanje motorjev, računa nagib, prikazuje podatke na LCD zaslonu in komunicira z računalnikom.

Program ob zagonu skalibrira senzor in nastavi začetne vrednosti. Kalibracija modela poteka v metodi calibrateRobot, po kalibraciji pa program linerano veča moč pogonskih motorjev do končne hitrosti v programski niti linearStart. S takim načinom začetka vožnje preprečimo sunkovito gibanje ob pospeševanju.

V času vožnje se program odvija v neskončni zanki, ki se izvaja v me-

todi main in skrbi za potek programa. Zanka ob vsakem obhodu preverja sporočila bluetooth. V primeru, da uporabnik želi spremembo smeri, vrednost zabeleži v spremenljivko userIn. V nadaljevanju izračuna čas, ki je potekel od prejšnjega obhoda zanke in s pomočjo metode GetLean pridobi podatke senzorja. Pridobljene podatke preda metodi PIDSteer ali fuzzyCalculateSteer, ki vrneta željen odziv. Ob koncu zanka odziv poda na izhod motorja in počaka 10 ms pred ponovnim obhodom.

5.1.1 Metoda GetLean

Metoda je zadolžena za periodično preverjanje hitrosti senzorja in računanje nagiba. Klicana je ob vsakem obhodu glavne zanke. Je del kode, ki bere in obdeluje podatke giroskopa. V razvojnem okolju lahko najdemo predlog uporabe HiTechnic giroskopa. Predlog deluje pravilno, vendar v obzir ne vzame drsenja vrednosti, ki jih podaja giroskop. Le-ta tudi ob mirovanju vrača minimalne spremembe v rotaciji, ki jih program ne gladi. Naša rešitev je sestavljena iz branja vrednosti senzorja, glajenja odmika, računanja hitrosti nagibanja in nagiba modela. Metoda je opisana s kodo na sliki 5.1. V kodi uporabljene spremenljivke predstavljajo:

`gyroRaw` - prebrana vrednost senzorja,
`gyroOffset` - odmik do srednje vrednosti senzorja,
`EMAOFFSET` - faktor glajenja odmika,
`gyroSpeed` - hitrost premika senzorja,
`lean` - integral hitrosti po času (nagib),
`tInterval` - pretečen čas od zadnjega obhoda zanke.

```
void GetLean()//calculates gyro speed and lean
{
  gyroRaw = SensorValue(GYRO); //get raw data from sensor
  gyroOffset = EMAOFFSET * gyroRaw + (1-EMAOFFSET) * gyroOffset; //fix offset
  gyroSpeed = gyroRaw - gyroOffset; //get speed
  lean += gyroSpeed*tInterval; //get lean
}
```

Slika 5.1: Branje vrednosti giroskopa.

Koda prebere vrednost senzorja in jo s koeficientom `EMAOFFSET` doda spremenljivki `gyroOffset`. Z uporabo preprostega nizkoprepustnega filtra povprečimo vrednost, pri kateri senzor predvidoma stoji v ravnovesni legi. Predstavljena rešitev rešuje problem drsenja senzorja, ki tudi ob mirovanju ne podaja ničelne vrednosti. Rešitev nam tako omogoča sorazmerno majhno napako pri majhnem nihanju okoli ravnovesne lege skozi dolgo časovno obdobje, v primeru, da je model dolgo v nagibu v isto smer, pa se napaka povečuje. Rešitev je za naš problem zadovoljiva, saj predvidevamo, da kolo vseskozi niha okoli ravnovesne lege.

V primeru, da bi senzor uporabljali na počasi rotirajočem modelu, ki ne niha okoli ene lege, bi bila napaka prevelika. Naš senzor brez glajenja odmika je v mirovanju ob času dvajsetih sekund zaznaval do 20° odmika. V primeru agresivnega glajenja odmika (velika vrednost `EMAOFFSET`), bi vrednost senzorja prehitro gladili, aplikacija premika ne bi zaznala. Ob uporabi premajhne vrednosti, drsenja ne kompenziramo dovolj, napaka ostaja. Ker drsenje ni konstantno, ampak je odvisno od okolice, tako ne moremo drsenja senzorja izmeriti in uporabljati konstante vrednosti, ki bi problem rešila. Rešitev je uporaba bolj natančnega senzorja, bolj pogosto branje vrednosti senzorjev ali uporaba več senzorjev in povprečenje njihovih napak.

S tem kratkim odsekom kode smo pridobili dva vhoda v naš sistem vodenja. Pri uporabi mehke logike ju lahko uporabimo kot možne vhodne spremenljivke. Pri regulaciji PID nam proporcionalni del predstavlja vrednost `gyroSpeed`, njegov integral, nagib, pa predstavlja spremenljivka `lean`. V primeru, da bi želeli imeti vse vrednosti, ki bi jih potrebovali za regulacijo PID izračunane vnaprej, bi tudi komponento D (odvod po času - pospešek) izračunali v tem delu kode.

5.1.2 Metoda CalibrateRobot

Ta del kode se izvede ob zagonu glavnega programa. Poskrbi za pravilno kalibracijo senzorja in motorjev.

Uporabnik pred zagonom robota naravna vodilno kolo, program pa v

tem delu kode nastavi trenutni odmik krmila na nič. Vrednost odmika je lahko uporabljena kot vhod v sistem vodenja, v našem primeru, pa nam najbolj pripomore pri analiziranju voženj. Ogrodje RobotC nam omogoča pridobivanje in nastavljanje vrednosti odmika s klicem nMotorEncoder[X], ki nam vrne ali vanj zapiše želeno vrednost odmika za motor X.

Vsi motorji se v tem delu programa ustavijo, vendar ostanejo aktivni - zavirajo kakršnekoli premike iz podane lege. Kolo ob kalibraciji v začetni legi drži uporabnik. Z blokiranjem motorjev minimiziramo premike kolesa v smeri vožnje v tem času, s tem pa delno pripomoremo k bolj natančni kalibraciji senzorja. Odziv giroskopskega senzorja ni neodvisen od prejete napetosti, saj se ob zmanjšanju napetosti zmanjša tudi njegova sredinska vrednost. Ob mirovanju motorji ne nižajo napetosti, ki jo kocka lahko dovaja. Z aktiviranjem zavore tako povzročimo, da se napetost zniža bliže vrednosti, ki bo uporabljena med vožnjo modela in s tem bolj točno kalibracijo.

V nadaljevanju program v času pol sekunde zajame 100 vrednosti giroskopa. Če je razlika med minimalno in maksimalno vrednostjo prevelika, program meritve zavrže in zajame 100 novih vrednosti. S tem preprečimo, da bi se model zagnal, medtem ko se uporabniku zamaje iz ravnovesne lege. Ob zadovoljivih meritvah se izračuna povprečna vrednost meritev, ki predstavlja začetno vrednost odmika giroskopa.

5.1.3 Metoda Main

Metoda main se požene ob zagonu programa. Njena naloga je skrb za potek programa in upravljanje s podatki.

Metoda ob zagonu programa ponastavi uporabljene vrednosti na začetne vrednosti in požene novo nit, ki skrbi za prikazovanje podatkov na zaslonu med uporabo modela. Metoda nato dovoli uporabniku spremembo želene končne moči motorjev. Ko uporabnik potrdi moč motorjev, metoda začne kalibracijo modela z metodo CalibrateRobot. Po končani kalibraciji požene pogonske motorje do želene hitrosti in preostanek vožnje skrbi za vodenje modela.

Vodenje poteka v neskončni zanki. Pogoj za prekinitev zanke je uporabniška prekinitev ali zaznan padec modela. Model zazna padec kot nagib konstrukcije, večji od 45°. V zanki se s pomočjo metode GetLean izračuna nagib modela in hitrost nagibanja. Pridobljene podatke se preda v nadaljnjo obdelavo želeni regulacijski metodi, njen odziv pa se aplicira na krmilni motor.

Ob prvotnih implementacijah smo ugotovili, da je odziv motorja prepočasen. V ta namen program pred zaključkom zanke počaka 10 ms, kar omogoči, da motor podano moč aplicira in se premika do naslednjega obhoda zanke.

5.2 Vodenje z regulacijo PID

Začetna implementacija rešitve je izvedena s pomočjo regulacije PID. Kratica PID predstavlja proporcionalno integrirno diferencirno vodenje sistema. Ta način vodenja je široko uporabljen v industriji, njegova izvedba je enostavna in omogoča široko prilagodljivost. Vodenje sprejme tri različne opise dogajanja v okolju. V našem primeru proporcionalni del predstavlja hitrost nagibanja, integrirani del nagib modela, diferencirni del pa predstavlja pospešek pri nagibanju. Z uporabo proporcionalnega dela v večini primerov pridobimo dober odziv sistema, z uporabo dodatnih komponent pa lahko odziv še poudarimo ali gladimo do želene vrednosti.

Splošna enačba za izhod u regulatorja PID je sledeča:

$$u(t) = K_p e(t) + K_i \int_0^t e(z)dz + K_d \frac{d}{dt} e(t) \quad , \tag{5.1}$$

kjer K_p predstavlja faktor množenja proporcionalnega dela, K_i faktor množenja integriranega dela, K_d faktor množenja diferencirnega dela, e napako, t čas vodenja in z spremenljivko, uporabljeno ob integraciji. Napaka e je izračunana kot:

$$e = (\text{želena vrednost vhoda}) - (\text{izmerjena vrednost vhoda}) \quad . \tag{5.2}$$

Regulacija na podlagi napake, ki se dogaja med vodenjem, popravlja odziv. Namen vodenja je napako ohranjati na minimalni vrednosti. S korekcijo faktorjev odziv regulacije nadziramo in prilagajamo našim potrebam.

Po vzoru referenčnega kolesa smo vodenje izvedli z regulacijo P, nato poskušali še z regulacijama PI in PID. Referenčno kolo je bilo uravnano samo z regulacijo P, kjer je bila moč motorja odvisna zgolj od hitrosti nagiba. Odločili smo se, da bomo tudi sami najprej kolo vodili na tak način, kasneje pa poskušali izboljšati delovanje.

Prosto dostopna koda [7] referenčnega modela, nam omogoča vpogled v delovanje sistema. Koda je zgrajena v okolju Matlab z orodjem Simulink. Shematiko vodenja modela smo preučili in sprogramirali sledečo kodo PID vodenja, kjer `gSpeed` predstavlja podano hitrost senzorja, `gLean` nagib in `gAcceleration` pospešek. Spremenljivka `sSpeed` predstavlja odziv sistema. Izsek kode je viden na sliki 5.2.

```
sSpeed= (
(gSpeed*(3.14/180)*factorP)+  //P
(gLean*(3.14/180)*factorI)+   //I
(gAcceleration*(3.14/180)*factorD)  //D
)
*100 // vrednost med -100 in 100
/((nAvgBatteryLevel*0.001098)-0.625) //kompenzacija baterije
*1.1; //kompenzacija trenja
```

Slika 5.2: Implementacija regulacije PID.

Vodenje tako vzame proporcionalni del (hitrost nagibanja), integrirani del (nagib) in diferencirni del (pospešek nagiba) v radianih, iz njih pa s pomočjo faktorjev pridobi odziv. Spremenljivka `factorP` predstavlja K_p, `factorI` K_i in `factorD` K_d. Pretvorba vhodov v radiane je izvedena hkrati z množenjem faktorjev, z množenjem z vrednostjo `(3.14/180)`. Algoritem kompenzira napolnjenost baterije (vrednost `nAvgBatteryLevel`) in izgube pri trenju motorja po eksperimentalno [7] pridobljenih enačbah. Vrednost `nAvgBatteryLevel` je med tipično uporabo med 8 in 7,2 voltov. Model hkrati

uporablja tri elektromotorje, zato je vodenje modela pod sedmimi volti nemogoče, saj akumulator ne zadosti potrebam po energiji. Faktor 1,1 je uporabljen za kompenziranje desetih odstotkov moči, ki jo motor uporabi za premagovanje trenja prenosa. Vodenje pred izhodom vrednost pomnoži s faktorjem 100, da pridobi odziv na območju od -100 do 100, ki ga uporablja krmilni motor.

Za uspešno stabilizacijo smo uporabili vrednosti `factorP` = 8, `factorI` = 0, `factorD` = 0. S tem načinom vodenja in faktorji nam je uspelo stabilizirati kolo pri različnih hitrostih vožnje. Vodenje poteka zgolj preko proporcionalnega dela, kotne hitrosti nagibanja. Naklon in pospešek se na vhod odzivata prepočasi in odziv preveč dušita. Z njima je vodenje zgladilo odziv na motnje, vendar se ob hitrejših spremembah ne odzivata dovolj hitro. Model ob uporabi manjših faktorjev nagiba ali pospeška ne pridobi na stabilnosti ali dušenju gibanja, ob uporabi večjih pa je odziv prepočasen, model pa se prevrne na stran.

Ob pregledu algoritma smo ugotovili, da vsebuje veliko konstantnih števil, uporablja pa zgolj proporcionalni del vodenja. Ker smo želeli algoritem optimizirati za hitrejše delovanje, smo izračunali povprečni odziv pri mejnih napetostih. Zaradi neuporabe, smo iz enačbe odstranili faktorja nagiba in pospeška. Vodenje smo nato poenostavili z združevanjem konstant. Izračunana faktorja algoritma sta tako pri napetostih 7,2 V - 2,11 in 8 V - 1,89. Če model vodimo ob povprečni napolnjenosti baterije, je uporabljena konstanta 2. Model smo nato poskušali voditi z uporabo enostavnejše enačbe:

```
sSpeed=2*gSpeed;
```

Slika 5.3: Koda regulacije P.

Pridobljena koda predstavlja regulator P, katerega odziv `sSpeed` je za faktor dve večji od kotne hitrosti `gSpeed`. Odziv brez kompenzacije izpraznjenosti baterije ni vidno slabši, saj je model kljub temu uspešno voden v istem

območju hitrosti in nagiba. Poenostavljeno enačbo vodenja bomo tako v nadaljevanju uporabljali kot referenco za implementacijo mehke logike in primerjavo odzivov sistemov.

5.3 Vodenje z mehko logiko

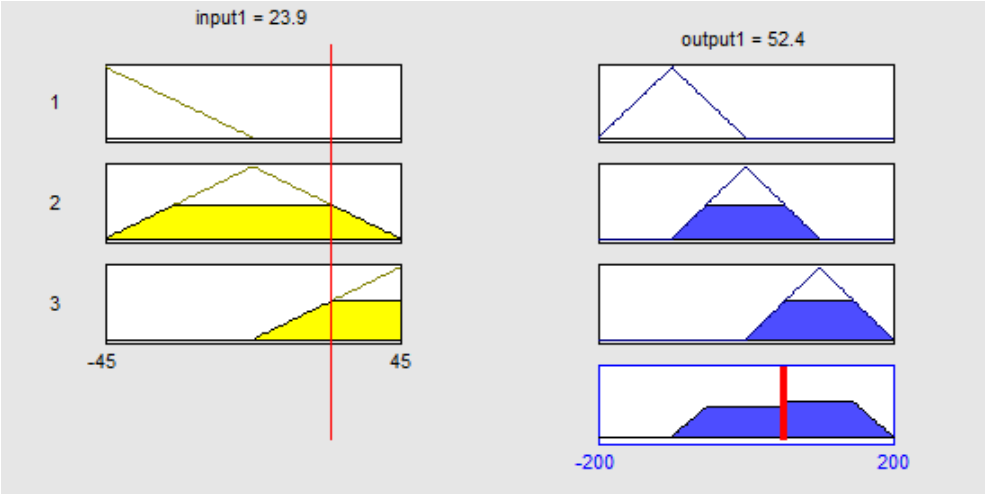
Glavni del naše naloge je bil implementacija krmilnika z uporabo mehke logike. Vsako spremembo ki smo jo napravili v obliki pripadnostnih funkcij, smo pred implementacijo preverili v okolju Matlab z uporabo paketa fuzzy. Tako smo si lažje in hitreje vizualizirali odziv, ki ga poda sistem s popravljenimi pripadnostnimi funkcijami glede na vhod.

S pomočjo regulacije PID in njenih faktorjev, smo poznali območja in potrebne vhode za uspešno stabilizacijo. Tako smo se odločili najprej približati odzivu regulacije PID, nadaljnje delo pa bo sestavljalo izpopolnjevanje vodenja.

Odločili smo se za implementacijo lastnega mehkega krmilnika, implementiranega skupaj s pravili v kodo. Ob takem načinu dela imamo popoln nadzor nad programom in njegovim izvajanjem, algoritem je hitrejši, morebitno razhroščevanje pa postane enostavnejše. Pri takem načinu je potrebno veliko pozornosti nameniti ob spreminjanju pripadnostnih funkcij, ki so v algoritmu zakodirane neposredno v postopek računanja. Obstajajo odprtokodne rešitve za enostavno implementacijo mehke logike, kot je na primer Free Fuzzy Logic Library, ki omogočajo lažjo manipulacijo pripadnostnih funkcij in računanje odzivov. Pripadnostne funkcije so v takih rešitvah podane kot tabela točk in pisni opis. V primeru, da bi naše vodenje obsegalo več vhodov v sistem in kompleksnejše funkcije, bi bila implementacija hitrejša, predvsem pa preglednejša z uporabo že razvitih rešitev.

V paketu fuzzy smo s pomočjo enostavnih funkcij definirali prvi odziv. S tremi vhodnimi in tremi izhodnimi trikotnimi funkcijami smo se zadovoljivo približali odzivu regulacije P. Prikazane funkcije na sliki 5.4 prikazujejo začetno približevanje rešitvi z mehko logiko. Na sliki 5.5 je prikazan odziv

sistema z uporabo mehke logike na podana pravila.

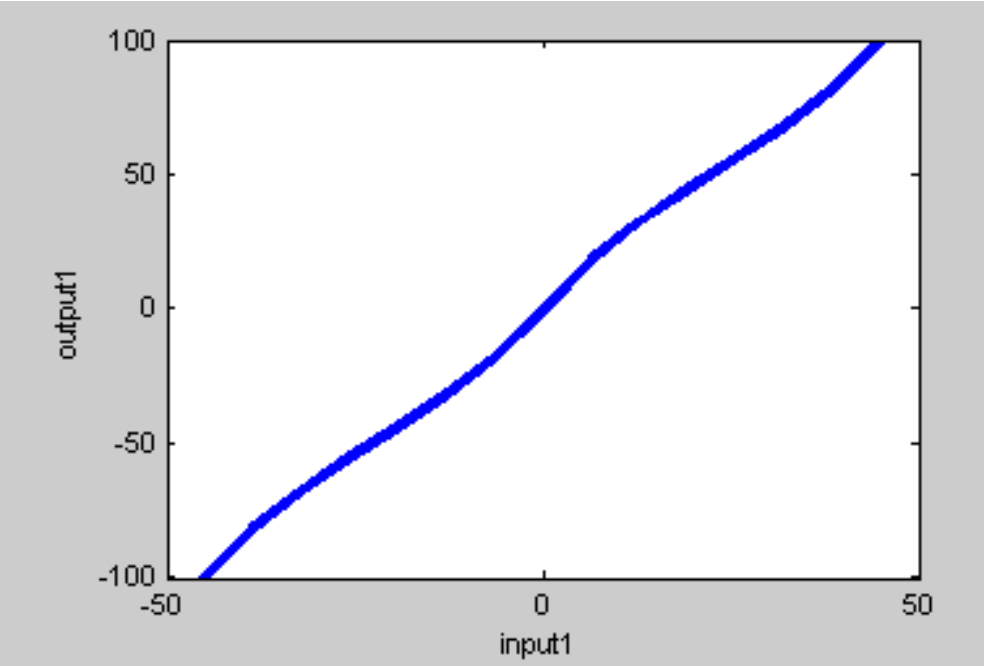


Slika 5.4: Primer bližanja odziva z mehko logiko krmilniku P.

V naslednjem koraku smo odziv prenesli na naš model. Ob vhodu v sistem je program izvedel mehčanje spremenljivke kotne hitrosti po pripadnostnih funkcijah, vidnih na sliki 5.4. Mehčanje je potekalo z računanjem odziva vsake funkcije in operacijo $\max(0, f(x))$, kjer je $f(x)$ funkcija pripadnosti, končna pripadnost pa tako poda le pozitivne rezultate. Ostrenje smo opravili s pomočjo centra gravitacije.

Ob računanju težišča smo naleteli na prvo omejitev strojne opreme. Naša implementacija algoritma je v prvi implementaciji integrirala skozi celoten obseg gibanja od -45 do 45 stopinj/sekundo. Na ta način smo dobili največjo natančnost, vendar je algoritem potreboval preveč sistemskih sredstev. Ob vodenju je sistem izračunaval odzive s precejšnjim zamikom. Ob eni vhodni spremenljivki je krmiljenje ohranjalo stabilno pot, vendar vidno slabše kot s proporcionalnim krmilnikom, ob dodajanju dodatnih spremenljivk pa je odziv že vidno zamujal, kolo pa ni bilo stabilno.

Sledila je poenostavljena implementacija iskanja težišča. Odločili smo se za uporabo povprečenja težišč vsake pripadnostne funkcije. Zaradi enostavne zgradbe funkcij je tak način izredno hiter in natančen. Algoritem deluje po



Slika 5.5: Odziv vodenja z mehko logiko.

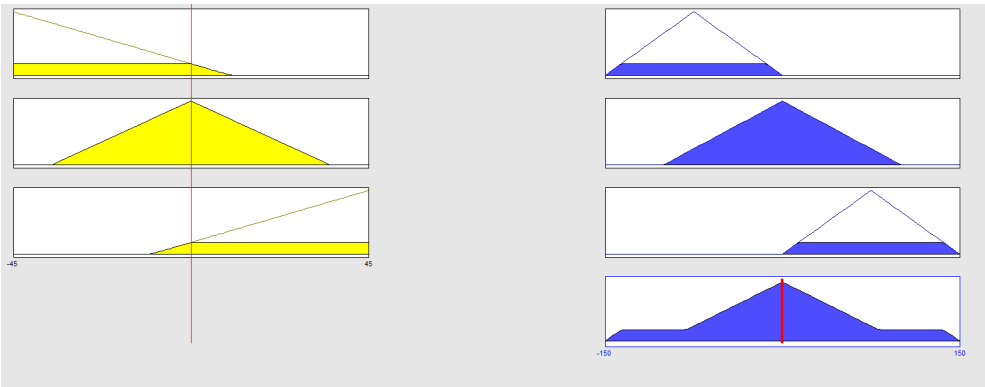
enačbi (2.6).

Zaradi znanih težišč vseh likov je računanje hitro in preprosto. Manjša odstopanja nastajajo pri težiščih prekrivanj, kjer liki niso simetrični. Ker se težišče lika spreminja linearno v odvisnosti od vrednosti funkcij, ki se prekrivata, ga lahko sproti interpoliramo iz težišč pri minimalni in maksimalni vrednosti prekrivanja.

Ob praktičnih poskusih se je izkazalo, da je odziv pri majhnih odstopanjih nekaj hitrejši, kot bi bilo potrebno zato v primerjavi z regulacijo PID ustvarja več nihanja.

S prilagajanjem funkcij smo odziv regulacije prilagodili našim potrebam. Ob majhnih vhodih v sistem smo dosegli milejši odziv. Model na tak način ohrani stabilnost, vožnja modela pa postane manj sunkovita, z daljšimi zavoji. Končne vhodne trikotne funkcije so porazdeljene simetrično. Leva pripadnostna funkcija linearno pada od -45°s do +10°/s, desna narašča od -10°/s do +45°/s. Sredinska pripadnostna funkcija poteka od -35°/s do 35°/s

z vrhom pri 0°/s. Vsaka vhodna funkcija vpliva na eno izhodno funkcijo. Razpon leve izhodne funkcije je od -150°/s do 0°/s, s težiščem pri -75°/s, desna je tudi v tem primeru zrcaljena preko 0 in tako poteka od 0°/s do 150°/s, s težiščem pri 75°/s. Sredinska funkcija zavzema vrednosti od -100°/s do 100°/s, z vrhom pri 0.

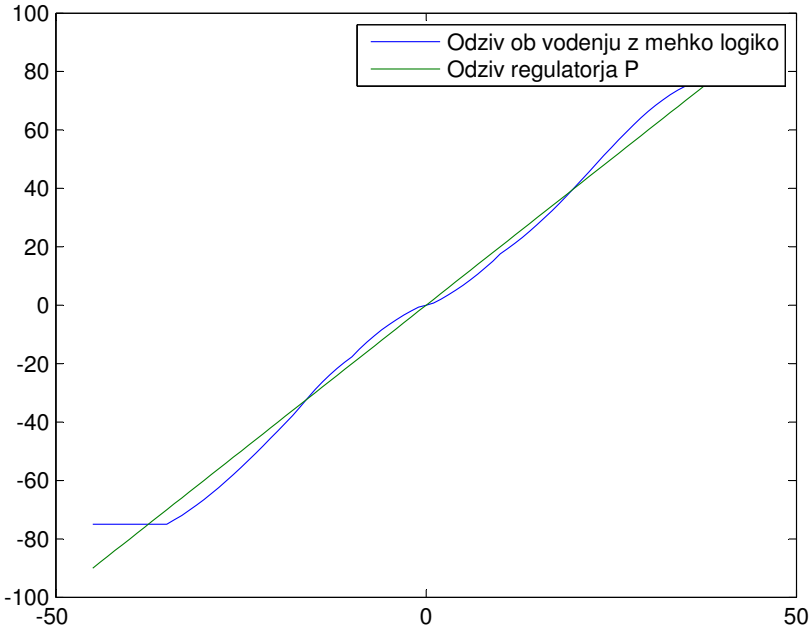


Slika 5.6: Končna oblika funkcij pri vodenju z mehko logiko.

Na sliki 5.6 so prikazane pripadnostne funkcije in odziv sistema ob ravnovesni legi. Na sliki 5.7 je prikazana razlika med odzivom sistema z uporabo proporcionalnega vodenja (zelena črta) in odzivom sistema, ki uporablja mehko logiko (modra črta).

V nadaljevanju dela smo z mehko logiko poskusili vpeljati nove spremenljivke. Podobno, kot pri proporcionalni regulaciji, smo z upoštevanjem nagiba poskušali kolo spraviti bližje stabilnemu položaju hitreje. Vožnja ob majhnih odklonih ni prinesla vidnih izboljšav, ob večjem odklonu pa je odziv postal pregrob. Med vožnjo je prihajalo do vse večjih odmikov od stabilne lege, kolo pa je izgubilo ravnotežje.

Odziv, prikazan na sliki 5.7, se v praksi izkaže kot najbolj primeren. Kolo se je ob vseh razmerah obnašalo primerljivo kot vodenje z regulacijo PID, pridobili pa smo nekoliko ublaženo nihanje vožnje.



Slika 5.7: Primerjava odzivov regulatorja P in mehke logike.

5.4 Uporabniško krmiljenje

Ob stabilizaciji modela med vožnjo smo želeli uporabniku dodati možnost upravljanja smeri modela. Uporabnik bi ob vožnji modela preko tipkovnice na računalniku krmilil model v želeno smer. V program smo dodali branje sporočil bluetooth, ki jih model sprejme od povezanega računalnika. Program ob vsakem obhodu zanke preverja pritisk tipke na povezanem računalniku in omogoča krmiljenje modela.

Ob zaznanem pritisku program povečuje ali zmanjšuje spremenljivko, ki kaže željen odmik. Na ta način dobimo skozi čas približno zvezno funkcijo, ki v primerjavi z diskretno omogoča bolj enakomerne odzive vodenja.

5.4.1 Vožnja z regulatorjem PID

Uporabnik ob pritisku tipke na računalniku kolesu pošilja ukaz za spremembo smeri. Ob daljšem pritisku je želena vrednost za spremembo smeri večja. V prvi implementaciji vodenja z uporabniškim vnosom smeri, smo vhod uporabnika sešteli z odzivom sistema vodenja. Ob takem načinu vodenja uporabnik spremeni odklon krmila in s tem povzroči spremembo smeri. Regulacija bi nato samostojno poskrbela za stabilizacijo modela ob nadaljnji vožnji.

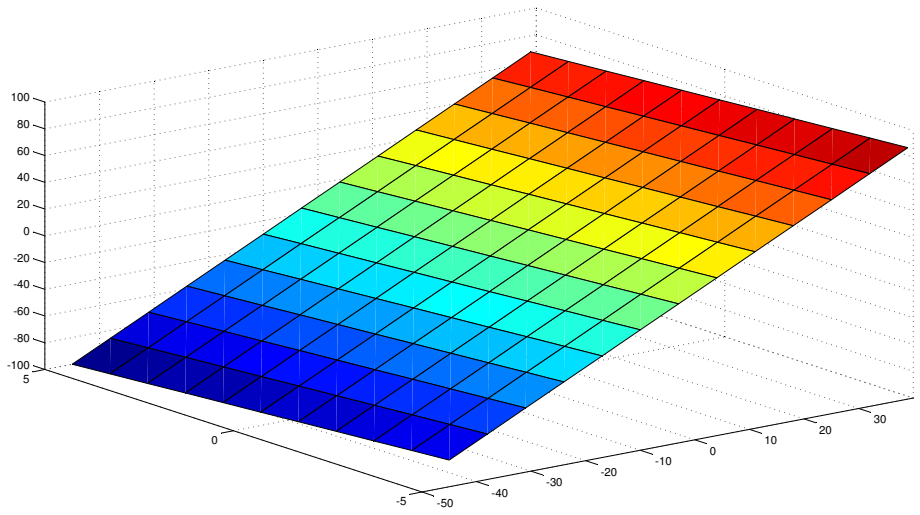
Ob praktičnih poskusih se je izkazalo, da je model preobčutljiv na tak način oddaljenega vodenja. V primeru, da je uporabnik podajal ukaz medtem ko je bil model v nihanju blizu skrajnih točk, je uporabniški vnos modelu lahko preusmeril odziv, posledica česar je bil padec.

Spremenili smo pristop in pred vhodom v regulacijski sistem zamaknili vhodne parametre za željen uporabniški naklon. V primeru, da uporabnik želi zaviti levo, sporočimo regulaciji podatek, da se model nagiba v nasprotno smer. Na ta način, algoritem sam poskrbi za padec v zavoj in vzdrževanje stabilne vožnje. Usmerjanje modela je postalo bolj enakomerno, hkrati pa obstaja manj možnosti, da se model prevrne. Odziv sistema z uporabo P vodenja ob uporabniškem vnosu predstavlja slika 5.8.

V praksi se je pokazalo, da je vodenje primerno ob majhnih odmikih od ravnovesne lege in uporabniškem ukazu po blagem zavoju. V primeru, da je kolo bilo v globokem nagibu (več 10 stopinj), ali je uporabnik zahteval preoster zavoj, je vodenje dopuščalo pospeševanje nagibanja, ob naslednjih obhodih pa je bil odziv krmilnega motorja prepočasen, da bi uspel padanje zaustaviti.

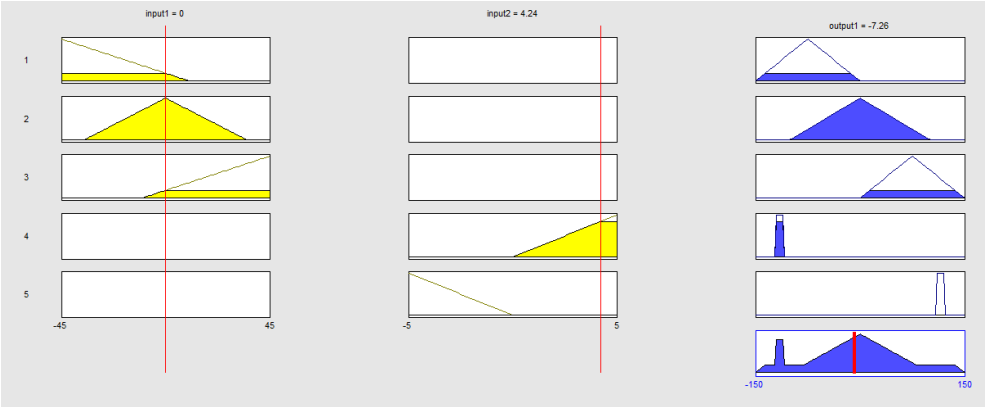
5.4.2 Vožnja z mehko logiko

Zaradi izkušenj s krmiljenjem proporcionalne regulacije, uporabniškega vnosa nismo uporabili neposredno na odzivu motorja. Zaradi podobnega odziva sistemov, tudi v tem primeru vodenje ne bi uspelo zadržati padca. Pri uporabi mehke logike, smo se odločili v sistem vpeljati še eno spremenljivko, ter ji dodati lastne izhodne funkcije.



Slika 5.8: Odziv regulacije PID na vhod.

Ob implementaciji novih pravil, se pokaže največja prednost mehke logike v primerjavi z ostalimi načini regulacije. V primeru zavijanja, smo tako postavili zgolj dve trapezoidni funkciji, ki sta odziv prilagajali skladno z uporabnikovim vnosom. Na sliki 5.9 je prikazana končna implementacija pripadnostnih funkcij. Izboljšanje pozicije in oblike funkcij je potekalo eksperimentalno. Odziv na vodenje ob uporabi pravil na sliki 5.9 je viden na sliki 5.10. V primerjavi z regulacijo PID, je odziv logično bolj smiseln. Ob močnem nagibu v desno je model nepotrebno dodatno siliti v zavoj, saj je že njegovo gibanje dovolj, da bo sam začel zavijati v želeno smer. Z boljšim pozicioniranjem in obliko funkcij bi lahko vodenje še dodatno izboljšali, že trenutna implementacija pa model vodi bolje kot regulator PID.



Slika 5.9: Pravila mehkega krmilnika z zavijanjem uporabnika.

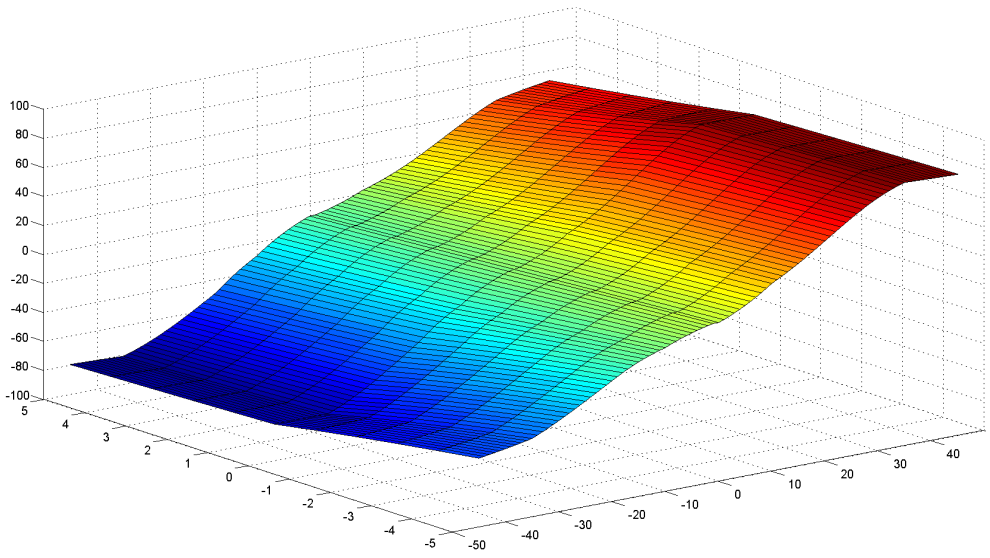
5.5 Beleženje stanja med vožnjo

Za potrebe analiziranja naših voženj, smo preko vmesnika bluetooth na konzolo razhroščevalnika zapisovali sprotne vrednosti senzorjev in naših odzivov. S tem smo pridobili pomembne informacije o reakcijah modela na naš sistem vodenja. Merjene količine so vsebovale čas vožnje, zasuk krmila, moč motorja krmila, hitrost nagibanja in nagnjenost nagibanja. S pridobljenimi podatki smo sproti preverjali pravilnost naših tez o stabilizaciji kolesa.

Z grafoma 5.11 in 5.12 smo tako potrdili pravilno implementacijo računanja odziva tudi na kocki, hkrati pa smo ob poskusih ugotovili, da ob takšnih odzivih kolo samostojno ohranja ravnotežje. Vsaka točka na grafu je bila posneta med vožnjo, nato pa prikazana na grafu kot odvisnost odziva sistema (moč motorja) na vhod v sistem (hitrost nagibanja).

Zagotavljanje ponovljivih okoliščin vožnje je pri naših poskusih problem, ki ga nismo uspeli rešiti. Problematična je začetna postavitev modela, ki je odvisna od uporabnika, zaradi prostega hoda v krmilnem motorju, pa je izredno težko zagotoviti enako pozicijo vodilnega kolesa. Analiziranje voženj ob različnih začetnih pogojih nam s tem ne podaja jasnih odgovorov.

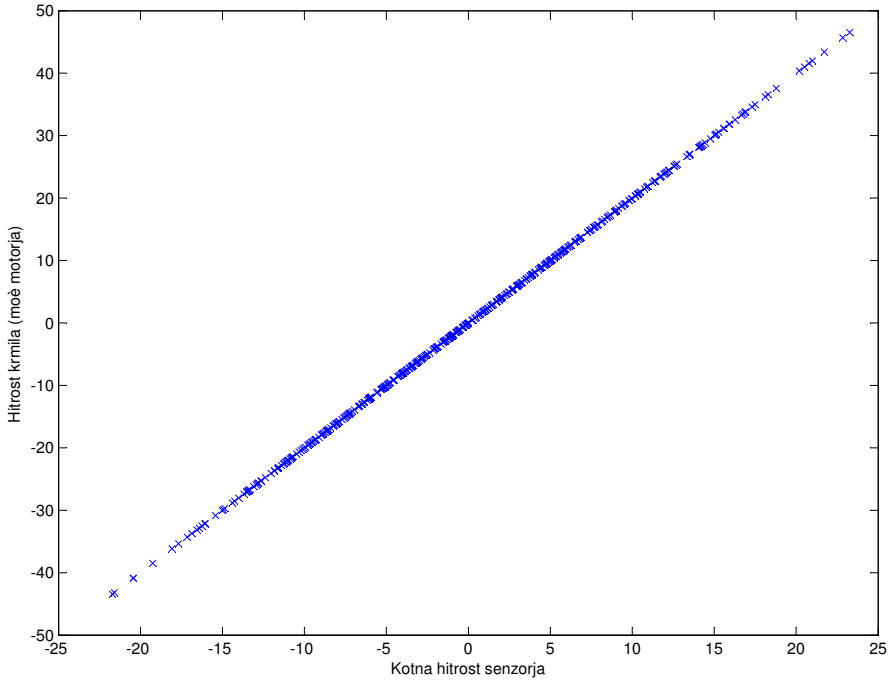
Z grafi na slikah 5.13 in 5.14 smo lahko na vodenje pogledali širše. Opazovali smo obnašanje modela skozi čas ob različnih zunanjih okoliščinah. Na



Slika 5.10: Odziv mehke regulacije na uporabniški vnos.

ta način smo lahko začetne parametre zanemarili in opazovali odvisnosti krivulj ob spremembah algoritma. Slika 5.13 nam prikazuje prvih šest sekund vožnje modela na ravni podlagi s proporcionalnim vodenjem. Opazimo, da ima krivulja nagiba (svetlo modra barva) ves čas minimalno odstopanje od ravnovesne lege. Rdeča krivulja nam prikazuje podatke pridobljene iz senzorja. Opazimo, skladno s faktorjem P, da je odziv sistema (zelena barva), po pričakovanjih glede na implementacijo, vedno za faktor 2 večji. Temno modra barva na grafu nam prikazuje odmik krmila od njegove začetne lege. Opazimo, da na ravni podlagi model uspešno ostaja v okolici ravnovesne lege, brez zunanjih motenj pa vožnja poteka v ravni liniji.

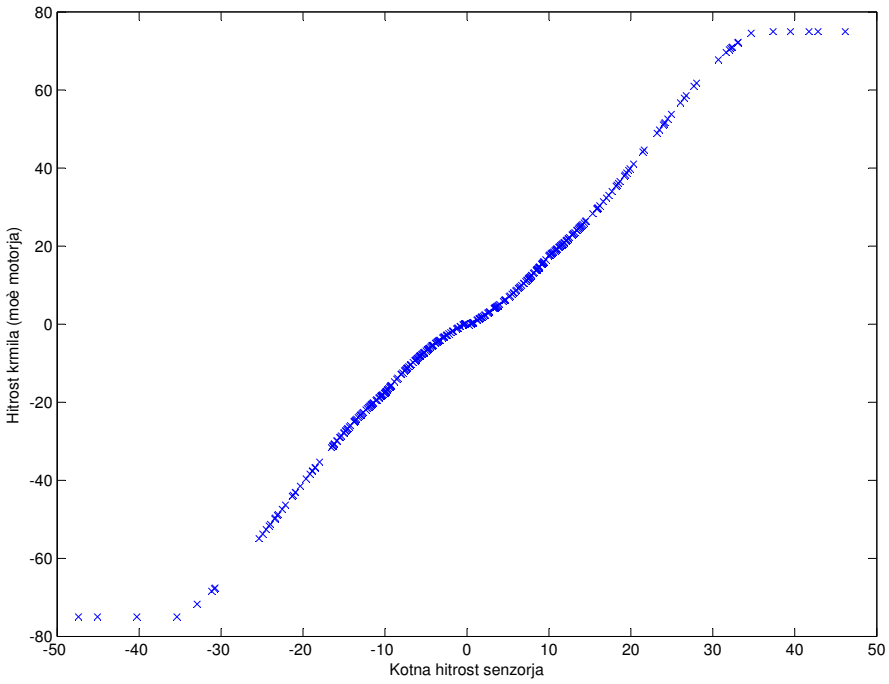
Na sliki 5.14, ki prikazuje posnetek vožnje na neravni podlagi z mehko logiko opazimo, da je nihanja mnogo več, a vodenje uspešno drži ravnotežje. Krmilno kolo večkrat popravlja svojo pozicijo, model pa uspešno nadaljuje



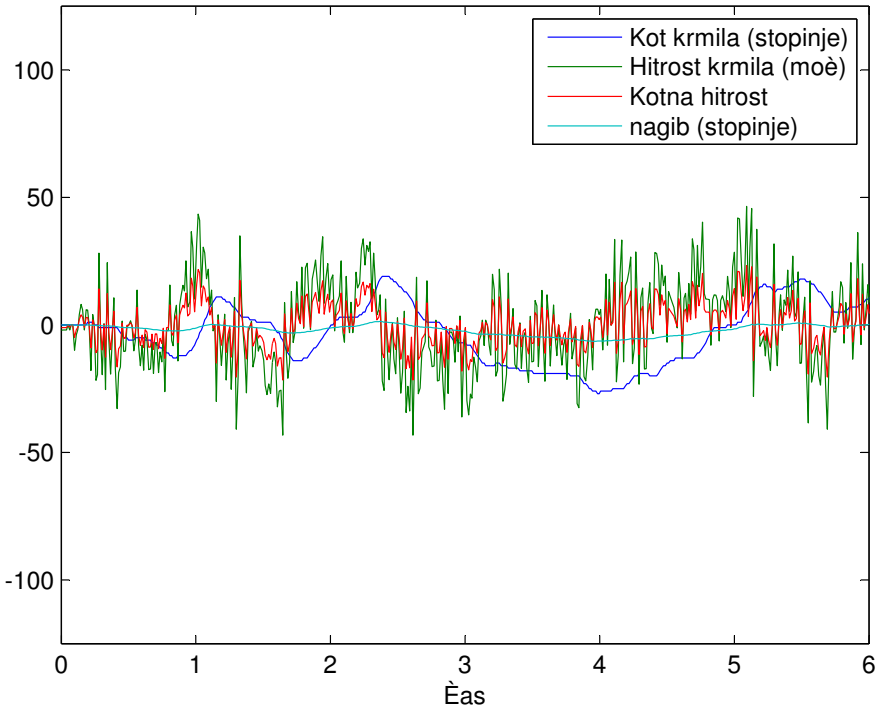
Slika 5.11: Posnetek odziva vodenja z regulacijo PID.

vožnjo. Zaradi motenj v podlagi vožnja ni potekala v ravni liniji. Po peti sekundi vožnje je model z boka prejel udarec, katerega vodenje ni uspelo pravočasno popraviti - sledil je padec kolesa.

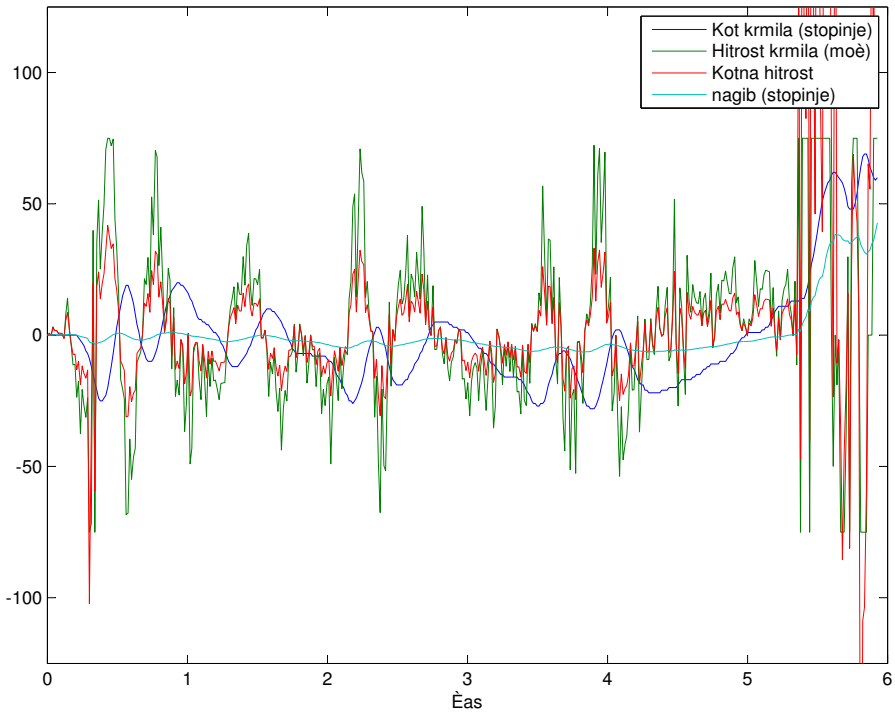
Ob analizi voženj obeh sistemov smo opazili, da model dobro kontrolira vožnjo ob počasnih spremembah, vodenje pa ne uspe popraviti nenadnih sprememb, kot je na primer rahel stranski potisk med vožnjo. K temu precej doprinese sama oblika modela in njegove značilnosti, kot so na primer dolga medosna razdalja in visoko težišče. V našem primeru se je izkazalo, da bi ob spremembah, ki bi omogočale reševanje takih situacij, model postal preveč nestabilen v normalni vožnji.



Slika 5.12: Posnetek odziva vodenja z uporabo mehke logike.



Slika 5.13: Posnetek vožnje z uporabo proporcionalnega vodenja.



Slika 5.14: Vožnja z uporabo krmilnika z mehko logiko na slabi podlagi s trkom.

Poglavje 6

Zaključek

V predstavljenem delu smo analizirali vodenje kolesa in ga uspešno implementirali na primeru. Vodenje s proporcionalnim krmilnikom in mehko logiko smo primerjali in ocenili glede na naša opažanja.

Izvedeno vodenje s proporcionalnim krmilnikom se je izkazalo za uspešno metodo ohranjanja ravnotežja kolesa. Največja prednost je enostavnost uporabe in razumevanja. Vodenje je uspešno stabiliziralo model tudi ob zmanjšanju hitrosti in slabi podlagi. Prilagajanje odziva je izredno preprosto, vendar omejeno na relativno preproste odzivne funkcije.

Uporaba mehke logike v modelu se je izkazala kot možna alternativa proporcionalnemu vodenju. Največja prednost je visoka stopnja prilagodljivosti odziva in dodajanje novih vhodnih spremenljivk. Z optimizacijo funkcij in algoritmov smo dosegli dovolj visoko hitrost obdelave informacij, da smo lahko obšli sistemske omejitve pri računanju odziva. Z njeno uporabo smo dosegli lepšo, bolj stabilno vožnjo.

Primerjava je v našem primeru pokazala, da je vodenje z mehko logiko bolj prilagodljivo in uspešno. Prepričani smo, da bi z dovolj temeljito optimizacijo enega in drugega vodenja, lahko dosegli še boljše rezultate pri obeh načinih, vendar so podane implementacije dovolj dober pokazatelj razlik v načinu vodenja. V kompleksnih sistemih je uporaba mehke logike smiselna, predvsem zaradi lažje predstave uporabniku bolj enostavna in hitrejša regu-

lacija PID, pa je primerna predvsem v sistemih, ki zahtevajo hitro odzivnost in imajo omejena sistemska sredstva.

V nadaljnjem delu bi model poskušali nadgraditi z avtomatskim sistemom speljevanja, ki bi iz poskusov odstranil možnost napake uporabnika pri zagonu modela. Omejitve Lego NXT kompleta ne omogočajo popolnoma objektivnih pogojev za testiranje, saj imajo komponente preveč prostega hoda. Dodatna nadgradnja obstoječega sistema bi lahko vsebovala dodatne senzorje, s katerimi bi se model izogibal trkom ali orientiral po svetlobi okolice.

Literatura

[1] L. A. Zadeh, “Fuzzy sets. Information and control”, Elsevier (1965)

[2] A. Dobnikar, B. Šter, “Mehko računanje za modeliranje, razpoznavanje in regresijo”, Založba FE in FRI, Ljubljana, 2008

[3] B. Kosko, ”Fuzzy Thinking: The New Science of Fuzzy Logic”, Hyperion, 1993.

[4] The Takagi-Sugeno fuzzy system. dostopno septembra 2013 na: http://www.atp.ruhr-uni-bochum.de/rt1/syscontrol/node129.html

[5] A fuzzy implementation, dostopno septembra 2013 na: http://apps.ensic.inpl-nancy.fr/benchmarkWWTP/RiskAnalysis/RiskWeb/RiskModule_070423_fichiers/Fuzzy_implementation_070423.pdf

[6] Fuzzy Sets and Fuzzy Techiques, dostopno septembra 2013 na: http://www.cb.uu.se/∼joakim/course/fuzzy/vt07/lectures/L11_4.pdf

[7] NXTbike-GS (self-balancing bike robot by steer-into-fall), dostopno septembra 2013 na: http://www.mathworks.com/matlabcentral/fileexchange/27694-nxtbike-gs-self-balancing-bike-robot-by-steer-into-fall

[8] RobotC for Mindstorms, dostopno septembra 2013 na: http://www.robotc.net/support/nxt/media/documents/ROBOTC%20for%20MINDSTORMS.pdf

[9] J. D. G. Kooijman, J. P. Meijaard, J. M. Papadopoulos, A. Ruina, A. L. Schwab, "A bicycle can be self-stable without gyroscopic or caster effects", Science, (2011), 332(6027), 339-342.

[10] Sharma, Himanshu Dutt, N. Umashankar, "A fuzzy controller design for an autonomous bicycle system." Engineering of Intelligent Systems, 2006 IEEE International Conference on. IEEE, 2006.

[11] Sharma, Himanshu Dutt, and N. Umashankar, "Stabilization of Autonomous Bicycle using Fuzzy Controller with Maximum Allowable Lean Constraint." Proceedings of 3rd International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2005). 2005.

[12] Sharma, Himanshu Dutt, and N. Umashankar, "A robotic model (ROBI) of autonomous bicycle system." Computational Intelligence for Modelling, Control and Automation, 2006 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on. IEEE, 2006.

[13] N. Umashankar and H. Sharma, "Adaptive neuro-fuzzy controller for stabilizing autonomous bicycles", Proc. IEEE Conf. Robotics and Biomimetics, pp.1652 -1657 2006

[14] Sharma, Himanshu Dutt, Sameer M. Kale, in N. Umashankar, "Simulation model for studying inherent stability characteristics of autonomous bicycle." Mechatronics and Automation, 2005 IEEE International Conference. Vol. 1. IEEE, 2005.

[15] JBike6, dostopno septembra 2013 na:
http://ruina.tam.cornell.edu/research/topics/bicycle_mechanics/JBike6_web_folder/

[16] Free Fuzzy Logic Library, dostopno septembra 2013 na:
http://ffll.sourceforge.net/index.html