SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
Zavod za automatiku i računalno inženjerstvo

DIPLOMSKI RAD br. 1533

ADAPTIVNI NEURONSKI REGULATOR S REFERENTNIM MODELOM

Marko Vincek

Zagreb, studeni 2006.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
POVJERENSTVO ZA DIPLOMSKI ISPIT

Zagreb, 7. srpnja 2006.

Zavod: **Zavod za automatiku i računalno inženjerstvo**
Predmet: **Automatizacija postrojenja i procesa**

DIPLOMSKI ZADATAK br. 1533

Pristupnik: **Marko Vincek**
Studij: Elektrotehnika
Smjer: Automatika

Zadatak: **Adaptivni neuronski regulator s referentnim modelom**

Opis zadatka:

Adaptivne metode upravljanja nelinearnim procesima često se oslanjaju na neuronske mreže kao univerzalne aproksimatore u obje osnovne sastavnice adaptivnog upravljanja: identifikaciji procesa i adaptaciji regulatora. U sklopu ovoga rada potrebno je analizirati osnovne strukture neuronskog upravljanja, te razraditi metodologiju upravljanja neuronskim regulatorom s referentnim modelom kojeg se podešava on-line posrednim učenjem. Ovako koncipirani regulator implementirati u programskom jeziku C pod Matlab okruženjem, te njegovu valjanost ispitati na eksperimentalnom postavu procesa magnetske levitacije.

Zadatak uručen pristupniku: 10. srpnja 2006. u 12:00
Rok za predaju rada: 10. studenog 2006. u 12:00

Mentor:
N. Perić
Prof. dr. sc. Nedjeljko Perić

Djelovođa:
Ivan Petrović
Prof. dr. sc. Ivan Petrović

Predsjednik povjerenstva za diplomski ispit:
N. Perić
Prof. dr. sc. Nedjeljko Perić

Zahvaljujem Mariu Vašku na trudu i pomoći pri izradi ovog rada i profesoru Nedjeljku Periću na mentorstvu, podršci i ohrabrenju.

SADRŽAJ

1. UVOD

Istraživanja i razvoj umjetnih neuronskih mreža motivirana su spoznajama o građi i načinu funkcioniranja ljudskog mozga te njegovim nevjerojatno velikim sposobnostima u rješavanju složenih problema. Dva su osnovna cilja tih istraživanja: prvi je razvoj novih struktura umjetnih neuronskih mreža koje bi funkcionirale na analogan način kao što funkcionira ljudski mozak i koje bi mogle oponašati barem neke njegove funkcije, a drugi je njihova primjena u rješavanju praktičnih problema.

Razvijeno je mnoštvo različitih struktura neuronskih mreža koje se u osnovi mogu podijeliti na statičke i dinamičke neuronske mreže. Sa stajališta primjene njihovo je najvažnije svojstvo sposobnost aproksimacije proizvoljnih kontinuiranih funkcija.U ovom se radu istražuje primjena statičkih neuronskih mreža u identifikaciji nelinearnih dinamičkih procesa, dok se dinamičke neuronske mreže koriste za sintezu adaptivnog sustava upravljanja sa referentnim modelom.

Primjena umjetnih neuronskih mreža u upravljanju nelinearnim procesima ubraja se među njihove najznačajnije primjene. U radu se obrađuju strukture upravljanja koje su sa stajališta teorije upravljanja dobro utemeljene i svojstva kojih su dobro istražena, a na kraju se daje opis strukture upravljanja s referentnim modelom uz on-line učenje (adaptaciju) parametara neuronskog regulatora.

Obrađeni postupci identifikacije i strukture upravljanja proveravaju se na primjeru nelinearnog procesa magnetske levitacije.

2. NEURONSKE MREŽE

2.1) Biološka osnova razvoja neuronskih mreža

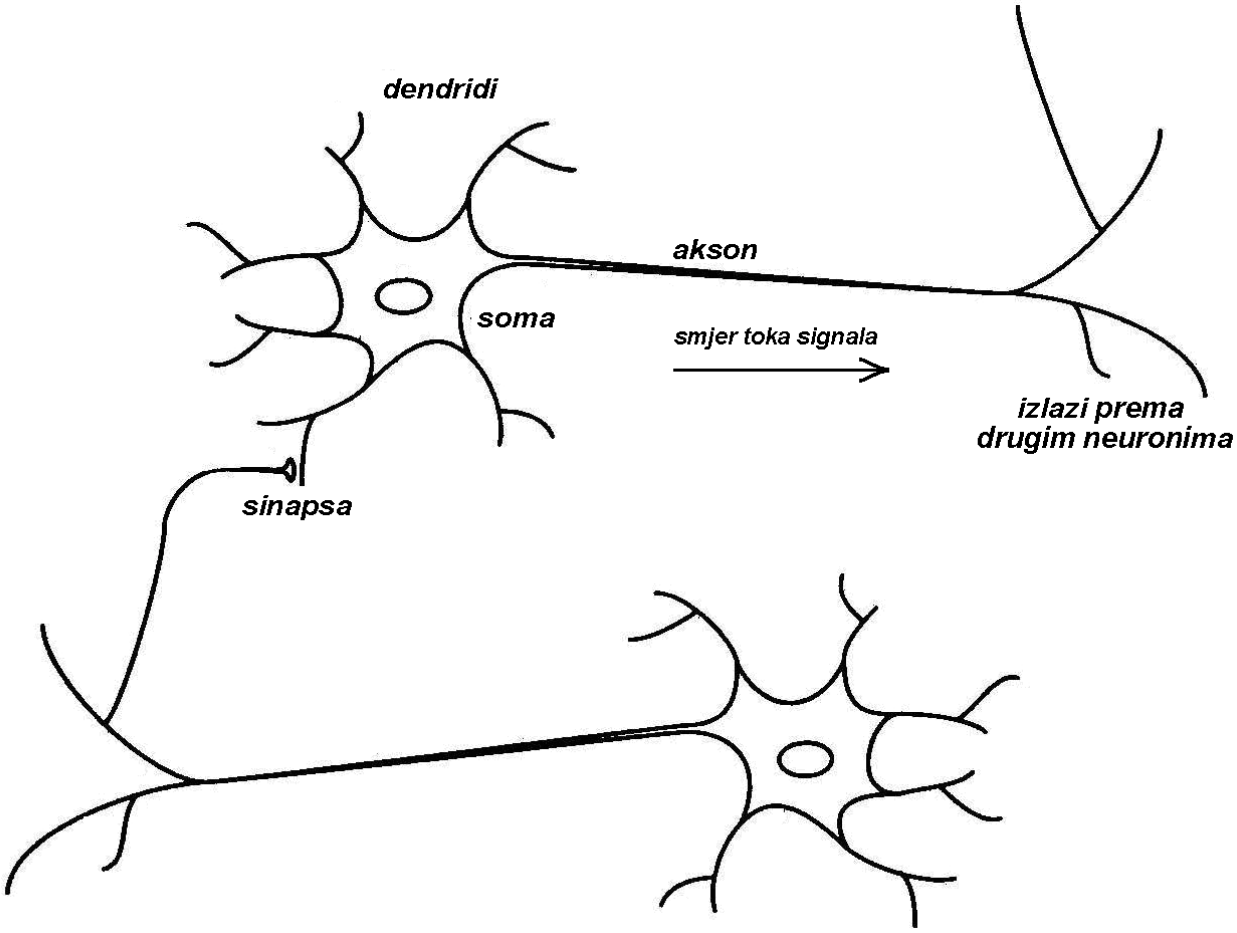
Ljudski je mozak previše složen da bi se postojećim metodama neurofiziologije mogao potpuno opisati i razumjeti način njegova djelovanja. Ipak, dosta je njegovih funkcionalnih aktivnosti već objašnjeno. Sastoji se od oko 10^{11} osnovnih živčanih stanica (neurona) organiziranih u module (slojeve, *engl.* layers) i međusobno povezanih u složenu mrežu s nekih 10^{15} međusobnih veza. Zbog tako velikoga broja veza među neuronima, još nema pravih spoznaja o broju modula i načinu na koji su oni organizirani. Ovako gusto povezana mreža neurona osigurava izuzetno veliku računsku i memorijsku moć ljudskoga mozga. Sve čovjekove aktivnosti i njegovo ponašanje uvjetovane su procesima koji se zbivaju unutar ove moćne biološke neuronske mreže.

Biološki neuron, kao osnovna gradivna jedinica biološke neuronske mreže, prima i obrađuje informacije od drugih neurona i/ili od osjetilnih organa. Sa stajališta obradbe signala, bez ulaženja u opisivanje elektrokemijskih procesa koji se događaju pri prijenosu i obradbi signala, biološki se neuron može pojednostavljeno prikazati kao stanica sastavljena od tijela (soma), mnoštva dendrita i aksona (Sl. 2.1.). Akson se može zamisliti kao tanka cjevčica čiji je jedan kraj povezan na tijelo neurona, a drugi je razdijeljen na mnoštvo grana. Krajevi ovih grana završavaju malim zadebljanjima koja najčešće dodiruju dendrile, a rjeđe tijelo drugoga neurona. Mali razmak između završetka aksona prethodnoga neurona i dendrida sljedećega neurona naziva se sinapsa. Akson jednoga neurona formira sinaptičke veze s mnoštvom drugih neurona. Impulsi, koji se generiraju u tijelu neurona, putuju kroz akson do sinapsi. Ovisno o učinkovitosti svakoga pojedinačnoga sinaptičkoga prijenosa, signali različita intenziteta dolaze do dendrita. Učinkovitost sinaptičkoga prijenosa kroz neku sinapsu ovisi o njezinome elektrokemijskom stanju, koje je rezultat prethodnih sinaptičkih prijenosa kroz nju. Sinapse, dakle, predstavljaju memorijske članove biološke neuronske mreže. Signali se od sinapsi dendritima prosljeđuju do tijela neurona, gdje se prikupljaju i obrađuju. Ovi signali mogu za tijelo neurona biti pobuđujući ili smirujući. Matematički gledano, pobuđujući i smirujući signali imaju suprotan predznak. Ako je njihova kumulativna vrijednost tijekom kratkog vremenskog intervala veća od praga osjetljivosti neurona, tijelo neurona generira impulse (tzv. aktivacijske potencijale) koji se šalju duž aksona prema drugim neuronima, a ako je manja, neuron ostaje nepobuđen i ne generira impulse.

Na osnovi ovoga opisa funkcioniranja biološkoga neurona može se zaključiti da se obradba signala u njemu odvija kroz dvije odvojene operacije:

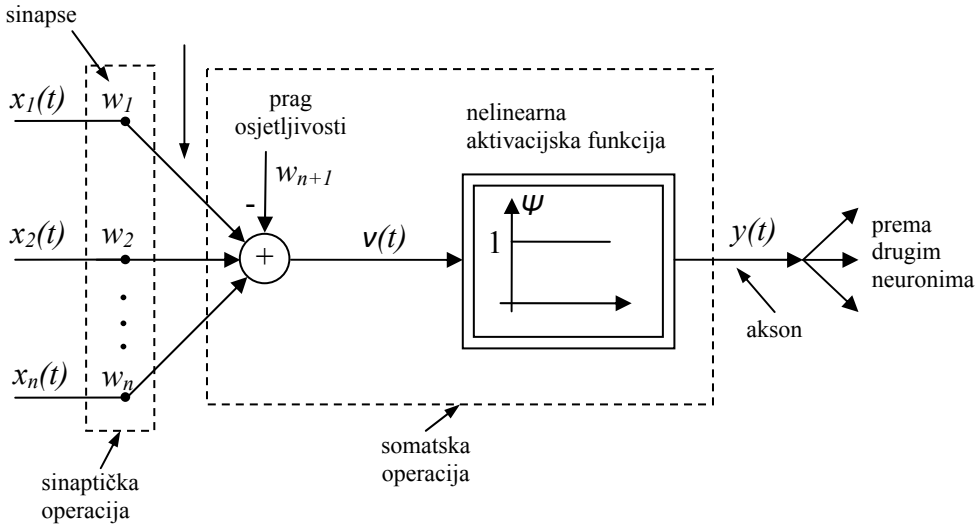
• *sinaptička operacija:* daje određeni značaj (težinu) svakom ulaznom signalu u neuron;

• *somatska operacija:* prikuplja sve "otežane" ulazne signale, te ako je njihova kumulativna vrijednost veća od praga osjetljivosti neurona, generira impulse koje šalje prema drugim neuronima.



Slika 2.1. Shematski prikaz biološkog neurona.

Prvi radovi o umjetnim neuronskim mrežama pojavili su se četrdesetih godina dvadesetog stoljeća. U njima se opisuje vrlo jednostavan model neurona koji, kao i biološki neuron, obrađuje signale kroz sinaptičku i somatsku operaciju (Sl. 2.2.). Ovaj vrlo jednostavan model neurona nazvan je *perceptron.* Sinaptička je operacija predstavljena množenjem svakog ulaznog signala x_i s težinskim koeficijentom w_i. Tako se otežani ulazni signali zbrajaju i njihov se zbroj uspoređuje s pragom osjetljivosti neurona w_{n+1}. Ako je zbroj otežanih signala veći od praga osjetljivosti neurona, nelinearna aktivacijska funkcija ψ generira izlazni signal neurona y iznosa jedan, a ako je manji, izlaz neurona je nula.



Slika 2.2. Shematski prikaz perceptrona.

Smatralo se da je povezivanjem velikog broja perceptrona moguće modelirati ljudski mozak. Međutim, iako po svojoj organizaciji podsjeća na biološki neuron, perceptron je prejednostavan, pa stoga ima ograničene sposobnosti predstavljanja.

2.2) Osnovni modeli umjetnih neurona

Opis biološkoga neurona u točki 2.1. samo načelno prikazuje način njegova funkcioniranja. U stvarnosti je funkcioniranje biološkoga neurona znatno složenije. K tomu, do danas su neurofiziolozi otkrili stotinjak različitih vrsta bioloških neurona u ljudskome mozgu, tako da je izradba modela koji vjerno opisuju njihove složene karakteristike vrlo složen zadatak. S druge strane, sa stajališta primjene neuronskih mreža uglavnom i nije potrebno koristiti složene modele neurona. Stoga, većina dosad razvijenih modela umjetnih neurona samo svojom strukturom podsjeća na biološke neurone, bez pretenzija da budu njihovi stvarni modeli. Unatoč velikome broju različitih modela neurona, moguće ih je svrstati u dvije osnovne skupine: statičke i dinamičke modele neurona. U ovom radu korišteni su statički modeli neurona te u nastavku donosim opis takvog neurona.

Perceptron ima skromne mogućnosti predstavljanja, što je u najvećoj mjeri posljedica diskontinuiteta aktivacijske funkcije. Osim toga, zbog diskontinuiteta aktivacijske funkcije otežano je učenje mreže, jer većina algoritama učenja za podešavanje težinskih koeficijenata unutarnjih slojeva mreže zahtijeva izračunavanje derivacije aktivacijskih funkcija. Ova se ograničenja perceptrona mogu prevladati primjenom kontinuirane, derivabilne aktivacijske funkcije. Iako mnoge funkcije zadovoljavaju ovaj uvjet, kao aktivacijske se funkcije najčešće koriste funkcije koje

pripadaju klasi sigmoidalnih funkcija[1], jer je dokazano da neuronske mreže izgrađene od najmanje tri sloja neurona sa sigmoidalnim aktivacijskim funkcijama mogu predstaviti (aproksimirati) proizvoljnu kontinuiranu funkciju. Prema tome, umjetni neuron sa sigmoidalnom aktivacijskom funkcijom, iako jednostavan, predstavlja vrlo korisnu aproksimaciju biološkoga neurona.

Opisani model neurona ne sadrži dinamičke članove pa njegov izlaz ovisi isključivo o trenutnim vrijednostima ulaznih signala i težinskim koeficijentima. Stoga se ovaj neuron naziva *statičkim neuronom,* a budući da opisani neuron predstavlja poopćenje perceptrona, često ga se upravo i naziva *perceptronom.* Veliki je broj neuronskih mreža izgrađen od perceptrona organiziranih u tri ili više slojeva. Te se mreže u ovome radu nazivaju *višeslojnim perceptronskim mrežama (engl.* MultiLayer Perceptron nertworks, u nastavku rada koristi se i kratica MLP mreže).

Matematički se perceptron može opisati sljedećim izrazima (Sl. 2.2.):

$$v(t) = \sum_{i=1}^{n} w_i(t) - w_{n+1} \qquad (2\text{-}1)$$

$$y(t) = \psi(v) \qquad (2\text{-}2)$$

gdje je:

$x_N(t) = [x_1(t), \quad \cdots, \quad x_n(t)]$ - vektor ulaznih signala neurona, pobudni vektor,

$w_S(t) = [w_1(t) \quad \cdots, \quad w_n(t)]$ - vektor sinaptičkih težinskih koeficijenata,

w_{n+1} - prag osjetljivosti neurona,

$v(t)$ - izlaz operacije konfluencije,

$\psi(v)$ - nelinearna aktivacijska funkcija,

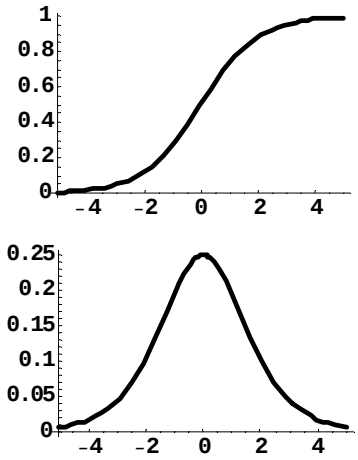
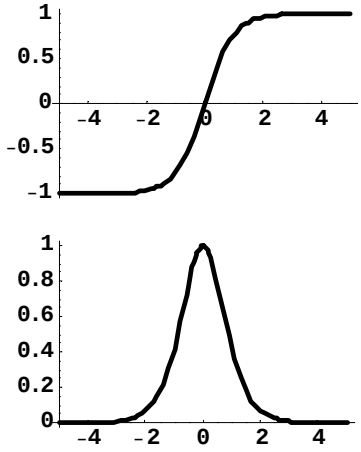
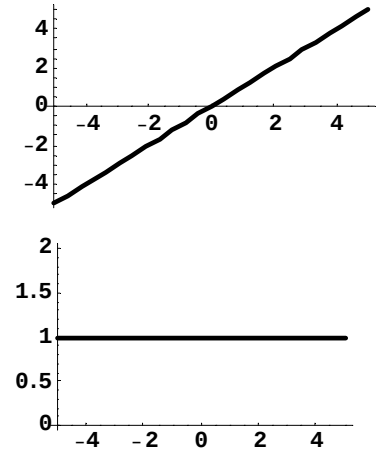
$y(t)$ - izlaz neurona.

Izraz (2-1) opisuje sinaptičku operaciju i prve dvije somatske operacije (prikupljanje otežanih ulaznih signala i usporedbu njihova zbroja s pragom osjetljivosti). Ove tri operacije zajedno čine tzv. *operaciju konfluencije*, a izraz (2-2) opisuje nelinearnu aktivacijsku funkciju. Dakle, s matematičkoga se stajališta umjetni neuron dijeli na *operaciju konfluencije* i *nelinearnu aktivacijsku funkciju.*

Nelinearna aktivacijska funkcija $\psi(v)$, izraz (2-2), preslikava izlaznu vrijednost operacije konfluencije $v(t)$ $[-\infty, \infty]$ u izlazni signal neurona $y(t)$ ograničenoga iznosa. Iznos izlaznoga signala

[1] Funkcija pripada klasi sigmoidalnih funkcija ako ispunjava sljedeće uvjete: a) monotono je rastuća funkcija od $v(t)$ u intervalu $(-\infty.\infty)$, b) asimptotski se približava donjoj graničnoj vrijednosti kako $v(t)$ teži $-\infty$, odnosno gornjoj kako $v(t)$ teži ∞ i c) ima samo jednu točku infleksije.

neurona najčešće je ograničen u području [0, 1] za unipolarne signale, a na [-1, 1] za bipolarne signale. Iako je veliki broj raznih funkcija koje omogućuju da neuronska mreža aproksimira proizvoljnu kontinuiranu funkciju, neke su funkcije prihvaćene kao standardne aktivacijske funkcije perceptrona: *logsig* i *tansig* funkcije. Aktivacijsko se pojačanje g_a uobičajeno izabire jediničnog iznosa te je tako izabrano i unutar ovog rada. U tablici 2.1. prikazana je i linearna aktivacijska funkcija (purelin), jer se ona često koristi kod neurona u izlaznome sloju mreže.

Naziv funkcije	*Izraz za funkciju i njezinu derivaciju*	*Grafički prikaz funkcije i derivacije*
Logsig	$\Psi(v) = \frac{1}{1+e^{-g_a v}}$ $\Psi'(v) = g_a \frac{e^{-g_a v}}{(1+e^{-g_a v})^2}$ $= g_a \cdot \Psi \cdot (1-\Psi)$	
Tansig	$\Psi(v) = \frac{2}{1+e^{-2g_a v}} - 1$ $\Psi'(v) = g_a \frac{4e^{-2g_a v}}{(1+e^{-2g_a v})^2}$ $= g_a \cdot (1-\Psi^2)$	
Linearna (purelin)	$\Psi(v) = g_a v$ $\Psi'(v) = g_a$	
napomena: $g_a > 0$ - aktivacijsko pojačanje (u primjerima $g_a = 1$)		

Tablica 2.1. Najčešće korištene aktivacijske funkcije kod MLP neuronskih mreža.

2.3) Osnovna svojstva neuronskih mreža

Nekoliko najvažnijih svojstava neuronskih mreža su:

• *Paralelno raspodijeljena obradba informacija*. Za razliku od konvencionalnih računskih tehnika, neuronske mreže prihvaćaju više ulaza paralelno i dobivene informacije obrađuju na raspodijeljen način. Drugim riječima, informacija spremljena u neuronsku mrežu raspodijeljena je na više računskih jedinica, što je potpuno suprotno konvencionalnome spremanju informacija u memoriju gdje je svaka posebna informacija (podatak) spremljena u svoj vlastiti memorijski prostor. Svojstvo raspodijeljenoga spremanja informacija daje neuronskim mrežama više prednosti, od kojih je najvažnija *redundantnost,* to jest otpornost na kvar. Redundantnost se može postići i kod klasičnih računskih tehnika, ali je kod neuronskih mreža ona inherentno svojstvo, slično kao kod bioloških sustava. Prema tome, neuronska će mreža raditi čak ako se i uništi neki njezin dio.

• *Učenje i adaptacija.* Svojstvo učenja i adaptacije čini neuronske mreže sposobnima obrađivati neprecizne i loše uščuvane podatke u nestrukturiranom i neodređenom okruženju. Na odgovarajući način naučena neuronska mreža ima svojstvo poopćavanja kada se na njezinome ulazu pojave podaci koji nisu bili u uzorku na osnovi kojeg je mreža naučena.

• *Univerzalni aproksimator.* Svojstvo neuronskih mreža da aproksimiraju proizvoljnu kontinuiranu nelinearnu funkciju do željene točnosti njihovo je najvažnije svojstvo sa stajališta modeliranja, identifikacije i upravljanja nelinearnim procesima.

• *Viševarijabilni sustavi.* Neuronske su mreže po svojoj strukturi viševarijabilni sustavi što ih čini lako primjenjivim za modeliranje, identifikaciju i upravljanje viševarijabilnim procesima.

• *Sklopovska implementacija.* Više je proizvođača razvilo specijalizirane sklopove za implementaciju neuronskih mreža koji omogućuju paralelnu raspodijeljenu obradbu u stvarnome vremenu.

• Neuronske su mreže računski vrlo zahtjevne. Izlaz svakog neurona rezultat je zbrajanja više umnožaka i izračunavanja nelinearne aktivacijske funkcije.

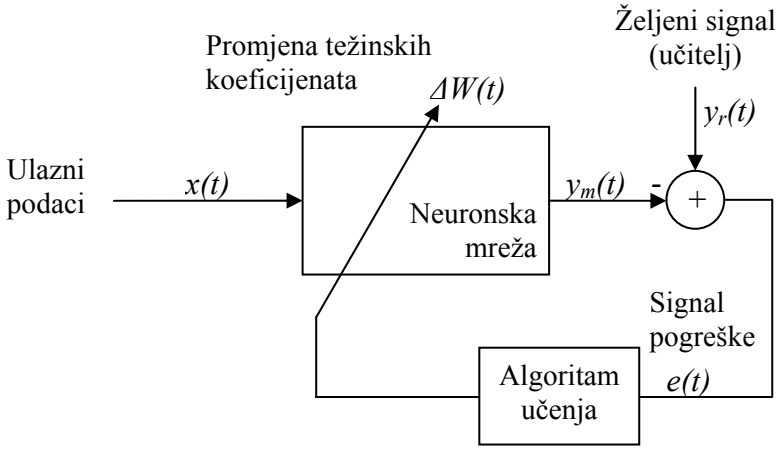
• Računska brzina neuronske mreže određena je brojem matematičkih operacija pojedinog sloja, a ne čitave mreže. Nadalje, svaki sloj mreže ima paralelnu građu, to jest svaki se neuron u sloju može promatrati kao lokalni procesor koji radi paralelno s ostalim neuronima.

• Neuronske mreže zahtijevaju veliki memorijski prostor. Naime, svaki pojedini neuron ima više sinaptičkih veza, a svakoj je od njih pridružen težinski koeficijent koji mora biti spremljen u memoriju. Povećanjem broja neurona u mreži memorijski zahtjevi rastu s kvadratom broja neurona.

2.4) Učenje neuronskih mreža

Većina neuronskih mreža zahtijeva učenje, to jest primjenu algoritama koji podešavaju iznose sinaptičkih težinskih koeficijenata. Ciljevi učenja mreže ovise o njezinoj primjeni, pa tako i izbor odgovarajućeg algoritma učenja. Iako je razvijen veliki broj raznih algoritama učenja neuronskih mreža, moguće ih je po načinu učenja podijeliti na algoritme učenja temeljene na pogrešci, algoritme učenja temeljene na izlazu mreže i algoritme učenja s ojačanjem.

Algoritmi učenja temeljeni na pogrešci često se nazivaju i algoritmi s "učiteljem" jer zahtijevaju vanjski referentni signal (učitelj) s kojim uspoređuju dobiveni odziv neuronske mreže generirajući signal pogreške. Na temelju signala pogreške algoritam učenja mijenja sinaptičke težinske koeficijente neuronske mreže s ciljem poboljšanja njezina vladanja, to jest smanjenja pogreške. Prema tome, ovi se algoritmi mogu primijeniti samo ako je unaprijed poznato željeno vladanje neuronske mreže, to jest podaci na osnovi kojih se mreža uči moraju sadržavati parove vrijednosti ulazno-izlaznih signala.



Slika 2.3. Shematski prikaz učenja neuronske mreže primjenom algoritma temeljenog na pogrešci.

Algoritmi učenja temeljeni na izlazu mreže nazivaju se i algoritmi bez "učitelja" jer ne zahtijevaju vanjski referentni signal. Podaci na osnovi kojih mreža uči sadrže samo vrijednosti ulaznih signala u mrežu.

Algoritmi učenja s ojačanjem zasnivaju se na tzv. signalu ojačanja koji daje kvalitativnu ocjenu vladanja neuronske mreže. Primjenjuje se u slučajevima kada vanjski referentni signal koji definira željeno vladanje neuronske mreže nije dostupan, ali je dostupan signal koji predstavlja kritičku ocjenu njezinog vladanja u smislu "dobro/loše".

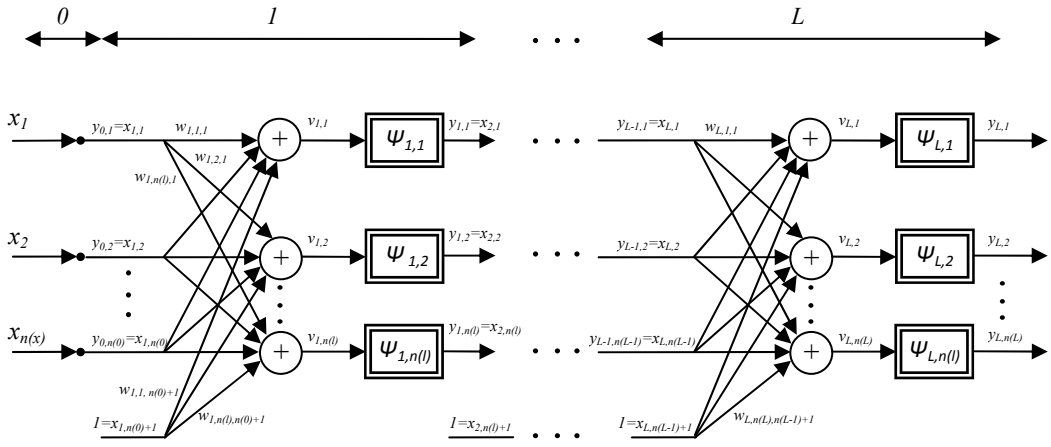
Gotovo svi algoritmi učenja neuronskih mreža pripadaju u jednu od ovih kategorija ili su njihova varijacija. Neke neuronske mreže imaju fiksne vrijednosti težinskih koeficijenata, a učenje se odvija mijenjanjem razine aktivacije neurona.

2.5) Statičke neuronske mreže

Statičke su neuronske mreže najčešće korištene neuronske mreže, osobito u primjenama kao što su identifikacija i upravljanje procesima, obradba signala te prepoznavanje oblika. Osnovni gradivni element statičkih neuronskih mreža jest neuron s bezmemorijskom aktivacijskom funkcijom. Iako i samo jedan neuron može aproksimirati jednostavnije nelinearne funkcije, općenito se aproksimacijska moć mreže povećava povećanjem broja neurona. Kod statičkih su neuronskih mreža neuroni organizirani na tzv. unaprijedni način (zato se ove mreže često nazivaju i unaprijedne neuronske mreže), što znači da svaki neuron može biti povezan s ulazima u mrežu i/ili s drugim neuronima, ali tako da se pri povezivanju ne formiraju povratne veze. Prema tome, statičke neuronske mreže ne sadrže nikakve dinamičke članove, a to ih čini strukturno stabilnima.

U nastavku se opisuju višeslojne perceptronske mreže (MLP mreže) koje su korištene pri izradi ovog rada.

Višeslojne perceptronske neuronske mreže izgrađene su od perceptron neurona organiziranih u serijski povezane slojeve (Sl. 2.4.). Slojevi se najčešće označuju brojevima 0, 1, 2, ... ,L. Nulti sloj samo prosljeđuje vektor ulaza u mrežu na ulaz prvog sloja. L-ti je sloj ujedno i izlazni sloj mreže, a slojevi između njih nazivaju se unutarnjim ili skrivenim slojevima jer ne daju i ne primaju vanjske signale.



Slika 2.4. Višeslojna perceptronska mreža (MLP mreža).

Svi neuroni u nekom sloju povezani su sa svim neuronima u dva susjedna sloja preko jednosmjernih, unaprijednih veza. Druge veze nisu dopuštene, to jest nema veza između neurona u istom sloju niti između neurona koji nisu u susjednim slojevima. Veze između neurona susjednih slojeva predstavljene su sinaptičkim težinskim koeficijentima koji djeluju kao pojačala signala na odgovarajućim vezama. Iznosi sinaptičkih težinskih koeficijenata određuju vladanje mreže,

odnosno njezina aproksimacija svojstva. Izračunavanje njihovih odgovarajućih iznosa ostvaruje se algoritmima učenja.

Iako nema teoretskoga ograničenja na broj unutarnjih slojeva neurona, uglavnom se koriste MLP mreže s jednim ili s dva unutarnja sloja, to jest dvoslojne ili troslojne MLP mreže. Teoretski je dokazano da MLP mreža s jednim unutarnjim slojem može aproksimirati proizvoljnu, kontinuiranu nelinearnu funkciju. Ovdje se u dokaz ovog teorema neće ulaziti. Opširnije se o aproksimaciji nelinearnih funkcija statičkim neuronskim mrežama može vidjeti u [1, str 20].

3. ALGORITMI UČENJA NEURONSKIH MREŽA

Neuronska je mreža u potpunosti određena tek kada je uz njezinu strukturu (određenu brojem, tipom i organizacijom neurona u mreži) definiran i algoritam učenja. Algoritam učenja podešava parametre mreže s ciljem postizanja njezinog željenog vladanja. U identifikaciji i upravljanju nelinearnim dinamičkim procesima najčešće je poznato željeno vladanje neuronske mreže pa se za njezino učenje primjenjuju algoritmi temeljeni na pogrešci. Zbog toga se u ovom poglavlju obrađuje samo ta grupa algoritama. Kao mjera pogreške koristi se neka kriterijska funkcija (kriterij kakvoće) $\Im(\Theta)$, koja može biti bilo koja pozitivna skalarna funkcija ovisna o parametrima neuronske mreže Θ. Algoritam učenja podešava parametre mreže dok kriterij kakvoće ne poprimi minimalni iznos odnosno iznos manji od unaprijed zadanog iznosa. Dakle, problem podešavanja parametara neuronske mreže svodi se na problem *nelinearnoga optimiranja* s kriterijskom funkcijom kao ciljnom funkcijom. Optimiranje se provodi na temelju skupa ulazno-izlaznih podataka sakupljenih eksperimentom, to jest funkciji koju neuronska mreža treba aproksimirati.

Osnovni problem koji kod primjene postupaka nelinearnog optimiranja za učenje neuronskih mreža treba razriješiti jest izračunavanje gradijenta kriterijske funkcije po parametrima mreže. Taj je problem dulje vrijeme usporavao istraživanja i primjenu neuronskih mreža, ali je uspješno razriješen primjenom *algoritma povratnoga prostiranja izlazne pogreške* kroz mrežu *(engl.* Back-Propagation algorithm, BP algoritam).

Dva su osnovna načina minimiziranja funkcije $\Im(\Theta)$ na kojima se temelje algoritmi učenja neuronskih mreža: nerekurzivni i rekurzivni.

3.1) Nerekurzivni algoritmi učenja neuronskih mreža

Prema nerekurzivnom se načinu funkcija $\Im(\Theta)$ minimizira tako da se promjene parametara mreže akumuliraju preko svih vektora mjernih podataka (preko bloka podataka) i tek nakon toga se stvarno promijene parametri mreže. Algoritmi učenja neuronskih mreža utemeljeni na ovome pristupu nazivaju se *nerekurzivnim algoritmima učenja*. U sklopu ovog rada nerekurzivne algoritme učenja neće se detaljnije opisivati. Međutim, potrebno je istaknuti da su pri identifikaciji modela procesa, parametri neuronske mreže naučeni primjenom modificiranog Levenberg-Marquardt-ovog algoritma [1 – str.44]. Isto tako, početni parametri neuronskog regulatora postavljeni su istim nerekurzivnim algoritmom učenja. Tako podešeni parametri neuronskog regulatora kasnije se adaptivnim algoritmom podešavaju tako da sustav što bolje prati referentni signal. Ovdje realizirani

adaptivni algoritam temelji se na rekurzivnom algoritmu učenja te se zbog toga detaljnije obrađuje takav postupak učenja.

3.2) Rekurzivni algoritmi učenja neuronskih mreža

Prema rekurzivnom se načinu učenja funkcija $\Im(\Theta)$ minimizira na temelju lokalne kriterijske funkcije $\Im_\nu(\Theta)$, to jest parametri mreže mijenjaju se nakon svakog vektora mjernih podataka. Algoritmi učenja neuronskih mreža utemeljeni na ovom pristupu nazivaju se *rekurzivnim algoritmima učenja (engl.* recursive, on-line, adaptive or data leaming algorithms).

Ako su svi mjerni podaci funkcije koja se aproksimira na raspolaganju prije početka postupka učenja mreže, za njezino se učenje mogu primijeniti i rekurzivni i nerekurzivni algoritmi. Međutim, ako svi uzorci nisu dostupni prije početka postupka učenja već se prikupljaju tijekom učenja, moguće je primijeniti samo rekurzivne algoritme učenja. U rješavanju problematike kojom se bavi ovaj rad primijenjen je rekurzivan algoritam učenja iz razloga što se parametri neuronskog regulatora podešavaju on-line, odnosno sa svakim novim uzorkom podataka regulator se adaptira trenutnim zahtjevima (promjenama) sustava.

Osnovni indeks koji se koristi pri opisu algoritma učenja jest indeks iteracije (korak izvođenja) algoritma *k.* U slučaju učenja mreže na temelju podataka koji u vremenskom slijedu pristižu tijekom procesa učenja, indeks iteracije algoritma *k* označava diskretno vrijeme.

Ciljna funkcija koju algoritmi učenja neuronskih mreža trebaju minimizirati jest kriterijska funkcija $\Im(\Theta)$. Algoritmi učenja minimiziraju funkciju $\Im(\Theta)$ podešavanjem iznosa parametara mreže Θ. Najčešće korišteni algoritmi učenja neuronskih mreža zasnivaju se na iterativnom postupku:

$$\Theta(k+1)=\Theta(k)+\Delta\Theta(k)=\Theta(k)+\alpha(k)s_d(k)\ , \qquad (3\text{-}1)$$

gdje je:

- $s_d(k)$ - smjer traženja minimuma u *k*-toj iteraciji (zasniva se na informaciji o $\Im(\Theta)$),
- $\alpha(k)$ - koeficijent učenja u *k*-toj iteraciji (određuje duljinu koraka u smjeru traženja optimuma).

U ovom radu pri podešavanju parametara neuronskog regulatora koristi će se *gradijentni postupak* minimizacije i to *postupak najbržeg spusta* te se u nastavku opisuje navedeni postupak.

Za postupak najbržeg spusta smjer traženja računa se na sljedeći način:

$$s_d(k) = -\nabla\Im(\Theta(k)) \,,$$

gdje je:

$\nabla\Im(\Theta)$ - gradijentni vektor kriterijske funkcije:

$$\nabla\Im(\Theta) = \left[\frac{\partial\Im(\Theta)}{\partial\Theta_1} \quad \frac{\partial\Im(\Theta)}{\partial\Theta_2} \quad \cdots \quad \frac{\partial\Im(\Theta)}{\partial\Theta_{n(\Theta)}} \right]^T .$$

Gradijent $\nabla\Im(\Theta)$ u bilo kojoj točki određenoj vrijednostima parametara mreže Θ jest vektor u smjeru njezinoga najvećeg lokalnog povećanja. Prema tome, najveće smanjenje kriterijske funkcije može se očekivati u smjeru negativnoga gradijenta (smjer najbržega spusta) kriterijske funkcije, pa izraz (3-1) poprima oblik:

$$\Theta(k+1) = \Theta(k) - \alpha(k)\nabla\Im(\Theta(k)) \quad . \qquad (3\text{-}2)$$

Negativni gradijent kriterijske funkcije određuje smjer traženja minimuma, ali ne i iznos koraka traženja. Iznos koraka traženja određen je iznosom koeficijenta učenja *α(k),* pa se ovisno o načinu određivanja koeficijenta učenja dobivaju razni postupci minimizacije u smjeru najbržega spusta. Općenito se minimum kriterijske funkcije ne može postići u jednom koraku pa se jednadžba (3-2) mora izvoditi kroz više koraka dok se ne pronađe minimum.

Kod osnovnoga algoritma koeficijent učenja jest skalar nepromjenljiva iznosa. Budući da je to prvi razvijeni algoritam učenja neuronskih mreža zasnovan na algoritmu povratnoga prostiranja izlazne pogreške, često se i sam tako naziva. Da bi se osigurala konvergentnost algoritma (3-2), koeficijent učenja mora biti veći od nule. Njegov iznos mora biti pažljivo izabran: ako je premali, algoritam konvergira suviše sporo tako da u većini slučajeva ne uspijeva ispuniti kriterij u dopustivom broju iteracija. Povećanje koeficijenta učenja povećava brzinu konvergencije, ali može rezultirati pojavom parazitnih oscilacija u okolini minimuma kriterijske funkcije, a pretjerano veliki iznos može prouzročiti i divergentnost algoritma. Dodatni je nedostatak algoritma najbržega spusta velika vjerojatnost "zaglavljivanja" u lokalnome minimumu.

Međutim, osim navedenih nedostataka, algoritam najbržega spusta ima i značajnih prednosti u odnosu na ostale algoritme učenja. Jednostavan je i zahtijeva manje računske moći i memorijskog prostora od ostalih algoritama. Osim toga, njegova velika prednost sa stajališta primjene za učenje neuronskih mreža jest paralelna struktura (zasebna jednadžba za učenje svakog parametra mreže) što ga čini kompatibilnim paralelnoj strukturi neuronskih mreža.

U literaturi se može naći veliki broj postupaka poboljšanja osnovnoga algoritma koji s više ili manje uspjeha rješavaju njegove navedene nedostatke. Većina postupaka poboljšanja iskustvenog su karaktera. Neki su od njih računski i memorijski vrlo zahtjevni, tako da poništavaju glavnu prednost osnovnog algoritma. Svi se oni mogu predstaviti kao specijalni slučajevi sljedećega općeg oblika algoritma:

$$\Delta\Theta(k) = -\alpha(k)\nabla\Im(\Theta(k)) + \gamma_m(k)\Delta\Theta(k-1)\,, \qquad (3\text{-}3)$$

gdje je:

γ_m - momentni koeficijent ($\gamma_m > 0$).

Dodavanjem momentnog člana može se ubrzati konvergencija učenja uz istodobno prigušenje parazitnih oscilacija. Ako se minimizacija trenutno odvija po glatkom dijelu kriterijske funkcije, tada je njezin gradijent približno konstantnog iznosa ($\nabla\Im(\Theta) \approx konst.$), pa se izraz (3-3) može pisati na sljedeći način:

$$\Delta\Theta(k) \cong -\frac{\alpha(k)}{1-\gamma_m(k)}\nabla\Im(\Theta(k))\,. \qquad (3\text{-}4)$$

Prema tome, stvarni se koeficijent učenja povećao $1/(1-\gamma_m)$ puta. U slučaju pojave parazitnih oscilacija momentni član pomaže njihovom smirivanju. Osnovni je postupak da se oba koeficijenta drže konstantnima cijelo vrijeme postupka učenja, primjerice na iznosima $\alpha(k)$=0.001-0.01 i γ_m(k)=0.8-0.9 [1] što je u sklopu ovog rada i učinjeno.

U sklopu diplomskog zadatka adaptivni algoritam upravljanja potrebno je isprobati na procesu magnetske levitacije. U nastavku je dan opis procesa. Prije opisa adaptivnog algoritma podešavanja parametara neuronskog regulatora dani su postupci identifikacije modela procesa (u adaptivnom

upravljanju služi za izračunavanje gradijenta kriterija kakvoće) i inverznog modela procesa koji predstavlja početne vrijednosti parametara regulatora.

4. OPIS LABORATORIJSKOG POSTAVA

U ovom se poglavlju opisuje laboratorijski postav na kojem je predviđeno provođenje laboratorijskih pokusa. Radi se o procesu u kojem se manifestiraju pojave magnetske levitacije.

Slika 4.1. Laboratorijski postav procesa magnetske levitacije.

4.1) Laboratorijski proces magnetske levitacije

Skica laboratorijskog procesa magnetske levitacije prikazana je na slici 4.2.

Slika 4.2. Skica laboratorijskog procesa magnetske levitacije.

Prikazan je pogled s bočne (lijeva skica) i prednje strane (desna skica). Gornja i donja zavojnica protjecane strujom proizvode magnetska polja kojima privlače ili odbijaju permanentne magnete. Jedan ili dva permanentna magneta kreću se duž staklene cijevi. Ukoliko narinemo struju kroz donju zavojnicu (upravljačkim naponom u_1) u smjeru da proizvedeno magnetsko polje odbija donji magnet, isti će levitirati na visini y_1 zbog proizvedene magnetske sile. Jakost polja magneta izrazito je velika. Magneti su napravljeni od rijetkih Zemljanih materijala (NeBFe) te su dizajnirani tako da izazovu značajne pomake magneta kako bi dobro demonstrirali principe levitacije.

Dva laserska senzora mjere udaljenost donjeg magneta od donje zavojnice y_1 i gornjeg od gornje y_2. Donji se senzor koristi za mjerenje pozicije magneta od donje zavojnice (do otprilike 8 cm visine), dok gornji senzor mjeri udaljenost magneta od gornje zavojnice (također mu je opseg do otprilike 8 cm).

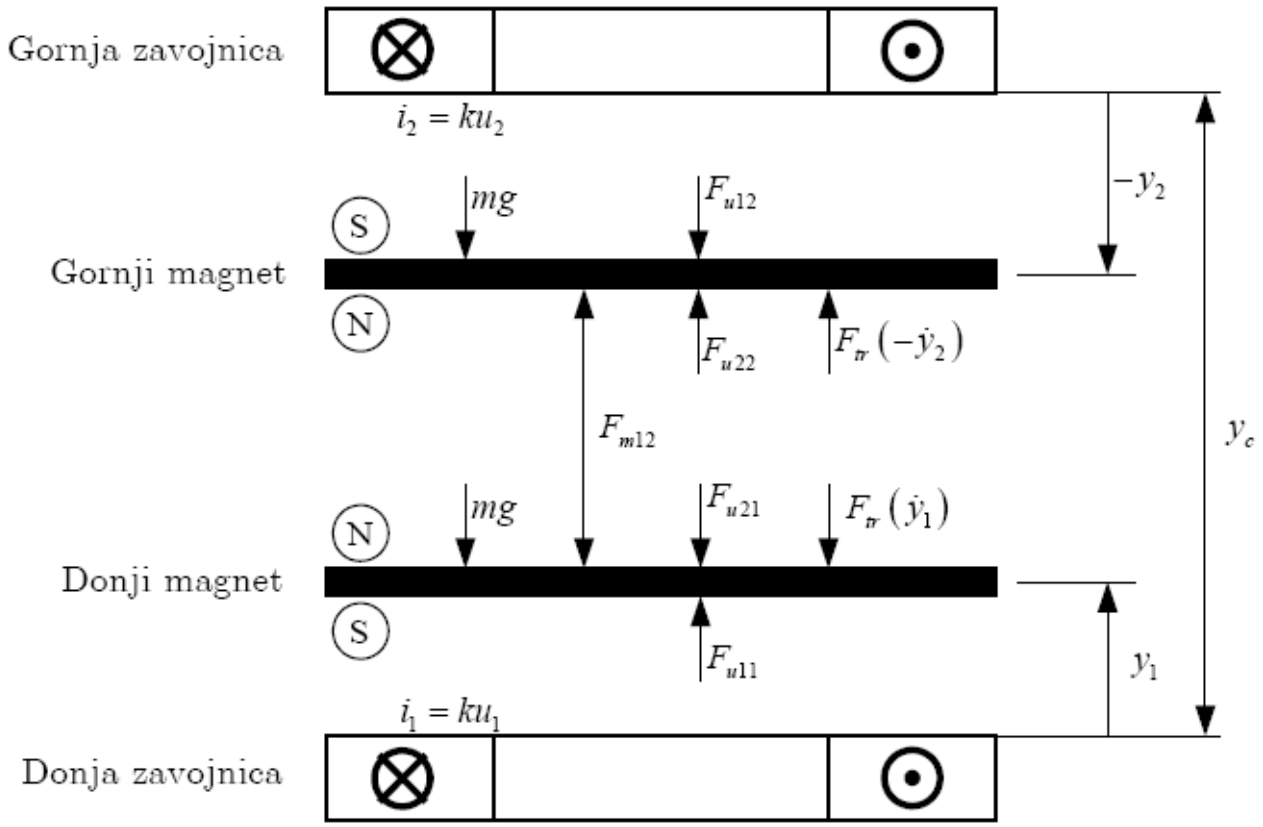
Osim postolja s magnetima, proces posjeduje i upravljačku kutiju. Ona je spojena s aktuatorima i senzorima preko konektora, te s upravljačkim računalom preko A/D-D/A akvizicijske kartice PCI1711 tvrtke Advantech. Eksperimenti će se u stvarnom vremenu odvijati korištenjem MATLAB-ovog okruženja Real-Time Workshop koje komunicira s akvizicijskom karticom.

Proces prima upravljačke napone u rasponu ±5 V. Pritom, za negativne napone zavojnica privlači pločicu pa u stacionarnom stanju nema odmaka pločice.

4.2) Matematički model procesa magnetske levitacije

U svrhu provedbe simulacijskih eksperimenata korišten je matematički model procesa. U nastavku se daje uvid u fizikalna zbivanja kod procesa magnetske levitacije na temelju kojih je izrađen matematički model.

Na slici 4.3. dan je shematski prikaz procesa.



Slika 4.3. Fizikalni opis procesa.

Prema slici 4.3., vrijede sljedeće jednadžbe gibanja za donji i gornji magnet:

$$\begin{aligned} m\ddot{y}_1 + F_{m12} &= F_{u11} - F_{u21} - F_{tr}(\dot{y}_1) - mg \\ m\ddot{y}_2 - F_{m12} &= F_{u22} - F_{u12} - F_{tr}(-\dot{y}_2) - mg \end{aligned} \qquad (4\text{-}1)$$

gdje je:

y_1 - udaljenost donjeg magneta od donje zavojnice [cm];

y_2 - udaljenost gornjeg magneta od gornje zavojnice [cm];

F_{u11} - sila između donje zavojnice i donjeg magneta [N];

F_{u12} - sila između donje zavojnice i gornjeg magneta [N];

F_{u21} - sila između gornje zavojnice i donjeg magneta [N];

F_{u22} - sila između gornje zavojnice i gornjeg magneta [N];

F_{m12} - sila između dva magneta [N];

F_{tr} - funkcional trenja baziran na LuGre modelu (posjeduje unutarnju varijablu stanja) [N].

Izrazi za magnetske sile su sljedeći:

$$
\begin{aligned}
F_{u11} &= \frac{u_1}{a\left(y_1 + b\right)^N} \\
F_{u12} &= \frac{u_1}{a\left(y_c + y_2 + b\right)^N} \\
F_{u21} &= \frac{u_2}{a\left(y_c - y_1 + b\right)^N} \\
F_{u22} &= \frac{u_2}{a\left(-y_2 + b\right)^N} \\
F_{m12} &= \frac{c}{\left(y_{12} + d\right)^N}
\end{aligned}
\qquad (4\text{-}2)
$$

gdje je:

$y_{12} = y_c + y_2 - y_1$ - udaljenost dva magneta [cm];

y_c - udaljenost gornje i donje zavojnice [cm].

Konstante *a, b, c, d* i *N* dobivaju se eksperimentalnim metodama. U detalje izvedbe LuGre modela sile trenja neće se ulaziti, upotrijebljen je radi numerički pouzdane i brze simulacije sustava s jedne strane, te dobrog opisa pojava trenja.

Za potrebe laboratorijskih pokusa, proces će se promatrati u SISO (Single Input Single Output) konfiguraciji, tj. jedini ulaz u proces je upravljački napon u_1 za struju donje zavojnice, a jedini izlaz bit će udaljenost između donjeg magneta i donje zavojnice y_1. U svrhu ove modifikacije procesa gornji magnet je otklonjen. Jednadžba gibanja za donji magnet, uz neaktivnu gornju zavojnicu i neprisustvo gornjeg magneta, glasi:

$$m\ddot{y}_1 = F_{u11} - F_{tr}\left(\dot{y}_1\right) - mg \ . \qquad (4\text{-}3)$$

Poremećaj na proces ostvarit će se dodavanjem dodatne mase na donji magnet, kada vrijede pune jednadžbe (4-1) uz $F_{u21}=F_{u22}=0$ radi neaktivne gornje zavojnice.

Osim nelinearnosti aktuatora, i senzorska je karakteristika također nelinearna, no ona se lako premosti invertiranjem nelinearnosti senzora. U shemi za izvođenje u stvarnom vremenu to je i učinjeno, dok ista nelinearnost uopće nije niti uključena u simulacijski model.

5. IDENTIFIKACIJA PROCESA PRIMJENOM NEURONSKIH MREŽA

Ovo poglavlje obrađuje identifikaciju NARX modela procesa na procesu magnetske levitacije primjenom neuronske mreže. Provodi se off-line postupak identifikacije nelinearnog modela procesa na laboratorijskom postavu iz prethodnog poglavlja. Postupak identifikacije procesa sastoji se od sljedećih koraka:

1) prikupljanje ulazno-izlaznih podataka, tj. mjernih vrijednosti ulaznih i izlaznih signala procesa;
2) izbor strukture modela procesa;
3) izbor kriterija kakvoće modela procesa;
4) estimacija parametara modela procesa;
5) izbor optimalne dimenzije modela i njegovo vrednovanje.

Prikupljanje mjernih vrijednosti ulaznih i izlaznih signala procesa provodi se posebno pripremljenim eksperimentom, koji treba provesti tako da se postigne najveća moguća informativnost mjernih podataka poštujući pri tome fizikalna ograničenja procesa. Kao pobudni signal za nelinearne procese najčešće se koristi pseudoslučajni signal ograničenog frekvencijskog područja (engl. Band Limited White Noise Signal, BLWNS).

Najčešće korištena struktura nelinearnog modela procesa prikladna za primjenu neuronskih mreža jest NARX struktura modela. Najopćenitiji se NARX model dobije primjenom nelinearne regresije nad prošlim mjernim uzorcima izlaznih i ulaznih signala procesa:

$$y(k) = f\left(y^{k-1}, u^{k-1}\right) + \xi(k) \ . \tag{5-1}$$

Prediktor za model (5-1) glasi:

$$\hat{y}(k) = f_N\left(\varphi(k), \Theta\right) = f_N\left(\left[\varphi_y(k), \varphi_u(k)\right], \Theta\right) \ , \tag{5-2}$$

gdje je:

$$\varphi(k) = \left[\varphi_y(k), \varphi_u(k)\right] = \left[y(k-1), \cdots, y(k-na), u(k-1), \cdots, u(k-nb)\right]^T \tag{5-3}$$

- regresijski vektor dimenzije $n(\varphi) = \sum_{i=1}^{n(y)} na(y_i) + \sum_{j=1}^{n(u)} nb(u_j)$,

$na = \left[na\left(y_1\right), \cdots, na\left(y_{n(y)}\right) \right]$ - vektor broja korištenih prošlih vrijednosti izlaznih signala procesa u regresijskom vektoru.

Kao parametrirana nelinearna funkcija $f_N(\varphi(k),\Theta)$ koristi se neuronska mreža koja ima svojstva univerzalnog aproksimatora.

Kao mjera kakvoće modela procesa uvodi se kriterijska funkcija:

$$\Im(\Theta) = \frac{1}{2}\sum_{v=1}^{N} e^T(v,\Theta)\cdot e(v,\Theta) = \frac{1}{2}\sum_{v=1}^{N}\sum_{i=1}^{n(L)} e_i^2(v,\Theta) = \frac{1}{2} e^{*T}(\Theta)\cdot e^*(\Theta) \quad , \tag{5-4}$$

gdje je:

$e(v,\Theta)$ - vektor predikcijske pogreške za v-ti vektor mjernih podataka;

$e^*(\Theta)$ - ukupna predikcijska pogreška na čitavom skupu mjernih podataka, vektor dimenzije $N_e = n(y)\cdot N$ koji se dobije slaganjem vektorâ $e(v,\Theta)$:

$$e^*(\Theta) = \left[e^T(1,\Theta), \cdots, e^T(N,\Theta) \right] = \left[e_1(1,\Theta), \cdots, e_{n(y)}(1,\Theta), \cdots, e_1(N,\Theta), \cdots, e_{n(y)}(N,\Theta) \right] .$$

Zadaća je postupaka estimacije parametara modela pronalaženje optimalnih vrijednosti parametara modela Θ^* uz koje je predikcijska pogreška najmanja. S obzirom da kriterij kakvoće $\Im(\Theta)$ predstavlja mjeru iznosa predikcijske pogreške, problem estimacije parametara svodi se na traženje njegova minimuma. Dakle, optimalne se vrijednosti parametara modela procesa Θ^* mogu definirati kao argument koji minimizira kriterij kakvoće:

$$\Theta^* = arg\,min\,\Im(\Theta) \, . \tag{5-5}$$

Za određivanje minimuma funkcije $\Im(\Theta)$ najčešće se koriste *gradijentni postupci nelinearnoga optimiranja.*

Jedan od najtežih problema kod identifikacije procesa primjenom neuronskih mreža jest izbor optimalne dimenzije (tj. optimalnog broja parametara) modela, odnosno mreže. Inherentno svojstvo neuronskih mreža jest veliki broj parametara, koji u najvećoj mjeri i doprinose njihovim aproksimacijskim svojstvima. Neuronska mreža s premalim brojem parametara ne može zadovoljavajuće aproksimirati nelinearnost procesa. Međutim, preveliki broj parametra može u velikoj mjeri narušiti svojstvo poopćavanja modela (neuronske mreže). Kako bi se poboljšalo svojstvo poopćavanja modela, koriste se razni postupci regularizacije. Najčešće se primjenjuju varijante poznate kao *eksplicitna* i *implicitna* regularizacija.

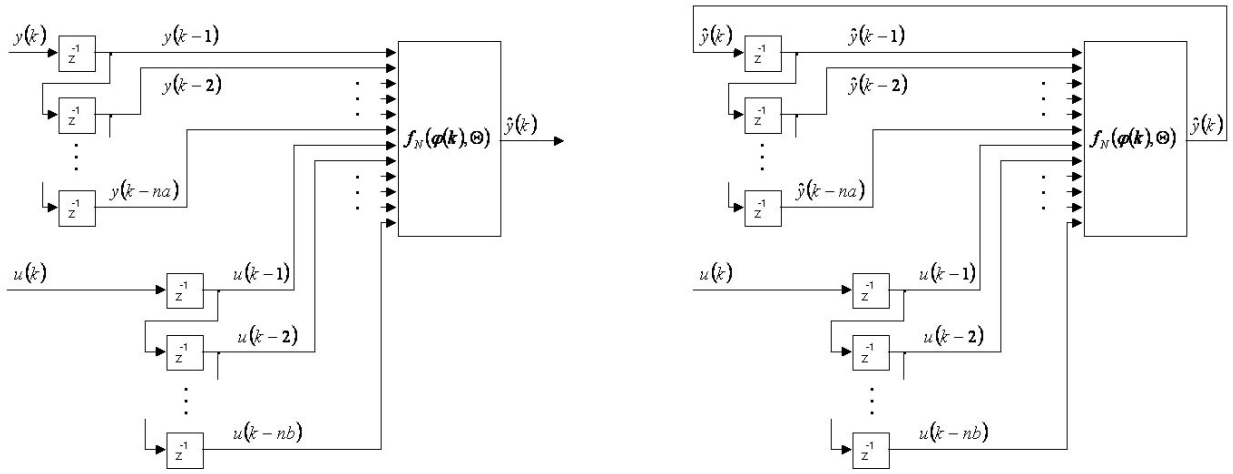
Završni dio postupka identifikacije jest vrednovanje estimiranoga modela procesa. Ako model zadovolji test vrednovanja, prihvaća ga se kao valjani model, a ako ne zadovolji postupak,

identifikacija se ponavlja. Najčešće se koristi modificirani klasični korelacijski postupak vrednovanja modela procesa.

Nakon identifikacije i validacije neuronskog modela procesa provodi se testiranje modela u dvije različite strukture.

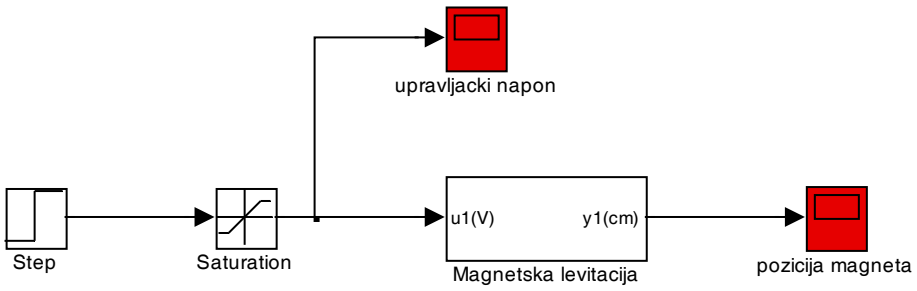
1) NARX struktura – sastoji se od neuronske mreže kojoj se izvana na ulaz mreže dovode vrijednosti izlaznih signala procesa *y(k)* iz prethodnih koraka uzorkovanja. Ovakva struktura naziva se NARX model (Nonlinear AutoRegressive model with eXogenous inputs). NARX model je model bez povratnih veza te je on strukturno stabilan model jer njegovi regresori ne ovise o parametrima modela. Ovakvi modeli opisuju nelinearni proces, uključivo i dodatnu smetnju i to na način da smetnju modeliraju kao dio dinamike procesa. Zbog toga što nema zasebnog modela smetnje, potrebna dimenzija regresijskog vektora može biti znatno veća nego što ju zahtjeva sama dinamika procesa.

2) NOE struktura (Nonlinear Output Error model) – NOE modeli se dobiju iz NARX modela zamjenom vektora izlaznih signala procesa *y(k)* vektorom izlaznih signala modela $\hat{y}(k)$. Zbog toga NOE modeli modeliraju samo dinamiku procesa, a ne i smetnju.

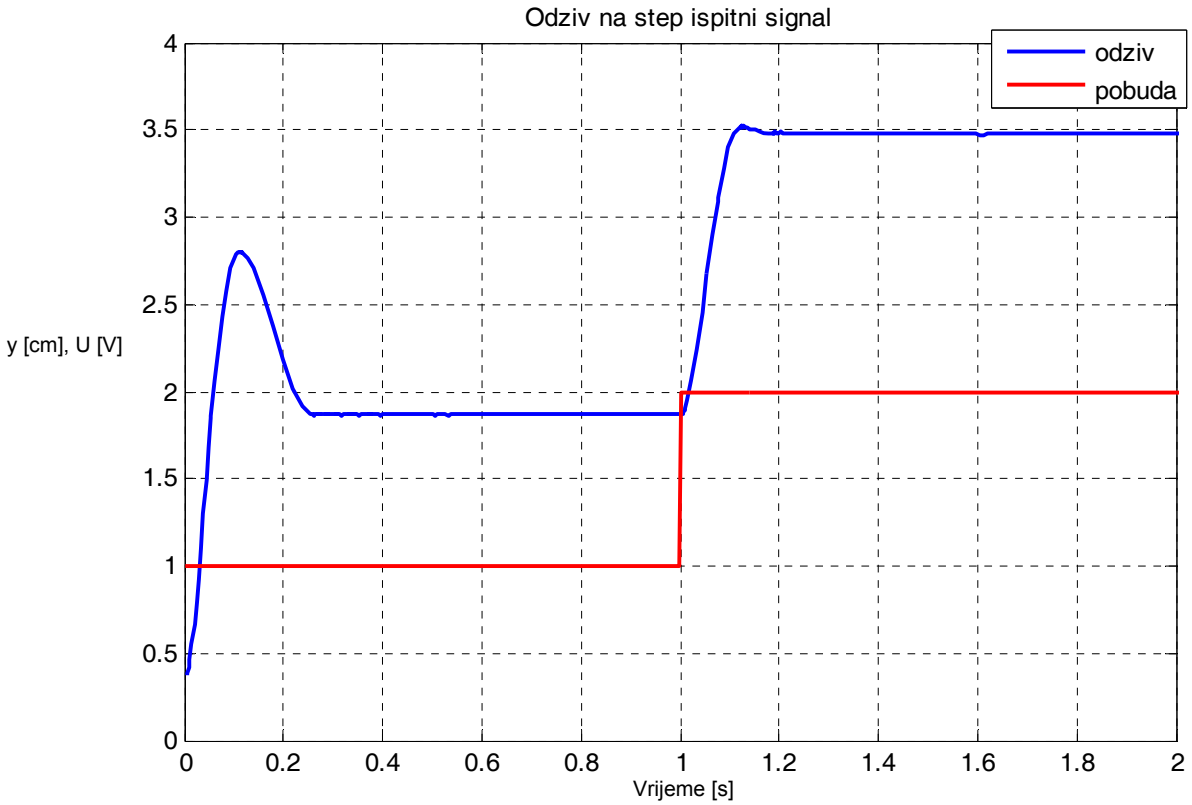


Slika 5.1. Blokovska shema NARX modela procesa (lijevo) i NOE modela procesa (desno).

5.1) Određivanje vremena uzorkovanja T



Slika 5.2. simulink model za određivanje vremena uzorkovanja modela.



Slika 5.3. Odziv modela procesa na step ispitni signal.

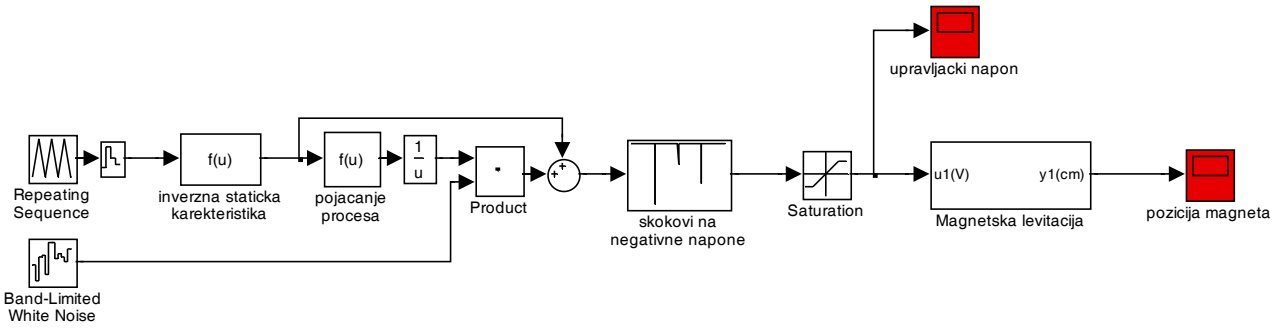
Vrijeme uzorkovanja odredimo kao vrijeme deset puta manje od vremena porasta (vremena potrebnog da se dosegne 63% vrijednosti novog stacionarnog stanja). Ovdje je potrebno pripaziti da ovu radnju obavimo negdje pri sredini radnog područja procesa. Iz tog razloga proces prvo dovedemo u radnu točku sa signalom upravljanja od u = $1V$ i nakon dostizanja stacionarnog stanja narinemo napon od 2V. Tijek provođenja ovog eksperimenta prikazan je grafom na slici 5.3..

Dakle, vrijeme porasta odredit ćemo kao vrijeme od $t = 1$ s do vremena kada pozicija magneta iznosi 63% od nove stacionarne vrijednosti ($y_{stac} \approx 3.5$ cm).

Na taj način dobiveno je vrijeme uzorkovanja $T = 0.0065$ s.

5.2) Prikupljanje ulazno-izlaznih podataka

Prikupljanje mjernih vrijednosti ulaznih i izlaznih signala procesa provodi se posebno pripremljenim eksperimentom, koji treba provesti tako da se postigne najveća moguća informativnost mjernih podataka poštujući pri tome fizikalna ograničenja procesa. Kao pobudni signal za nelinearne procese najčešće se koristi pseudoslučajni signal ograničenog frekvencijskog područja (Band Limited White Noise Signal, BLWNS). Na slici 5.4. prikazana je simulacijska shema koja će nam poslužiti za snimanje ulazno izlaznih signala procesa i učenje neuronske mreže.



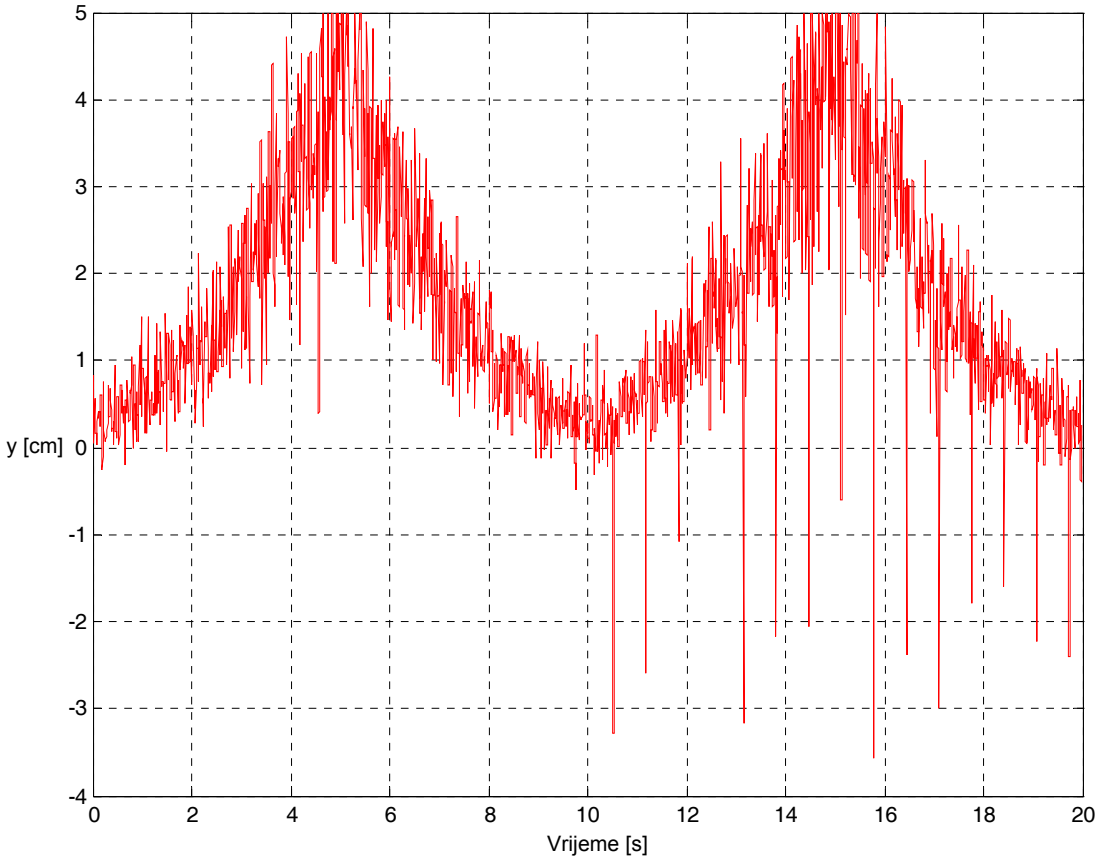
Slika 5.4. Model za prikupljanje identifikacijskih podataka.

Realizacijom inverzne funkcije statičke karakteristike procesa postiže se da su sve vrijednosti y kao radne točke jednako zastupljene. Pojačanje sustava (derivacija statičke karakteristike) također je nelinearno, te je zato umnožak "varijanca šuma BLWN signala * pojačanje procesa" prilagođen da ima sličan iznos. Time svaku radnu točku podjednako "razdrmamo".

U podsustavu *skokovi na negativne napone* realizirao sam skokove upravljačkog signala na negativne vrijednosti. Negativnim iznosom upravljačkog signala zavojnica privlači magnet te se time potpomaže gravitacijskoj sili da magnet brže pada. Budući da djelovanje negativnih napona u slučaju kada je pločica magneta na minimalnoj poziciji nema smisla tako i nema potrebe za dugotrajnim izlaganjem procesa takvim signalima. Zbog toga su u upravljačku sekvencu uključeni kratkotrajni skokovi na negativne vrijednosti. Ovakvim signalom za učenje nastoji se neuronski

model procesa pripremiti za situacije kada nastane potreba za naglim smanjenjem visine levitiranja magneta.

Prikaz takvog upravljačkog signala dan je na slici 5.5.



Slika 5.5. Prikaz upravljačkog signala (signala za učenje neuronske mreže).

5.3) Validacija naučenog modela nakon učenja

Za validaciju modela koristit će se simulacijska shema prikazana kao na slici 5.4. samo s drugim „seedom“ u BLWNS signalu.

Pritom se model proces smatra valjanim ako su ispunjeni sljedeći uvjeti:

$$
\begin{aligned}
&R_{ee}(\tau)=\delta(\tau)\\
&R_{eu}(\tau)=0, \qquad \forall\tau\\
&R_{e(eu)}(\tau)=0, \qquad \tau\geq 1\\
&R_{u^2 e}(\tau)=0, \qquad \forall\tau\\
&R_{u^2 e^2}(\tau)=0, \qquad \forall\tau
\end{aligned}
\tag{5-6}
$$

Naravno, ti uvjeti ne mogu biti egzaktno ispunjeni te se model može smatrati ispravnim ako se iznosi korelacijskih funkcija nalaze unutar određenih područja povjerenja.

Korelacijske funkcije ćemo normirati kako bi se vrijednosti koje one poprimaju ograničile na područje [-1, 1].

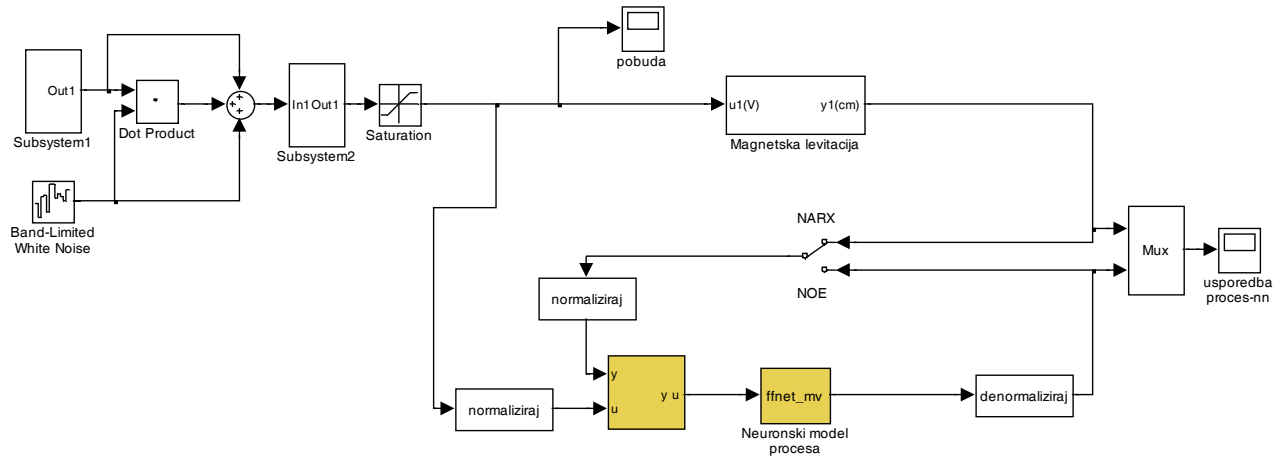
- normirana autokorelacijska funkcija:

$$R_{xx_n}(\tau) = \frac{R_{xx}(\tau)}{R_{xx}(0)} \qquad (5\text{-}7)$$

- normirana međukorelacijska funkcija

$$R_{xy_n}(\tau) = \frac{R_{xy}(\tau)}{\sqrt{R_{xx}(0) \cdot R_{xy}(0)}} \qquad (5\text{-}8)$$

Dobiveni model procesa testirat ćemo pomoću sheme prikazane na slici 5.6., gdje ćemo ispitati ponašanje NARX i NOE strukture modela.



Slika 5.6. Simulink model za ispitivanje NARX/NOE strukture modela.

Koristimo dvoslojnu MLP neuronsku mrežu s *tansig* aktivacijskim funkcijama u skrivenom sloju i *purelin* aktivacijskom funkcijom u izlaznom sloju uz početnu dimenziju regresijskog vektora *na=nb=3* i *20* neurona u skrivenom sloju mreže.

5.4) Metode regularizacije

U metodi bez regularizacije svaki se parametar neuronske mreže proračunava tokom izvođenja pokusa i postavlja na neku vrijednost. Ako je broj izabranih parametara prevelik, može doći do toga da neuronska mreža počne opisivati šum. Drugim riječima, može doći do predetaljnog opisivanja ponašanja procesa i na taj način beskorisnog podešavanja parametara sve dok kriterijska funkcija ne ispuni zadani kriterij.

Metoda identifikacije uz eksplicitnu regularizaciju pogodna je za on-line učenje neuronske mreže jer se uvodi dodatni faktor kazne k_r. Veća važnost daje se većim svojstvenim vrijednostima Hessian matrice, a zanemaruju se manje svojstvene vrijednosti. Drugim riječima, suvišni neuroni se postavljaju u nulu i oni praktički nemaju nikakav utjecaj na neuronsku mrežu. Faktor kazne je ujedno i negativna strana ove metode jer predstavlja još jedan parametar više za podešavanje.

Identifikacija uz implicitnu regularizaciju također je pogodna za on-line učenje jer se učenje zaustavlja u trenutku kad je vrijednost kriterijske funkcije u trenutnom koraku manja od kriterijske funkcije u prethodnom koraku na identifikacijskim podatcima, a kriterijska funkcija u trenutnom koraku postane veća od kriterijske funkcije u prethodnom koraku nad validacijskim podatcima. Tako se zapravo ne stiže u globalni minimum kriterijske funkcije već malo dalje. Na taj način postignuto je da se ne modelira šum jer prostor šuma unosi lokalne minimume. Kako algoritam učenja ne bi zaglavio u nekom od tih lokalnim minimuma, učenje se zaustavlja nešto prije.

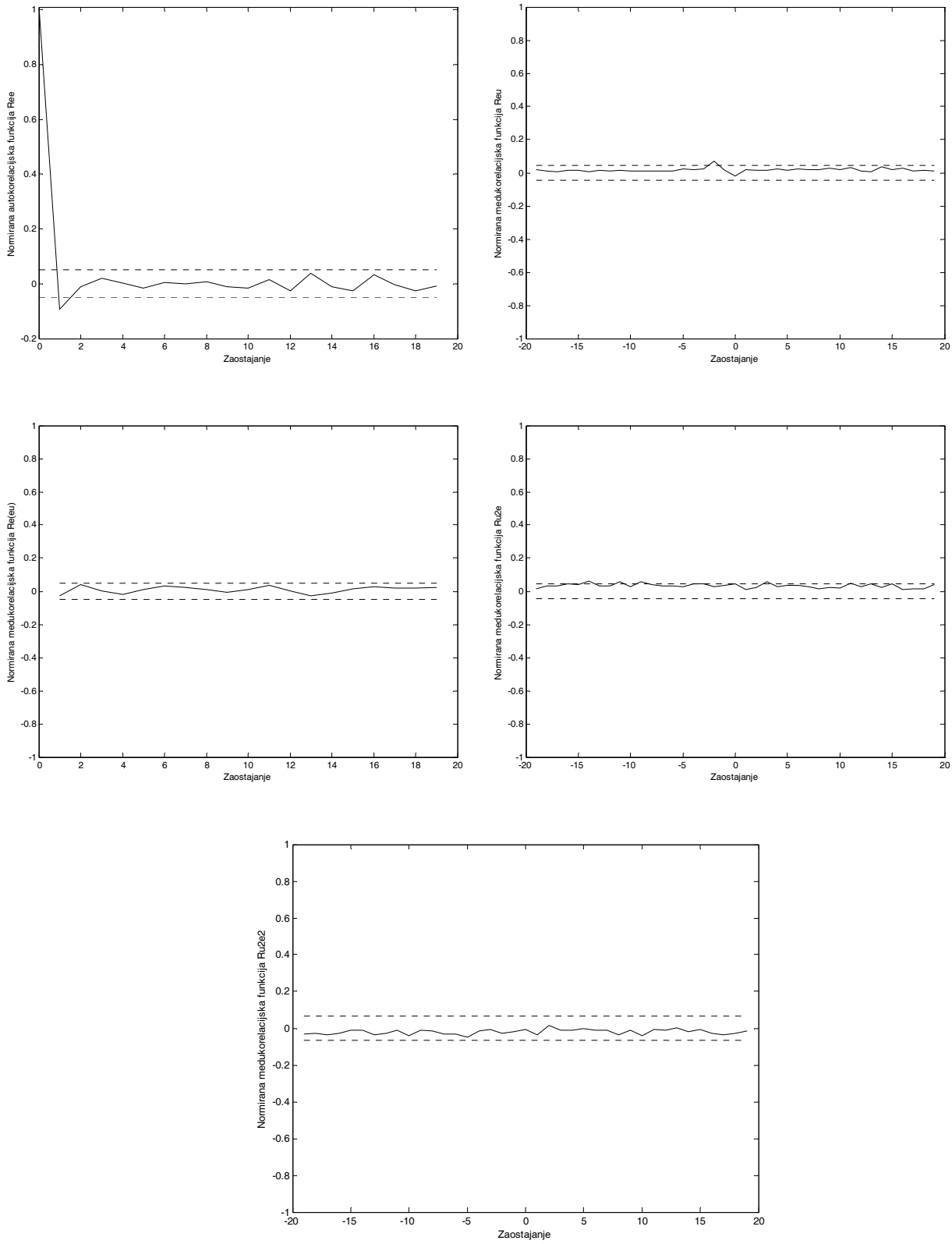
Simulacijskim eksperimentima isprobane su sve navedene metode regularizacije.

Nakon obrade rezultata donosim zaključak kako je identifikacija uz eksplicitnu regularizaciju dovela do kvalitetnijeg modela procesa od identifikacije bez regularizacije ili pak uz implicitnu regularizaciju. U nastavku su prikazani rezultati identifikacije eksplicitnom regularizaciju uz faktor kazne $k_r = 10^{-4}$. Model procesa identificiran na ovaj način korišten je i u nastavku ovog rada.

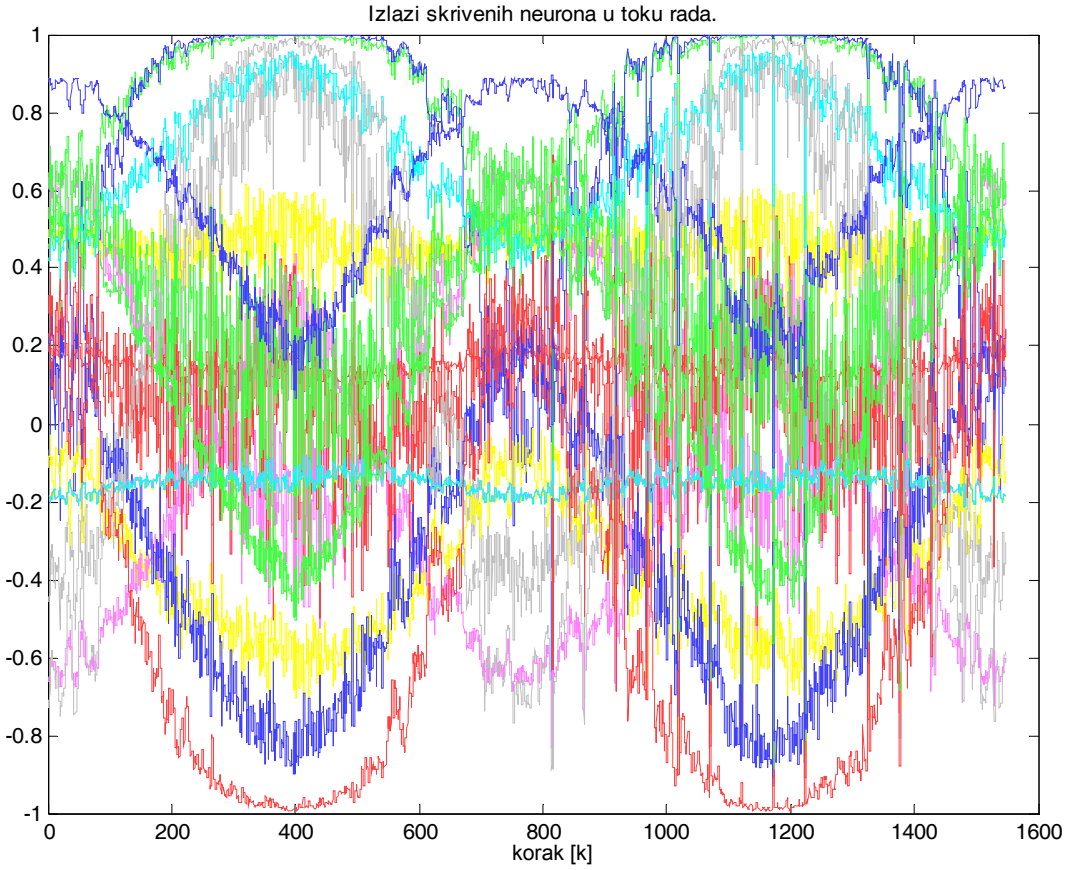
Validacija postupka identifikacije dana je korelacijskim funkcijama na slici 5.7. gdje su sve funkcije unutar granica povjerenja te je neuronski model valjan.

Odzivi NARX i NOE struktura modela prikazani su slikama 5.9..

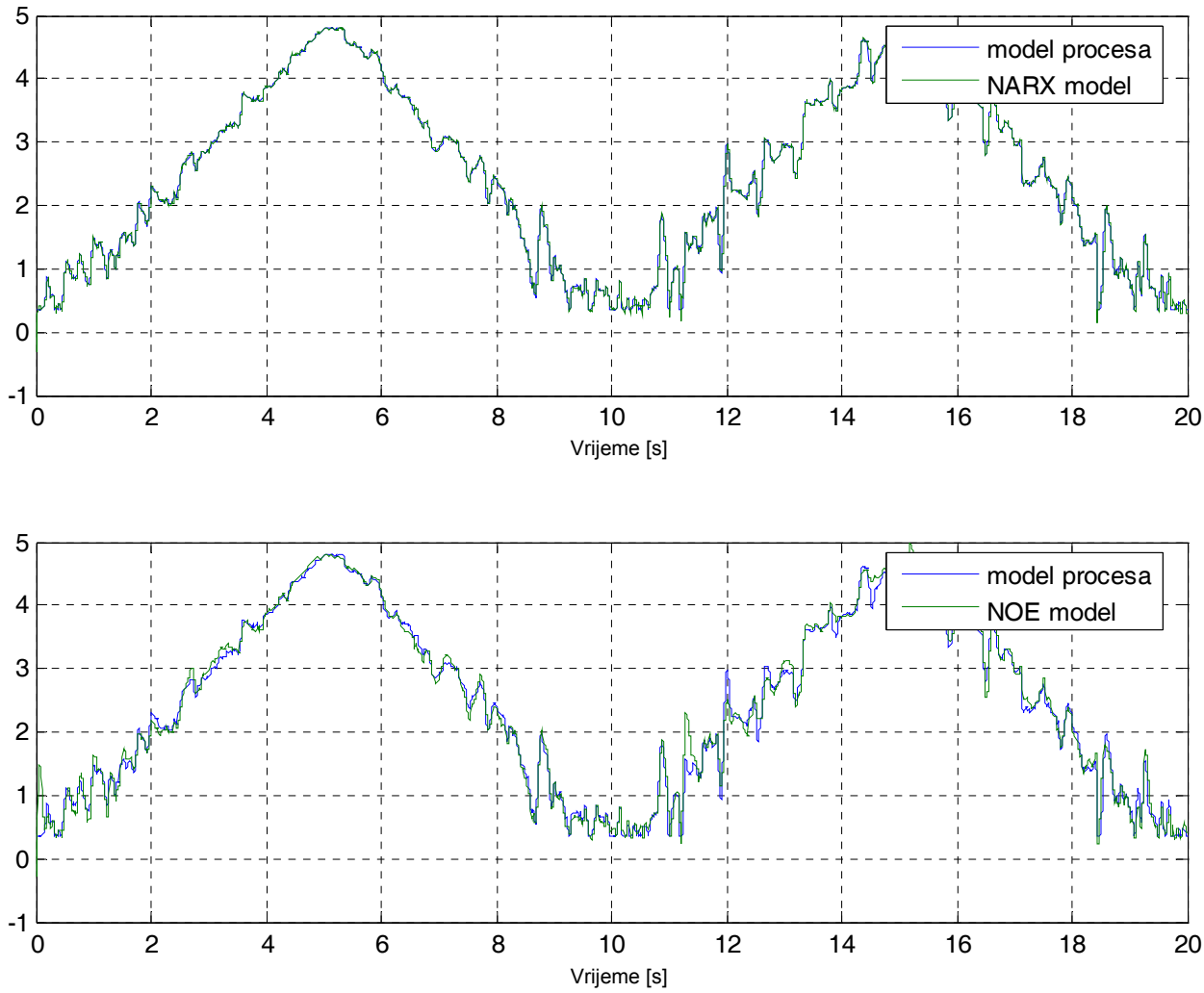
korelacijske funkcije:



Slika 5.7. korelacijske funkcije pri identifikaciji uz eksplicitnu regularizaciju.



Slika 5.8. izlazi neurona u skrivenom sloju pri identifikaciji uz eksplicitnu regularizaciju.



Slika 5.9. Usporedba matematičkog modela procesa i NARX/NOE modela identificiranog primjenom neuronske mreže uz eksplicitnu regularizaciju na validacijskim podacima.

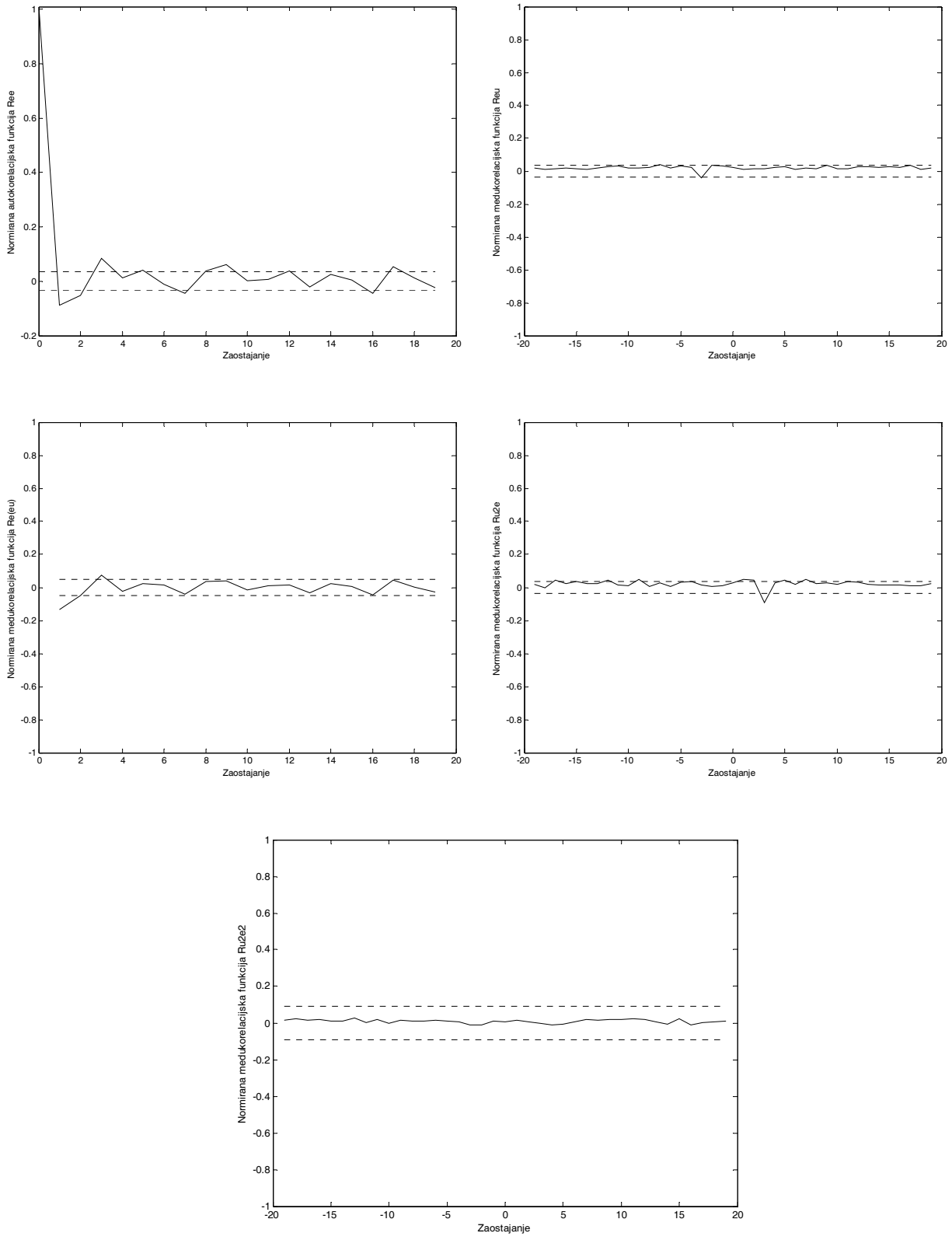
6. POKUS NA LABORATORIJSKOM POSTAVU

U radu na pokusu korišteni su sljedeći parametri:

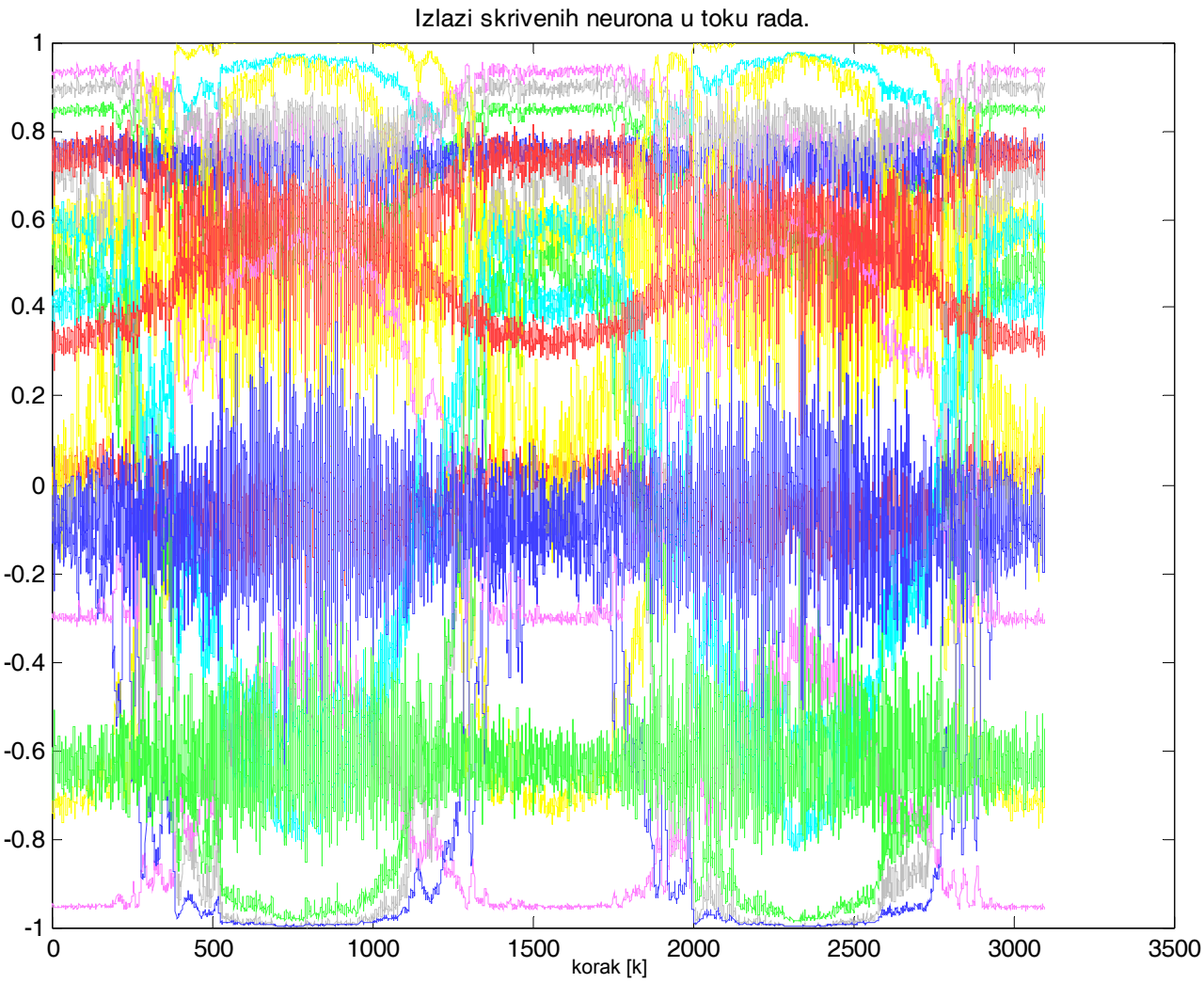
T = 0.0065 s
nu=3 i ny=3
skriveni sloj S1=20 neurona
izlazni sloj S2=1 neurona

Učenje je provedeno eksplicitnom regularizacijom te su i izlazi neurona skrivenog sloja u skladu sa onim dobivenim simulacijom. Validacija korelacijskim postupkom dana je na slici 6.1. i ona daje zadovoljavajuće rezultate jer su iznosi korelacijskih funkcija uglavnom unutar granica povjerenja.

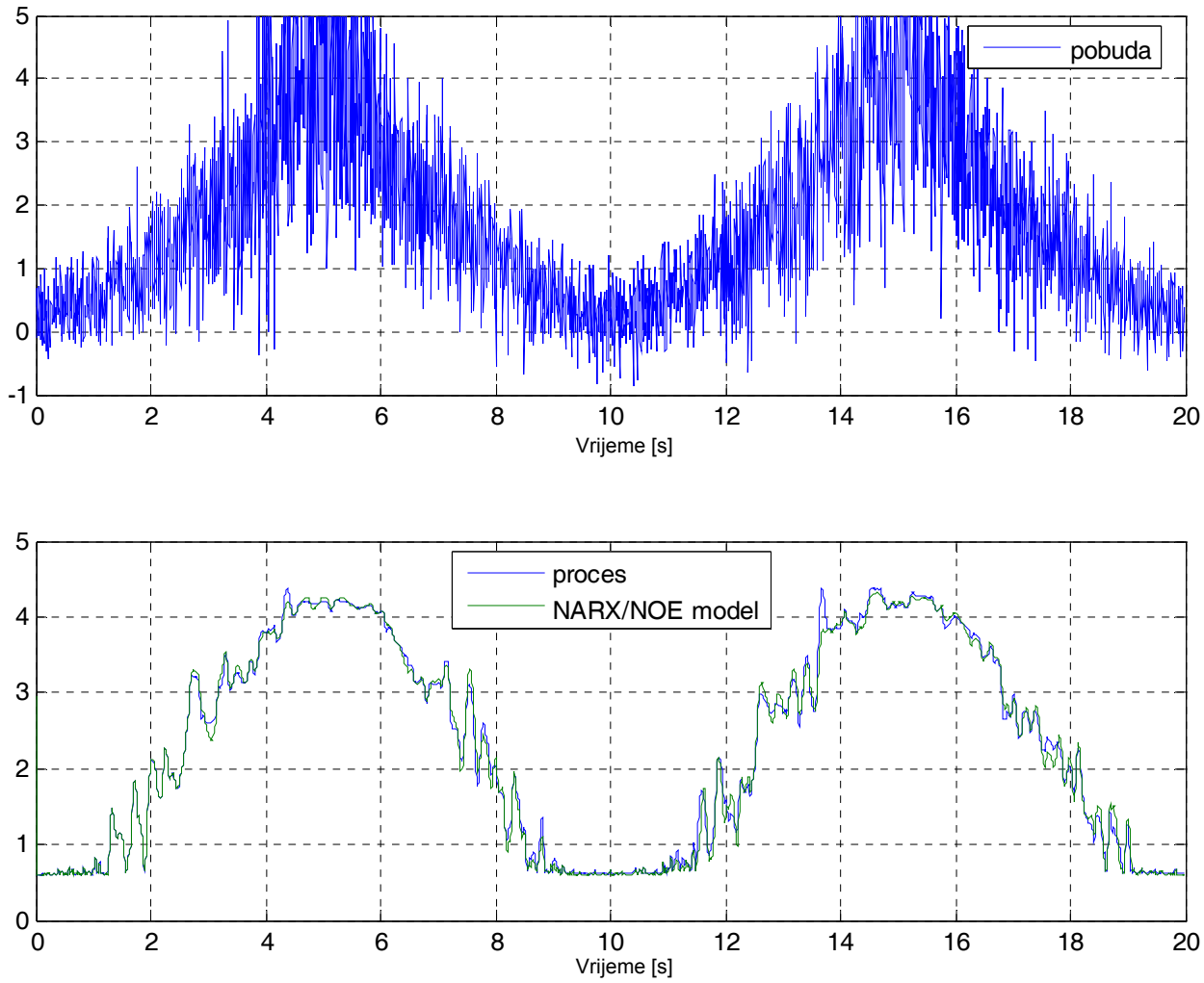
korelacijske funkcije:



Slika 6.1. korelacijske funkcije pri identifikaciji uz eksplicitnu regularizaciju.



Slika 6.2. izlaz neurona u skrivenom sloju.



Slika 6.3. Usporedba odziva procesa i NARX/NOE modela procesa identificiranog primjenom neuronske mreže uz eksplicitnu regularizaciju.

Usporedba odziva procesa i odziva NARX/NOE modela procesa identificiranog primjenom neuronske mreže. U vremenu od [0, 3] sekunde mreža je spojena u NARX konfiguraciji, a nakon toga u NOE konfiguraciji.

NARX model daje uvijek bolje rezultate od NOE modela. No NOE model je puno praktičniji prvenstveno iz razloga što u regresoru koristi svoje izlaze (a ne izlaze procesa).

NARX model je model bez povratnih veza (njegovi regresori ne ovise o parametrima modela) i zbog toga je on strukturno stabilan i numerički postupci estimacije njegovih parametara su puno jednostavniji nego kod modela s povratnim djelovanjem. Nedostaci NARX modela su što se

smetnja modelira kao dio dinamike procesa pa nema zasebnog modela smetnje zbog čega potrebna dimenzija regresora može biti puno veća nego što ju dinamika procesa zahtijeva.

NOE modeli su modeli s povratnim djelovanjem. Takav model procesa modelira samo dinamiku procesa, ne i smetnju. U slučaju NOE modela vidljiva je greška u odzivu jer NOE model vrši simulaciju u otvorenom krugu, bez informacija o stvarnom izlazu procesa, dok NARX vjerno slijedi sustav u sva tri slučaja.

7. STRUKTURE UPRAVLJANJA ZASNOVANE NA NEURONSKIM MREŽAMA

Među najznačajnije primjene umjetnih neuronskih mreža ubraja se njihova primjena u upravljanju nelinearnim procesima. U nastavku se obrađuju tri strukture upravljanja koje su sa stajališta teorije upravljanja dobro utemeljene i svojstva kojih su dobro istražena. To su strukture:

♦ Inverzno upravljanje *(engl.* Inverse Control)

♦ Upravljanje s referentnim modelom *(engl.* Model Reference Control)

♦ Upravljanje s unutarnjim modelom *(engl.* Internal Model Control)

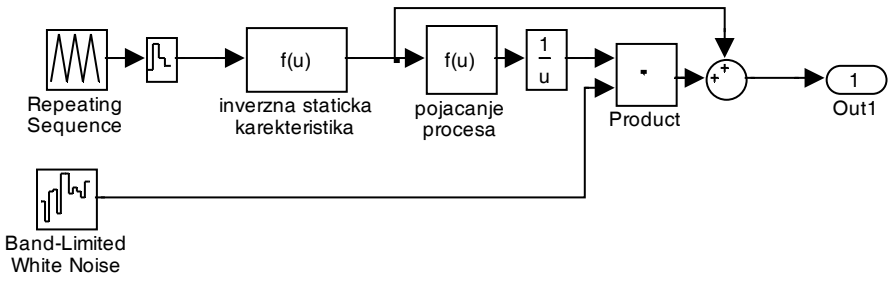
Zajednička je značajka svih ovih struktura upravljanja zasnovanost na identificiranome neuronskome modelu procesa. Osim modela procesa, identificira se i inverzni model procesa te se ove strukture mogu nazvati i strukture upravljanja zasnovane na inverznom modelu procesa.

Strukture upravljanja zasnovane na inverznom modelu koriste inverzni model procesa ili inverzni model modela procesa kao regulator. Za tvorbu inverznog modela procesa kod upravljanja nelinearnim procesima najčešće se primjenjuju neuronske mreže. Takav inverzni model naziva se inverznim neuronskim regulatorom.

7.1) Identifikacija inverznog modela

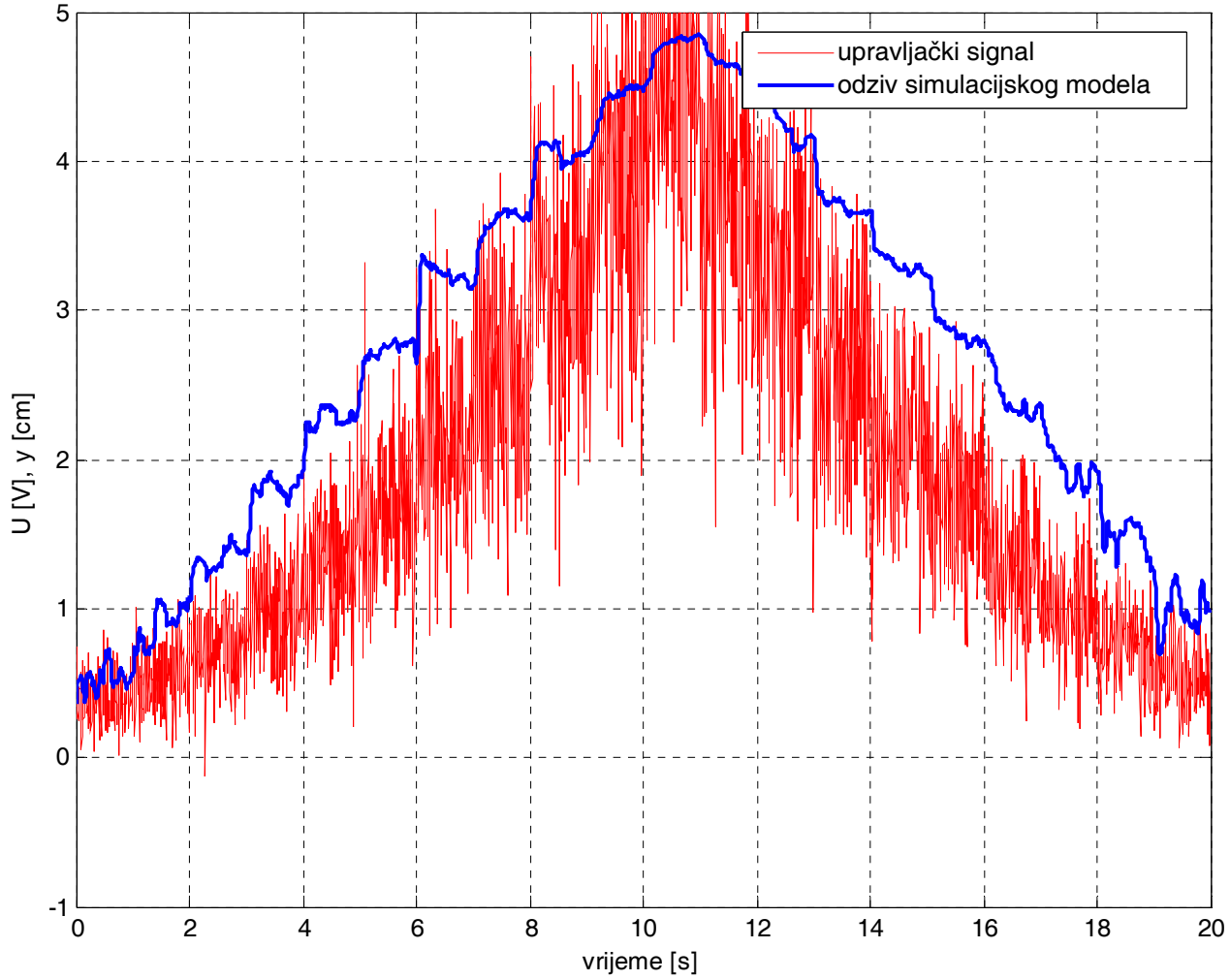
Identifikacija inverznog modela procesa provedena je uz eksplicitnu regularizaciju ($K_R=10^{-2}$), uz dimenzije regresijskog vektora *nui=3; nyi=3;*. Broj neurona skrivenog sloja mreže iznosi *S1i=20*.

Na slici 7.1. prikazana je realizacija signala za učenje inverznog modela, a slika 7.2. prikazuje izgled konačnog signala i odziv matematičkog modela magnetske levitacije na takav signal.



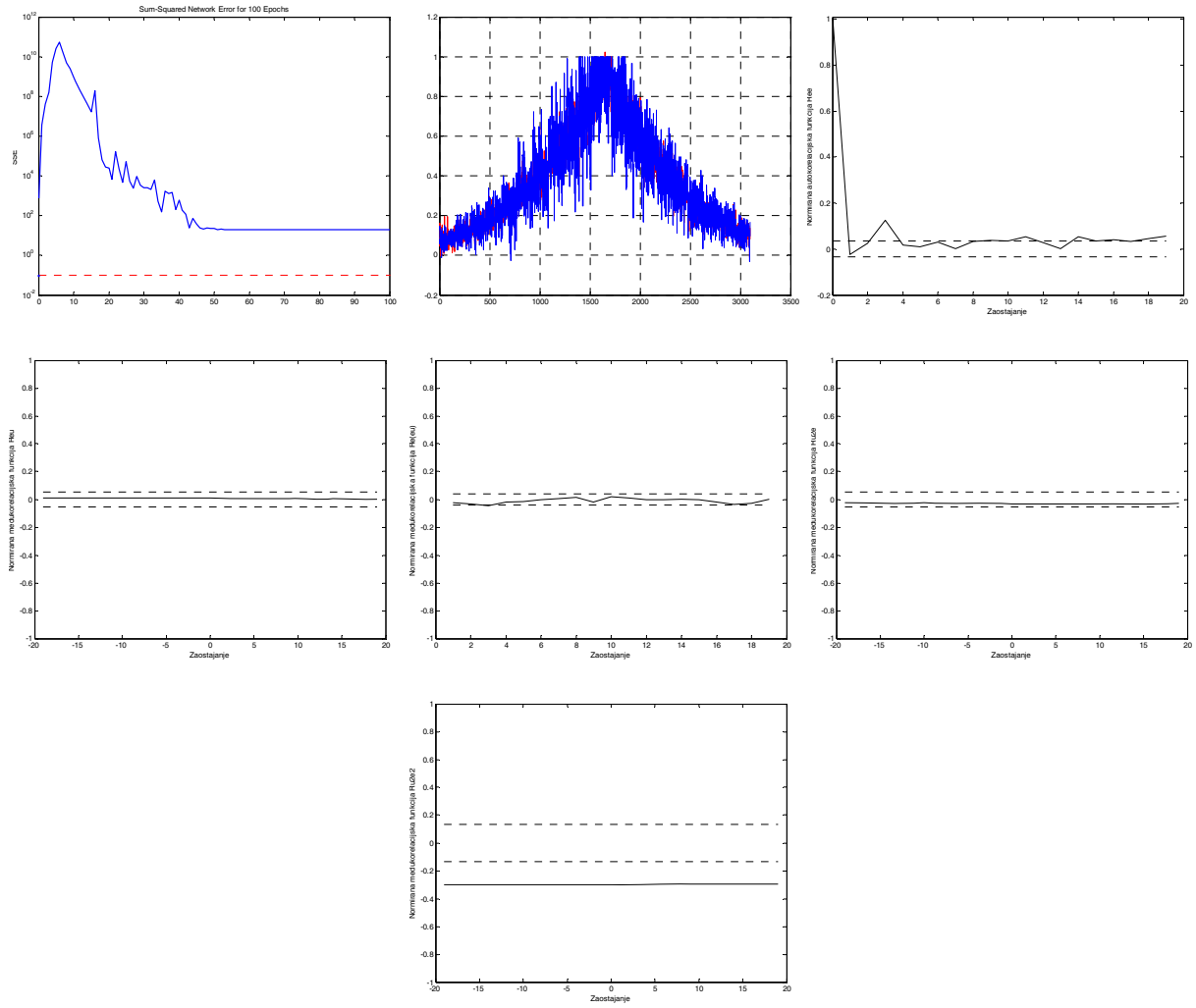
Slika 7.1. simulink shema generiranja signala za učenje.

Na signal stepeničastog oblika superponira se signal bijelog šuma. Blok *pojačanje procesa* i blok invertiranja signala ubačeni su iz razloga da umnožak varijance signala bijelog šuma i pojačanja procesa uvijek budu jednaki. Na taj način svaka točka nelinearnog procesa jednako je „razdrmana“.



Slika 7.2. prikaz signala za učenje i odziva matematičkog modela procesa na takav signal.

Konačno, za takav upravljački signal provedena je identifikaciju modela procesa i identifikacija inverznog modela! Srednja kvadratična pogreška, odziv mreže na validacijskim podacima, stanje neurona u skrivenom sloju mreže i korelacijske funkcije dobivene pri identifikaciji prikazani su na slici 7.3..



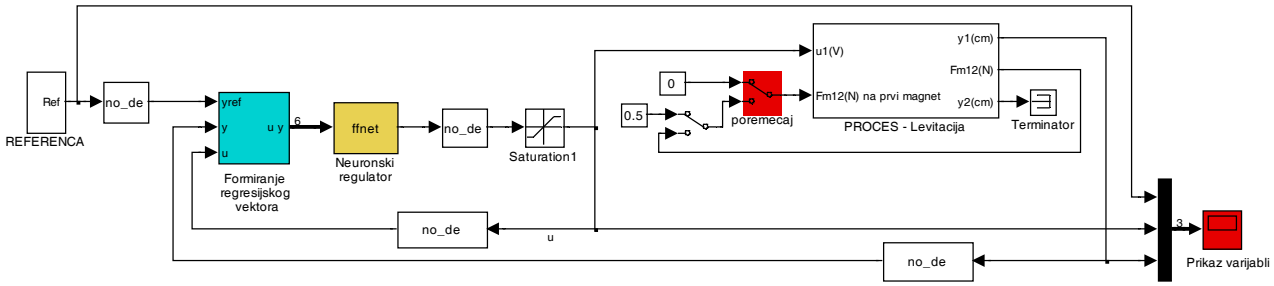
Slika 7.3. validacijski grafovi za identifikaciju inverznog modela procesa.

7.2) Inverzno upravljanje

Inverzno upravljanje bazira se na primjeni inverznog modela koji se spaja u seriju s procesom tvoreći na taj način sustav s trenutačnim odzivom jediničnog pojačanja između ulaza u inverzni model (y_r) i izlaza iz procesa (y). Na taj način se poništava sam proces i teoretski bi odziv takvog sustava trebao biti jednak ulaznom signalu. Dakle, inverzni model procesa, predstavljen neuronskom mrežom, djeluje kao regulator. Koncepcijski, ovo je najosnovnija struktura upravljanja s neuronskim regulatorom.

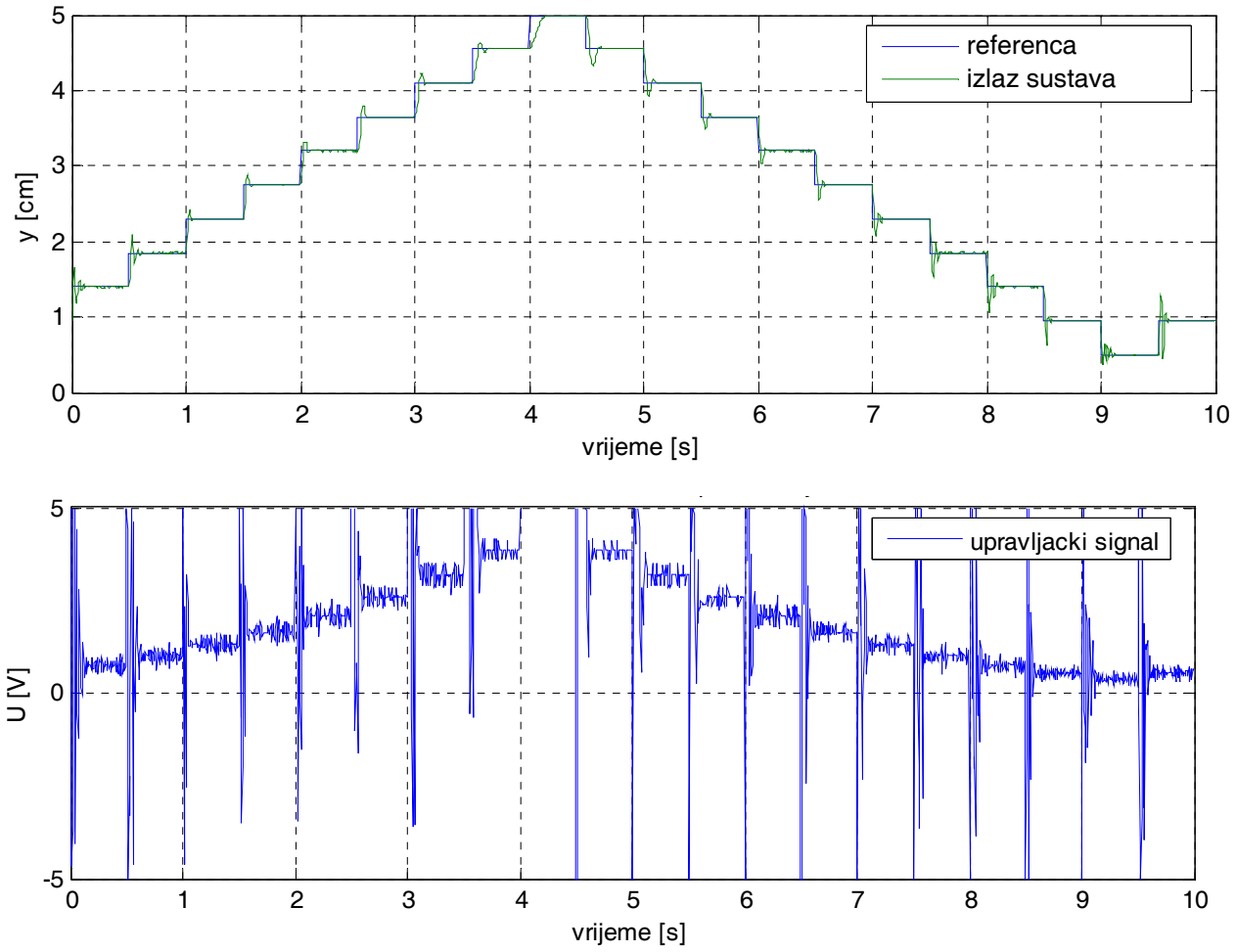
Nedostatak inverznog upravljanja je što ne postoji jedinstveno rješenje inverznog problema. Isto tako, ako je proces neminimalno-fazni, inverzni model je nestabilan. Još jedan veliki nedostatak je

taj što je inverzno upravljanje upravljanje u otvorenoj petlji pa ne osigurava kompenzaciju poremećaja.

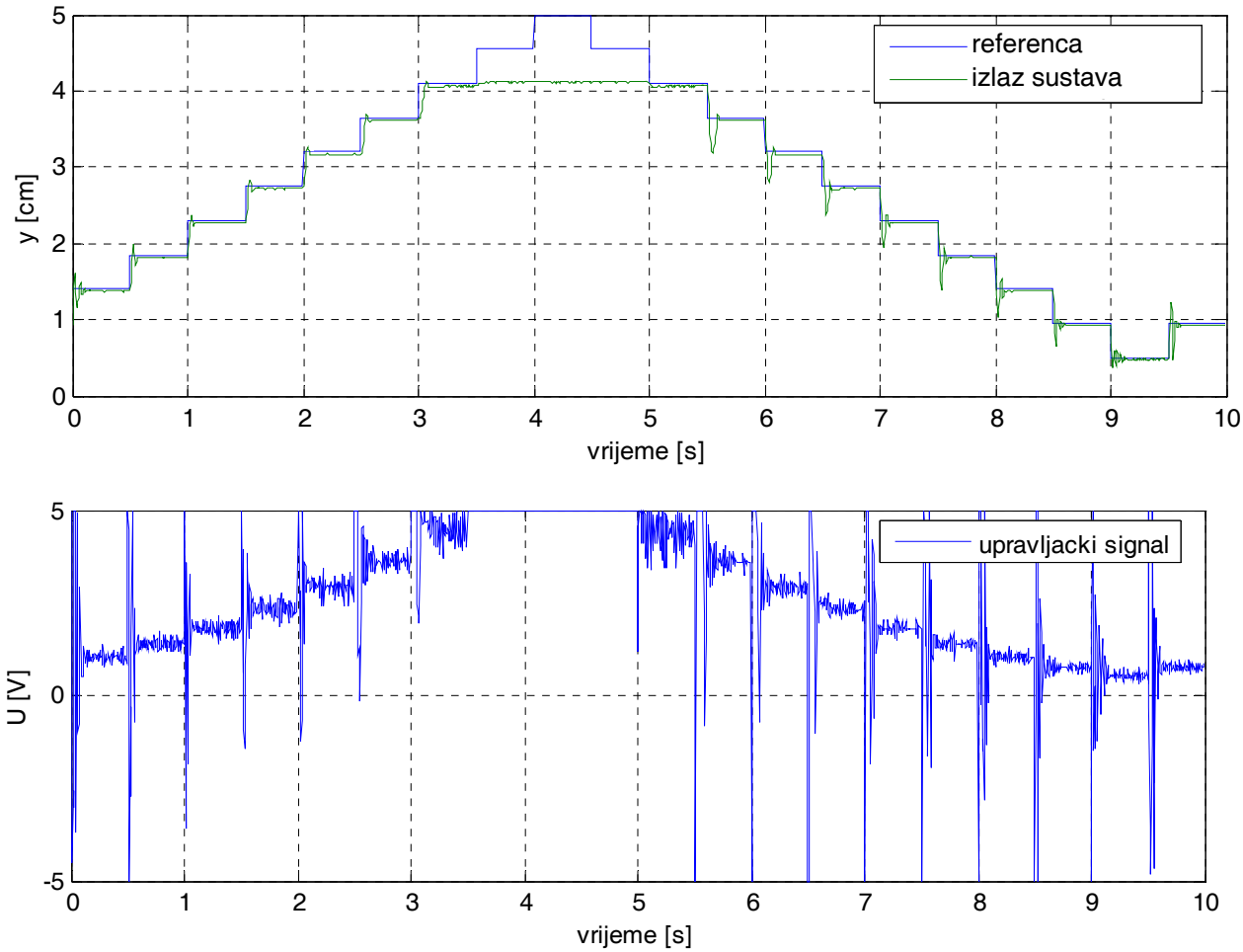


Slika 7.4. Simulink shema inverzne strukture upravljanja

Odzivi sustava upravljanja po principu inverznog upravljanja dani su slijedećim slikama:



*Slika 7.5. Odziv **inverzne strukture** upravljanja **bez poremećaja***



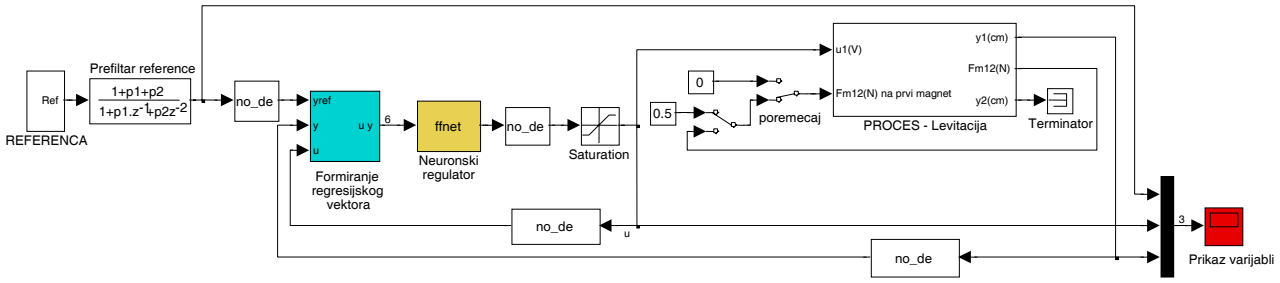
*Slika 7.6. Odziv **inverzne strukture** upravljanja **uz uključen poremećaj***

Iz odziva na slikama 7.5. i 7.6. možemo zaključiti kako postoji znatno forsiranje upravljačkog signala na samim skokovima referentnog signala. Razlog tome je što se pokušava natjerati sustav da trenutačno prati skokovitu promjenu reference za što je potrebna beskonačno velika energija. U tom bi slučaju upravljački signal pri promjeni reference trebao poprimiti beskonačno veliku vrijednost. Da bi se to spriječilo, u granu referentne vrijednosti uvodi si prefiltar kako bi skokove referentne vrijednosti „izgladio“ .

Na slici 7.5. u trenutku $t = 4$ s pri skoku referentne vrijednosti sa 4.5 cm na 5 cm primijećeno je da sustavu treba nešto više vremena da dostigne novu stacionarnu vrijednost. To se objašnjava time što je upravljački signal ograničen na maksimalnih 5 V dok bi za brže dostizanje novog položaja bile potrebne nešto veće vrijednosti upravljačkog signala. Iz sličnog razloga na slici 7.6. kada je uključen poremećaj sustav nema dovoljno snage za ostvarenje položaja iznad 4 cm.

U inverznom upravljanju da bi se postigao odziv sustava jednak ulazu, uz ulazni signal skokovitog oblika, upravljački signal bi trebao biti toliko velik da se takve amplitude ne mogu ostvariti zbog fizikalnih ograničenja, a sama forsiranja upravljačkog signala koja se pritom javljaju

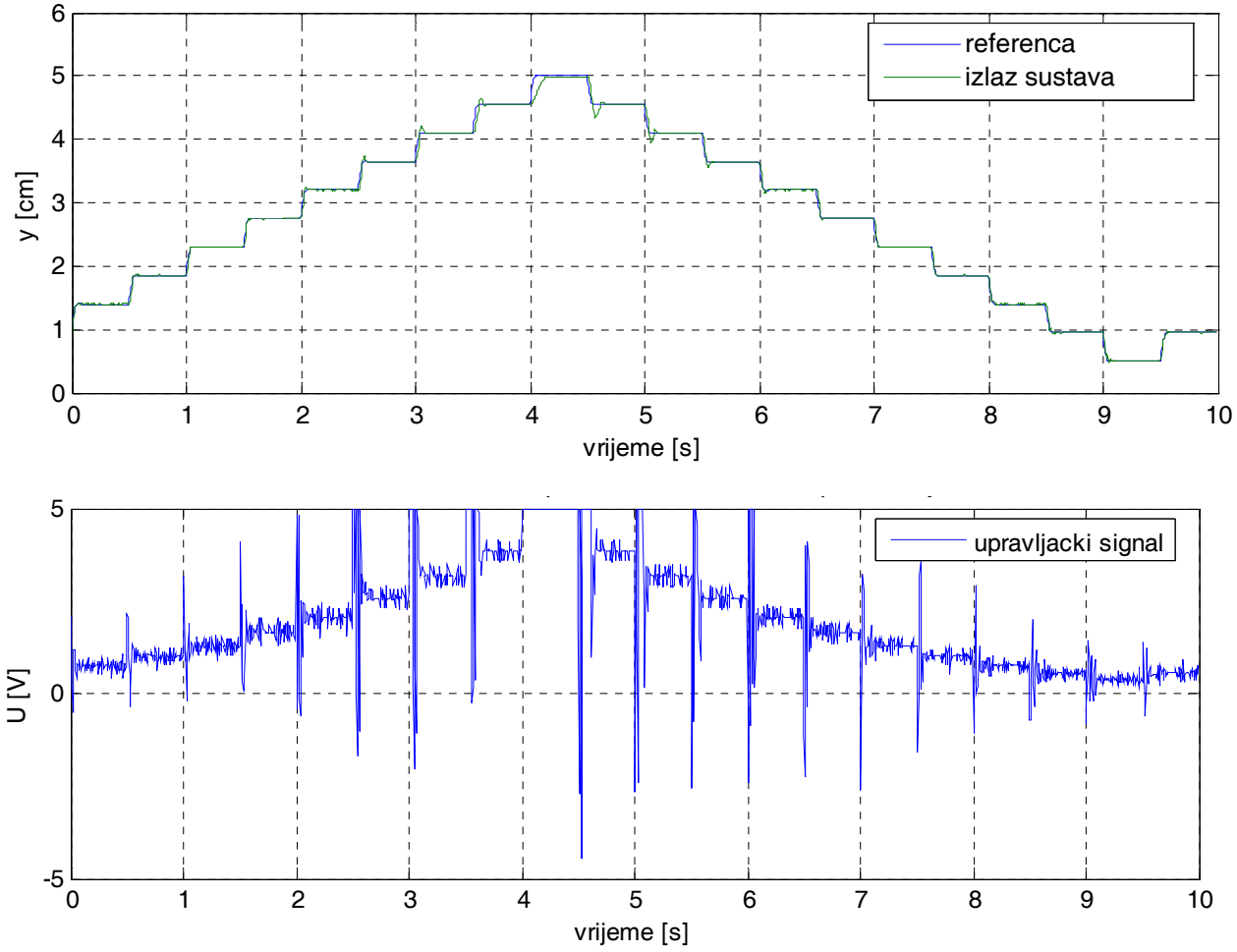
vrlo su neugodna za fizikalne komponente sustava. Ove se pojave ublažavaju uvođenjem prefiltra reference kojim se definira trajanje prijelazne pojave.



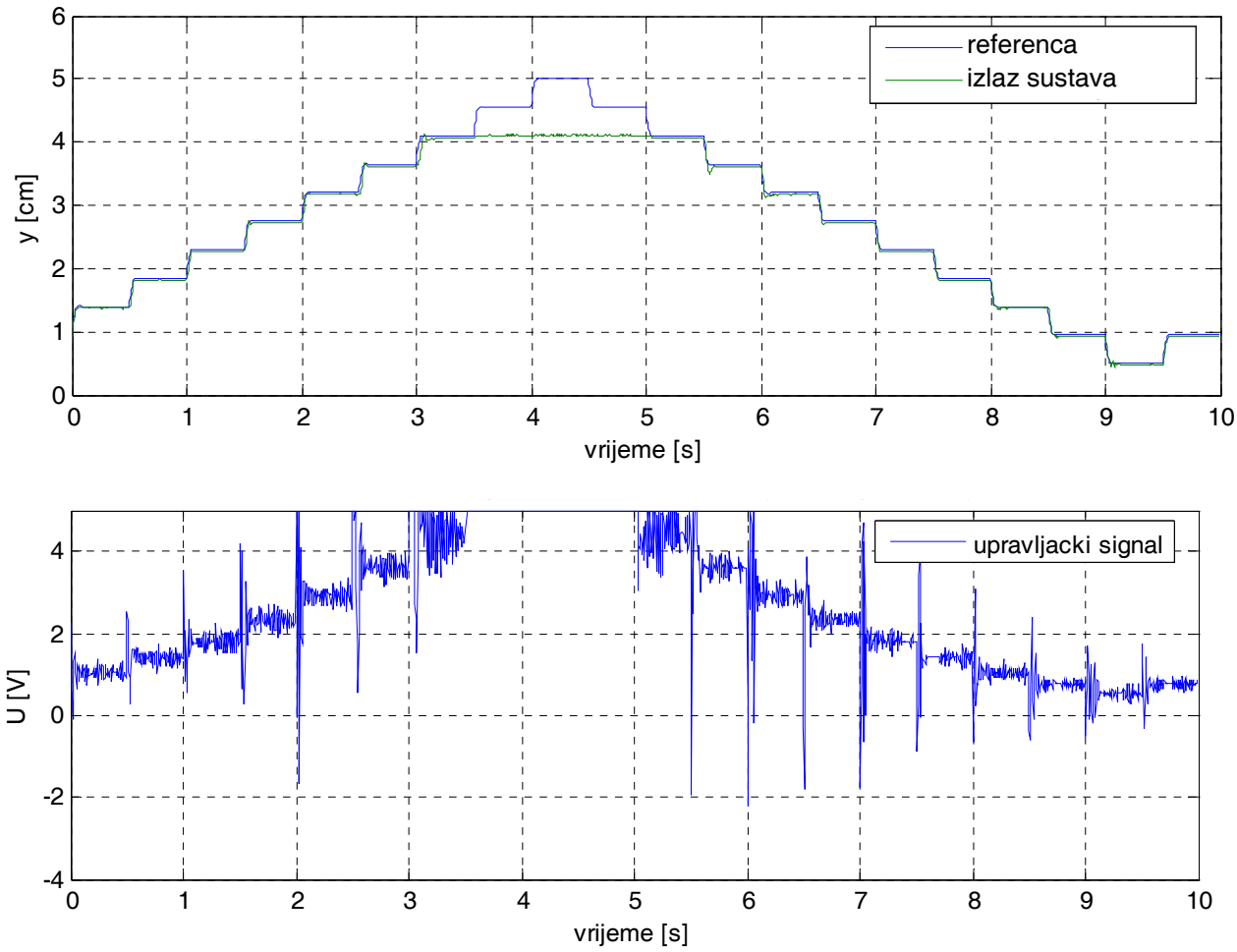
Slika 7.7. Simulink shema strukture inverznog upravljanja s prefiltrom reference.

Parametri prefiltra reference odabrani su tako da njegovi koeficijenti kvalitete iznose:

- relativni koeficijent prigušenja: $\zeta = 0.7$
- vrijeme prvog maksimuma: $t_m = 50$ ms



*Slika 7.8. Odziv strukture upravljanja **s prefiltrom** reference **bez poremećaja***



*7.9. Odziv strukture upravljanja **s prefiltrom** reference uz **uključen poremećaj***

U strukturi upravljanja gdje smo dodali prefiltar u granu reference dobivamo manje forsiranje upravljačkog signala jer sada se više ne forsiraju skokovite promjene referentnog signala. Ovaj način upravljanja pogodniji je za postrojenja jer ne dolazi do forsiranja izvršnih elemenata, a samim time smanjena je mogućnost kvara istih.

Ovakvom strukturom upravljanja dobivamo i bolje slijeđenje reference no još uvijek postoji pogreška u stacionarnom stanju kada na proces djeluje poremećaj/opterećenje.

Također, vidljivo je kako u odzivima kada je uključeni poremećaj postoji manja pogreška u stacionarnom stanju.

7.3) Adaptivno upravljanje s referentnim modelom

Kao što je već rečeno, primjena inverznog modela procesa kao regulatora u idealnome slučaju rezultira trenutačnim odzivom sustava s jediničnim pojačanjem. Ako neuronski regulator ne predstavlja idealni inverzni model procesa, vladanje regulacijskog sustava može se značajno narušiti. Primjenom on-line posrednog učenja neuronskog regulatora može se postići da on prilično dobro opisuje inverzni model procesa. Međutim, za postizanje trenutačnog odziva mogu biti potrebne velike amplitude upravljačkoga signala u dinamičkim stanjima, koje se zbog fizikalnih ograničenja procesa ne mogu ostvariti. Ove se nepoželjne pojave mogu značajno ublažiti uvođenjem referentnog modela, kojim se definira željeno vladanje sustava. Pri izboru referentnog modela treba voditi računa o fizikalnim ograničenjima upravljanoga procesa. Ova struktura upravljanja detaljnije je obrađena u 8. poglavlju.

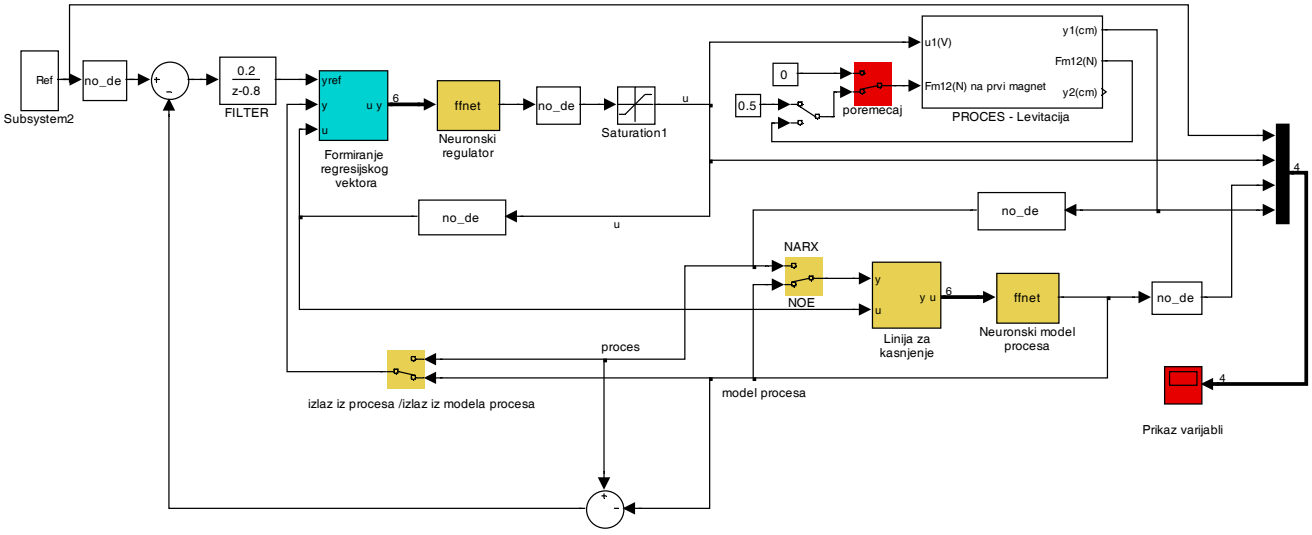
7.4) Upravljanje s unutarnjim modelom (IMC)

I ovaj način upravljanja se zasniva na inverznom modelu procesa. No, razlika u odnosu na prethodna dva načina upravljanja je u tome što se ovdje upravljanje vrši u zatvorenoj petlji. Dakle, postoji povratna veza po signalu razlike između procesa i njegovog modela a rezultat je kompenzacija vanjskog poremećaja.

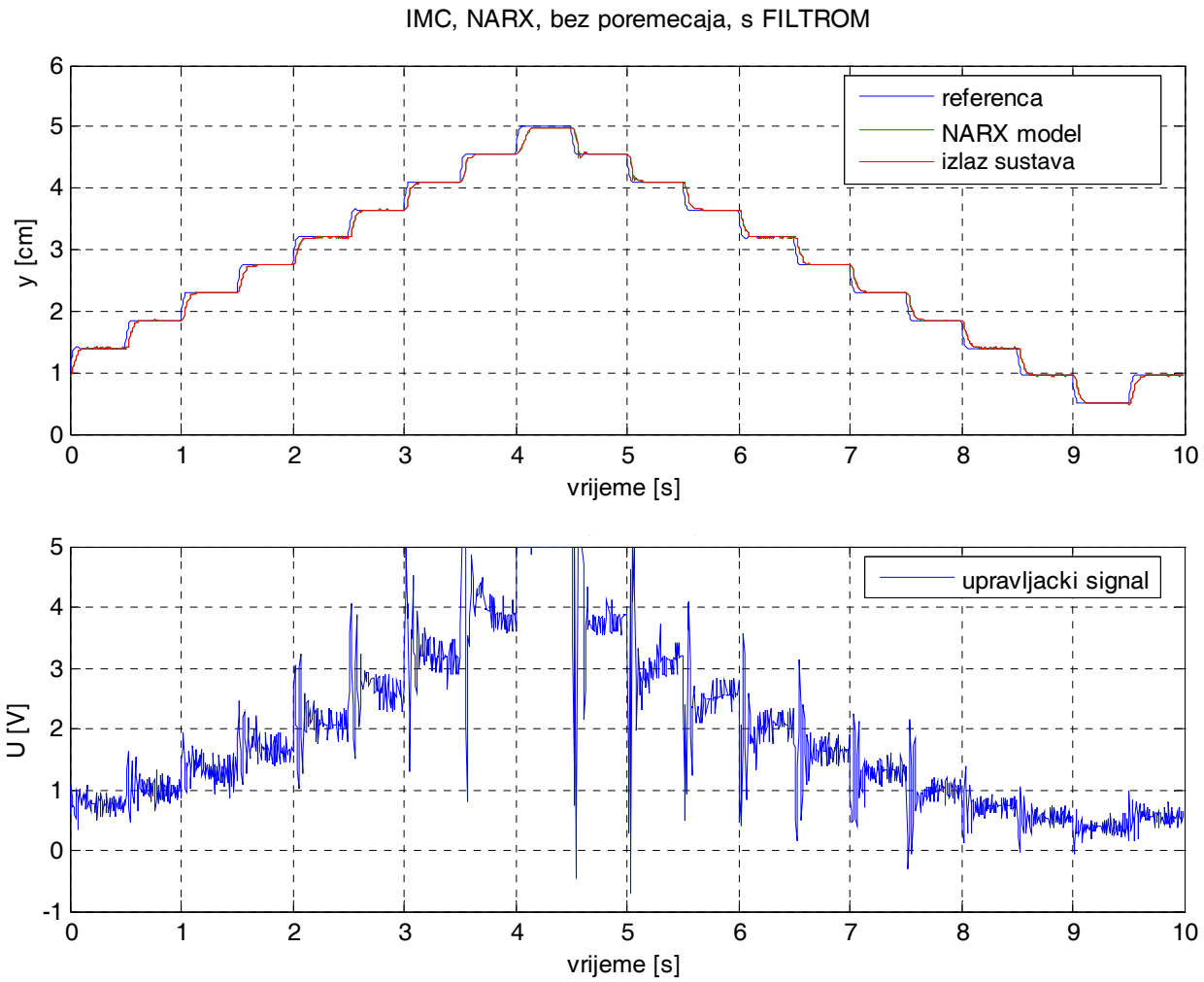
Ovo upravljanje može se primijeniti isključivo za upravljanje procesima koji su stabilni u otvorenoj petlji. Ipak, IMC upravljanje ima nekoliko svojstava koje ga čine iznimno prikladnim za upravljanje industrijskim procesima:

- ako su proces i regulator stabilni te ako je model procesa idealno naučen, zatvoreni je sustav također stabilan
- ako postoji inverzni model modela procesa i ako se koristi kao regulator te ako je s tim regulatorom sustav stabilan, upravljanje je idealno (u svakom trenutku je $y = y_r$ bez obzira na vanjske poremećaje)
- sinteza sustava upravljanja je jasna i jednostavna

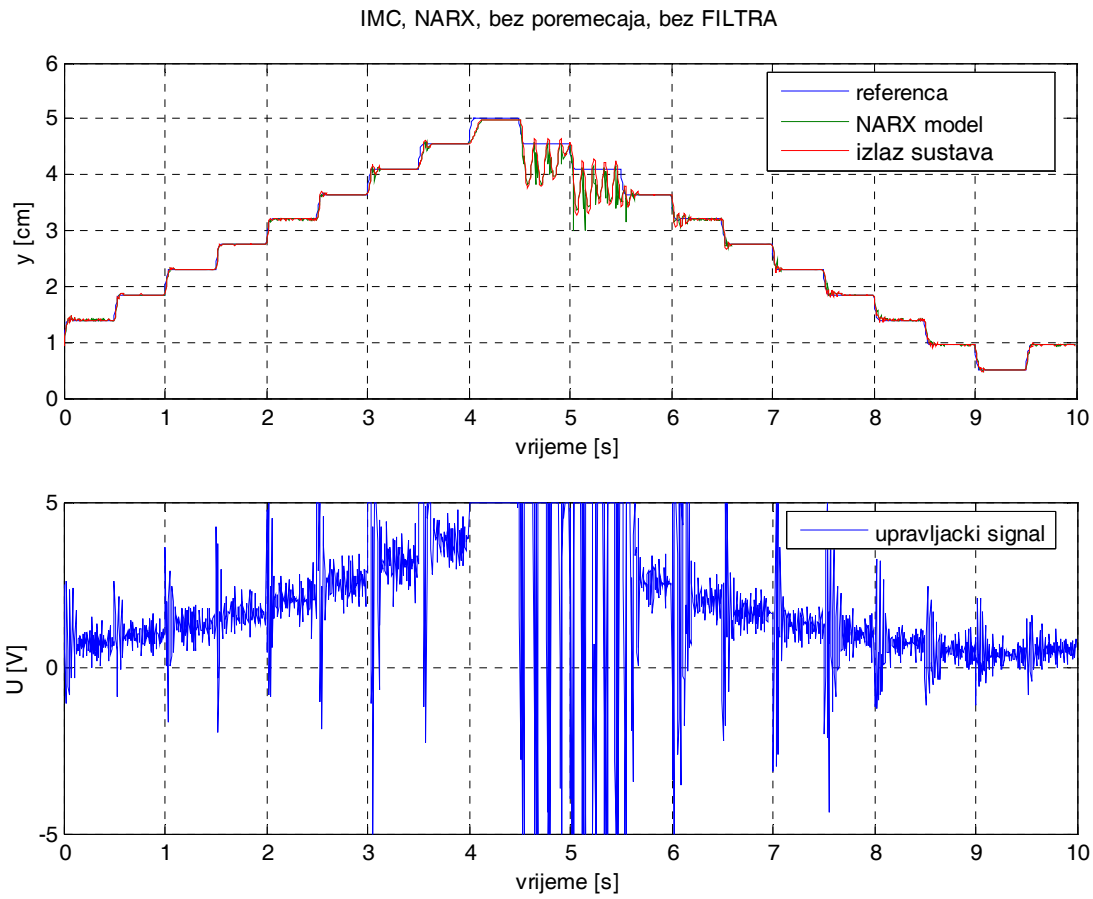
U stvarnosti je, naravno, nemoguće dobiti idealan model procesa, a idealno bi upravljanje zahtijevalo beskonačno pojačanje regulatora što bi dovelo do problema sa stabilnošću strukture. Zbog toga se u strukturu uvodi filtar koji se projektira tako da smanjuje pojačanje zatvorenog regulacijskog kruga čime se povećava robusnost sustava upravljanja uz istodobno udaljavanje vladanje sustava od idealnoga [1].



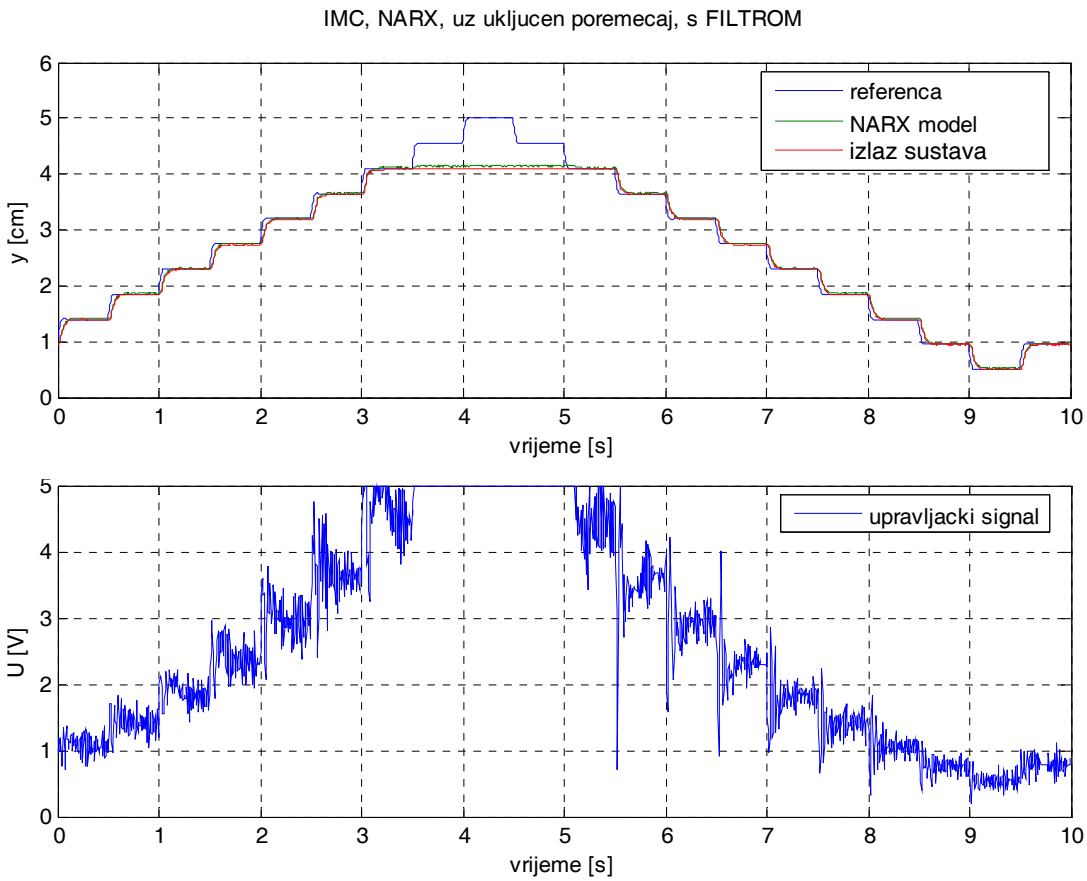
Slika 7.10. Simulink shema strukture upravljanja s unutarnjim modelom (IMC struktura).



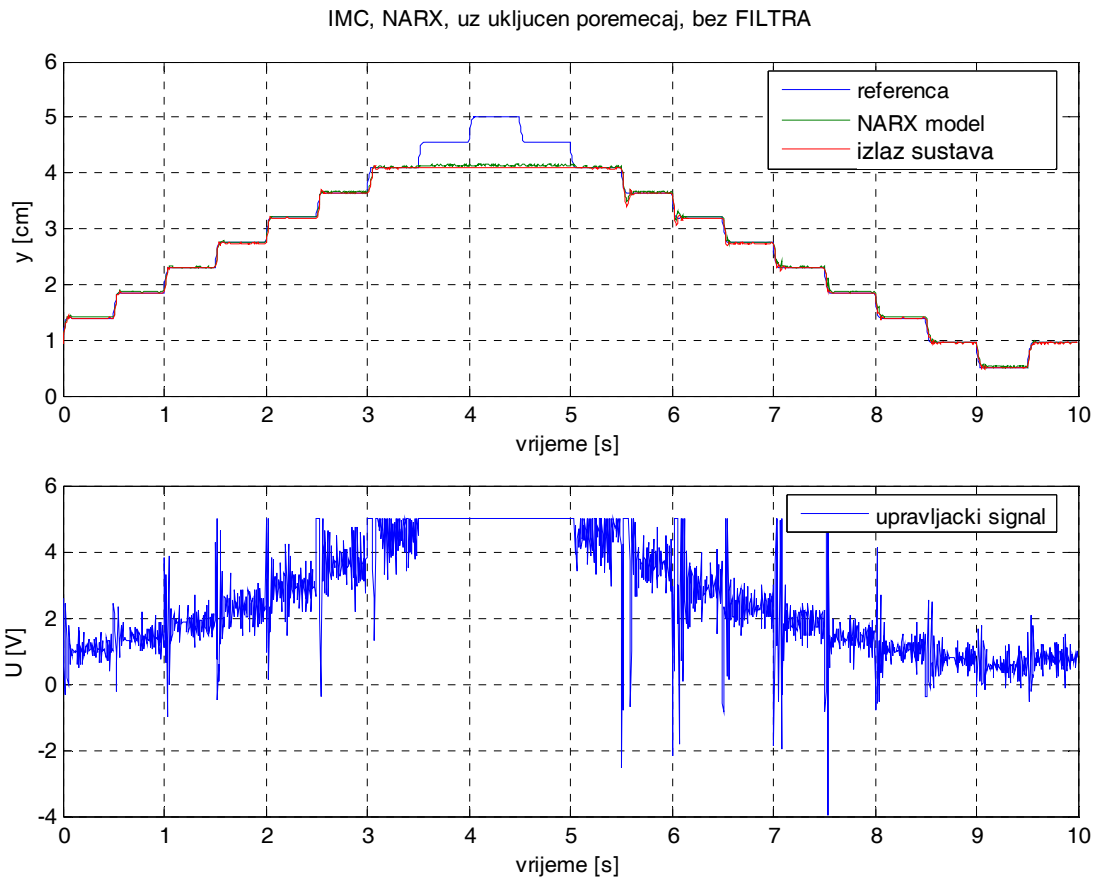
Slika 7.11. Struktura upravljanja s unutarnjim modelom uz NARX model procesa s filtrom



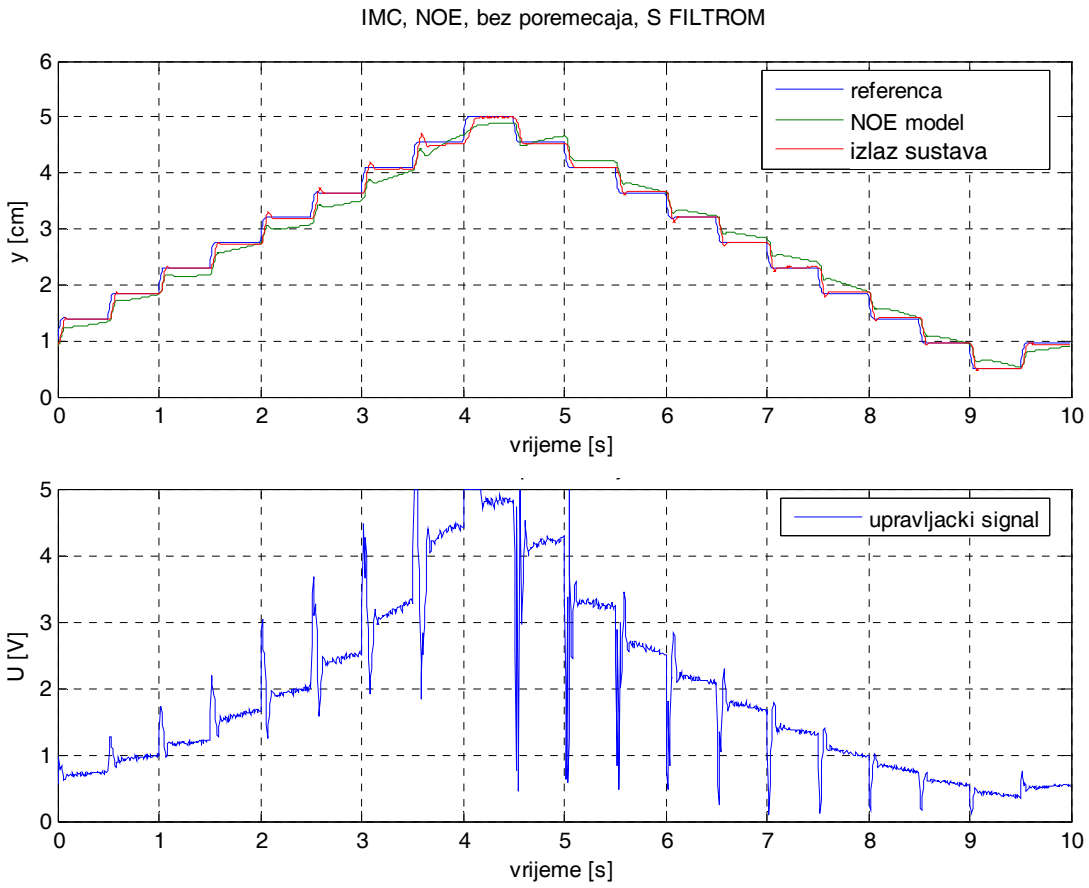
Slika 7.12. Struktura upravljanja s unutarnjim modelom uz NARX model procesa bez filtra



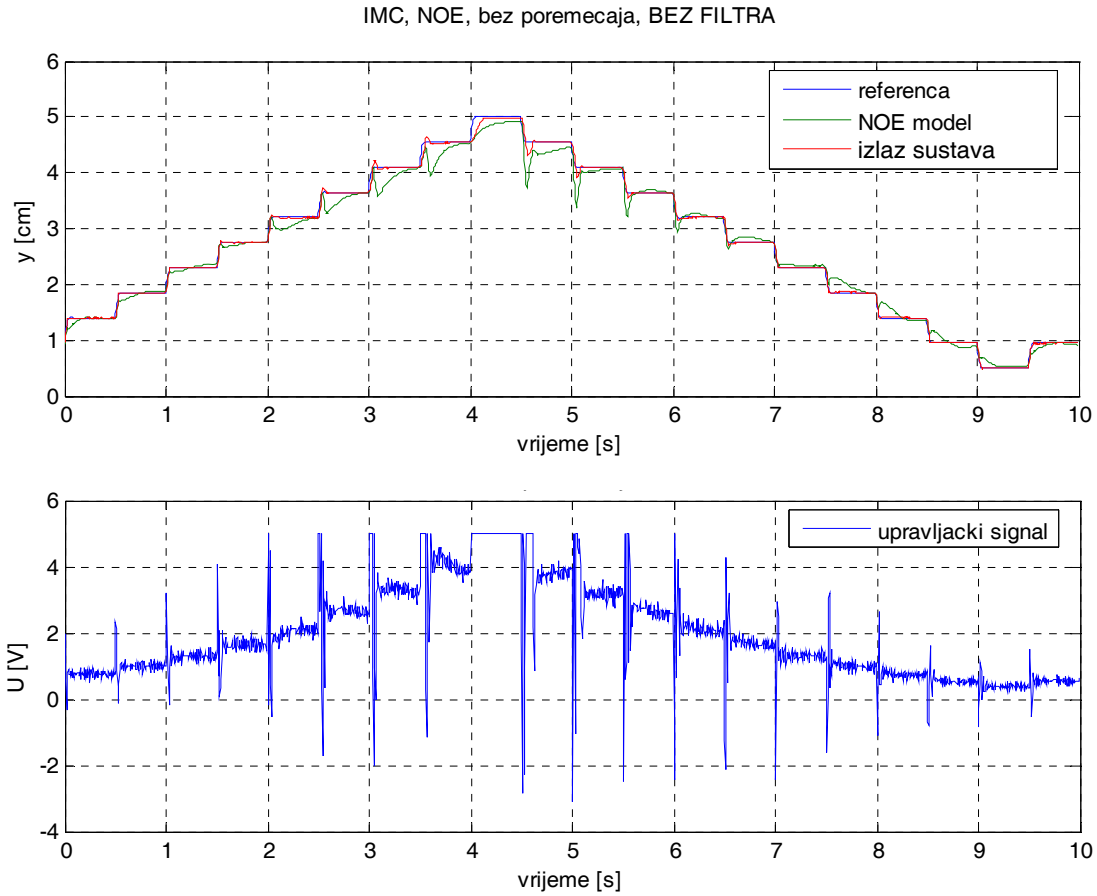
Slika 7.13. Upravljanje s unutarnjim modelom uz NARX model procesa uz uključen poremećaj s filtrom



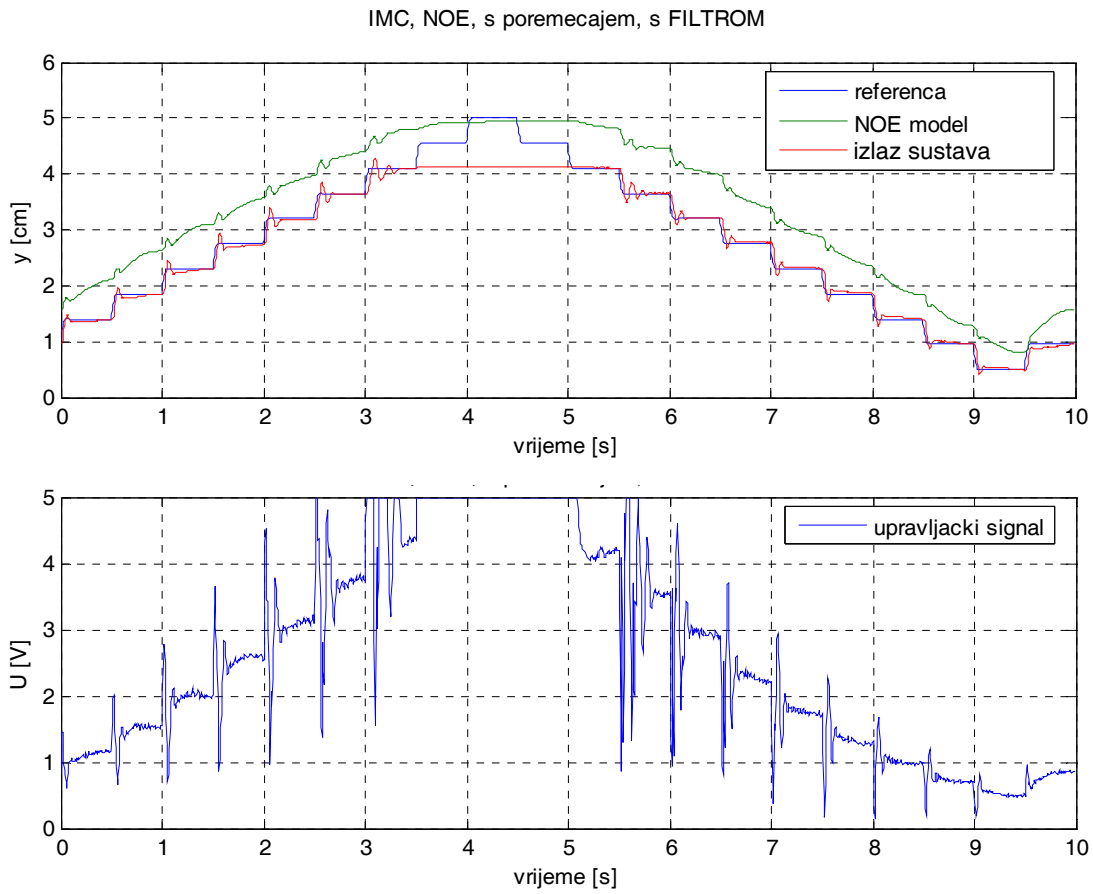
Slika 7.14. Upravljanje s unutarnjim modelom uz NARX model procesa uz uključen poremećaj bez filtra



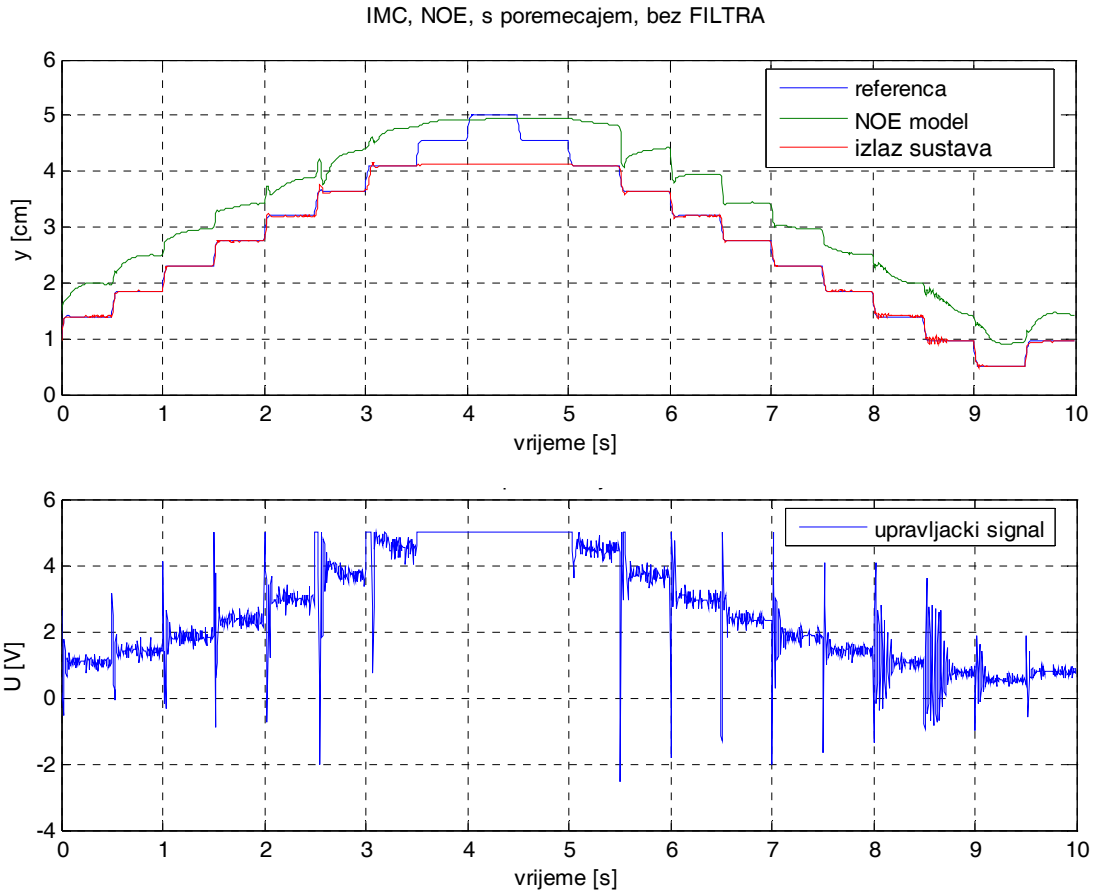
Slika 7.15. Upravljanje s unutarnjim modelom uz NOE model procesa bez poremećaja s filtrom



Slika 7.16. Upravljanje s unutarnjim modelom uz NOE model procesa bez poremećaja bez filtra



Slika 7.17. Upravljanje s unutarnjim modelom uz NOE model procesa uz uključen poremećaj s filtrom



Slika 7.18. Upravljanje s unutarnjim modelom uz NOE model procesa uz uključen poremećaj bez filtra

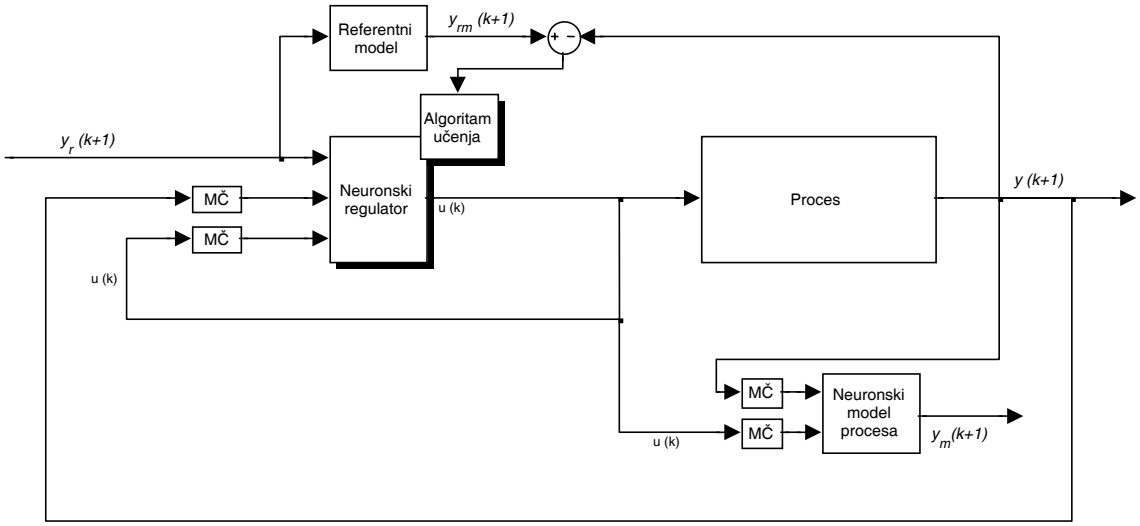
Ovom strukturom upravljanja riješen je problem pogreške slijeđenja reference pri djelovanju poremećaja. Razlog tome je upravljanje u zatvorenoj petlji. Dakle, postoji povratna veza po signalu razlike između procesa i njegovog modela a rezultat je kompenzacija vanjskog poremećaja.

Nadalje, ova struktura upravljanja s NOE modelom procesa i uz uključen filtar ima puno mirniji i prihvatljiviji upravljački signal.

Za usporedbu dani su i odzivi sustava bez filtra na kojima se može vidjeti njegov utjecaj. Isključivanjem filtra dobiva se idealnije ponašanje, ali na račun robusnosti sustava. Na slici 7.12. primjećuje se znatno pogoršanje vladanja sustava i velika oscilatornost odziva pri $t \approx 5s$ kada je filtar isključen. S druge strane na slikama 7.18. i 7.16. isključenje filtra uzrokuje poboljšanje odziva i točnije praćenje referentnog signala.

8. POSREDNO ON-LINE UČENJE NEURONSKOG REGULATORA

Posredno učenje inverznog neuronskog regulatora zasniva se na identificiranome modelu procesa, koji se koristi za izračunavanje gradijenta kriterija kakvoće po parametrima regulatora. Na slici 8.1. prikazana je blokovska shema sustava upravljanja s referentnim modelom uz posredno on-line učenje regulatora. Za ovakav načina učenja pretpostavlja se da je neuronski model procesa unaprijed identificiran off-line postupkom identifikacije i da se zatim ne mijenja te da je identificirani model procesa NARX strukture. Dakle, model procesa nije sastavni dio sustava upravljanja, već služi samo za učenje neuronskog regulatora.



Slika 8.1. Blokovska shema sustava upravljanja s referentnim modelom (uz on-line učenje).

Optimalne vrijednosti parametara inverznog neuronskog regulatora u *k-tom* koraku učenja dobivaju se minimiziranjem kriterija kakvoće:

$$\Im_R(\Theta_R(k)) = \frac{1}{2}\sum_{i=k-\tau}^{k}(y_r(i) - y(i))^2 \qquad (8\text{-}1)$$

gdje τ predstavlja broj prošlih mjernih uzoraka koji se uzimaju u obzir.

Primjenom algoritma povratnog prostiranja dobije se izraz za izračunavanje gradijenta kriterija kakvoće (8-1) po parametrima regulatora:

$$\frac{\partial \Im\left(\Theta_{\mathrm{R}}(\mathrm{k})\right)}{\partial \Theta_{\mathrm{R}}} = \sum_{i=k-\tau}^{k} [y(i) - y_r(i)] \frac{\partial \hat{y}^{+}(i)}{\partial \Theta_{\mathrm{R}}} \tag{8-2}$$

gdje je $\frac{\partial \hat{y}^{+}(i)}{\partial \Theta_{\mathrm{R}}}$ redna parcijalna derivacija izlaza neuronskog modela po parametrima regulatora i računa se:

$$\frac{\partial^{+} \hat{y}(i)}{\partial \Theta_{\mathrm{R}}} = \sum_{p=1}^{nb} \frac{\partial \hat{y}(i)}{\partial u(i-p)} \frac{\partial u^{+}(i-p)}{\partial \Theta_{\mathrm{R}}} + \sum_{r=1}^{na} \frac{\partial \hat{y}(i)}{\partial \hat{y}(i-r)} \frac{\partial \hat{y}^{+}(i-r)}{\partial \Theta_{\mathrm{R}}} \tag{8-3}$$

$$\frac{\partial^{+} u(i)}{\partial \Theta_{\mathrm{R}}} = \frac{\partial u(i)}{\partial \Theta_{\mathrm{R}}} + \sum_{p=1}^{nb-1} \frac{\partial u(i)}{\partial u(i-p)} \frac{\partial u^{+}(i-p)}{\partial \Theta_{\mathrm{R}}} + \sum_{r=0}^{na-1} \frac{\partial u(i)}{\partial \hat{y}(i-r)} \frac{\partial \hat{y}^{+}(i-r)}{\partial \Theta_{\mathrm{R}}} \tag{8-4}$$

pri čemu se može uzeti da je (prema [1 - str 136]):

$$\frac{\partial^{+} u(i-p)}{\partial \Theta_{\mathrm{R}}} = 0 \qquad za\ i < nb - 1$$

$$\frac{\partial^{+} \hat{y}(i-r)}{\partial \Theta_{\mathrm{R}}} = 0 \qquad za\ i < na - 1$$

Gornji izrazi predstavljaju granicu do koje je u prošlost potrebno računati. Derivacije ranije u prošlosti sve manje utječu na konačan izraz. Ipak, pri izradi adaptivnog algoritma za vremenski trenutak $i=k-\tau$ derivacije se računaju prema gornjim napomenama dok se za derivacije bliže trenutnom vremenskom indeksu u obzir uzimaju sve do tada izračunate prošle derivacije.

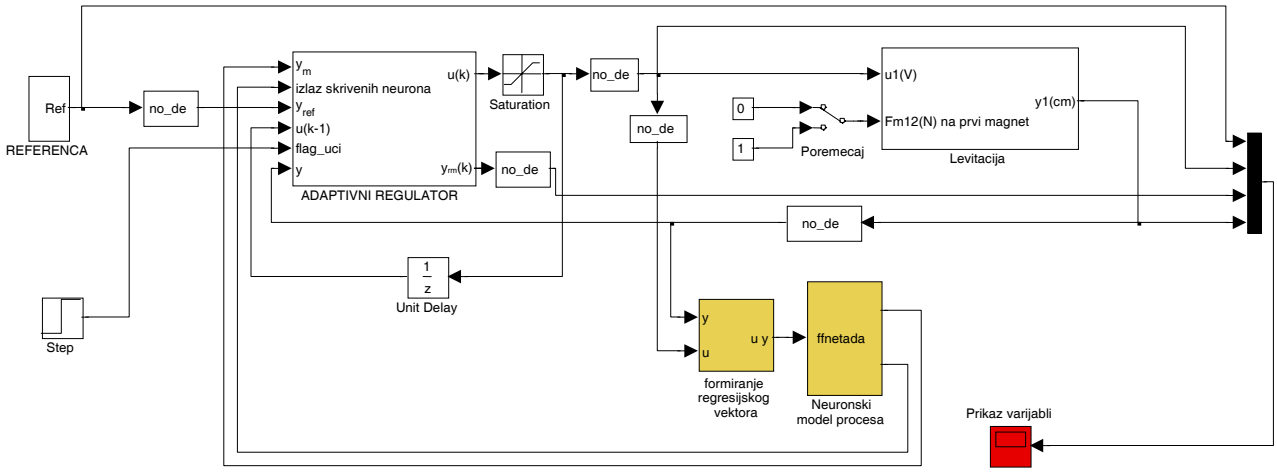
Posredno učenje inverznog neuronskog regulatora znatno je složenije od neposrednog učenja. Međutim, ono ima opravdanja u nekim slučajevima. Primjerice, ako nelinearnost procesa nije bijektivna neposrednim se učenjem može dobiti krivi inverzni model, dok se kod posrednog učenja izborom početnih vrijednosti parametara neuronskog regulatora može osigurati dobivanje inverznog modela sa željenim svojstvima. S obzirom da i tijekom učenja regulatora proces treba biti upravljan, smisleno je prvo provesti neposredno učenje regulatora i dobivene vrijednosti parametara postaviti kao početne vrijednosti za on-line posredno učenje. U tom se slučaju on-line posredno učenje može promatrati kao poboljšanje regulatora naučenog off-line, neposrednim učenjem. Postupak učenja regulatora zaustavlja se kada kriterij (8-1) poprimi zadani mali iznos. Nadalje, prednost posrednog učenja nad neposrednim jest i njegova usmjerenost eksplicitnom cilju, koji u kontekstu upravljanja podrazumijeva željeno vladanje izlaznog signala procesa. Naime, za razliku od neposrednog učenja,

posredno se učenje zasniva na minimizaciji pogreške između željenog vladanja i stvarnog vladanja izlaza procesa, odnosno njegova modela (izraz (8-1)). Ovo svojstvo čini posredno učenje posebno prikladnim za optimiranje neuronskog regulatora u primjenama u kojima je unaprijed poznata čitava trajektorija željenog vladanja izlaznog signala procesa (npr. u robotici).

Pri tome model procesa služi samo za izračunavanje gradijenta kriterija kakvoće po parametrima regulatora. Parametri inverznog regulatora izračunavaju se rekurzivnim algoritmom, opisanim u potpoglavlju 3.2, koji minimiziraju kriterij kakvoće (8-1).

Inverzno upravljanje jest upravljanje u otvorenoj petlji. To je njegov veliki nedostatak jer ne osigurava kompenzaciju vanjskog poremećaja. Ovaj se nedostatak može djelomično ublažiti, ako se postupak učenja parametara regulatora provodi trajno tijekom rada procesa (adaptivno upravljanje). Naime, učenjem regulatora, prema izrazima (8-1) i (8-2), pogreška između stvarne i željene vrijednosti izlaza procesa svest će se na nulu. Međutim, regulator više neće predstavljati inverzni model procesa. Brzina kompenzacije poremećaja ovisi o svojstvima algoritma učenja regulatora.

8.1) Realizacija adaptivnog algoritma



Slika 8.2. Simulacijska shema sustava upravljanja s referentnim modelom uz on-line učenje.

Adaptivni regulator, odnosno neuronska mreža sa mogućnosti on-line učenja realiziran je kao S-funkcija za Simulink u programskom okruženju Matlab. Funkcija je napisana u C programskom jeziku i podijeljena je u nekoliko podfunkcija.

- **`mdlInitializeSizes`**
  - definira broj parametara, ulaznih i izlaznih signala te broj radnih varijabli

- **`mdlInitializeSampleTimes`**
  - definira vrijeme uzorkovanja
- **`mdlInitializeConditions`**
  - funkcija za rezervaciju memorije i inicijalizaciju radnih varijabli
- **`f_df`**
  - funkcija za proračun derivacija aktivacijskih funkcija neuronske mreže (za neuronski model i za neuronski regulator)
- **`f_dymdup_dymdyp`**
  - funkcija za proračun derivacija izlaza neuronskog modela procesa po ulazima neuronski model
- **`f_dudthetaR_dudup_dudyp`**
  - funkcija za proračun derivacija upravljačkog signala koji daje neuronski regulator po parametrima regulatora i po ulazima (regresoru) u regulator
- **`f_dy_plus_dthetaR`**
  - funkcija za proračun izraza (8-3)
- **`f_du_plus_dthetaR`**
  - funkcija za proračun izraza (8-4)
- **`f_dI_dthetaR`**
  - funkcija za proračun izraza (8-2)
- **`simulacija_NM`**
  - proračun upravljačkog signala (simulacija neuronske mreže sa parametrima regulatora)
- **`mdlOutputs`**
  - u ovoj funkciji računaju se potrebne promjene parametara na temelju izraza (8-2) i (3-3) te se podešavaju parametri regulatora prema izrazu (3-1)
- **`mdlTerminate`**
  - pri završetku simulacije oslobađa se prethodno rezervirana memorija

Programski kôd priložen je na kraju diplomskog rada, a simulacijska shema prikazana je slikom 8.2. U bloku pod nazivom *adaptivni regulator* ugrađena je navedena funkcija koja kao ulaze prima slijedeće signale:

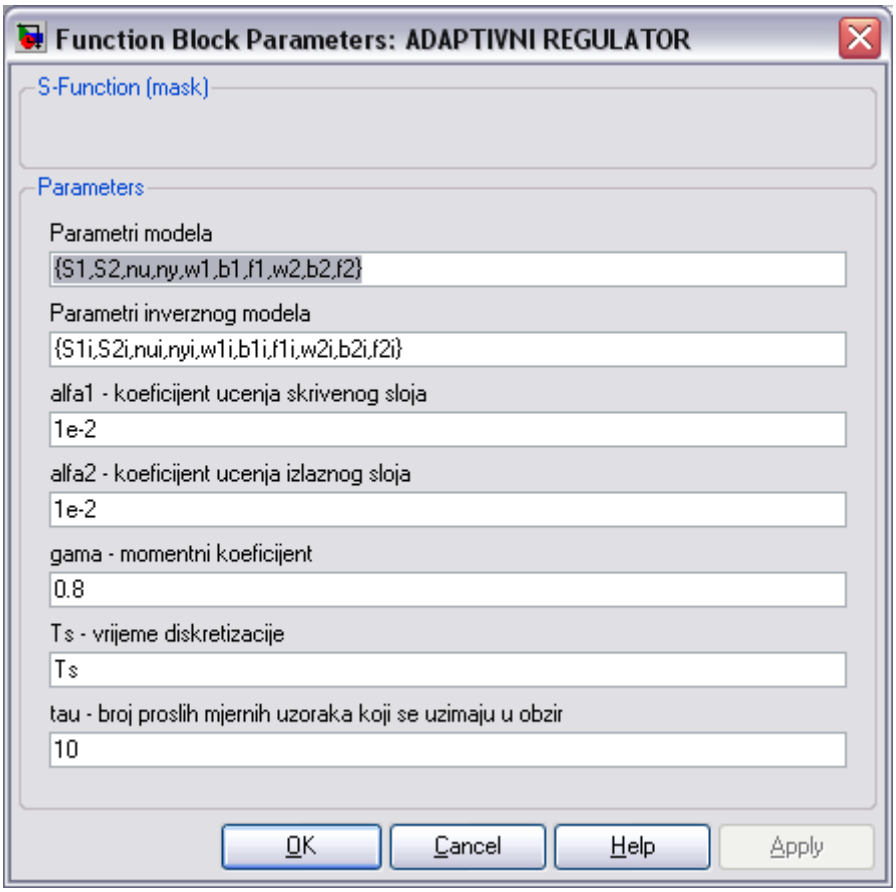
- y_m – izlaz iz neuronskog modela procesa
- *izlaz skrivenih neurona* – izlaz skrivenog sloja neuronskog modela
- y_{ref} – trenutna vrijednost referentnog signala

- *u(k*-1) – vrijednost upravljačkog signala iz prošlog koraka
- *flag_uci* – zastavica kojom se definira podešavaju li se parametri regulatora u trenutnom koraku ili ne. Ukoliko se želi uključiti algoritam učenja na ovaj ulaz postavlja se 1, a u suprotnome 0.
- *y* – izlaz iz procesa

Funkcija na izlaz bloka *adaptivni regulator* šalje upravljački signal *u(k)* proračunat simulacijom neuronske mreže uz parametre inverznog modela podešene u ovisnosti o stanju zastavice za učenje. Signal koji funkcija postavlja na drugi izlaz jest filtrirani iznos reference, odnosno referenca „provučena“ kroz referentni model $y_{rm}(t)$.

Svi signali koje se prenose u blok adaptivnog regulatora i unutar njega obrađuju moraju se normirati na interval [-1, 1]. U tu svrhu ostvarena je dodatna funkcija pod imenom *no_de.c* čija je uloga upravo normiranje signala na navedeni interval ili vraćanje normiranog signala na originalan interval (denormalizacija).

Osim toga realizirana funkcija prima niz parametara. U tu svrhu izrađena je maska bloka adaptivnog regulatora kako bi se pojednostavilo unošenje brojnih parametara. Maska bloka prikazana je na slici 8.3..



Slika 8.3. Parametri funkcije adaptivnog regulatora.

- *S1* – broj neurona u skrivenom sloju mreže
- *S2* – broj neurona u izlaznom sloju mreže (*S2=1*)
- *nu* – broj mjernih uzoraka upravljačkog signala kojima se puni regresijski vektor
- *ny* – broj mjernih uzoraka izlaznog signala kojima se puni regresijski vektor
- *w1* – težine skrivenog sloja neuronske mreže
- *b1* – pragovi osjetljivosti skrivenog sloja mreže
- *f1* – aktivacijska funkcija skrivenog sloja
- *w2* – težine izlaznog sloja neuronske mreže
- *b2* – pragovi osjetljivosti izlaznog sloja mreže
- *f2* – aktivacijska funkcija izlaznog sloja (uobičajeno *purelin*)

Parametri sa sufiksom *i* odnose se na inverzni model procesa.

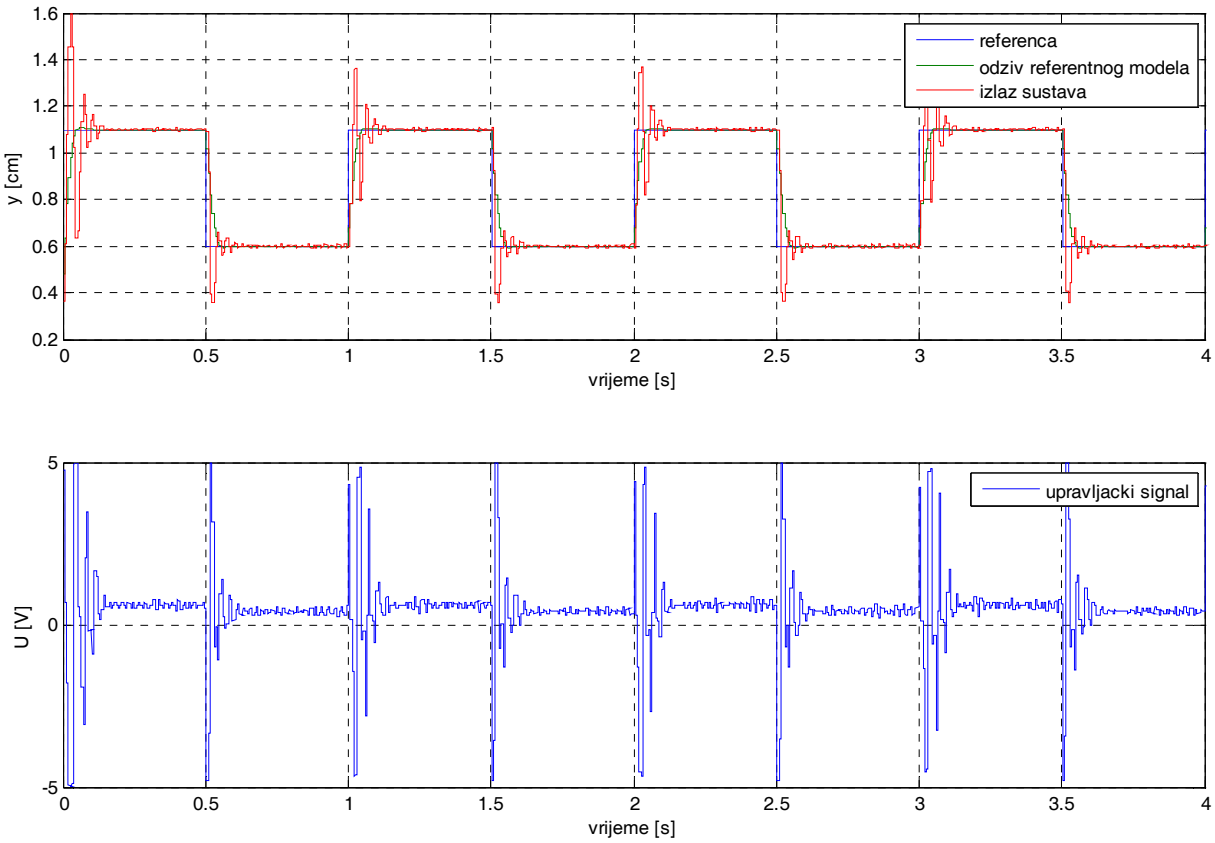
8.2) Prikaz rezultata rada algoritma on-line učenja

U simulacijama i provedbi eksperimenata kod sustava upravljanja s on-line učenjem korišten je inverzni model procesa sa *S1i=9* neurona u skrivenom sloju mreže. Na taj način broj parametara za podešavanje adaptivnim algoritmom smanjen je sa *n=161* (za *S1i=20, nui=nyi=3*) na *n=73* (za *S1i=9, nui=nyi=3*). U praktičnoj primjeni ovo smanjenje broja parametara neuronskog regulatora rezultira znatnim ubrzanjem izvođenja adaptivnog algoritma, ali bez znatnih narušavanja svojstva aproksimacije inverznog modela procesa.

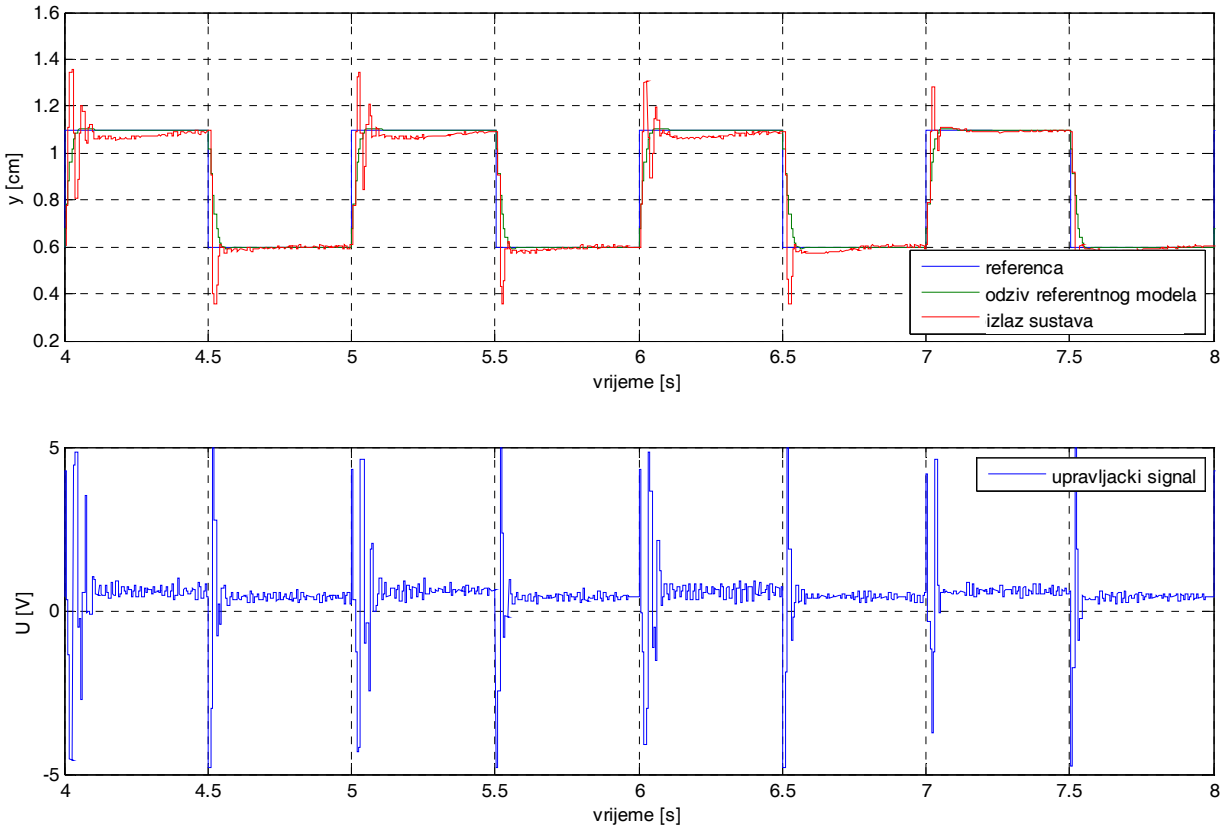
Parametri referentnog modela odabrani su tako da njegovi koeficijenti kvalitete iznose:

- relativni koeficijent prigušenja: $\zeta = 0.7$
- vrijeme prvog maksimuma: $t_m = 50$ ms

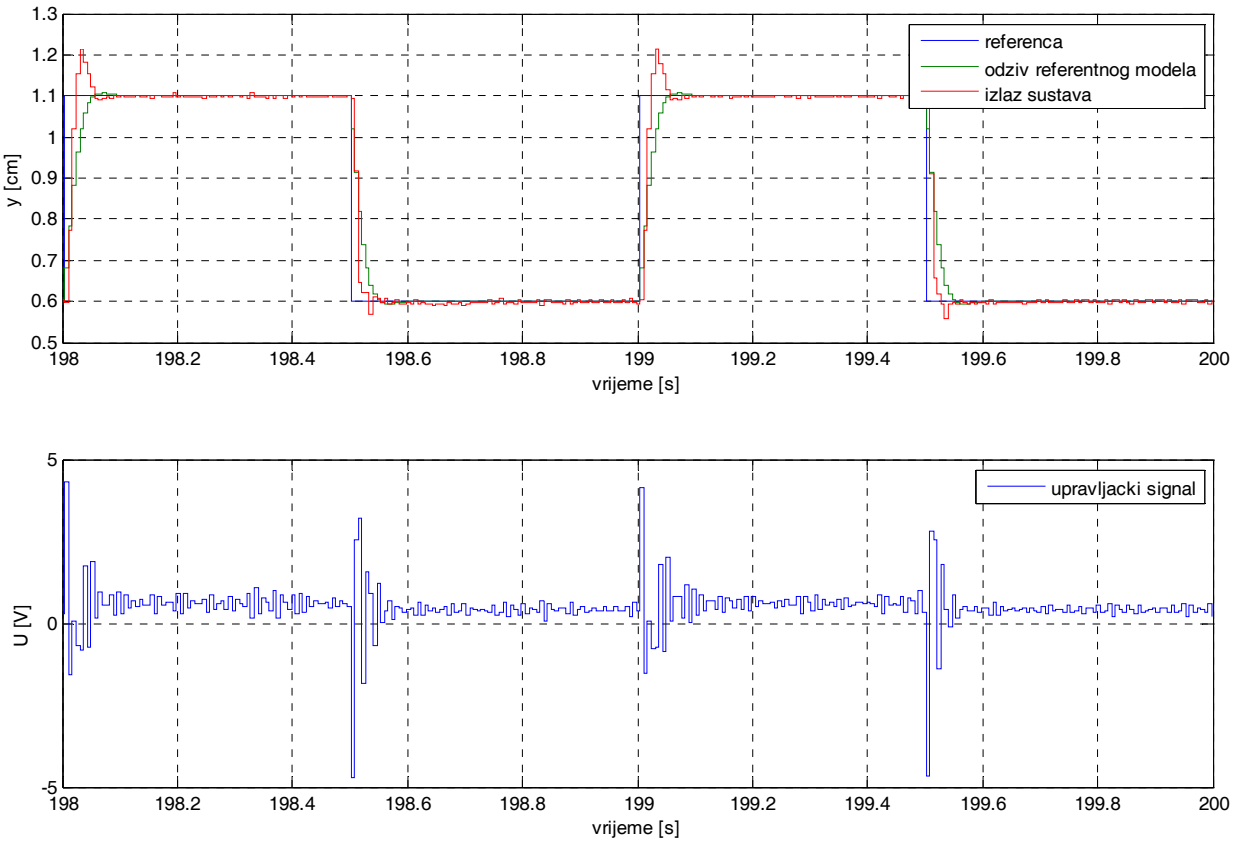
Na slikama 8.4. do 8.6. prikazan je tijek učenja i poboljšanja u kvaliteti upravljanja nakon podešavanja parametara regulatora on-line adaptivnim algoritmom.



*Slika 8.4. Odziv sustava **prije početka učenja**.*



*Slika 8.5. Odziv sustava **na početku učenja**.*

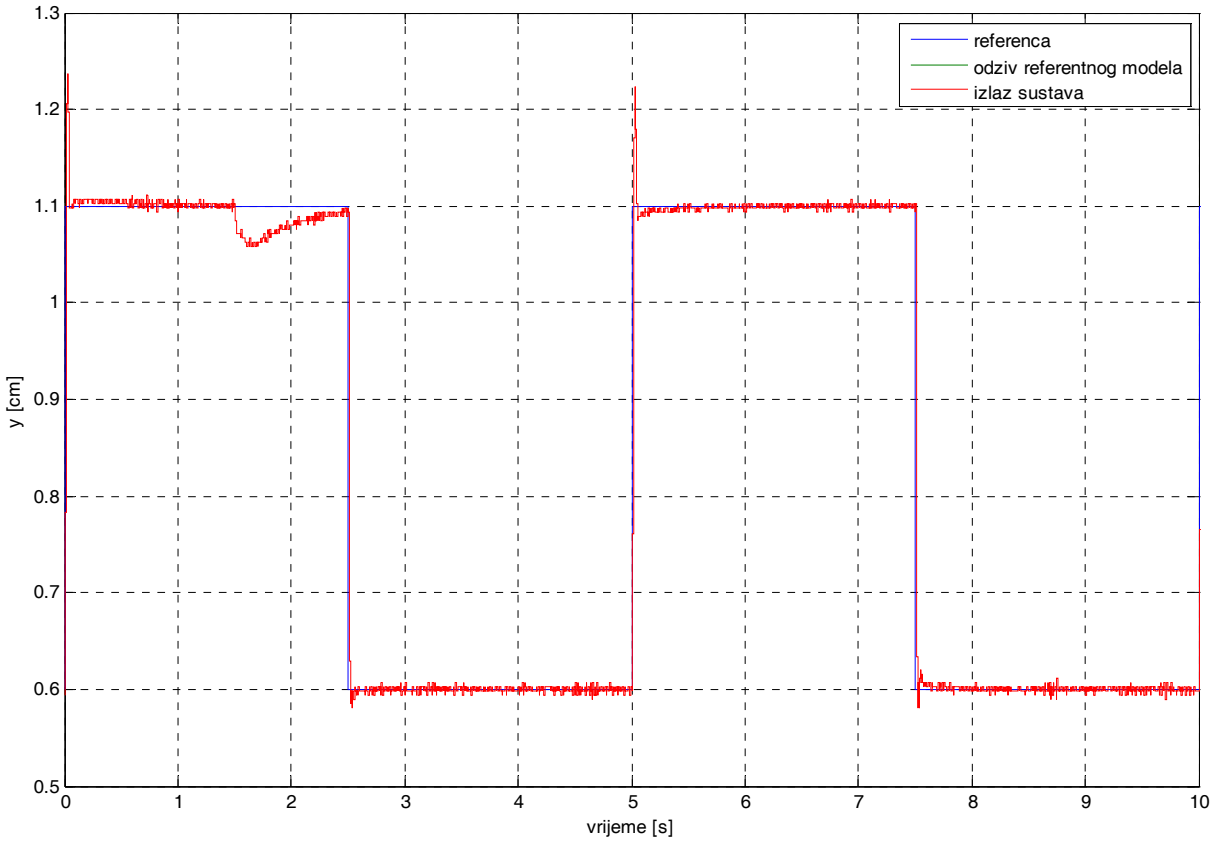


*Slika 8.6. Prikaz kvaliteta upravljanja **dovoljno dugo nakon početka učenja**.*

Eksperiment je proveden tako da se on-line učenje uključilo u *t=4* s postavljanjem jedinice na peti ulaz adaptivnog regulatora (ulaz *flag_uci*). Na taj način imamo uvid u ponašanje sustava sa početnim parametrima regulatora do četvrte sekunde, a nakon toga počinje podešavanje parametara prema algoritmu opisanom ranije.

Ako se usporede odzivi na slici 8.4. prije nego što se aktivira algoritam učenja sa odzivima na slici 8.5. gdje se uključilo on-line učenje, može se već u prvim promjenama referentnog signala uočiti pomak u kvaliteti upravljanja. Uočljivo je smjenjenje nadvišnja i kraće trajanje prijelazne pojave. Daljnjim učenjem u odzivu procesa primjećuju se stalna poboljšanja u smislu približavanja odzivu referentnog modela, smanjenja nadvišenja i smanjivanjem trajanja prijelazne pojave. Prema tome, iznos kriterijske funkcije (8-1) se smanjuje. Nakon dovoljno dugo vremena nakon početka učenja primjećuje se prestanak poboljšanja kvalitete upravljanja što je prikazano slikom 8.6.. Sada se parametri regulatora više ne mijenjaju i kriterijska funkcija dostigla je svoj cilj - lokalni minimum.

Adaptivno upravljanje isprobano je i u slučaju skokovite promjene vanjskog poremećaja. U tom slučaju adaptivni algoritam tako podesi parametre regulatora da se poništi utjecaj poremećaja i greška sustava svede na nulu (slika 8.7).



*Slika 8.7. Odziv sustava pri **skokovitoj promjeni poremećaja** u t=1.5 s.*

9. ZAKLJUČAK

Struktura upravljanja zasnovana po principu inverznog upravljanja zadovoljavajuće obavlja posao slijeđenja referentne vrijednosti. Takav sustav upravljanja nije u mogućnosti potpuno kompenzirati djelovanje vanjskog poremećaja pa iz tog razloga postoji pogreška u stacionarnom stanju kada na sustav djeluje vanjski poremećaj. Model procesa čak i sa uključenim poremećajem dobro prati referentni signal. Oscilacije upravljačke veličine su velike jer se forsira praćenje skokovite reference što zahtjeva dovođenje velike količine energije u sustav. Uz inverzno upravljanje s prefiltrom reference smanjuju se oscilacije upravljačke veličine jer se skokovite promjene referentne veličine filtriraju kroz prefiltar reference.

Pri upravljanju s unutarnjim modelom (IMC upravljanje) riješen je problem statičkog odstupanja procesa od referentnog signala. Na račun toga NOE model procesa više neće dobro slijediti referentni signal jer u njemu nije modelirana vanjska smetnja. Aktivnost upravljačkog signala u stacionarnom stanju svedena je na zadovoljavajuću razinu.

U osmom poglavlju prilikom testiranja adaptivnog algoritma na strukturi upravljanja s referentnim modelom pokazale su se prednosti takvog načina upravljanja. On-line učenje ima svoje prednosti jer parametre regulatora podešava konstantno tijekom rada sustava i ako početni parametri nisu dovoljno dobro postavljeni to se registrira povećanjem kriterijske funkcije. Na temelju velikog iznosa kriterijske funkcije i njenog gradijenta proračunava se potrebna promjena svakog parametra neuronskog regulatora kako bi se smanjio iznos kriterijske funkcije. Na taj način dolazi se do poboljšanja kvalitete upravljanja.

Ovakav način on-line podešavanja parametara regulatora ima svoje opravdanje u realnim sustavima gdje dolazi do promjena parametara na procesu uzrokovani primjerice promjenom temperature okoline ili nekim drugim faktorima. U takvim uvjetima ovakav adaptivni regulator ima sposobnost prilagođavanja novonastalim uvjetima te održavanja zadane kvalitete upravljanja.

PRILOG – PROGRAMSKI KÔD ADAPTIVNOG REGULATORA (ADA.C)

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME ada
#include "simstruc.h"
#include <malloc.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>

#define p1 -1.29040000000000    // konstante filtra 2. reda
#define p2  0.45170657636058    // (za referentni model)
// brojnik  = 1 + p1 + p2
// nazivnik = 1 + p1*z^-1 + p2*z^-2

#define LOGSIG(x) ( 1.0/(1+exp(-1.0*(x))) - 1.0 )
#define TANSIG(x) ( 2.0/(1+exp(-2.0*(x))) - 1.0 )

#define u1(element) (*uPtrs1[element])   //izlaz iz modela procesa
#define u2(element) (*uPtrs2[element])   //izlaz iz skrivenog sloja modela
#define u3(element) (*uPtrs3[element])   //referenca
#define u4(element) (*uPtrs4[element])   //proslo stanje regulatora
#define u5(element) (*uPtrs5[element])   //flag_uci
#define u6(element) (*uPtrs6[element])   //izlaz iz procesa

//parametri se unose slijedećim redosljedom...
//S1m,S2m,num,nym,w1m,b1m,f1m,w2m,b2m,f2m,S1,S2,nu,ny,w1,b1,f1,w2,b2,f2,
//Tdisc,alfa1,alfa2,gama,tau  (alfa=0.001-0.01 i gama=0.8-0.9)

// S1m - broj neurona 1.sloja neuronskog modela
// S2m - broj neurona 2.sloja neuronskog modela
// num - broj prošlih vrijednosti upravlj.signala u regresoru neuronskog modela
// nym - broj prošlih vrijednosti izlaza procesa u regresoru neuronskog modela
// w1m - težine 1.sloja neuronskog modela procesa
// b1m - pragovi 1.sloja neuronskog modela procesa
// f1m - aktivacijska funkcija 1.sloja neuronskog modela procesa
// w2m - težine 2.sloja neuronskog modela procesa
// b2m - pragovi 2.sloja neuronskog modela procesa
// f2m - aktivacijska funkcija 2.sloja neuronskog modela procesa
// S1  - broj neurona 1.sloja neuronskog regulatora
// S2  - broj neurona 2.sloja neuronskog regulatora
// nu  - broj prošlih vrijednosti upravlj.signala u regresoru neuronskog regulatora
// ny  - broj prošlih vrijednosti izlaza procesa u regresoru neuronskog regulatora
// w1  - težine 1.sloja neuronskog regulatora
// b1  - pragovi 1.sloja neuronskog regulatora
// f1  - aktivacijska funkcija 1.sloja neuronskog regulatora
// w2  - težine 2.sloja neuronskog regulatora
// b2  - pragovi 2.sloja neuronskog regulatora
// f2  - aktivacijska funkcija 2.sloja neuronskog regulatora
// T   - vrijeme diskretizacije
// alfa1-koeficijent učenja 1.sloja regulatora
// alfa2-koeficijent učenja 2.sloja regulatora
// gama -momentni koeficijent
// tau - broj prošlih mjernih uzoraka koji se uzimaju u obzir

#define S1m      (int_T) mxGetPr(ssGetSFcnParam(S, 0))[0]
#define S2m      (int_T) mxGetPr(ssGetSFcnParam(S, 1))[0]
#define num      (int_T) mxGetPr(ssGetSFcnParam(S, 2))[0]
#define nym      (int_T) mxGetPr(ssGetSFcnParam(S, 3))[0]
#define w1m   ((real_T*) mxGetPr(ssGetSFcnParam(S, 4)))
#define b1m   ((real_T*) mxGetPr(ssGetSFcnParam(S, 5)))
#define f1m      (int_T) mxGetPr(ssGetSFcnParam(S, 6))[0]
#define w2m   ((real_T*) mxGetPr(ssGetSFcnParam(S, 7)))
#define b2m   ((real_T*) mxGetPr(ssGetSFcnParam(S, 8)))
#define f2m      (int_T) mxGetPr(ssGetSFcnParam(S, 9))[0]
#define S1       (int_T) mxGetPr(ssGetSFcnParam(S,10))[0]
#define S2       (int_T) mxGetPr(ssGetSFcnParam(S,11))[0]
```

```
#define nu      ((int_T) mxGetPr(ssGetSFcnParam(S,12))[0] - 1)
#define ny       (int_T) mxGetPr(ssGetSFcnParam(S,13))[0]
#define w1r   ((real_T*) mxGetPr(ssGetSFcnParam(S,14)))
#define b1r   ((real_T*) mxGetPr(ssGetSFcnParam(S,15)))
#define f1       (int_T) mxGetPr(ssGetSFcnParam(S,16))[0]
#define w2r   ((real_T*) mxGetPr(ssGetSFcnParam(S,17)))
#define b2r   ((real_T*) mxGetPr(ssGetSFcnParam(S,18)))
#define f2       (int_T) mxGetPr(ssGetSFcnParam(S,19))[0]
#define T       (real_T) mxGetPr(ssGetSFcnParam(S,20))[0]
#define alfa1   (real_T) mxGetPr(ssGetSFcnParam(S,21))[0]
#define alfa2   (real_T) mxGetPr(ssGetSFcnParam(S,22))[0]
#define gama    (real_T) mxGetPr(ssGetSFcnParam(S,23))[0]
#define tau     ((int_T) mxGetPr(ssGetSFcnParam(S,24))[0] + 1)
    // ovdje je tau uvećan za jedan radi jednostavnijeg baratanja sa poljama varijabli

/* polja varijabli organizirana su na način da su najstariji uzorci stavljeni na kraj
 * polja, a najsvježiji na početak. Primjer: regresor neuronskog modela
 * je dimenzije num+nym=6 pa tako prvih 6 vrijednosti u polju regresora su trenutne
 * vrijednosti regresora, a posljednjih 6 vrijednosi najstarije vrijednosti regresora
 */

#define NUM_ARGS 25

/*************************************************************/
/**********  Funkcija MDL_INITIALIZE_SIZES  ******************/
/*************************************************************/
static void mdlInitializeSizes(SimStruct *S){
    ssSetNumSFcnParams(S, NUM_ARGS);
    if (ssGetNumSFcnParams(S)!=ssGetSFcnParamsCount(S)){
        return;
    }
    ssSetNumContStates(S,0);
    ssSetNumDiscStates(S,0);

    if(!ssSetNumInputPorts(S,6)) return;
    ssSetInputPortWidth(S,0,S2m);   //izlaz iz modela procesa
    ssSetInputPortWidth(S,1,S1m);   //izlaz iz skrivenog sloja modela
    ssSetInputPortWidth(S,2,1);     //referenca
    ssSetInputPortWidth(S,3,1);     //proslo stanje regulatora
    ssSetInputPortWidth(S,4,1);     //flag_uci
    ssSetInputPortWidth(S,5,1);     //izlaz iz procesa

    ssSetInputPortDirectFeedThrough(S,0,1);
    ssSetInputPortDirectFeedThrough(S,1,1);
    ssSetInputPortDirectFeedThrough(S,2,1);
    ssSetInputPortDirectFeedThrough(S,3,1);
    ssSetInputPortDirectFeedThrough(S,4,1);
    ssSetInputPortDirectFeedThrough(S,5,1);
    if (!ssSetNumOutputPorts(S,2)) return;
    ssSetOutputPortWidth(S,0,1);
    ssSetOutputPortWidth(S,1,1);

    ssSetNumSampleTimes(S,1);
    ssSetNumIWork(S,1);
    ssSetNumRWork(S,0);
    ssSetNumPWork(S,40);
    ssSetNumModes(S,0);
    ssSetNumNonsampledZCs(S,0);
}

/*************************************************************/
/********  Funkcija MDL_INITIALIZE_SAMPLE_TIMES  ***********/
/*************************************************************/
//inicijalizacija vremena diskretizacije
static void mdlInitializeSampleTimes( SimStruct *S){
    ssSetSampleTime(S,0,T);
    ssSetOffsetTime(S,0,0);
}
```

```
/*********************************************************/
/**********  Funkcija MDL_INITIALIZE_SIZES  ***************/
/*********************************************************/
/*
 * funkcija za rezervaciju memorije
 * i inicijalizaciju radnih varijabli
 */
#define MDL_INITIALIZE_CONDITIONS
#if defined (MDL_INITIALIZE_CONDITIONS)
static void mdlInitializeConditions(SimStruct *S){
    real_T *pok;
    int_T i;

    if(ssGetSFcnParamsCount(S)!=NUM_ARGS)
        ssSetErrorStatus(S,"Pogrešan broj ulaznih argumenata");

    if((pok=(real_T*)calloc(S1m,sizeof(real_T)))!=NULL)            /* df1m */
        ssSetPWorkValue(S,0,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc(S2m,sizeof(real_T)))!=NULL)            /* df2m */
        ssSetPWorkValue(S,1,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc((ny+nu+1)*S1,sizeof(real_T)))!=NULL)   /* w1 */
        ssSetPWorkValue(S,2,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");
    pok=ssGetPWorkValue(S,2);
    for (i=0;i<S1*(nu+ny+1);i++)
        pok[i]=w1r[i];

    if((pok=(real_T*)calloc(S1,sizeof(real_T)))!=NULL)             /* b1 */
        ssSetPWorkValue(S,3,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");
    pok=ssGetPWorkValue(S,3);
    for (i=0;i<S1;i++)
        pok[i]=b1r[i];

    if((pok=(real_T*)calloc(S1*S2,sizeof(real_T)))!=NULL)          /* w2 */
        ssSetPWorkValue(S,4,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");
    pok=ssGetPWorkValue(S,4);
    for (i=0;i<S2*S1;i++)
        pok[i]=w2r[i];

    if((pok=(real_T*)calloc(S2,sizeof(real_T)))!=NULL)             /* b2 */
        ssSetPWorkValue(S,5,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");
    pok=ssGetPWorkValue(S,5);
    for (i=0;i<S2;i++)
        pok[i]=b2r[i];

    if((pok=(real_T*)calloc((tau+nu)*S1,sizeof(real_T)))!=NULL)      /* df1 */
        ssSetPWorkValue(S, 6,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc((tau+nu)*S2,sizeof(real_T)))!=NULL)      /* df2 */
        ssSetPWorkValue(S, 7,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc((tau+nu)*(nu+ny+1)*S1,sizeof(real_T)))!=NULL)/* dudw1 */
        ssSetPWorkValue(S, 8,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc((tau+nu)*S1,sizeof(real_T)))!=NULL)    /* dudb1 */
        ssSetPWorkValue(S, 9,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");
```

```
if((pok=(real_T*)calloc((tau+nu)*S1*S2,sizeof(real_T)))!=NULL) /* dudw2 */
    ssSetPWorkValue(S,10,pok);
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc((tau+nu)*S2,sizeof(real_T)))!=NULL)    /* dudb2 */
    ssSetPWorkValue(S,11,pok);
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc(S1,sizeof(real_T)))!=NULL)             /* d2d1 */
    ssSetPWorkValue(S,12,pok);
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc(S1m,sizeof(real_T)))!=NULL)            /* d2d1m */
    ssSetPWorkValue(S,13,pok);
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc((tau+ny)*num,sizeof(real_T)))!=NULL)   /* dymdup */
    ssSetPWorkValue(S,14,pok);
else ssSetErrorStatus(S,"Nedovoljno memorije");

 if((pok=(real_T*)calloc((tau+ny)*nym,sizeof(real_T)))!=NULL)  /* dymdyp */
     ssSetPWorkValue(S,15,pok);
 else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc((tau+nu)*(nu+1),sizeof(real_T)))!=NULL)/* dudup */
    ssSetPWorkValue(S,16,pok);
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc((tau+nu)*(ny),sizeof(real_T)))!=NULL)  /* dudyp */
    ssSetPWorkValue(S,17,pok);
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc((tau+nu)*S1,sizeof(real_T)))!=NULL)    /* reg1 */
    ssSetPWorkValue(S,18,pok);
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc(S2,sizeof(real_T)))!=NULL)             /* reg2 */
    ssSetPWorkValue(S,19,pok);
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc(nu,sizeof(real_T)))!=NULL)             /* up */
    ssSetPWorkValue(S,20,pok);
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc(ny+tau,sizeof(real_T)))!=NULL)         /* yp */
    ssSetPWorkValue(S,21,pok);
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc((tau+nu)*(ny+nu+1),sizeof(real_T)))!=NULL)/* r */
    ssSetPWorkValue(S,22,pok);
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc((tau+nu)*S1*(nu+ny+1),sizeof(real_T)))!=NULL)
    ssSetPWorkValue(S,23,pok);                          /* du_plus_dw1 */
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc((tau+nu)*S1*S2,sizeof(real_T)))!=NULL)
    ssSetPWorkValue(S,24,pok);                          /* du_plus_dw2 */
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc((tau+nu)*S1,sizeof(real_T)))!=NULL)
    ssSetPWorkValue(S,25,pok);                          /* du_plus_db1 */
else ssSetErrorStatus(S,"Nedovoljno memorije");

if((pok=(real_T*)calloc((tau+nu)*S2,sizeof(real_T)))!=NULL)
    ssSetPWorkValue(S,26,pok);                          /* du_plus_db2 */
else ssSetErrorStatus(S,"Nedovoljno memorije");
```

```
    if((pok=(real_T*)calloc((nu+ny+1)*S1*2,sizeof(real_T)))!=NULL)
        ssSetPWorkValue(S,27,pok);                          /* delta_w1 */
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc((S1*S2)*2,sizeof(real_T)))!=NULL)/* delta_w2 */
        ssSetPWorkValue(S,28,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc((S1)*2,sizeof(real_T)))!=NULL)   /* delta_b1 */
        ssSetPWorkValue(S,29,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc((S2)*2,sizeof(real_T)))!=NULL)   /* delta_b2 */
        ssSetPWorkValue(S,30,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc((nu+ny+1)*S1,sizeof(real_T)))!=NULL)/*dI_dw1*/
        ssSetPWorkValue(S, 31,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc(S1,sizeof(real_T)))!=NULL)        /* dI_db1 */
        ssSetPWorkValue(S, 32,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc(S1*S2,sizeof(real_T)))!=NULL)     /* dI_dw2 */
        ssSetPWorkValue(S,33,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc(S2,sizeof(real_T)))!=NULL)        /* dI_db2 */
        ssSetPWorkValue(S,34,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc(2+tau,sizeof(real_T)))!=NULL)       /* ref */
        ssSetPWorkValue(S,35,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc((tau+ny-1)*S1*(nu+ny+1),sizeof(real_T)))!=NULL)/*dy_plus_dw1*/
        ssSetPWorkValue(S,36,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc((tau+ny-1)*S1,sizeof(real_T)))!=NULL)   /* dy_plus_db1 */
        ssSetPWorkValue(S,37,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc((tau+ny-1)*S1*S2,sizeof(real_T)))!=NULL)/* dy_plus_dw2 */
        ssSetPWorkValue(S,38,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");

    if((pok=(real_T*)calloc((tau+ny-1)*S2,sizeof(real_T)))!=NULL)   /* dy_plus_db2 */
        ssSetPWorkValue(S,39,pok);
    else ssSetErrorStatus(S,"Nedovoljno memorije");
}
#endif

void f_df(SimStruct *S){
    // podfunkcija za proračun derivacija aktivacijskih funkcija
    // vidjeti izraze u tablici 2.1.
    real_T  *df1m, *df2m, *reg1, *reg2,*df1,*df2;
    int_T i;
    InputRealPtrsType uPtrs1=ssGetInputPortRealSignalPtrs(S,0);  //izlaz modela procesa
    InputRealPtrsType uPtrs2=ssGetInputPortRealSignalPtrs(S,1);  //izlaz skrivenog sloja modela
    df1m = ssGetPWorkValue(S, 0); // derivacije aktivacijskih funkcija 1.sloja modela
    df2m = ssGetPWorkValue(S, 1); // derivacije aktivacijskih funkcija 2.sloja modela
    df1  = ssGetPWorkValue(S, 6); // derivacije aktivacijskih funkcija 1.sloja regulatora
    df2  = ssGetPWorkValue(S, 7); // derivacije aktivacijskih funkcija 2.sloja regulatora
    reg1 = ssGetPWorkValue(S,18); // reg1 - izlazi prvog sloja regulatora
    reg2 = ssGetPWorkValue(S,19); // reg2 - izlazi drugog sloja regulatora
```

```
    //derivacije aktivacijskih funkcija neuronskog modela procesa
    if (f2m==1) for(i=0;i<S2m;i++) df2m[i]=u1(i)*(1-u1(i));
    else if (f2m==3) for(i=0;i<S2m;i++) df2m[i]=(1-u1(i)*u1(i));
    else if (f2m==2) for(i=0;i<S2m;i++) df2m[i]=1;

    if (f1m==1) for(i=0;i<S1m;i++) df1m[i]=u2(i)*(1-u2(i));
    else if (f1m==3) for(i=0;i<S1m;i++) df1m[i]=(1-u2(i)*u2(i));
    else if (f1m==2) for(i=0;i<S1m;i++) df1m[i]=1;

    //derivacije aktivacijskih funkcija neuronskog regulatora
    // prvo se derivacije izračunate u prošlim koracima pomknu za jedan korak u prošlost
    // naredbom memmove
    memmove(&df2[S2],   df2, (tau+nu-1)*S2  *sizeof(real_T));
    if (f2==1) for(i=0;i<S2;i++) df2[i]=reg2[i]*(1-reg2[i]);
    else if (f2==3) for(i=0;i<S2;i++) df2[i]=(1-reg2[i]*reg2[i]);
    else if (f2==2) for(i=0;i<S2;i++) df2[i]=1;

    memmove(&df1[S1],   df1, (tau+nu-1)*S1  *sizeof(real_T));
    if (f1==1) for(i=0;i<S1;i++) df1[i]=reg1[i]*(1-reg1[i]);
    else if (f1==3) for(i=0;i<S1;i++) df1[i]=(1-reg1[i]*reg1[i]);
    else if (f1==2) for(i=0;i<S1;i++) df1[i]=1;
}

void f_dymdup_dymdyp(SimStruct *S){
    // proračun izlaza iz neuronskog modela po njegovom regresoru
    real_T  *d2d1m,*dymdup,*df2m,*df1m,*dymdyp;
    int_T   i, j, n;
    df1m   = ssGetPWorkValue(S, 0); //derivacije aktivacijskih funkcija 1.sloja modela
    df2m   = ssGetPWorkValue(S, 1); //derivacije aktivacijskih funkcija 2.sloja modela
    d2d1m  = ssGetPWorkValue(S,13); //derivacije izlaza 2.sloja po izlazima 1.sloja modela
    dymdup = ssGetPWorkValue(S,14); //derivacije izlaza modela po ulazima (za prošle u)
    dymdyp = ssGetPWorkValue(S,15); //derivacije izlaza modela po ulazima (za prošle y)

    // zapamte se derivacije izračunate u prošlim koracima
    memmove(&dymdyp[nym],   dymdyp, (tau+ny-1)*nym *sizeof(real_T));
    memmove(&dymdup[num],   dymdup, (tau+ny-1)*num *sizeof(real_T));
    for (j=0; j<nym+num; j++){
        for (i=0; i<S1m; i++){
            if (j==0){
                d2d1m[i] = 0;
                for (n=0; n<S2m; n++){
                    d2d1m[i] += w2m[i*S2m + n]*df2m[n];
                }
            }
            if (j<nym) {
                if (i==0) dymdyp[j] = 0;
                dymdyp[j] += d2d1m[i]*df1m[i]*w1m[j*S1m + i];
            }
            else {
                if (i==0) dymdup[j-nym] = 0;
                dymdup[j-nym] += d2d1m[i]*df1m[i]*w1m[j*S1m + i];
            }
        }
    }
}

void f_dudthetaR_dudup_dudyp(SimStruct *S){
    real_T  *d2d1,*dudw2,*dudb2,*dudw1,*dudb1,*df2,*w2,*reg1,*df1,*r,*dudup,*w1,*dudyp;
    int_T   i, j, n, p, rr, k;
    w1     = ssGetPWorkValue(S, 2); // težine 1.sloja regulatora
    w2     = ssGetPWorkValue(S, 4); // težine 2.sloja regulatora
    df1    = ssGetPWorkValue(S, 6); // derivacije aktivacijskih funkcija 1.sloja regulatora
    df2    = ssGetPWorkValue(S, 7); // derivacije aktivacijskih funkcija 2.sloja regulatora
    dudw1  = ssGetPWorkValue(S, 8); // derivacije upr.signala po težinama 1.sloja regulatora
    dudb1  = ssGetPWorkValue(S, 9); // derivacije upr.signala po pragovima 1.sloja regulatora
    dudw2  = ssGetPWorkValue(S,10); // derivacije upr.signala po težinama 2.sloja regulatora
    dudb2  = ssGetPWorkValue(S,11); // derivacije upr.signala po pragovima 2.sloja regulatora
```

```
    d2d1   = ssGetPWorkValue(S,12); // derivacije izlaza 2.sloja po izlazima 1.sloja regulatora
    dudup  = ssGetPWorkValue(S,16); // derivacije izlaza regulatora po ulazima (za prošle u)
    dudyp  = ssGetPWorkValue(S,17); // derivacije izlaza regulatora po ulazima (za prošle y)
    reg1   = ssGetPWorkValue(S,18); // reg1 - izlazi prvog sloja regulatora
    r      = ssGetPWorkValue(S,22); // vektor regresora

    // proračun parcijane derivacije izlaza iz regulatora
    // po parametrima regulatora za (tau+nu) koraka u prošlost
    for (k=0; k<tau+nu; k++){
        for (j=0; j<ny+nu+1; j++){
            for (i = 0; i < S1; i++){
                if (j==0){
                    d2d1[i] = 0;
                    for (n=0; n<S2; n++){
                        d2d1[i] += df2[S2*k + n]*w2[S2*i+n];
                        dudw2[k*S2*S1 + S2*i + n] = df2[S2*k + n]*reg1[k*S1 + i];
                        if (i==0){
                            dudb2[k*S2 + n] = df2[k*S2 + n];
                        }
                    }
                }
                dudw1[S1*(ny+nu+1)*k+S1*j+i] = d2d1[i]*df1[S1*k + i]*r[(ny+nu+1)*k + j];
                if (j==0){
                    dudb1[S1*k + i] = df1[S1*k + i];
                }
            }
        }

        //proračun derivacije izlaza iz regulatora po njegovim ulazima
        for (p=1; p<=nu; p++){
            dudup[k*(nu+1) + p] = 0;
            for (i=0; i<S1; i++) {
                dudup[k*(nu+1) + p] += d2d1[i]*df1[S1*k + i]*w1[(ny+p)*S1+i];
            }
        }
        for (rr=0; rr<ny; rr++){
            dudyp[k*ny + rr] = 0;
            for (i=0; i<S1; i++){
                dudyp[k*ny + rr] += d2d1[i]*df1[S1*k + i]*w1[(1+rr)*S1+i];
            }
        }
    }
}

void f_dy_plus_dthetaR(SimStruct *S, int_T k){
    // proračun derivacija prema izrazu (8-3)
    real_T  *dy_plus_dw1,*dy_plus_db1,*dy_plus_dw2,*dy_plus_db2;
    real_T  *du_plus_dw1,*du_plus_db1,*du_plus_dw2,*du_plus_db2;
    real_T  *dudup,*dudyp,*dymdup, *dymdyp;
    real_T  *dymdw1,*dymdb1,*dymdw2,*dymdb2;
    int_T   i,j,p,rr;
    dymdw1      = ssGetPWorkValue(S,36); // derivacije izlaza modela po težinama 1.sloja regul.
    dymdb1      = ssGetPWorkValue(S,38); // derivacije izlaza modela po pragovima 1.sloja regul.
    dymdw2      = ssGetPWorkValue(S,39); // derivacije izlaza modela po težinama 2.sloja regul.
    dymdb2      = ssGetPWorkValue(S,40); // derivacije izlaza modela po pragovima 2.sloja regul.
    dudup       = ssGetPWorkValue(S,16); // derivacije izlaza regulatora po ulazima (za prošle u)
    dudyp       = ssGetPWorkValue(S,17); // derivacije izlaza regulatora po ulazima (za prošle y)
    dymdup      = ssGetPWorkValue(S,14); // derivacije izlaza modela po ulazima (za prošle u)
    dymdyp      = ssGetPWorkValue(S,15); // derivacije izlaza modela po ulazima (za prošle y)
    du_plus_dw1 = ssGetPWorkValue(S,23); // redna derivacija izlaza regulatora po parametrima w1
    du_plus_dw2 = ssGetPWorkValue(S,24); // redna derivacija izlaza regulatora po parametrima w2
    du_plus_db1 = ssGetPWorkValue(S,25); // redna derivacija izlaza regulatora po parametrima b1
    du_plus_db2 = ssGetPWorkValue(S,26); // redna derivacija izlaza regulatora po parametrima b2
    dy_plus_dw1 = ssGetPWorkValue(S,36); // redna derivacija izlaza modela po parametrima w1
    dy_plus_dw2 = ssGetPWorkValue(S,37); // redna derivacija izlaza modela po parametrima w2
    dy_plus_db1 = ssGetPWorkValue(S,38); // redna derivacija izlaza modela po parametrima b1
    dy_plus_db2 = ssGetPWorkValue(S,39); // redna derivacija izlaza modela po parametrima b2
```

```
    for (i=0; i<S1; i++){
        dy_plus_db1[k*S1+i] = 0;
        for (j=0; j<(nu+ny+1); j++){
            dy_plus_dw1[k*(nu+ny+1)*S1+j*S1+i] = 0;
            for (p=0; p<num; p++){
                if (k+p+1>=tau+num-1) continue;
                dy_plus_dw1[k*(nu+ny+1)*S1+j*S1+i] += dymdup[k*num+p] *
du_plus_dw1[(k+p+1)*(nu+ny+1)*S1+j*S1+i];
                if (j==0) dy_plus_db1[k*S1+i] += dymdup[k*num+p] * du_plus_db1[(k+p+1)*S1+i];
            }
            for (rr=0; rr<nym; rr++){
                if (k+rr+1>=tau+nym-1) continue;
                dy_plus_dw1[k*(nu+ny+1)*S1+j*S1+i] += dymdyp[k*nym+rr] *
dy_plus_dw1[(k+rr+1)*(nu+ny+1)*S1+j*S1+i];
                if (j==0) dy_plus_db1[k*S1+i] += dymdyp[k*nym+rr]*dy_plus_db1[(k+rr+1)*S1+i];
            }
        }
    }
    for (i=0; i<S2; i++){
        dy_plus_db2[k*S2+i] = 0;
        for (j=0; j<S1; j++){
            dy_plus_dw2[k*S2*S1+j*S2+i] = 0;
            for (p=0; p<num; p++){
                if (k+p+1>=tau+num-1) continue;
                dy_plus_dw2[k*S2*S1+j*S2+i] += dymdup[k*num+p] *
du_plus_dw2[(k+p+1)*S2*S1+j*S2+i];
                if (j==0) dy_plus_db2[k*S2+i] += dymdup[k*num+p] * du_plus_db2[(k+p+1)*S2+i];
            }
            for (rr=0; rr<nym; rr++){
                if (k+rr+1>=tau+nym-1) continue;
                dy_plus_dw2[k*S2*S1+j*S2+i] += dymdyp[k*nym+rr] *
dy_plus_dw2[(k+rr+1)*S2*S1+j*S2+i];
                if (j==0) dy_plus_db2[k*S2+i] += dymdyp[k*nym+rr] * dy_plus_db2[(k+rr+1)*S2+i];
            }
        }
    }
}

void f_du_plus_dthetaR(SimStruct *S){
    // proračun derivacija prema izrazu (8-4)
    real_T  *dudup,*dudyp,*dymdup,*dymdyp,*dudw1,*dudb1,*dudw2,*dudb2;
    real_T  *dy_plus_dw1,*dy_plus_db1,*dy_plus_dw2,*dy_plus_db2;
    real_T  *du_plus_dw1,*du_plus_db1,*du_plus_dw2,*du_plus_db2;
    int_T   i, j, k, p, rr, brojac;
    dudw1  = ssGetPWorkValue(S, 8); // derivacije upr.signala po težinama 1.sloja regulatora
    dudb1  = ssGetPWorkValue(S, 9); // derivacije upr.signala po pragovima 1.sloja regulatora
    dudw2  = ssGetPWorkValue(S,10); // derivacije upr.signala po težinama 2.sloja regulatora
    dudb2  = ssGetPWorkValue(S,11); // derivacije upr.signala po pragovima 2.sloja regulatora
    dudup       = ssGetPWorkValue(S,16); // derivacije izlaza regulatora po ulazima (za prošle u)
    dudyp       = ssGetPWorkValue(S,17); // derivacije izlaza regulatora po ulazima (za prošle y)
    dymdup      = ssGetPWorkValue(S,14); // derivacije izlaza modela po ulazima (za prošle u)
    dymdyp      = ssGetPWorkValue(S,15); // derivacije izlaza modela po ulazima (za prošle y)
    du_plus_dw1 = ssGetPWorkValue(S,23); // redna derivacija izlaza regulatora po parametrima w1
    du_plus_dw2 = ssGetPWorkValue(S,24); // redna derivacija izlaza regulatora po parametrima w2
    du_plus_db1 = ssGetPWorkValue(S,25); // redna derivacija izlaza regulatora po parametrima b1
    du_plus_db2 = ssGetPWorkValue(S,26); // redna derivacija izlaza regulatora po parametrima b2
    dy_plus_dw1 = ssGetPWorkValue(S,36); // redna derivacija izlaza modela po parametrima w1
    dy_plus_dw2 = ssGetPWorkValue(S,37); // redna derivacija izlaza modela po parametrima w2
    dy_plus_db1 = ssGetPWorkValue(S,38); // redna derivacija izlaza modela po parametrima b1
    dy_plus_db2 = ssGetPWorkValue(S,39); // redna derivacija izlaza modela po parametrima b2
    brojac = ssGetIWorkValue(S, 0);      // trajanje simulacije (u koracima)
```

```
    for  (k=tau+nu-1; k>=0; k--){
        if (brojac<k) continue;

        for (i=0; i<S1; i++){
            du_plus_db1[k*S1 + i] = dudb1[k*S1 + i];
            for (j=0; j<(nu+ny+1); j++){
                du_plus_dw1[k*(nu+ny+1)*S1+j*S1+i] = dudw1[k*(nu+ny+1)*S1 + j*S1 + i];
                for (p=1; p<=nu; p++){
                    if (k+p>=tau+nu) continue;
                    du_plus_dw1[k*(nu+ny+1)*S1+j*S1+i] +=
dudup[k*(nu+1)+p]*du_plus_dw1[(k+p)*(nu+ny+1)*S1+j*S1+i];
                    if (j==0) du_plus_db1[k*S1+i] += dudup[k*(nu+1)+p] *
du_plus_db1[(k+p)*S1+i];
                }
                for (rr=0; rr<=(ny-1); rr++){
                    if (k+rr>=tau+ny-1) continue;
                    if (rr==0 && i==0 && j==0) f_dy_plus_dthetaR(S, k+rr);
                    du_plus_dw1[k*(nu+ny+1)*S1+j*S1+i] +=
dudyp[k*ny+rr]*dy_plus_dw1[(k+rr)*(nu+ny+1)*S1+j*S1+i];
                    if (j==0) du_plus_db1[k*S1+i] += dudyp[k*ny+rr] * dy_plus_db1[(k+rr)*S1+i];
                }
            }
        }
        for (i=0; i<S2; i++){
            du_plus_db2[k*S2+i] = dudb2[k*S2+i];
            for (j=0; j<S1; j++){
                du_plus_dw2[k*S2*S1+j*S2+i] = dudw2[k*S2*S1+j*S2+i];
                for (p=1; p<=nu; p++){
                    if (k+p>=tau+nu) continue;
                    du_plus_dw2[k*S2*S1+j*S2+i] += dudup[k*(nu+1)+p] *
du_plus_dw2[(k+p)*S2*S1+j*S2+i];
                    if (j==0) du_plus_db2[k*S2+i] += dudup[k*(nu+1)+p] *
du_plus_db2[(k+p)*S2+i];
                }
                for (rr=0; rr<=(ny-1); rr++){
                    if (k+rr>=tau+ny-1) continue;
                    du_plus_dw2[k*S2*S1+j*S2+i] += dudyp[k*ny+rr] *
dy_plus_dw2[(k+rr)*S2*S1+j*S2+i];
                    if (j==0) du_plus_db2[k*S2+i] += dudyp[k*ny+rr] * dy_plus_db2[(k+rr)*S2+i];
                }
            }
        }
    }
}

void f_dI_dthetaR(SimStruct *S){
    // proračun derivacija prema izrazu (8-2)
    int_T   i, j, k;
    real_T  *dI_dw1, *dI_db1, *dI_dw2, *dI_db2;
    real_T  *ref, *yp;
    real_T  *dy_plus_dw1,*dy_plus_db1,*dy_plus_dw2,*dy_plus_db2;

    InputRealPtrsType uPtrs3=ssGetInputPortRealSignalPtrs(S,2);  //referenca
    InputRealPtrsType uPtrs6=ssGetInputPortRealSignalPtrs(S,5);  //izlaz iz procesa

    yp          = ssGetPWorkValue(S,21); // vrijednosti izlaza procesa
    dI_dw1      = ssGetPWorkValue(S,31); // derivacije kriterijske funkcije po parametrima w1
    dI_db1      = ssGetPWorkValue(S,32); // derivacije kriterijske funkcije po parametrima b1
    dI_dw2      = ssGetPWorkValue(S,33); // derivacije kriterijske funkcije po parametrima w2
    dI_db2      = ssGetPWorkValue(S,34); // derivacije kriterijske funkcije po parametrima b2
    ref         = ssGetPWorkValue(S,35); // vrijednosti odziva referentnog modela
    dy_plus_dw1 = ssGetPWorkValue(S,36); // redna derivacija izlaza modela po parametrima w1
    dy_plus_dw2 = ssGetPWorkValue(S,37); // redna derivacija izlaza modela po parametrima w2
    dy_plus_db1 = ssGetPWorkValue(S,38); // redna derivacija izlaza modela po parametrima b1
    dy_plus_db2 = ssGetPWorkValue(S,39); // redna derivacija izlaza modela po parametrima b2
```

```
    //inicijalizacija (postavljanje derivacija kriterijske funkcije na nulu)
    for(i=0;i<S1;i++){
        dI_db1[i]=0;
        for (j=0;j<(nu+ny+1);j++){
            dI_dw1[j*S1+i]=0;
        }
    }
    for(i=0;i<S2;i++){
        dI_db2[i]=0;
        for (j=0;j<S1;j++){
            dI_dw2[j*S2+i]=0;
        }
    }
    //proračun derivacija kriterijske funkcije prema izrazu (8-2)
    for(i=0;i<S1;i++){
        for (k=0; k<tau; k++){
            for (j=0;j<(nu+ny+1);j++)
                dI_dw1[j*S1+i]+=(yp[k]-ref[k])*dy_plus_dw1[k*S1*(nu+ny+1) + j*S1 + i];
            dI_db1[i]+=(yp[k]-ref[k])*dy_plus_db1[k*S1 + i];
        }
    }
    for(i=0;i<S2;i++){
        for (k=0; k<tau; k++){
            for (j=0;j<S1;j++)
                dI_dw2[j*S2+i]+=(yp[k]-ref[k])*dy_plus_dw2[k*S2*S1 + j*S2 + i];
            dI_db2[i]+=(yp[k]-ref[k])*dy_plus_db1[k*S2 + i];
        }
    }
}

void simulacija_NM(SimStruct *S){
    // simulacija neuronskog regulatora
    int_T   i, j;
    real_T  *w1,*w2,*b1,*b2,*r,*reg1,*reg2,*y;
    w1   = ssGetPWorkValue(S, 2); // težine 1.sloja regulatora
    b1   = ssGetPWorkValue(S, 3); // pragovi 1.sloja regulatora
    w2   = ssGetPWorkValue(S, 4); // težine 2.sloja regulatora
    b2   = ssGetPWorkValue(S, 5); // pragovi 2.sloja regulatora
    reg1 = ssGetPWorkValue(S,18); // reg1 - izlazi prvog sloja regulatora
    reg2 = ssGetPWorkValue(S,19); // reg2 - izlazi drugog sloja regulatora
    r    = ssGetPWorkValue(S,22); // vektor regresora
    y    = ssGetOutputPortRealSignal(S,0); // veličina koja se šalje na 1. izlaz bloka

    memmove(&reg1[S1],  reg1,   (tau+nu-1)*(S1) *sizeof(real_T));
    //prvi sloj
    for(i=0;i<S1;i++)   {
        reg1[i]=0;
        for(j=0;j<nu+ny+1;j++)
            reg1[i]+=w1[j*S1+i]*r[j];
        reg1[i]=reg1[i]+b1[i];

        if (f1==1) reg1[i]=LOGSIG(reg1[i]);
        else if (f1==3) reg1[i]=TANSIG(reg1[i]);
        else if (f1==2) reg1[i]=reg1[i];
    }
    //drugi sloj
    for(i=0;i<S2;i++)   {
        reg2[i]=0;
        for(j=0;j<S1;j++)
            reg2[i]+=w2[j*S2+i]*reg1[j];
        reg2[i]=reg2[i]+b2[i];

        if (f2==1) reg2[i]=LOGSIG(reg2[i]);
        else if (f2==3) reg2[i]=TANSIG(reg2[i]);
        else if (f2==2) reg2[i]=reg2[i];
    }
    y[0]=reg2[0];
}
```

```
/*********************************************************/
/*************  Funkcija MDL_OUTPUTS  ********************/
/*********************************************************/
static void mdlOutputs(SimStruct *S,int_T tid){
    int_T   i,brojac,j,flag_uci;
    real_T  *r;
    real_T  *w1,*b1,*w2,*b2;
    real_T  *up, *yp,*ref,*y1;
    real_T  *delta_w1,*delta_w2,*delta_b1,*delta_b2, *dI_dw1,*dI_dw2,*dI_db1,*dI_db2;
    InputRealPtrsType uPtrs3=ssGetInputPortRealSignalPtrs(S,2); //referenca
    InputRealPtrsType uPtrs4=ssGetInputPortRealSignalPtrs(S,3); //proslo stanje regulatora
    InputRealPtrsType uPtrs5=ssGetInputPortRealSignalPtrs(S,4); //flag_uci
    InputRealPtrsType uPtrs6=ssGetInputPortRealSignalPtrs(S,5); //izlaz iz procesa
    y1 = ssGetOutputPortRealSignal(S,1);  // veličina koja se šalje na 2. izlaz bloka

    w1       = ssGetPWorkValue(S, 2); // težine 1.sloja regulatora
    b1       = ssGetPWorkValue(S, 3); // pragovi 1.sloja regulatora
    w2       = ssGetPWorkValue(S, 4); // težine 2.sloja regulatora
    b2       = ssGetPWorkValue(S, 5); // pragovi 2.sloja regulatora
    up       = ssGetPWorkValue(S,20); // prošle vrijednosti izlaza regulatora
    yp       = ssGetPWorkValue(S,21); // prošle vrijednosti izlaza procesa
    r        = ssGetPWorkValue(S,22); // vektor regresora
    delta_w1 = ssGetPWorkValue(S,27); // promjena parametra w1
    delta_w2 = ssGetPWorkValue(S,28); // promjena parametra w2
    delta_b1 = ssGetPWorkValue(S,29); // promjena parametra b1
    delta_b2 = ssGetPWorkValue(S,30); // promjena parametra b2
    dI_dw1   = ssGetPWorkValue(S,31); // derivacije kriterijske funkcije po parametrima w1
    dI_db1   = ssGetPWorkValue(S,32); // derivacije kriterijske funkcije po parametrima b1
    dI_dw2   = ssGetPWorkValue(S,33); // derivacije kriterijske funkcije po parametrima w2
    dI_db2   = ssGetPWorkValue(S,34); // derivacije kriterijske funkcije po parametrima b2
    ref      = ssGetPWorkValue(S,35); // izlaz referentnog modela
    brojac   = ssGetIWorkValue(S, 0); // trajanje simulacije (u koracima)
    flag_uci=(int_T) **uPtrs5;        // zastavica učenja

    //inicijalno se sva prosla stanja izlaza postavljaju na trenutnu vrijednost izlaza
    if (brojac==0)  {
        for(i=0; i<ny+tau; i++)
            yp[i]=u6(0);
        ref[1]=u6(0);  // definiranje prošlih vrijednosti referentnog modela
        ref[2]=ref[1];
    }
    brojac++;
    ssSetIWorkValue(S, 0, brojac);

    //filtriranje reference (referentni model) - filtar drugog reda
    ref[0]=u3(0)*(1+p1+p2)-p1*ref[1]-p2*ref[2];
    y1[0]=ref[0]; // izlaz referentnog modela šalje se na 2.izlaz bloka
    memmove(&ref[1], ref, (tau+1)*sizeof(real_T));

    //azuriranje vektora prošlih stanja ulaza i izlaza
    memmove(&up[1], up, (nu-1)*sizeof(real_T));
    memmove(&yp[1], yp, (tau+ny-1)*sizeof(real_T));
    up[0]=u4(0);            //proslo stanje upravljacke velicine
    yp[0]=u6(0);            //sadasnje stanje procesa

    //formiranje regresijskog vektora
    memmove(&r[nu+ny+1], r,  (nu+ny+1)*(tau+nu-1) *sizeof(real_T));
    r[0]=u3(0);
    memmove(&r[1],      yp, (ny)*sizeof(real_T));
    memmove(&r[1+ny],   up, (nu)*sizeof(real_T));

    simulacija_NM(S);
    f_df(S);
    f_dymdup_dymdyp(S);
```

```
    if (flag_uci==1) {
        f_dudthetaR_dudup_dudyp(S);
        f_du_plus_dthetaR(S);
        f_dI_dthetaR(S);

        // proračun promjena parametara u smjeru traženja minimuma - izraz (3-3)
        for(i=0; i<S1; i++){
            delta_b1[i]=-alfa1*dI_db1[i] + gama*delta_b1[S1+i];
            for(j=0; j<(nu+ny+1); j++){
                delta_w1[S1*j+i]=-alfa1*dI_dw1[S1*j+i] + gama*delta_w1[(nu+ny+1)*S1+S1*j+i];
            }
        }
        for(i=0; i<S2; i++){
            delta_b2[i]=-alfa2*dI_db2[i] + gama*delta_b2[S2+i];
            for(j=0; j<S1; j++){
                delta_w2[S2*j+i]=-alfa2*dI_dw2[S2*j+i] + gama*delta_w2[S1*S2+S2*j+i];
            }
        }
        //ovdje odmah zapišem trenutne delte težina kao prošle za slijedeći korak
        memmove(&delta_w1[(nu+ny+1)*S1],    delta_w1,   (nu+ny+1)*S1    *sizeof(real_T));
        memmove(&delta_w2[S2*S1],           delta_w2,   (S2*S1)         *sizeof(real_T));
        memmove(&delta_b1[S1],              delta_b1,   (S1)            *sizeof(real_T));
        memmove(&delta_b2[S2],              delta_b2,   (S2)            *sizeof(real_T));

        // ažuriranje parametara regulatora - izraz (3-1)
        for(i=0; i<S1; i++){
            b1[i]=b1[i]+delta_b1[i];
            for(j=0; j<(nu+ny+1); j++){
                w1[S1*j+i]+=delta_w1[S1*j+i];
            }
        }
        for(i=0; i<S2; i++){
            b2[i]=b2[i]+delta_b2[i];
            for(j=0; j<S1; j++){
                w2[S2*j+i]+=delta_w2[S2*j+i];
            }
        }
    }
}

/*********************************************************/
/***************  Funkcija MDL_TERMINATE  ****************/
/*********************************************************/
static void mdlTerminate(SimStruct *S){
    // oslobađanje ranije rezervirane memorije
    int_T   i;
    for (i=0;i<ssGetNumPWork(S);i++){
        free(ssGetPWorkValue(S,i));
    }
}

#ifdef MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

LITERATURA

[1] Ivan Petrović, Nedjeljko Perić: *INTELIGENTNO UPRAVLJANJE SUSTAVIMA – 2.dio: neuronsko upravljanje., 2004/2005*

[2] Ivan Petrović i suradnici: *INTELIGENTNO UPRAVLJANJE SUSTAVIMA - Upute za laboratorijske vježbe - 2. dio: Upravljanje primjenom neuronskih mreža*

[3] The MathWorks, Inc. 1998-2006.: *Simulink, Simulation and Model-Based Design,WRITING S-FUNCTIONS*

[4] The MathWorks, Inc. 1998-2006.: *Simulink, Simulation and Model-Based Design, USING SIMULINK*

SAŽETAK

Adaptivni Neuronski Regulator s Referentnim Modelom

U novije se vrijeme intenzivno razvija teorija neuronskih mreža. Istodobno se istražuju i mogućnosti njihove primjene. Jedno od značajnijih područja primjene neuronskih mreža jest identifikacija i upravljanje nelinearnim procesima.

U ovom radu opisana je primjena neuronskih mreža u identifikaciji i upravljanju procesima. U sklopu zadatka izrađen je neuronski model nelinearnog procesa i inverzni neuronski model istog procesa. Obrađene su osnovne strukture neuronskog upravljanja zasnovane na inverznom modelu procesa te su one provjerene na simulacijskom modelu nelinearnog procesa magnetske levitacije.

Centralni dio rada obrađuje metodologiju adaptacijskog algoritma neuronskog regulatora odnosno opisuje se podešavanje parametara takvog regulatora posrednim on-line načinom učenja.

Ključne riječi: neuronske mreže, adaptivno upravljanje, nelinearan proces, identifikacija

ABSTRACT

Model Reference Adaptive Control Based on Neural Networks

The neural network theory has recently been considerably developed. At the same time, the possibilities of their practical applications are examined too. One of the most significant areas where neural networks can be applied is the area of identification and control of non-linear processes. This paper describes how neural networks can be used in process identification and control. The neural model of nonlinear process has been made, as well as the inverse model of the same process. Essential control structures involving neural networks based on inverse process model are discussed. Those control structures are tested on the simulation model of non-linear magnetic levitation process.

The main thesis part deals with the methodology of model reference adaptive control based on neural networks. The algorithm for indirect on-line tuning of neural networks weights is presented.

Keywords: neural network, adaptive control, nonlinear process, identification

POPIS UPOTREBLJAVANIH OZNAKA

y_1 udaljenost donjeg magneta od donje zavojnice [cm];

y_2 udaljenost gornjeg magneta od gornje zavojnice [cm];

F_{u11} sila između donje zavojnice i donjeg magneta [N];

F_{u12} sila između donje zavojnice i gornjeg magneta [N];

F_{u21} sila između gornje zavojnice i donjeg magneta [N];

F_{u22} sila između gornje zavojnice i gornjeg magneta [N];

F_{m12} sila između dva magneta [N];

F_{tr} funkcional trenja baziran na LuGre modelu [N].

y_{12} udaljenost dva magneta [cm];

y_c udaljenost gornje i donje zavojnice [cm].

Θ skup parametara neuronske mreže (neuronskog modela)

Θ^* optimalne vrijednosti parametara

$\Im(\Theta)$ kriterijska funkcija (kriterij kakvoće)

$e(v,\Theta)$ vektor predikcijske pogreške za *v*-ti vektor mjernih podataka;

$e^*(\Theta)$ ukupna predikcijska pogreška na čitavom skupu mjernih podataka

k indeks iteracije, korak izvođenja

$y(k)$ izlazni signal procesa

$\hat{y}(k)$ izlaznih signala modela procesa

$y_r(k)$ referentni signal

$u(k)$ upravljački signal

$R_{xx_n}(\tau)$ normirana autokorelacijska funkcija

$R_{xy_n}(\tau)$ normirana međukorelacijska funkcija

na broj korištenih prošlih vrijednosti izlaznog signala procesa u regresijskom vektoru *(na≡ny)*

nb broj korištenih prošlih vrijednosti upravljačkog signala u regresijskom vektoru *(nb≡nu)*

S1 broj neurona u skrivenom sloju neuronske mreže

S2 broj neurona u izlaznom sloju neuronske mreže

T vrijeme diskretizacije

ζ relativni koeficijent prigušenja

t_m vrijeme prvog maksimuma

ŽIVOTOPIS

Rođen sam 8.travnja 1981.g. u Zagrebu. Osnovnu školu završio sam u Zagrebu te 1995. godine upisujem *II. ekonomsku školu* u Zagrebu. Na ekonomskoj školi maturirao sam 1999. godine sa odličnim uspjehom. Iste godine upisao sam redovni studij na *Fakultetu elektrotehnike i računarstva* u Zagrebu. Na završnim godinama studija opredjeljujem se za smjer Automatika na Zavodu za automatiku i procesno računarstvo (današnji Zavod za automatiku i računalno inženjerstvo), a za svog mentora izabirem prof.dr.sc. Nedjeljka Perića. Tijekom studija honorarno sam radio na televiziji Nova TV gdje sam godinu i pol obavljao poslove voditelja programa u režiji, a nakon toga tri mjeseca bio sam pomoćnik voditelja tehničke službe u istoj firmi.
Civilni vojni rok odslužio sam 2006. godine na općinskom sudu u Zagrebu.