ELKHA, *Vol. 11, No.1, April 2019, pp. 12 - 18*
ISSN: 1858-1463 (print), 2580-6807 (online)

Implementasi Kendali PID Menggunakan Jaringan Syaraf Tiruan *Backpropagation*

Johanes Nico Sukamto[1)] dan Gunawan Dewantoro[2)]

[1, 2)] Fakultas Teknik Elektronika dan Komputer, Universitas Kristen Satya Wacana, Indonesia
email: [1)] johanesnicoo@gmail.com, [2)] gunawan.dewantoro@staff.uksw.edu

***Abstrak*– Jaringan syaraf tiruan adalah salah satu representasi dari *Artificial Intelligence* yang dapat melatih suatu sistem menjadi cerdas. Pada penelitian ini, jaringan syaraf tiruan akan dilatih dengan metode *Backpropagation* untuk menggantikan pengendali konvensional, yaitu pengendali PID yang diimplementasikan dengan Matlab. Dataset pelatihan jaringan syaraf tiruan diambil dari input dan output pengendali PID pada sebuah sistem *closed loop*. Dengan *setpoint* dan *plant* yang sama, jaringan syaraf tiruan dapat memiliki unjuk kerja yang menyerupai pengendali PID. Pengujian dilakukan dengan *tools* simulink di Matlab. Hasil data yang dilatih dapat dilihat dari grafik dan parameter pelatihan. Pada parameter pelatihan dapat menunjukkan jumlah *epochs, validation checks* dan *gradient descent,* sedangkan hasil grafik dapat menjelaskan informasi *Performance, Training State dan Regression*. Percobaan dilakukan dua kali, dengan gangguan luar dan tanpa gangguan luar. Hasil percobaan akan membandingan kestabilan dan kehandalan terhadap gangguan dari luar dibandingkan dengan pengendali PID.**

Kata Kunci: Artificial Intelligence, jaringan syaraf tiruan, backpropagation, pengendali PID

I. PENDAHULUAN

Pengendali *Proportional-Integral-Derivative* (PID) adalah pengendali konvensional yang paling umum dipakai di dunia industri. Sementara itu, perkembangan teknologi yang pesat membuat implementasi kecerdasan buatan (*Artificial Intelligence*) semakin sentral seiring dengan berkembangnya era revolusi industri 4.0 [1]. Beberapa aplikasi kecerdasan buatan telah banyak dipakai sebagai pengendali cerdas. Konsep logika fuzzy sebagai kendali cerdas telah diterapkan pada berbagai aplikasi robotika [2],[3],[4]. Pada desain kendali logika fuzzy, masukan mengalami fuzzifikasi sehingga menjadi variabel fuzzy. Masukan ini merupakan pembacaan sensor sesuai dengan aplikasi yang dipakai. Inferensi logika fuzzy dapat menggunakan metode Mamdani, Sugeno ataupun yang lainnya. Setiap himpunan fuzzy masukan menggunakan sejumlah membership function dan rule base. Hasil keluarannya kemudian mengalami defuzzifikasi agar kembali menjadi bilangan crisp tunggal yang umumnya diperoleh dengan menggunakan metode *Center of Gravity* (CoG). Nilai crisp keluaran ini digunakan sebagai penggerak aktuator robot agar dapat menyelesaikan misi yang diharapkan. Penelitian di atas masih merupakan simulasi dengan tool berbasis Matlab dan Simulink. Jaringan syaraf tiruan (JST) adalah salah satu metode dari kecerdasan buatan di mana *Artificial Neural Network* (ANN) dapat mengolah sejumlah data. Beberapa penelitian telah mencoba menerapkan ANN sebagai pengendali cerdas. JST *B-spline* berbasis mikrokontroler ATmega16 telah digunakan untuk mengendalikan kecepatan DC motor *brushless* [5],[6]. *B-spline* merupakan salah satu JST dengan jenis *Associative Memory Networks* (AMN) yang mampu mengendalikan suatu sistem proses secara *real-time*. JST *B-spline* yang digunakan memiliki variasi orde 1, 2, dan 3. Sinyal pengendalian *plant* merupakan penjumlahan dari keluaran *gain* proporsional dan JST *B-spline*. Selain itu, JST juga telah dimodifikasi agar dapat bekerja secara adaptif pada robot manipulator [7], [8], [9]. Pada simulasi yang dilakukan, kendali *full state feedback* dan *output feedback* digunakan pada sistem yang dikendalikan. Unjuk kerja dari JST adaptif dapat menjamin sinyal sistem lingkar tertutup dapat terbatas (*bounded*). Seiring dengan berkembangnya perangkat keras komputasi, maka mulai banyak penggunaan JST berlapis banyak atau *deep learning*. Pada penelitian [10] dan [11], algoritma *deep learning* dipakai untuk melatih jaringan *deep belief framework* untuk merancang pengendali. Dataset pelatihan berasal dari masukan dan keluaran pengendali PID. Jaringan yang dipakai terdiri dari dua lapisan, masing-masing terdiri dari 50 *neuron*. Karena dalamnya *node* pada jaringan ini, maka dibutuhkan kerja komputasi yang cukup intensif dan membutuhkan sumber daya yang memadai.

Pada penelitian ini, JST yang ringan akan akan dieksplorasi untuk mengetahui potensinya sebagai pengendali dengan data pelatihan dari pengendali PID. Pengendali PID dipilih sebagai acuan karena saat ini masih menjadi pilihan utama sebagai pengendali konvensional di level industri. Berbagai unjuk kerja kendali digunakan sebagai parameter pembandingan, termasuk tanggapan waktu dan kehandalan terhadap gangguan luar. Apabila unjuk kerja JST dapat menyamai pengendali PID, maka pengembangan lebih lanjut akan memungkinkan untuk menerapkan kendali cerdas pad sebuah sistem proses.

II. METODOLOGI PENELTIAN

2.1. *Backpropagation*

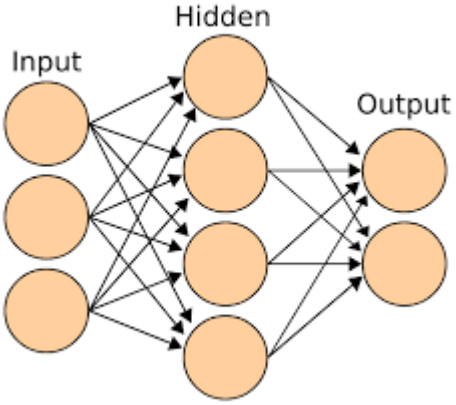
JST *backpropagation* merupakan suatu jaringan dengan pelatihan terbimbing di dalamnya terdapat lapisan-lapisan neuron [12]. Jaringan ini membaca galat keluaran sebagai dasar perubahan nilai bobot-bobotnya dalam arah balik (backward). Untuk mendapatkan galat ini, maka harus dilakukan dulu tahap perambatan maju (*forward propagation*). Fungsi aktivasi yang digunakan pada umumnya adalah fungsi *tansig*, *logsig*, dan *purelin*, karena ketiganya memenuhi syarat sifat kontinyu, dapat

Manuscript received 05-12-2018; revised 02-02-2019; accepted 05-02-2019

diferensialkan, dan bukan fungsi turun. Setiap nilai data pada JST *backpropagation* akan diinisialisasi terlebih dahulu. Dataset yang akan dipakai dalam pelatihan diberikan dalam bentuk vektor. Tiap-tiap data masukan mempunyai target yang juga disajikan dalam bentuk vektor. Target acuan merupakan hasil pemetaan dari vektor masukan. Metode pelatihan yang dipakai merupakan proses pembelajaran dalam rangka mengenali data kemudian direpresentasikan dalam bentuk nilai bobot-bobot. Secara garis besar, terdapat 3 tahap dalam pelatihan jaringan syaraf tiruan *backpropagation*, yaitu tahap maju (*feed forward*), tahap balik (*backward*), dan tahap pengaturan bobot. Dalam tahap *feedforward*, vektor masukan dirambatkan maju dari lapisan masukan ke arah lapisan keluaran. Sebaliknya, pada tahap *backward* tiap-tiap unit keluaran menghitung galat keluaran yang merupakan selisih antara hasil keluaran aktual dengan target yang bersesuaian. Galat tersebut kemudian akan dirambatkan balik. Sedangkan tahap pengaturan bobot bertujuan untuk meminimalkan galat yang ada. Ketiga tahap tersebut berlangsung terus-menerus hingga memenuhi syarat penghentian (*stopping criterion*).

2.2. Arsitektur *Backpropagation*

Pada JST arsitektur *backpropagation*, arsitekturnya terdiri dari tiga lapisan, yaitu lapisan masukan, lapisan tersembunyi, dan lapisan keluaran seperti ditunjukkan oleh Gambar 1. Pada lapisan masukan belum terjadi proses penghitungan, namun pada lapisan masukan terjadi pengiriman sinyal masukan X ke lapisan tersembunyi. Pada lapisan tersembunyi dan lapisan keluaran terjadi proses penghitungan tergantung dari bobot dan bias tiap neuron. Hasilnya adalah penghitungan nilai keluaran dari lapisan tersembunyi dan lapisan keluaran berdasarkan fungsi aktivasi yang dipakai.

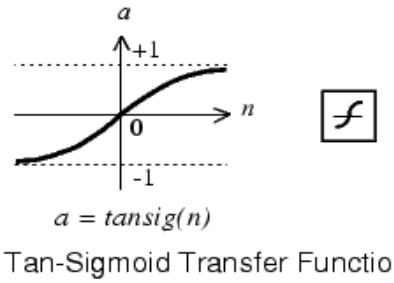


Gambar 1. Arsitektur *backpropagation*

2.3. Fungsi Aktivasi *Tansig* dan *Purelin*

Fungsi aktivasi merupakan fungsi umum yang akan dipakai untuk mentransfer masukan menuju keluaran yang diharapkan [13]. Keluaran dari fungsi aktivasi inilah yang akan mengatur besarnya nilai bobot. Pemilihan fungsi aktivasi banyak tergantung pada kebutuhan dan keluaran yang diinginkan. Pada penelitian ini fungsi aktivasi yang dipakai adalah fungsi aktivasi *tansig* dan linier/*purelin*.

Fungsi aktivasi *tansig* merupakan *sigmoid tangen* berlaku sebagai fungsi alih. Fungsi ini akan memetakan nilai masukan pada keluaran menggunakan rumus *hyperbolic tangen sigmoid*. Rentang keluaran dari fungsi ini adalah dari -1 hingga 1. Grafik dari fungsi *tansig* ditunjukkan oleh Gambar 2.

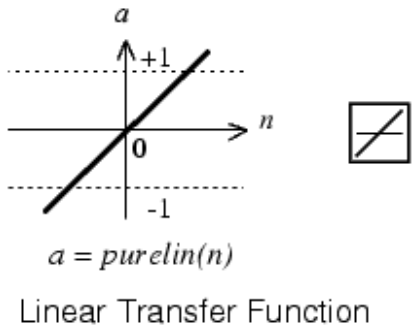


Gambar 2. Fungsi aktivasi *tansig*

Sedangkan rumus fungsi aktivasi *tansig* ditunjukkan oleh persamaan (1).

$$a = \tan sig(n) = \frac{2}{1+\exp(-2*n)} - 1 \qquad (1)$$

Fungsi aktivasi linier/*purelin* akan memetakan masukan ke keluaran secara proporsional. Grafik dari fungsi *purelin* ditunjukkan oleh Gambar 3.



Gambar 3. Fungsi aktivasi *purelin*

Sedangkan rumus fungsi aktivasi *purelin* ditunjukkan oleh persamaan (2).

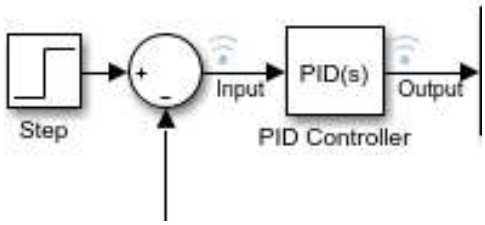
$$a = kn \qquad (2)$$

di mana *k* adalah sembarang konstanta

III. HASIL DAN PEMBAHASAN

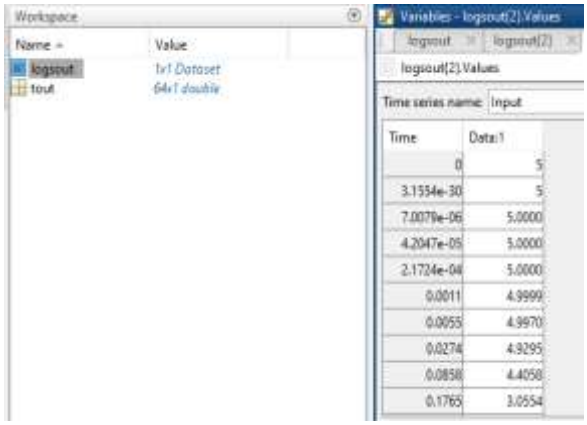
3.1. Pengambilan Data Masukan dan Keluaran

Pengambilan data masukan dan keluaran dari pengendali PID ditujukan untuk mendapatkan data masukan dan target yang akan dipakai untuk pelatihan jaringan syaraf tiruan *backpropagation*. Data masukan dan keluaran didapatkan menggunakan *tools view log signal* yang terdapat pada Simulink seperti ditunjukkan Gambar 4. Data yang didapat berupa matriks 63×1. Karena format matriks yang digunakan pada toolbox nntool adalah matriks baris (bukan matriks kolom) maka data pada matriks tersebut akan dirubah dengan fungsi *transpose* agar mendapatkan matriks 1×63. Data masukan dan target tersebut dipakai untuk melatih jaringan syaraf tiruan *backpropagation*

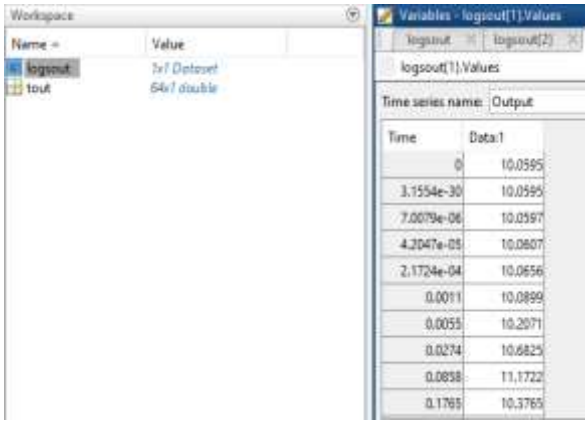


Gambar 4. *Tools Log View Signal*

Ketika Simulink dijalankan maka data masukan dan target akan terbaca pada *workspace* Matlab yang ditunjukkan pada Gambar 5 dan 6.



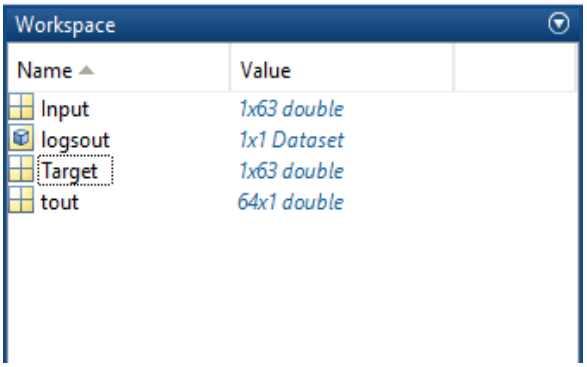
Gambar 5. *Workspace* dan data masukan



Gambar 6. *Workspace* dan data target

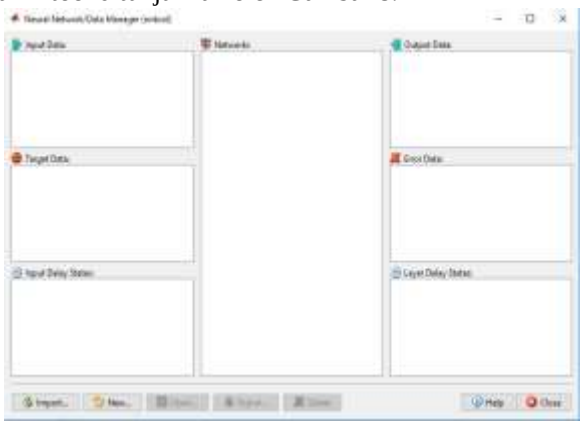
3.2. Pelatihan JST *Backpropagation*

Dataset yang telah didapat lalu dimasukan ke dalam *workspace* dengan masukan dari PID diberi nama input dan keluaran dari PID diberi nama target, seperti ditunjukkan Gambar 7.



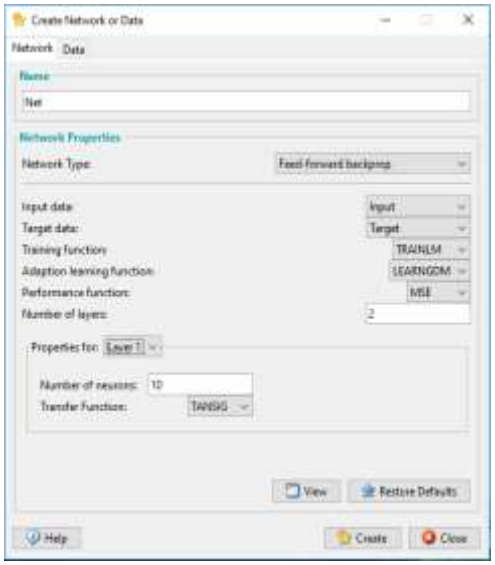
Gambar 7. *Input* dan Target di *Workspace*

Setelah mendapatkan data input dan target untuk pelatihan, pada *command window* Matlab digunakan toolbox nntool untuk memulai pelatihan JST *backpropagation.* Tampilan awal nntool ditunjukkan oleh Gambar 8.



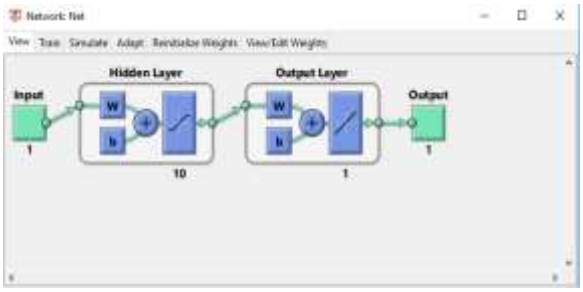
Gambar 8. *Tools* nntool di Matlab

Untuk melatih data pada nntool, data *input* dan target harus di *import* terlebih dahulu dari *workspace* ke dalam *Neural Network Data Manager*. Setelah itu buat *network* pelatihan dengan berbagai variasi pilihan seperti pada Gambar 9.



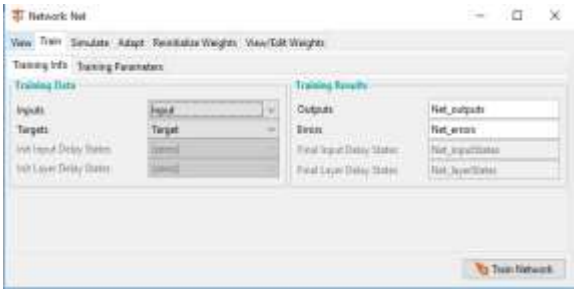
Gambar 9. *Tools new*

Gambar 9 menunjukan jendela pada *tools new*. Nama jaringan yang dipakai adalah net, tipe *network* yang dipakai adalah *feed-forward backpropagation*. Pelatihan ini memakai 2 lapisan, di mana lapisan pertama memakai fungsi aktivasi *tansig* dan lapisan kedua memakai fungsi aktivasi *pureline*. Setelah itu klik *create* dan *network* akan muncul pada jendela *Neural Network data Manager*, lalu jalankan *network* yang telah dibuat. Hasil dari jaringan yang dibuat ditunjukkan Gambar 10.



Gambar 10. Arsitektur jaringan yang dibuat

Perintah *train* pada jendela *network* berisi *tools training info* dan *training parameters*, seperti ditunjukkan Gambar 11. Pada *training info* digunakan untuk memasukan input dan target. Gambar 12 menunjukkan jendela *training parameters* di mana terdapat banyak fungsi yang akan mempengaruhi proses pelatihan.



Gambar 11. Pemilihan *input* dan target

Gambar 12. Parameter pelatihan

Epochs berfungsi untuk jumlah langkah pembelajaran pada jaringan syaraf tiruan *backpropagation*, *min_grad* atau *gradient descent* yaitu performa maksimum gradient, *max_fail* adalah nilai maksimum validasi kegagalan, *mu* adalah fungsi kontrol parameter algoritma yang dipakai untuk melatih *neural network* dan *time* adalah waktu maksimum yang digunakan untuk melatih jaringan dalam satuan detik.

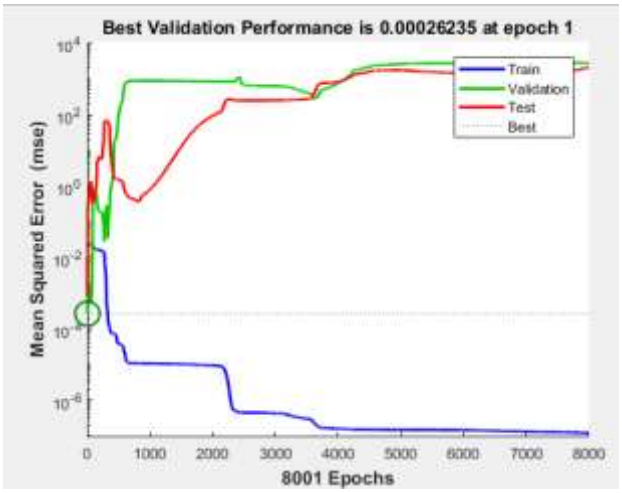
Parameter pelatihan yang digunakan adalah *epochs* sebesar 10.000, *min_grad* sebesar 1e-09, *max_fail* sebesar 8000 dan *mu* sebesar 0.001. Setelah parameter pelatihan diubah maka proses selanjutnya adalah melatih *neural network* dengan perintah *train network*.

Pelatihan jaringan syaraf tiruan *backpropagation* berhenti pada saat *Validation check/max_fail* mencapai nilai 8000 dan *epochs* berada dinilai 8001, seperti ditunjukkan seperti pada Gambar 13.

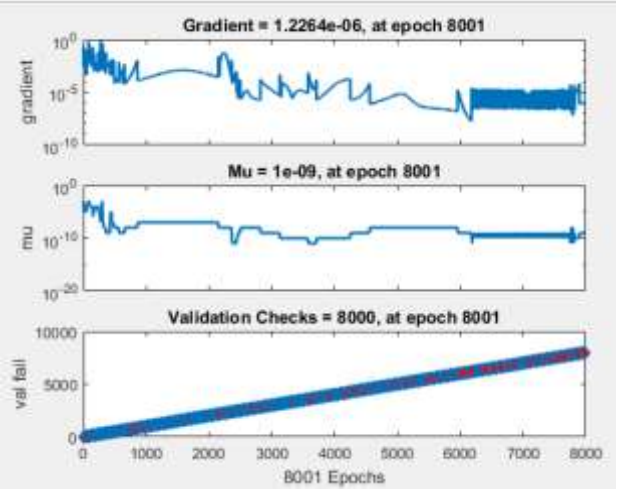
Gambar 13. Proses pelatihan jaringan

Pada jendela *plots* terdapat *perfomance*, *training state* dan *regression*, ketiga *plots* ini akan menunjukan hasil pelatihan menggunakan grafik.

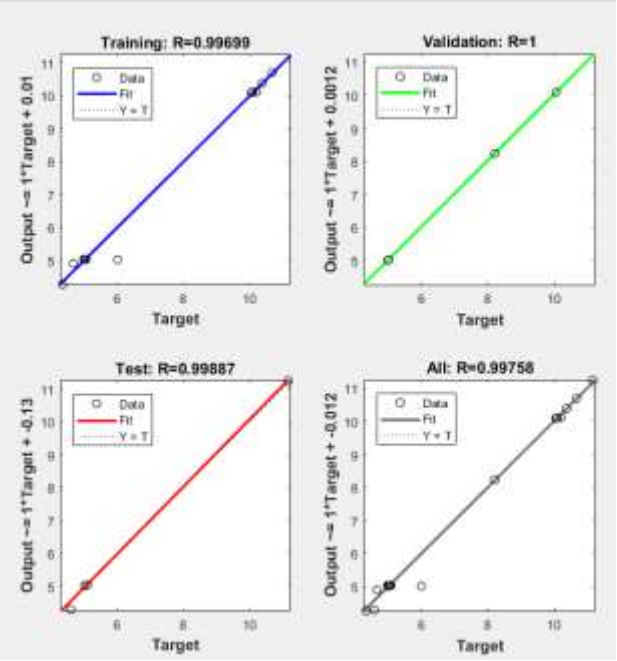
Gambar 14 menunjukkan bahwa *best validation performance* bernilai 0.00026235 pada *epoch* ke-1 dari 8001 *epochs*. Sementara itu Gambar 15 menunjukkan bahwa t*raining state gradient* terbesar bernilai 1.2264e-06 pada *epochs* ke 8001. Untuk mengetahui hasil regresi, maka sebaran data hasil pelatihan diletakkan di atas garis fit yang serupa dengan garis linier putus-putus Y=T. Terlihat bahwa data yang dilatih sudah menyerupai garis fit seperti ditunjukkan oleh Gambar 16.



Gambar 14. Plot unjuk kerja selama pelatihan



Gambar 15. Plot *training state*

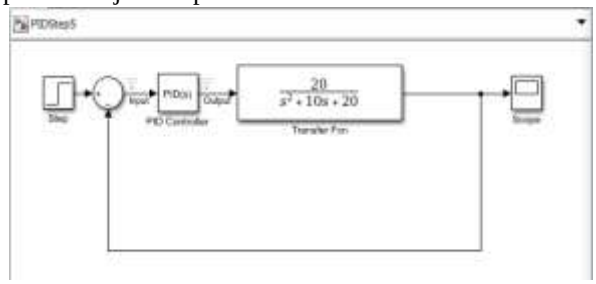


Gambar 16. Plot hasil regresi

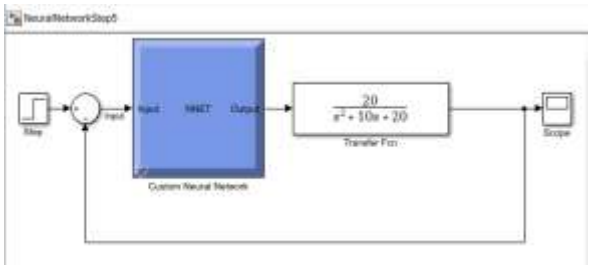
3.3 Pengujian Hasil JST *Backpropagation*

Setelah melatih JST *backpropagation* dengan dataset pelatihan yang ada, selanjutnya akan membandingkan kinerja JST *backpropagation* yang telah dilatih dengan pengendali PID. Pada tahap ini menggunakan *tools* Simulink untuk membandingkan PID dan JST *backpropagation*.

Untuk membuat ANN di *simulink*, *Network* yang telah dilatih dengan nama Net harus di *export* dahulu ke dalam *workspace* Matlab, lalu menggunakan *syntax* gensim(Net), ANN akan langsung terbentuk pada simulink. Di dalam simulink pengendali PID dan ANN harus dibuat serupa seperti ditunjukkan pada Gambar 17 dan 18.

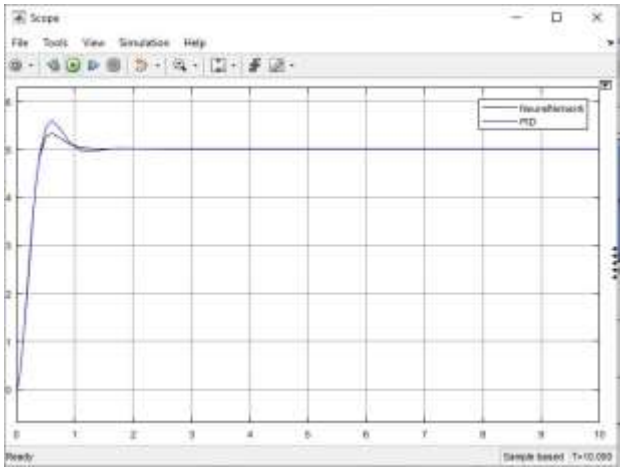


Gambar 17. Pengendali PID Pada *Simulink*



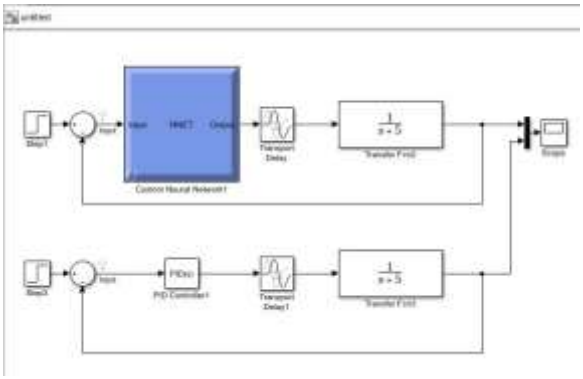
Gambar 18. ANN pada *Simulink*

Untuk membandingkan keduanya, *setpoint* dan *plant* harus serupa. *Setpoint* yang dipakai berupa *step* bernilai 5 dan *plant* yang dipakai berupa *transfer function* dari Motor DC dengan sistem orde 2. Grafik perbandingan keduanya dapat dilihat pada Gambar 19.



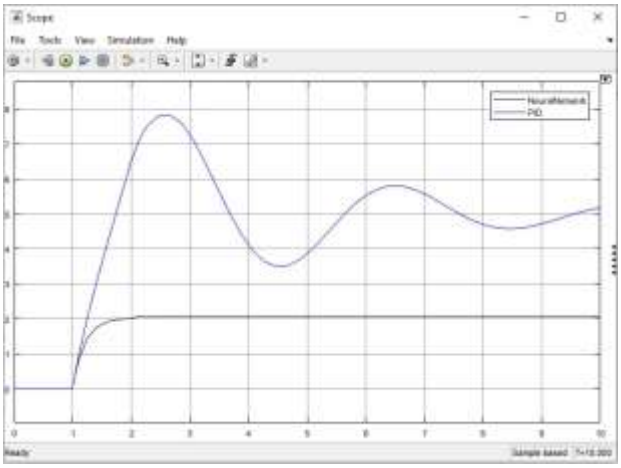
Gambar 19. Hasil *scope* PID dan *Neural Network*

Percobaan selanjutnya JST akan diuji dengan mengganti *plant* dengan sistem orde 1 dan menambahkan *transport delay* seperti ditunjukkan Gambar 20.



Gambar 20. PID dan *Neural Network* pada *Simulink*

Percobaan mengganti *plant* sistem orde 1 dan menambahkan *transport delay* menghasilkan grafik perbandingan seperti pada gambar 21.



Gambar 21. Hasil pengujian pada *plant* orde 1

Terlihat bahwa tanggapan waktu pada JST *backpropagation* memiliki unjuk kerja yang lebih baik, seperti ditunjukkan pada Tabel 1. Namun ketika mengujikan plant lain, JST memiliki tanggapan transien yang lebih baik namun memiliki *steady-state error* yang masih tinggi seperti ditunjukkan Tabel 2.

Tabel 1. Perbandingan Unjuk Kerja Sistem orde 2

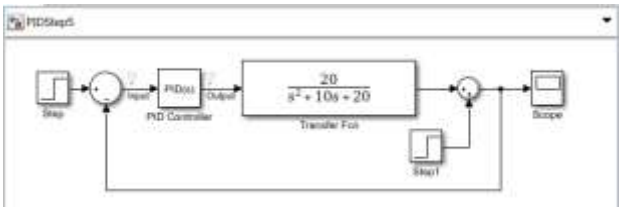
	***Rise time* (ms)**	***Overshoot* (%)**	***Steady State Error***
PID	289	11,798	0
JST	279	6,989	0

Tabel 2. Perbandingan Unjuk Kerja Sistem orde 1

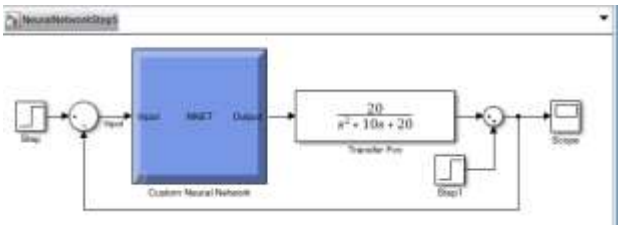
	***Rise time* (ms)**	***Overshoot* (%)**	***Steady State Error***
PID	533	71,552	0,12
JST	504	0,505	2,94

3.4. Pengujian Unjuk Kerja dengan Gangguan Luar

Pada saat membandingkan pengendali PID dengan JST terlihat bahwa JST lebih baik dari pengendali PID, terlihat pada Gambar 19. Pada pengujjian selanjutnya akan membandingkan kembali antara pengendali PID dan JST *backpropagation* saat mendapatkan gangguan dari luar. Gangguan yang diberikan berupa *step* yang memiliki nilai 10% dari *setpoint* dan gangguan diberikan pada detik ke-5, seperti ditunjukkan Gambar 22 dan 23.

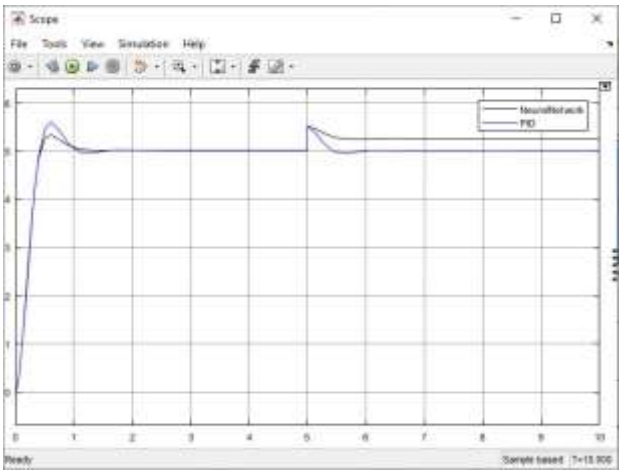


Gambar 22. Gangguan luar pada pengendali PID



Gambar 23. Gangguan luar pada JST *backpropagation*

Gangguan luar dibuat untuk melihat apakah jaringan syaraf tiruan *backpropagation* dapat mengkompensasi gangguan dari luar seperti yang bisa dilakukan oleh pengendali PID. Hasil yang didapat akan ditunjukkan pada Gambar 24.



Gambar 24. *Scope* hasil gangguan luar pengendali PID dan *Neural Network*

Dapat dilihat dari hasil simulasi diatas, bahwa pengendali PID dari menahan gangguan dari luar dan kembali ke *setpoint*. Sedangkan JST *backpropagation* juga dapat mengkompensasi gangguan dari luar, akan tetapi belum bisa mengembalikan tanggapan ke *setpoint*.

IV. KESIMPULAN

JST *backpropagation* dapat dilatih untuk menyerupai kerja dari pengendali PID konvensional. Berdasarkan unjuk kerja tanggapan waktu, prosentase overshoot pada JST *backpropagation* lebih kecil dibandingkan pengendali PID. Selain itu *rise time* JST *backpropagation* juga lebih cepat. Namun, JST *backpropagation* belum bisa mengembalikan nilai ke *setpoint* saat mengkompensasi gangguan dari luar. Jika plant yang digunakan saat pelatihan berbeda dengan saat pengujian, terlihat JST masih memiliki *steady-state error* yang tinggi. Ke depan, penelitian akan lebih difokuskan untuk memvariasikan dataset pelatihan, topologi jaringan, dan parameter pelatihan untuk mengoptimalkan kinerja JST.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Universitas Kristen Satya Wacana yang telah memberikan dukungan finansial pada penelitian ini.

DAFTAR PUSTAKA

[1] Lu Y., "Industry 4.0: A Survey on Technologies, Applications and Open Research Issues," *Journal of Industrial Information Integration*, vol. 6, pp. 1-10, June 2017.

[2] Wajiansyah A., Supriadi, Nur S. Wicaksono A.B., "Implementasi Fuzzy Logic Pada Robot Line Follower," *Jurnal Teknologi Informasi dan Ilmu Komputer*, vol. 5, no. 4, pp. 395 - 402, 2018.

[3] Timur M.B.B, Gaffar A.F.O, Wajiansyah A., "Desain dan Implementasi Kendali Cerdas untuk Robot Quadpod (Berkaki Empat) – Studi Kasus Robot Pemadam Api (RPA)," *Jurnal Teknologi Terpadu*, vol. 5, no.2, pp. 140-145, 2017.

[4] Dewantoro G., Susilo D., dan Adi P.P, "Implementasi Pengendali Logika Fuzzy pada Navigasi Robot Penjejak Dinding," *Majalah Ilmiah Teknologi Elektro*, vol. 16, no. 2, pp. 72-77, 2017.

[5] Widaningrum L., Setiyono B., Riyadi M. A., "Perancangan Kontroler Jaringan Syaraf Tiruan B-Spline Berbasis Mikrokontroler Atmega16 Sebagai Kendali Kecepatan Motor Brushless DC (BLDC)," *Transient*, vol. 6, no. 3, September 2017.

[6] Solanki S., "Brushless DC Motor Drive during Speed Regulation with Artificial Neural Network Controller." *International Journal of Engineering Research and Applications*, vol. 6, no. 6, pp. 01-05, 2016.

[7] He W., David A. O., Yin Z., Sun C.,"Neural Network Control of a Robotic Manipulator with Input Deadzone and Output Constraint," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 6, pp. 759-770, June 2016.

[8] Sun C.; He W., Hong J.,"Neural Network Control of a Flexible Robotic Manipulator Using the Lumped Spring-Mass Model," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 8, pp. 1863-1874, Aug. 2017.

[9] Artale V., Collotta M., Milazzo C., Pau G., Ricciardello A., "Real-Time System based on a Neural Network and PID Flight Control", *Applied Mathematics & Information Sciences*, Vol. 10 , No. 2, pp. 395-402, 2016.

[10] Cheon K., Kim J., Hamadche M., Lee D., "On Replacing PID Controller with Deep Learning Controller for DC Motor System," *Journal of Automation and Control Engineering*, vol. 3, no. 6, pp. 452-256, 2015.

[11] Zhou J., Shan W., Duan Z., Stability and case studies of linear continuous-time systems under deep belief network controllers," *Proc. in Chinese Automation Congress (CAC)*, 2017.

[12] Muhammad A., "On replacing PID controller with ANN controller for DC Motor Position Control," *International Journal of Research Studies in Computing*, vol. 2, no. 1, pp. 22-30, April 2013.

[13] Nurmila N., Sugiharto A., Sarwoko E. A.,"Algoritma Back Propagation Neural Network Untuk Pengenalan Pola Karakter Huruf Jawa," *Jurnal Masyarakat Informatika*, vol. 1, no. 1, pp. 1-10, December 2010.