Data-Driven Predictive Control Menggunakan Algoritma Nearest Neighbor Untuk Sistem Yang Tidak Stabil

Herlambang Saputra[*1], Ahmad Bahri Joni Malyan [2], Ahyar Supani [3], Indarto[4]

[1,2,3,4] Jurusan Teknik Komputer, Politeknik Negeri Sriwijaya
Jl. Srijaya Negara Palembang 30139 - Indonesia
e-mail: [1]herlambang@polsri.ac.id , [2]bahrijoni@yahoo.co.id
[3]ahyaryuli@yahoo.com, [4]indarto@polsri.ac.id

Abstrak

Sebuah sistem dapat dikendalikan dengan baik apabila menggunakan algoritma yang tepat dan handal. Algorima yang dipakai dalam penelitian ini adalah sebuah teknik data-driven dengan menggunakan Nearest Neighbor sebagai algoritma dalam menentukan input u terbaik untuk sistem. Teknik data-driven terbagi dua bagian yaitu matriks database A dan vektor informasi. Ketika matriks database telah terbentuk, langkah berikutnya adalah penghitungan vektor informasi secara real-time dibandingkan dengan matriks database tersebut dengan cara penghitungan jarak vektor informasi dan matriks database menggunakan l_2 −norm. Pada proses pemilihan jarak terdekat menggunakan dua buah pendekatan, yang pertama pendekatan jarak terdekat yang kedua menggunakan aturan-aturan baru. Hasil akhirnya adalah simulasi berupa grafik, penghitungan error dan waktu komputasi.
Kata kunci: *Data-driven, Nearest Neighbor, Vektor Informasi*

Abstract

A system can be controlled very well if the system used a proper algorithm. The algorithm in this reserach was a data-driven technique with nearest neighbor that used to get input u for the system. Data-driven technique consisted of two parts, the first part was a matrix database A and a second part was an information vector. When the matrix database was made, the next step was to compare real-time the information vector and matrix database with l_2 −norm. The nearest query vector used two kinds approach, first approach only used the nearest vector and the second approach used a new rule. The last for this reseach was a graph simulation, error calculated and computation time.

Keywords: *Data-driven, Information Vector, Nearest Neighbor*

1. PENDAHULUAN

Untuk mengendalikan sebuah robot atau sebuah sistem memerlukan algoritma *controller* yang tepat dan handal, salah satu metode yang sedang berkembang adalah metode *data-driven predictive control* dimana metode ini banyak sekali digunakan oleh industri kimia [1], [2]. Sebuah *controller* pada penelitian ini didesain dari observasi *input* dan *output* [3] yang kemudian dipetakan dan diformulasikan menjadi sebuah vektor matematik. Secara umum metode yang digunakan adalah sebuah pendekatan *nonlinear* dimana analisis secara matematik dalam identifikasi sistem tidak dilakukan. Metode ini sering juga disebut sebagai *machine learning* yang diterapkan pada sistem kendali dan diharapkan mampu memberikan solusi yang menjanjikan dalam menyelesaikan permasalahan dalam bidang sistem kendali. Secara khusus pada penelitian ini penggunaan algoritma *Nearest Neighbor* (NN) dipakai dalam melakukan proses *switching* sistem kendali. Konsep dasar dari cara kerja algortima ini adalah mencari vektor terdekat yang berada dalam *database* [4]. *Database* tersebut atau sering juga disebut sebagai *training data*

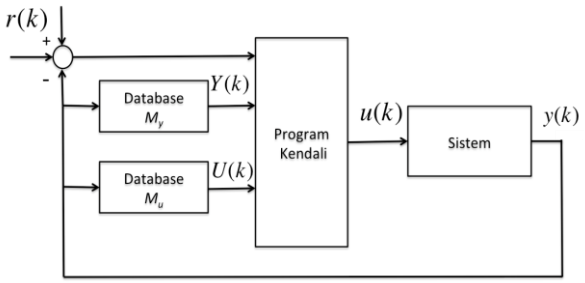
memiliki vektor-vektor yang telah disimpan dan kemudian dihitung jaraknya dengan vektor-vektor informasi. Untuk penghitungan jarak antara dua vektor *database* dan vektor informasi menggunakan $l_2 - norm$.

Selama proses pencarian dengan menghitung jarak vektor untuk mencari vektor terdekat dari *database*, semua *input* dan *output* terbaru yang diperoleh dari sistem kemudian disimpan kembali dan digunakan untuk proses penghitungan pada iterasi berikutnya. Untuk NN berperan dalam menentukan *input* $u(k)$ untuk sistem yang kemudian menghasilkan *output* $y(k)$. Keduanya *input* dan *output* tersebut disimpan kembali seperti yang telah dijelaskan pada awal paragraf.

Pada penelitian ini, para peneliti juga menggunakan metode *locality sensitive hashing* (LSH) sebagai salah satu metode yang diharapkan mampu memberikan solusi lebih baik untuk mempersingkat waktu perhitungan oleh NN. Metode LSH adalah sebuah metode praklasifikasi yang ditentukan klasifikasinya berdasarkan nilai *hash* yang tujuannya untuk mempersingkat pencarian didalam *database*. Kontribusi utama dari penelitian ini adalah memberikan sebuah algoritma alternatif pada dunia sistem kendali untuk sistem yang tidak stabil berupa sebuah metode berbasis *machine learning* sebagai *controller* untuk sistem diskrit pada sistem yang tidak stabil.

2. METODE PENELITIAN

Pada Gambar 1 dapat dilihat secara jelas mengenai alur data pada penelitian ini :



Gambar 1. Deskripsi sistem kendali

Pada penelitian ini terdapat dua bagian, bagian pertama adalah proses pembentukan matriks *database A* dan proses kedua adalah proses yang berlangsung secara *real-time* dengan membandingkan data yang didapat dengan matriks *database* untuk menghasilkan $u(k)$ dan $y(k+1)$. Untuk penjelasan masing-masing variabel akan dijelaskan pada bagian berikutnya pada tulisan ini. Adapun tahap-tahapan yang dilakukan oleh para peneliti pada penelitian ini adalah :

Inisialisasi. Menentukan nilai-nilai variabel berikut n, m, N, h_u dan h_y. Proses penentuan nilai-nilai variabel tidak ada acuan khusus semuanya ditentukan oleh operator. Untuk waktu dimulai dari $k = 0$.

Langkah 1. Inisialisasi nilai $k \leq max(n, m)$, ulangi langkah ini. Masukkan nilai $u(k)$ bernilai diskrit yang telah ditentukan kemudian hitung nilai *output* $y(k)$. Setelah itu simpan semua nilai $u(k)$ dan $y(k)$. Karena sistem yang digunakan adalah sistem diskrit maka proses kemudian pindah atau naik ke waktu berikutnya ($k \leftarrow k + 1$).

Langkah 2. Berikutnya adalah sinyal referensi, pada penelitian ini sinyal referensi adalah sebuah

fungsi *step* $r(t)$. Fungsi *step* dari sinyal referensi kemudian disusun menjadi sebuah vektor.

Langkah 3. Pada langkah satu adalah langkah-langkah dalam proses pembuatan *database*. Setelah *database* terbentuk berupa matriks *A* berikutnya adalah menggunakan metode *NN* dan LSH dalam mengklasifikasi *database* dan vektor informasi **b**. Hasil dari proses tersebut adalah jumlah vektor yang terdekat dengan vektor informasi **b**, jika ditentukan lima terdekat maka data yang diambil adalah lima vektor terdekat. Satu dari lima vektor ini akan diambil sebagai solusi.

Langkah 4. *Data-driven predictive control* juga diterapkan [5] dalam mengumpulkan pendahuluan *Input/Output* data dari sistem untuk membangun sebuah *controller*.

$$\mathrm{a}_i \coloneqq \begin{bmatrix} \mathrm{y}_p(k_i) \\ \mathrm{y}_f(k_i) \\ \mathrm{u}_p(k_i) \end{bmatrix} \in \Re^d, i = 1, 2, \dots, N, \tag{1}$$

$$\mathrm{c}_i \coloneqq \mathrm{u}_f(k_i) \in \Re^{h_u}, i = 1, 2, \dots, N, \tag{2}$$

dimana

$$d = n + h_y + m, \tag{3}$$

$$y_p = \begin{bmatrix} y(k-n+1) \\ \vdots \\ y(k) \end{bmatrix} \tag{4}$$

$$u_p = \begin{bmatrix} y(k-m) \\ \vdots \\ y(k-1) \end{bmatrix} \tag{5}$$

Untuk variabel p adalah *past data* dan f adalah *future data*. Ide dasar dari *data-driven predictive control* terdiri dari dua tahap :

- Pilih jumlah p vektor terdekat a_{ij} terhadap vektor *query*

$$\mathbf{b} = \begin{bmatrix} \mathrm{y}_p(k) \\ \mathrm{r}(k) \\ \mathrm{u}_p(k) \end{bmatrix} \tag{6}$$

  Vektor **b** adalah vektor informasi yang terbentuk secara *real-time* dan sebuah sinyal referensi $\mathrm{r}(k)$ yang diletakkan di tengah vektor. Untuk sinyal referensi dapat dilihat pada kajian pustaka.
- Kemudian generalisasi terhadap *input* yang akan digunakan sebagai *weights* x_{ij} untuk mendapatkan nilai *input* u untuk sistem, *input* yang diharapkan didapat dari (7):

$$\hat{u}_f(k) = \begin{bmatrix} \hat{u}(k|k) \\ \vdots \\ \hat{u}(k+h_u-1|k) \end{bmatrix} \tag{7}$$

$$= \sum_{j=1}^{k} x_{ij} \mathrm{u}_f(k_{ij}) = \sum_{j=i}^{k} x_{ij} \mathrm{c}_{ij} \tag{8}$$

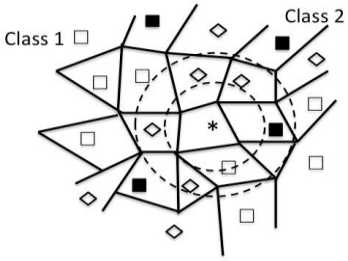
Langkah 5. Aplikasikan element pertama $\hat{u}(k|k)$ dari $\hat{u}_f(k)$ ke sistem $u(k)$. Naikkan waktu diskrit (k) sebagai berikut $k \leftarrow k + 1$ dan kembali ke **Langkah 3**.

Langkah 6. Pada langkah terakhir adalah proses simulasi pembuatan grafik (*plotting*) dari *output* $y(k)$ dan referensi $\mathrm{r}(k)$. Untuk menghitung tingkat *error* dari kedua grafik tersebut menggunakan MSE (*Mean Square Error*) dimana semakin kecil nilainya dalam artian mendekati nol semakin baik hasilnya.

Nakpong dan Yamamoto [6], pada penelitian mereka menggunakan metode data-driven yang digunakan untuk mengendalikan robot Two-Wheeled (e-nuvo). Robot tersebut bergerak berdiri dengan menggunakan dua buah roda penopang sebagai sebuah pendulum untuk membuat robot tersebut seimbang sehingga dapat bergerak tegak kedepan dan belakang. Pada paper di jelaskan bahwa mereka membangun sebuah database yang dipakai sebagai data-driven untuk mengendalikan robot e-nuvo.

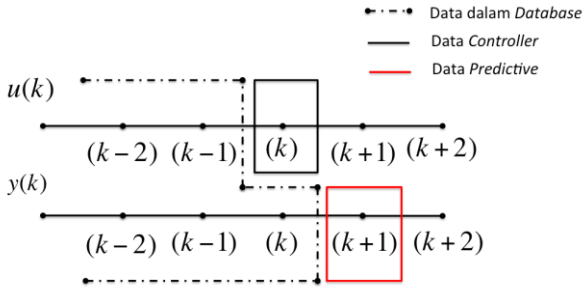
Peneliti pada penelitian ini Herlambang Saputra dan Shigeru Yamamoto [7], membandingkan tiga buah metode yaitu LWA (*Locally Weighted Average*), *Least-norm solution*, dan $l_1 - norm$ *optimization* [8], [9]. Ketiga metode tersebut menggunakan *data-driven predictive control* dalam mencari *controller output* $u(k)$, hanya saja ketika *database* sudah terbentuk, algoritma yang digunakan untuk mencari *input* sistem masing-masing menggunakan algoritma yang berbeda.

Konaka [10] dalam penelitiannya memperkenalkan algortima *neighbor* sebagai salah satu solusi untuk *controller* u. Ada beberapa langkah untuk mendapatkan nilai $u(k)$. Pada tahap pertama kali yang dilakukan adalah memperoleh *training data*, sebuah *training data* diambil secara random dari lima *input* $u(k)$ secara terbatas dan kemudian mendapatkan *output* $y(k)$. Langkah berikutnya menyusun vektor tersebut dan kemudian menormalisasikan semua vektor yang terlibat. Tahap kedua adalah mencari vektor informasi yang diambil secara *real-time*. Pada Gambar 2, terdapat gambar bintang yang mengindikasikan sebuah informasi vektor, untuk gambar-gambar yang lainnya seperti persegi, persegi kosong dan belah ketupat masing-masing mengindikasikan sebuah *class* yang terdapat dalam *database*. Sedangkan lingkaran dalam dan lingkaran luar menggambarkan tentang daerah sekitar vektor informasi (*neighbor*).



Gambar 2. Ilustrasi dari *nearest neighbor*

Sebuah ilustrasi gambar tentang struktur *predictive control* dapat dilihat pada Gambar 3.



Gambar 3. Gambaran Umum *Predictive Control*

Bentuk standar digunakan karena mampu mengakomodir antara teori sistem linear dan sistem kendali [11]. Ada beberapa hal yang dapat terjadi pada waktu k :

- Tujuannya adalah untuk memperoleh nilai $y(k)$

- Untuk memperoleh *input* sistem dibutuhkan nilai $u(k)$
- Kemudian gunakan *input* $u(k)$, tujuannya untuk mendapatkan nilai *predictive* $y(k+1)$

Berdasarkan asumsi, waktu *delay* bisa terjadi antara $y(k)$ dan penggunaan $u(k)$ dikarenakan perhitungan $u(k)$ untuk mendapatkan $y(k\ +\ 1)$ bukan untuk mendapatkan nilai $y(k)$.

Gambaran Umum Sistem

Sebuah Sistem pada Gambar 1 dapat dimodelkan menggunakan persamaan matematika (*state space*) seperti pada persamaan (9) :

$$x(k+1) = Ax(k) + u(k)$$

$$y(k) = C_y x(k)$$

$$z(k) = C_Z x(k) \tag{9}$$

$$x \in \Re^{n_x}, y \in \Re^{n_y}, u \in U = \{u^{(1)}, \dots, u^{(N)}\} \tag{10}$$

dimana M_y dan M_u mendefenisikan tentang memori yang telah disimpan pada proses sebelumnya. Untuk variabel-varibael $x, u,$ dan y adalah sebuah *state* dalam *state space*, *input* vektor dan *output* vektor, secara berurutan. Sedangkan n adalah dimensional pada *state* vektor dan k menunjukan *index* waktu. Pada variabel y dan z untuk kasus yang lebih besar terkadang terjadi tumpang tindih maka diasumsikan bahwa $y\ \equiv\ z$, kemudian A adalah sebuah matriks dan C didefenisikan untuk C_y dan C_z.

Matriks Database dan Elemen Terkait

Sebuah matriks didesain untuk menggantikan sebuah model, matriks tersebut adalah matriks *database* A. Untuk pembuatan *database* dibutuhkan data dari sebuah proses pendahuluan berupa *input/output*. Vektor-vektor dalam *database Y* dan *U* dapat diexpresikan oleh persamaan (11),

$$Y(k) = [y(k)^T, y(k-1)^T, \dots, y(k-n_a)^T]^T\ \in \Re^{(n_a+1)n_y},$$

$$U(K) = [u(k-1)^T, \dots, u(k-n_b)^T]^T\ \in U^{n_b} \tag{11}$$

Pada persamaan (11) terdapat variabel n_a yang merupakan jumlah contoh *database output* dan n_b adalah jumlah contoh *database input*. Untuk $u(k)$ tidak terdapat dalam $U(k)$ dikarenakan nilai $u(k)$ ditentukan oleh *controller*.

Untuk menghasilkan *database*, operator harus membuat sebuah asumsi terhadap beberapa parameter seperti l (panjang data), m (jumlah dari *output*), n (jumlah dari *input*) dan P (jumlah dari beberapa langkah ke depan). Parameter-parameter tersebut harus didefenisikan sebelum proses pembuatan matriks. Tujuannya agar diketahui dimensi dari matriks A dan sebagai deklarasi memori di komputer.

Pada langkah pertama adalah pembuatan vektor yang terdiri dari u^f, y^f dan $r(k)$ adalah sinyal referensi. Variabel P menunjukkan beberapa langkah ke depan yang ditentukan oleh operator dan untuk desain vektornya dapat dilihat pada :

$$u^f = \begin{bmatrix} u(k+1) \\ u(k+2) \\ \vdots \\ u(k+P) \end{bmatrix} \tag{12}$$

$$y^f = \begin{bmatrix} y(k+1) \\ y(k+2) \\ \vdots \\ y(k+p) \end{bmatrix} \tag{13}$$

Sedangkan referensi sinyal $\mathrm{r}(k)$ digunakan pada saat proses penghitungan vektor informasi adalah

$$r(k+1) = \begin{bmatrix} r(k+1) \\ r(k+2) \\ \vdots \\ r(k+P) \end{bmatrix} \tag{14}$$

Pada bagian ini parameter l, m dan n menghasilkan vektor-vektor berikut (15), (16) dan (17). Simbol y^p adalah *past output*, u^p adalah *past input*. Semua data dihasilkan dari eksperimen pendahuluan :

$$y^p(k) = \begin{bmatrix} y\big(k-(m-1)\big) \\ \vdots \\ y(k) \end{bmatrix} \tag{15}$$

$$u^p(k) = \begin{bmatrix} u(k-n) \\ \vdots \\ u(k-1) \end{bmatrix} \tag{16}$$

Langkah berikutnya adalah menghitung nilai *weight* matriks $\psi_1 \ldots \psi_k$ dan *weight* matriks terdiri dari tiga vektor. Matriks tersebut adalah matriks ψ_l dan bisa dilihat pada persamaan (17) :

$$\psi_l = \begin{bmatrix} y_l^p \\ y_l^f \\ u_l^p \end{bmatrix} \tag{17}$$

Matriks ψ_l adalah matriks yang cukup penting dan matriks ψ_l adalah A. Matriks tersebut digunakan untuk dihitung dengan vektor informasi $\varphi(k)$ untuk mencari nilai vektor x. Langkah terakhir adalah membuat matriks *database M* dan matriks *M* terdiri dari dua elemen ψ_l dan *input* u^f. Pada matriks ini terdapat penambahan parameter u^f, tujuannya untuk memesan tempat pada memori komputer. Tetapi ketika proses penghitungan tidak menggunakan u^f dan kegunaan dari u^f akan digunakan pada matriks C. Kesimpulannya *database* digunakan untuk memprediksi nilai $\hat{u}(k+1)$.

Pada *database* dan vektor informasi terdapat *Locality Sensitive Hashing* (LSH) [12] atau sering disebut fungsi *hash* dan digunakan untuk mengklasifikasikan kedua buah vektor, baik vektor pada *database* dan vektor informasi, adapun fungsi *hash* tersebut adalah :

$$h: \Re^d \to Z, h_{a,b}(v) = \left\lfloor \frac{a^T v + b}{r} \right\rfloor \tag{18}$$

dimana $r > 0$ adalah sebuah parameter dari fungsi *hash*, $a \in \Re^d$ dipilih secara bebas dari sebuah distribusi normal, b bernilai skalar dari sebuah distribusi normal dan $\lfloor x \rfloor$ adalah sebuah fungsi *floor*.

Vektor Informasi

Untuk mencari *weights* x_{ij} pada [5], dapat diformulasikan sebagai:

$$Ax = \mathbf{b}, \tag{19}$$

dimana

$$A = [\mathrm{a}_{i1}\ \mathrm{a}_{i2} \cdots \mathrm{a}_{ik}] \in \Re^{d \times k}, \tag{20}$$

$$x = [\ x_{i1}\ x_{i2} \cdots x_{ik}]^T \in \Re^{k}. \tag{21}$$

Pada *database* adalah proses pembuatan matriks A dan vektor informasi adalah proses pembuatan vektor **b** yang terjadi secara *real-time*. Untuk pembuatan proses vektor **b** telah dibahas pada persamaan (6) pada metode penelitian.

3. HASIL DAN PEMBAHASAN

3.1 Sistem Tidak Stabil dan Sinyal Referensi

Pada penelitian ini penulis memilih sebuah sistem yang tidak stabil, sistem tersebut direpresentasikan oleh sebuah persamaan matematika :

$$y(k) = y(k-1) - 0.16 * y(k-2) + u(k-1) - 1.5 * u(k-2) \tag{22}$$

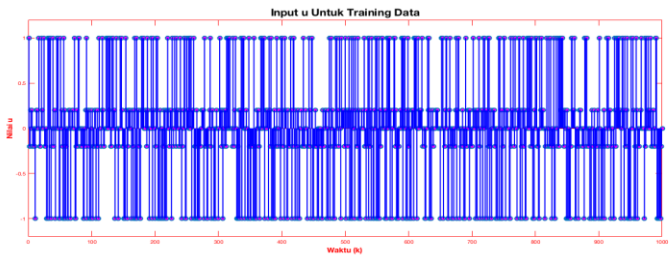
dimana k adalah waktu diskrit, u adalah *input* dan y adalah *output*. Untuk sinyal referensi yang dipakai adalah sebuah fungsi step dan fungsi tersebut adalah :

$$r(k) = \begin{cases} 0.5, & 0 \le k \le 200 \\ -0.5, & 201 \le k \le 400 \\ 0.8, & 401 \le k \le 600 \\ -0.8, & 601 \le k \le 800 \\ 0, & 801 \le k \le 1000 \end{cases} \tag{23}$$

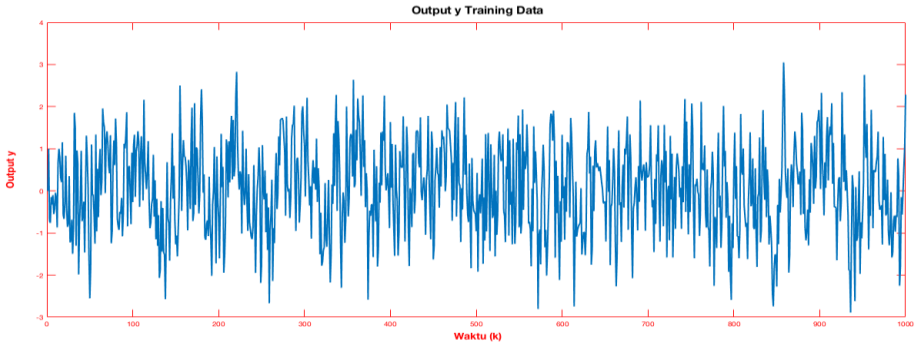
3.2 Matriks Database

Pada tahap awal peneliti melakukan inisialisasi beberapa variabel awal seperti n_a = 3, n_b = 3, n_f = 2, r = 1.5 dan k = 1000. Variabel tersebut ditetapkan untuk mendapatkan berapa jumlah vektor pada *training data* yang akan terbentuk, dalam penelitian ini akan terbentuk vektor dengan ordo 8 × 1 sebanyak 1000 data.

Langkah berikutnya adalah memilih *input* u untuk sistem yang diambil secara acak. Adapun *input* yang digunakan adalah $u(k) = \{-1, -0.2, 0, 0.2, 1\}$. Grafik *input* u dan *output* y dapat dilihat pada Gambar 4 dan Gambar 5.

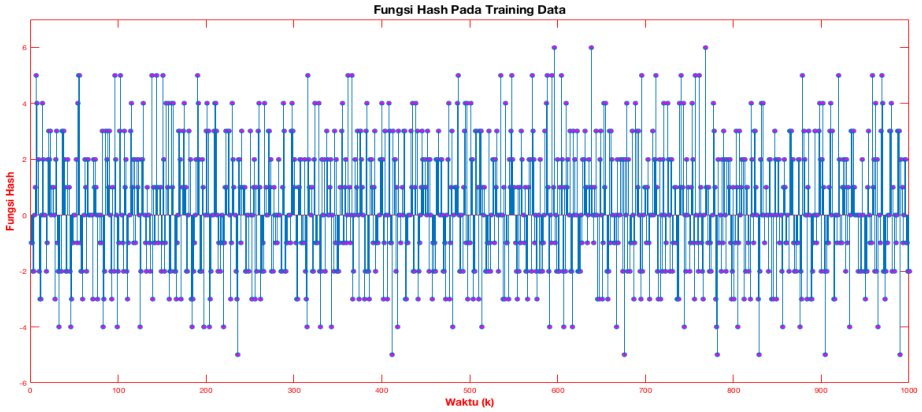


Gambar 4. Random *Input* u pada *Database*



Gambar 5. Random *Output y* pada *Database*

Apabila *input* dan *output* telah terbentuk, langkah selanjutnya menyusun *input* dan *output* kedalam sebuah vektor sebanyak 1000 buah yang disimpan di dalam matriks *database*. Setelah itu hitung fungsi *hash* yang berfungsi untuk mempercepat perhitungan. Untuk fungsi *hash* dapat dilihat pada Gambar 6.



Gambar 6. Fungsi *Hash* Pada *Training Data*

3.3 Vektor Informasi

Pada bagian ini, matriks *database* sudah terbentuk yaitu matriks A dengan ordo 8 × 1000. Beberapa variabel juga harus ditentukan agar jumlah vektor pada vektor informasi sama dengan vektor pada *training data*. Serta ada beberapa variabel tambahan seperti a dan b yang dipilih secara acak serta nilai r = 1.5 yang berguna dalam penghitungan fungsi *hash*.

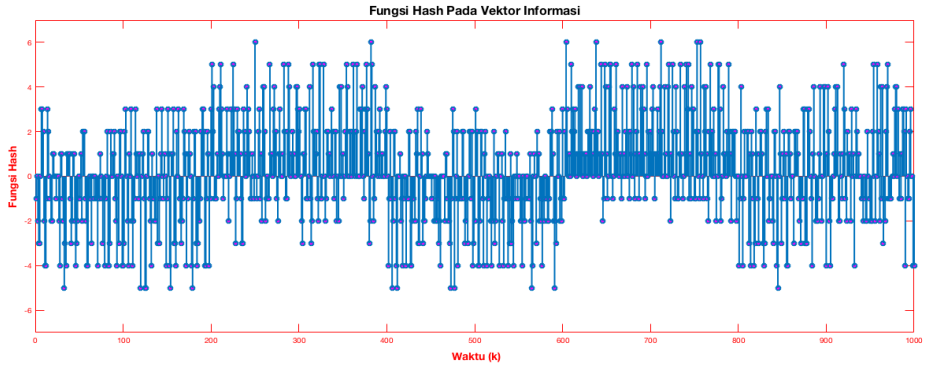
Pada *training data* fungsi *hash* dihitung setelah mendapatkan nilai y tetapi pada vektor informasi nilai fungsi *hash* dihitung untuk mendapatkan nilai $y(k + 1)$ dimana nilai $y(k)$ sudah di dapatkan pada perhitungan sebelumnya $y(k - 1)$ secara *real time* dari waktu ke waktu.

Untuk mendapatkan vektor terdekat pada setiap waktu dilakukan proses penghitungan jarak antara vektor informasi dan vektor *database*. Langkah awal pada waktu k adalah membandingkan satu buah vektor pada vektor informasi dengan seribu vektor pada vektor *database* dimana vektor terpendek yang memiliki fungsi *hash* yang sama diambil enam angka terdekat. Adapun formula penghitungan jarak pada *software* Matlab dapat dihitung dengan :

$$d = \text{norm}((\text{normalvektorvc}(:, k) - (\text{normalvektorvl}(:, n))), 2); \quad (24)$$

dimana *d* adalah jarak antara kedua vektor, normalvektorvc adalah vektor pada *training data* dan

normalvektorvl pada vektor informasi. Variabel k dan n adalah variabel waktu pada *training data* dan vektor informasi. Untuk fungsi *hash* dapat dilihat pada Gambar 7.



Gambar 7. Fungsi *Hash* Pada Vektor Informasi

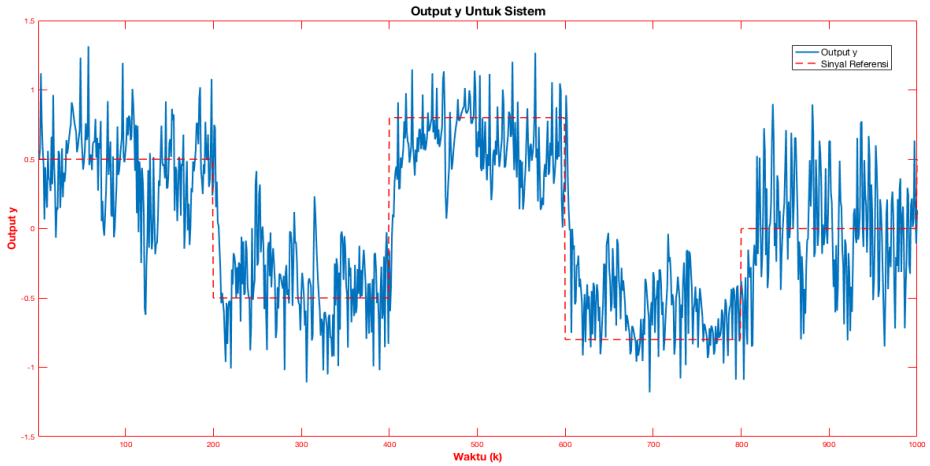
Pada penelitian ini penulis melakukan teknik pemilihan u terbaik menggunakan dua buah algoritma. Algoritma pertama penulis hanya memakai vektor terdekat yang diambil dan yang kedua penulis menerapkan sejumlah aturan yang mungkin dimana vektor terdekat belum tentu yang diambil. Aturan tersebut adalah :

- Jika ada enam nilai vektor terdekat di dalam keranjang maka untuk referensi yang bernilai positif, nilai u terdekat yang bernilai negatif diambil. Jika tidak ada yang bernilai negatif di dalam enam tetangga terdekat, maka kita memilih $u = 0$.

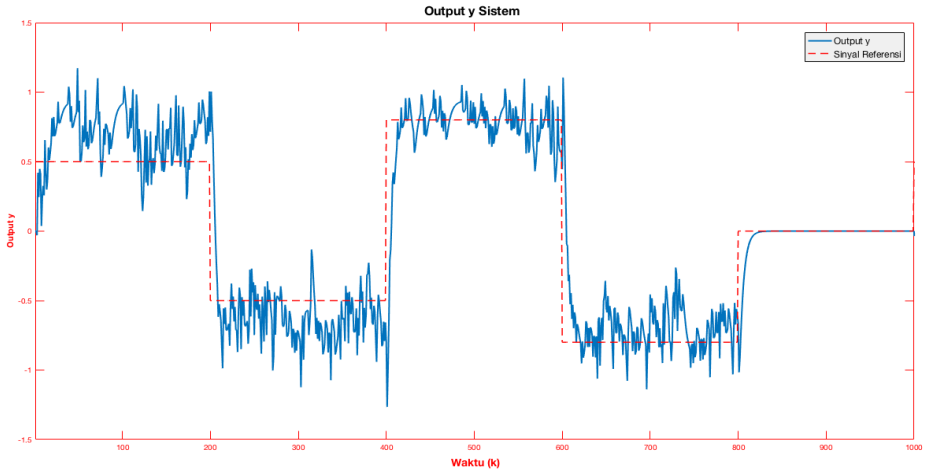
- Begitu juga sebaliknya, jika referensi bernilai negatif maka u terdekat positif yang diambil. Jika tidak ada yang bernilai positif di dalam enam tetangga terdekat, maka kita memilih $u = 0$.

- Untuk referensi yang bernilai nol, maka nilai u yang dipilih adalah nol.

Hasil akhir dari penelitian ini adalah *output y* yang dapat dilihat pada Gambar 8 dan Gambar 9. Grafik *output y* yang menggunakan algoritma pertama dapat dilihat pada Gambar 8 dan Gambar 9 menggunakan algoritma kedua.



Gambar 8. *Output y* dari Sistem dengan algoritma pertama



Gambar 9. *Output y* dari Sistem dengan algoritma kedua

Indikator MSE dan waktu komputasi dipakai untuk menghitung perbandingan kedua buah algoritma. Berdasarkan Gambar 8 dan 9 hasil yang diperoleh seperti pada Tabel 1.

Tabel 1. Perbandingan *Output y*

Algoritma	MSE	Waktu Komputasi
1	0.136234	5.732332 detik
2	0.078508	9.472880 detik

4. KESIMPULAN

Terlihat jelas jika kedua buah algoritma tersebut dapat mengikuti sinyal referensi dengan baik. Algorima satu memberikan nilai MSE lebih besar dari algoritma 2 tetapi dalam hal waktu komputasi memberikan waktu yang lebih pendek dari algoritma 2.

UCAPAN TERIMAKASIH

Para penulis pada paper ini mengucapkan terima kasih kepada Politeknik Negeri Sriwijaya atas bantuan hibah Penelitian yang dibiayai oleh Dana PNPB Polsri Nomor. 6434/PLG.2.1/PL/2017 pada Tanggal 7 Agustus 2017.

DAFTAR PUSTAKA

[1] C. E. Garcia, D. M. Prett, and M. Morari, " Model Predictive Control : Theory and Practice a Survey," *Automatica*, vol. Vol.25, no. Iss.3, pp. 335–348, 1989.

[2] K. Fujiwara, M. Kano, S. Hasebe and A. Takinami, " Soft-sensor development using correlation-based just-in-time modeling," *AIChE Journal*, vol. 55, no. 7, pp. 1754–1765, 2009.

[3] A. Stenman, *Model on Demand : Algorithms, Analysis and Applications, PhD thesis*. No. 571, Department of Electric Engineering Linkoping University, 1999.

[4] J. Ohta and S. Yamamoto, " Database-Driven Tuning of PID Controllers," *Trans. of the*

Society of Instrument and Control Engineers, vol. 40, pp. 664– 669, June 2004.

[5] S. Yamamoto, " A new model-free predictive control method using input and output data," in *Advanced Material Research*, vol. 1042 of *3rd International Conference on Key Engineering Materials and Computer Science (KEMCS 2014)*, (Singapore), pp. 182–187, Trans Tech Publication, August 2014.

[6] N. Nakpong and S. Yamamoto, " Just-In-Time predictive control for a two-wheeled robot," in *2012 Tenth International Conference on ICT and Knowledge Engineering*, no. 3, (Thailand), pp. 95–98, Nov. 2012.

[7] Saputra and Yamamoto, " Comparative study of model-free predictive control and its database maintenance for unstable system," *SICE Journal of Control, Measurement and System Integration*, vol. 8, pp. 390–395, Nov 2015.

[8] S. Yamamoto, " A model-free predictive control by l_1-minimization," in *Proceedings of the 10th Asian Control Conference*, ASCC, 2015.

[9] A. Y. Yang, A. Ganesh, Z. Zhou, S. Sastry, and M. Y, " A review of fast l_1-minimization algorithms for robust face recognition, *http://arxiv.org/abs/1007.3753*," 2010.

[10] E.Konaka," Design of Discrete Predictive Controller Using Approximate Nearest Neighbor Method," in *Proceedings of the 18th IFAC World Congress*, (Milano (Italy)), pp. 10213–10218, Aug. 2011.

[11] J. Maciejowski, " *Predictive Control with Constraints*," London: Prentice Hall, 2002.

[12] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. *In proc. of the 20th ACM Symposium on Computational Geometry (SCG'04)*, Brooklyn, New York, USA, pages 253–262, 2004.