AUTONOMOUS MOBILE ROBOT BERBASIS *PLAYER/STAGE* MENGGUNAKAN PARALLEL SELF-ORGANIZING FEATURE MAPS UNTUK PEMETAAN LINGKUNGAN GLOBAL YANG TIDAK DIKETAHUI

Mochamad Hariadi, Muhtadin, Mauridhy Hery Purnomo, Muhammad Rivai
Bidang Studi Teknik Sistem Komputer
Jurusan Teknik Elektro - Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember, Surabaya-60111
Email: mochar@ee.its.ac.id

ABSTRAK

Autonomous mobile robot adalah salah satu jenis robot yang dikembangkan dengan kemampuan untuk mengendalikan dirinya sendiri walaupun dalam lingkungan yang tidak diketahui. Untuk dapat melakukan pengendalian secara mandiri, bisa dilakukan dengan melalui proses pembelajaran secara mandiri tanpa supervisi (unsupervised) dengan mempertimbangkan input dari sensor-sensor yang dipakai. Pada saat robot melakukan pengenalan terhadap lingkungannya, diperlukan perosesan komputasi yang berat dengan waktu yang lama. Penelitian ini akan membahas tentang penggunaan Kohonen Self-Organizing Feature Maps (SOFM) atau (SOM) sebagai metode pembelajaran Autonomous mobile robot dalam mengenali lingkungannya. Proses pembelajaran dilakukan dengan menggunakan parallel processing menggunakan LAM-MPI, Simulasi dilakukan dengan menggunakan software simulasi Player/ Stage. Hasil simulasi menunjukkan bahwa SOM menampilkan performa yang baik dalam memetakan lingkungan yang tidak diketahui tanpa supervisi. Hasil pemrosesan dengan menggunakan parallel processing juga menunjukkan dicapainya kecepatan yang signifikan dalam proses pembelajaran robot untuk mengenali lingkungannya. Planning dengan menggunakan A mampu untuk memberikan jalur yang efektif bagi robot dalam mencapai titik tujuan.*

Kata Kunci: autonomous, pemetaan, path-planning, komputasi parallel, parallel SOM

1. PENDAHULUAN

Autonomous mobile robot biasa digunakan pada banyak keperluan, misalnya pengendalian otomatis pada jalan raya, eksplorasi pada lokasi yang berbahaya, dan lain-lain. Aplikasi ini harus berupa metode yang mampu bertahan dan mampu beradaptasi dengan berbagai lingkungan yang berubah-ubah[1].

Agar sebuah robot dapat mempunyai kemampuan untuk mengendalikan dirinya secara mandiri, maka robot harus memiliki pengetahuan tentang lingkungan tempat dia berada. Jika robot tidak mengetahui keberadaannya dalam lingkungannya, maka robot tidak dapat menentukan pergerakannya secara efektif untuk menemukan target yang dituju[3]. Ada banyak isu yang bisa digunakan untuk menjelaskan tentang pengendalian secara mandiri pada *autonomous mobile robot*. Salah satu isu yang sering digunakan adalah *path planning*[1]. Pada *path planning*, ada dua kategori yang bisa digunakan, kategori pertama adalah *global path planning*, pada kategori ini diperlukan pengetahuan secara menyeluruh terhadap area terlebih dahulu, kategori yang kedua adalah *local path planning*, pada kategori ini digunakan pada lingkungan yang belum diketahui. Pada *global path planning*, robot harus mempunyai pengetahuan yang lengkap tentang suatu area terlebih dahulu saat mulai dijalankan dan planning dilaksanakan secara *offline*. Pada kategori *local path planning*, robot akan mempergunakan sensor, sensor yang dipakai bisa berupa sensor *ultrasonic*, sensor *laser*, ataupun dengan menggunakan sistem pengamatan dengan kamera untuk menentukan area yang ditempatinya dan planning dilaksanakan secara online berdasarkan sensor-sensor tersebut[1,2].

Pada *local path planning*, robot tidak perlu mengetahui seluruh bagian dari lingkungannya, robot hanya perlu mengetahui keadaan *obstacle* disekitarnya saja, oleh karena itu tidak diperlukan ekplorasi secara menyeluruh terhadap lingkungannya. Dengan menggunakan metode ini maka proses pengenalan lingkungan bisa lebih cepat dan dapat dilakukan pelatihan secara *online* [4,6]. Metode ini mempunyai kelemahan yaitu akan kesulitan untuk menentukan jalur yang efektif menuju titik tujuan, hal ini disebabkan karena perngetahuan robot yang hanya terbatas pada bagaimana menghindari *obstacle* didekatnya [8].

Metode yang berbeda digunakan pada global path planning, pada metode ini robot terlebih dahulu melakukan eksplorasi terhadap semua bagian dari lingkungannya[9], data hasil eksplorasi disimpan dalam bentuk kelompok *neuron* [7]. Metode ini telah kami gunakan pada penelitian awal kami di [10] dengan SOM sebagai algoritma pembelajarannya.

Dengan adanya pengetahuan yang menyeluruh, maka robot akan mampu menghindari halangan dan mampu memilih jalur yang paling efektif untuk mencapai titik tujuan. Kelemahan dari metode ini adalah lamanya proses eksplorasi sehinga sulit untuk melakukan eksplorasi secara *online*.

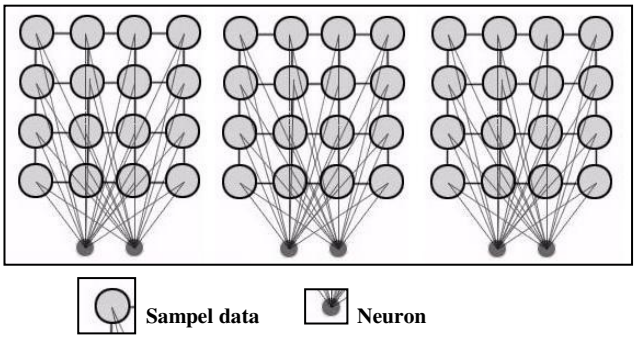
Penelitian di [10] mempunyai kendala kecepatan proses. dalam ini menyajikan bagaimana cara untuk meningkatkan kecepatan proses pembelajaran bagi robot dalam mengenali lingkungannya. Percepatan dilakukan dengan cara menjalankan proses pembelajaran menggunakan komputasi *parallel*. Pemrosesan *parallel* dilaksanakan dilaksanakan dengan menggunakan *Message Passing Interface (MPI)* [13]. Dari hasil yang dicapai, nampak bahwa diperoleh percepatan pembelajaran yang signifikan dibandingkan dengan pembelajaran menggunakan processor tunggal.

2. DASAR TEORI

2.1 *Kohonen Self Organizing Feature Maps*

Metode *SOFM* atau SOM memungkinkan untuk menggambarkan data multidimensi kedalam dimensi yang lebih kecil, biasanya satu atau dua dimensi. Proses penyederhanaan ini dilakukan dengan mengurangi vektor yang menghubungkan masing-masing *node*. Cara ini disebut juga dengan *Vector Quantization*. Teknik yang dipakai dalam metode *SOM* dilakukan dengan membuat jaringan yang menyimpan informasi dalam bentuk hubungan *node* dengan training set yang ditentukan[11].

Salah satu hal yang menarik dalam metode SOM adalah kemampuannya untuk belajar secara mandiri (*unsupervised learning*). Pada metode belajar secara mandiri, sebuah *network* akan belajar tanpa adanya target terlebih dahulu[12]. Hal ini berbeda dengan beberapa metode neural network yang lain seperti back propagation, perceptron, dan sebagainya yang memerlukan adanya target saat proses *learning* dilaksanakan.



Gambar 1. Jaringan Parallel SOM sederhana

Arsitektur Jaringan Paralel SOM

Pada penelitian ini, jaringan SOM dimodifikasi sedemikian rupa sehingga memungkinkan untuk dilakukan proses pembelajaran secara paralel, jaringan SOM tersebut dimodifikasi menjadi jaringan Paralel SOM.

Neuron dibentangkan keseluruh bagian lingkungan yang ingin dikenali, kemudian seluruh area itu dibagi-bagi menjadi area yang lebih kecil. Masing-masing area tersebut akan diproses oleh satu *processor*. Gambar 1 pembagian 3 area dari satu lingkungan utuh.

Pada arsitektur paralel menggunakan LAM MPI, diperlukan sebuah komputer yang berfungsi sebagai *master*, *master* ini akan membagi-bagi tugas ke beberapa komputer/host lain yang disebut *slave*, pada proses pembelajaran dengan menggunakan parallel SOM, *master* bertugas untuk melakukan pembagian lingkungan menjadi area-area yang lebih kecil, kemudian *master* mengirimkan area tersebut untuk diproses pada masing-masing *slave*, setelah semua proses dari *slave* selesai, maka *master* akan mengumpulkan kembali hasil pemrosesan dari masing-masing area dan menggabungkan menjadi satu lingkungan yang utuh, secara detail pembelajaran paralel SOM pada saat eksplorasi dapat dijelaskan dibawah ini :

a. Inisialisasi *weight* pada masing-masing *node*.
   Sebelum pembelajaran dimulai, *weight* masing-masing *node* diberikan inisialisasi terlebih dahulu.
b. *Master* akan membagi lingkungan menjadi area-area yang lebih kecil, masing-masing area akan dikirimkan kepada *slave* untuk dilakukan proses pembelajaran.
c. Input berupa sebuah vektor merupakan hasil deteksi dari salah satu sensor pada robot.
d. Apabila vektor input masuk pada satu area tertentu, maka proses selanjutnya dikerjakan oleh komputer *slave* yang bersankutan.
e. Setiap *node* dihitung dan dicari *weight* yang paling mendekati vektor input. Weight pemenang disebut dengan *Best Matching Unit* (*BMU*).
   Untuk menentukan *BMU*, salah satu metode yang dipakai adalah dengan menghitung semua *node* dan menghitung jarak *Euclidean* antara *weight* masing-masing *node* dengan vektor input. *Node* yang mempunyai vektor paling mendekati vektor input, maka ditentukan sebagai *BMU*.[10]
f. Menentukan *node* tetangga *BMU*.
   Setelah *BMU* ditentukan, langkah selanjutnya adalah menentukan *node* mana saja yang menjadi *node* tetangga *BMU*. Langkah pertama yang perlu dilakukan adalah menentukan jarak / radius dari *BMU*. Radius ini yang akan menjadi penentu apakah *node* disekitarnya masuk sebagai *node* tetangga atau tidak, jika ada *node* yang berada dalam radius tersebut, maka dianggap sebagai *node* tetangga, sebaliknya apabila terletak diluar radius tersebut maka akan dianggap bukan sebagai *node* tetangga. Radius ini akan semakin

mengecil seiring dengan bertambahnya proses iterasi pada saat proses *learning*.[10]

g. Update *Weight,*
Setelah *node* tetangga BMU ditentukan, maka langkah selanjutnya adalah melakukan update weight pada semua *node* tetangga BMU termasuk juga BMU itu sendiri. [10]

h. Setelah proses dari masing-masing komputer *slave* selesai, maka *Master* akan mengumpulkan hasil dari masing-masing komputer *slave* dan menggabungkannya menjadi satu peta yang utuh.

2.2 *Player/Stage*

Player/Stage adalah simulator robot yang dibuat berdasarkan sistem operasi *linux*. Simulator tersebut dibuat untuk mensimulasikan robot serta lingkungan tempat simulasi serta pemrograman *interface* robot secara fleksibel dalam lingkungan 2 dimensi.

Player/Stage mengimplementasikan konfigurasi *client/server*. Sebuah file world mendefinisikan lingkungan dan semua obyek didalamnya, termasuk robot, serta spesifikasi kemampuan *hardware* dari masing-masing robot. File ini yang akan digunakan untuk menyediakan "dunia" tempat robot berinteraksi. Kemudian *client* program akan terhubung ke *server* pada *port* tertentu untuk bisa mengontrol robot.

Program *client* dapat dijalankan pada mesin/komputer yang sama dengan server ataupun dari mesin/komputer yang terhubung dengan *server* melalui jaringan. *Player* dapat juga berfungsi sebagai *server* untuk *real-robot*, sehingga program *client* yang digunakan untuk menjalankan robot dalam *simulator*, dapat juga digunakan untuk menjalankan *real-robot*.[10]

3. DESAIN SISTEM

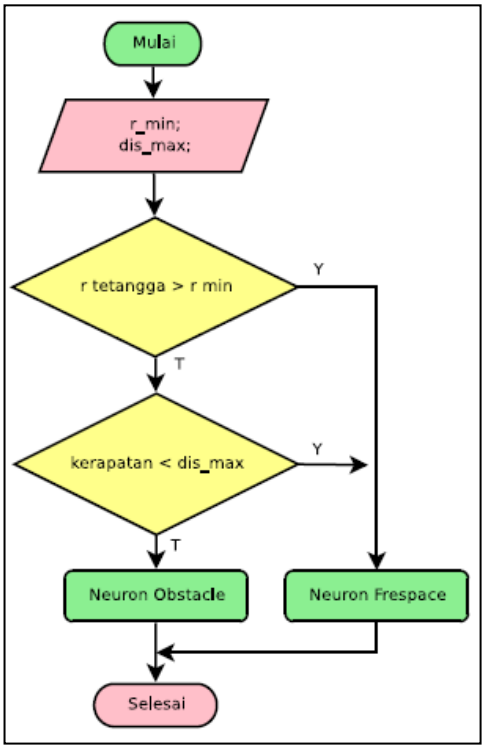
Simulasi yang digunakan adalah *Player/Stage* dengan menggunakan *player* sebagai *server* dan *stage* sebagai simulatornya. *Player* menyediakan *interface*, *device* dan *driver* yang akan digunakan untuk membangun sebuah robot beserta lingkungannya. definisi *interface*, *device* dan *driver* ini dituliskan dalam sebuah file konfigurasi *player*. *Interface*, *device* dan *driver* yang sudah didefiniskan dalam file konfigurasi *stage* akan didefinisikan pula modelnya yang ditulis dalam file konfigurasi *stage*.

Pada penelitian ini, robot yang dibangun memiliki *sonar* yang sebanyak 10 buah *sonar* yang disebar di tepi robot. 7 *sonar* diletakkan dibagian depan robot, dari ujung kanan menuju ujung kiri dengan posisi hadap *sonar* mempunyai selang 30 derajat dari masing-masing *sonar*, 3 *sonar* berada dibelakang dengan posisi hadap *sonar* 30 derajat dari masing-masing *sonar*. Pada penelitian ini, *sonar* digunakan untuk mengetahui adanya *obstacle* pada sudut dan jarak tertentu dari robot.[10]

Proses *learning* dilakukan pada layer SOM. Input yang digunakan dari pelatihan tersebut berasal dari data yang dikumpulkan oleh robot dengan menggunakan *sonar* yang dipunyai oleh robot.

Hasil data *sonar* yang didapat, kemudian diproses secara parallel kedalam beberapa *node*. Setiap *node* akan melakukan proses pada wilayah tertentu dalam peta lingkungan robot.

Setelah proses *learning* selesai, neural pada layer SOM akan membentuk bidang yang mendekati dengan bentuk peta. Dari kondisi tersebut, *neuron* dapat digolongkan sebagai *neuron freespace* dan *neuron obstacle* dengan algoritma seperti ditunjukkan diagram pada gambar 2. Setelah semua *neuron* digolongkan menjadi *neuron freespace* dan *neuron obstacle* , proses selanjutnya adalah membuat blok dan mengolongkannya menjadi blok *freespace* dan blok *osbstacle*.



Gambar 2. Penggolongan *neuron*

Setelah penggolongan blok selesai, proses selanjutnya adalah pencarian path menggunakan A*. Algoritma pencarian jalur yang tepat dengan menggunakan A* dapat dijelaskan sebagai berikut :

a. Ambil titik awal dan titik tujuan, cari blok yang sesuai dengan posisi titik awal dan posisi tujuan tersebut.
b. Masukkan blok awal kedalam *open* c.
c. Cek blok semua blok yang berhubungan dengan blok awal, abaikan blok tersebut jika blok tersebut berupa blok *obstacle* . Dan masukkan blok-blok tersebut kedalam *open*

list, untuk masing-masing blok tadi, set *parent* blok tersebut ke blok asal.

d. Hapus blok asal tadi dari *open list* dan masukkan kedalam *closed list*.

e. Dari semua blok dalam *open list*, cari nilai Fn terkecil, ambil sebagai blok yang aktif.

f. Jika blok aktif mempunyai nilai Hn lebih besar dari 0 maka Cek semua blok yang berhubungan dengan blok aktif untukmasingmasing blok, jika blok tersebut adalah blok *freespace* , lakukan langkah-langkah sebagai berikut :

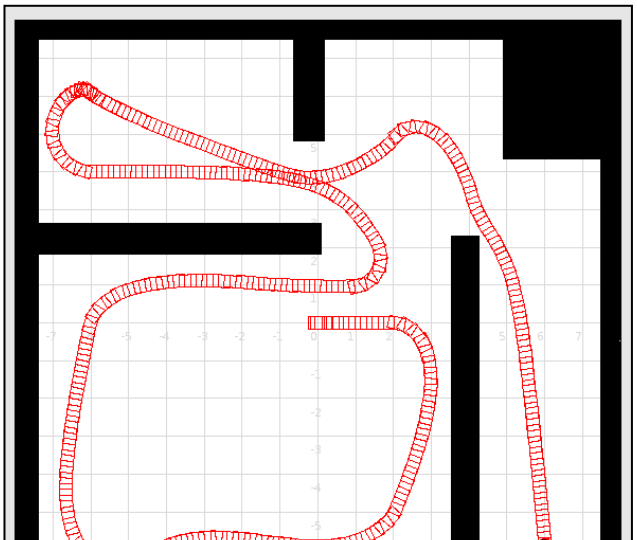
   i. Jika blok tersebut belum ada dalam *open list* dan belum ada juga dalam *closed list*, masukkan blok tersebut kedalam *open list*, dan set *parent* blok tersebut ke blok aktif.

   ii. Jika blok tersebut sudah ada dalam *open list*, Bandingkan nilai Gn dari *parent* blok tersebut dengan nilai Gn dari blok aktif, jika nilai Gn blok aktif lebih kecil daripada nilai Gn *parent* blok tersebut, maka update *parent* blok tersebut menjadi blok aktif.

   iii. Jika blok tersebut sudah ada dalam *closed list*, abaikan saja blok tersebut.

g. Jika blok aktif mempunyai nilai Hn = 0, maka blok aktif adalah blok tujuan, dan proses selesai. Jalur yang dapat ditempuh dapat dirunut kembali melalui *parent* dari blok tujuan sampai blok asal.

h. Jika *open list* sudah kosong, sementara blok tujuan belum ditemukan, berarti tidak ada jalur yang bisa dilewati dari titik awal ke titik tujuan.



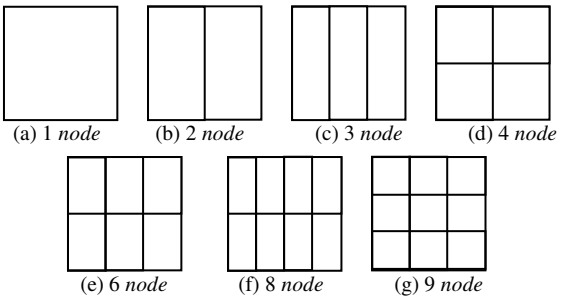
Gambar 3. Robot menjelajah ruangan

4. PENGUJIAN

Pengujian dilakukan dengan mengunakan *Stage* sebagai simulator dengan *client* deprogram menggunakan C++.

Proses pembelajaran diawali dengan menjalankan robot menjelajahi seluruh daerah, robot hanya perlu berjalan dan melakukan scaning terhadap *obstacle* yang ada pada lingkungan tersebut. Gambar 3 menunjukkan jalur yang dilalui robot saat robot menjelajahi ruangan.

Proses pembelajaran dilakukan secara parallel dengan membagi menjadi beberapa *node*. Pada penelitian ini, proses parallel dicobakan pada beberapa *node* yang berbeda kemudian dicatat hasilnya. Pembagian *node* didasarkan pada kuadran tempat *neuron* berada, seperti ditunjukkan gambar 4.

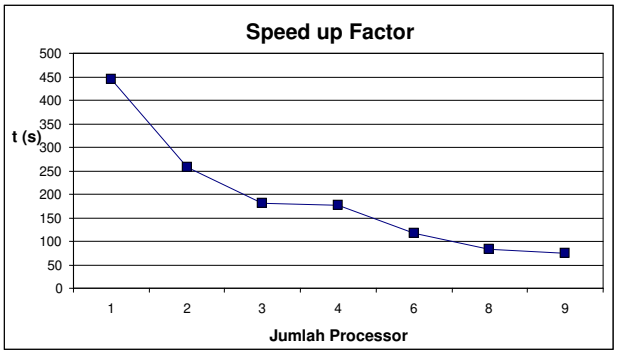


Gambar 4. Pembagian *Node*

Dengan pengujian berdasarkan pembagian seperti tersebut diatas didapatkan *speed factor* seprti ditunjukkan tabel 1. Dari hasil percobaan, didapatkan bahwa semakin banyak *node* yang digunakan, maka semakin cepat proses pembelajaran dilaksanakan. Perbandingan jelas terlihat ketika pembelajaran dilakukan dengan menggunakan single processor, maka waktu yang diperlukan adalah **445 detik**, sedangkan saat pembelajaran dilaksanakan dengan menggunakan 9 processor, maka proses pembelajaran hanya memerlukan waktu **74 detik**. Hasil percobaan secara lengkap ditunjukkan oleh tabel 1 dan grafiknya ditunjukkan oleh gambar 5.

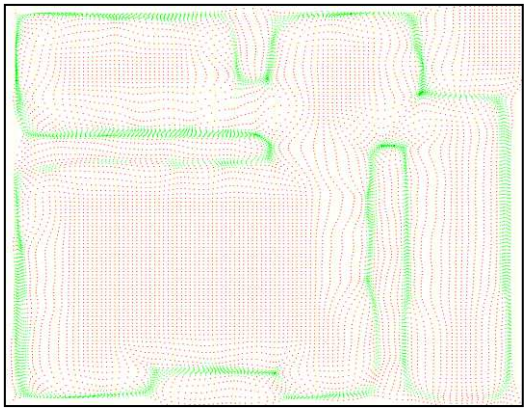
Tabel 1. Tabel Hasil percobaan

Node	Waktu (detik)
1	445
2	258
3	180
4	176
6	116
8	82
9	74



Gambar 5. Perbandingan Speed up Factor

Setelah proses learning selesai, kemudian komputer server mengumpulkan hasil proses pembelajaran dari komputer *slave*, maka akan diperoleh sebuah peta utuh seperti ditunjukkan pada gambar 6.

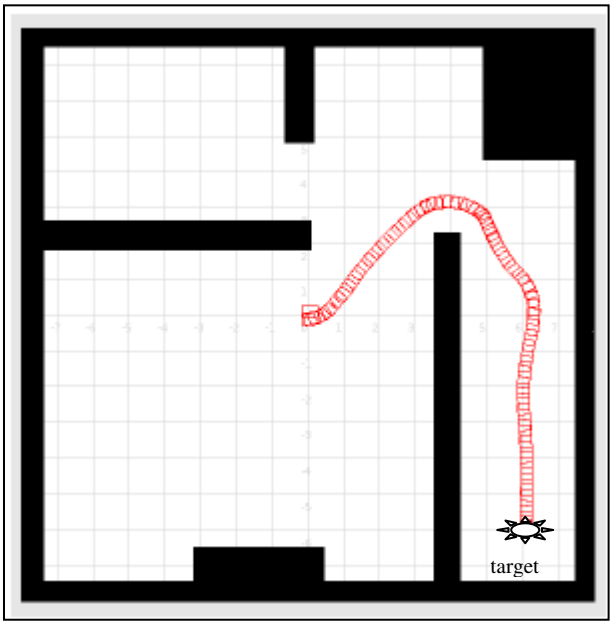


Gambar 6. Hasil pengenalan lingkungan

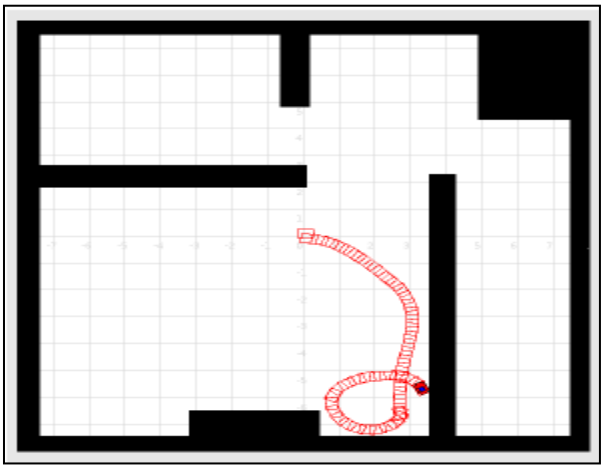
Gambar 6 menunjukkan grid *neuron* yang sudah mengalami proses pembelajaran dengan menggunakan sebagian dari peta yang sudah dijelajahi oleh robot, pada kondisi ini, robot hanya mengalami pembelajaran terhadap sebagian dari peta saja. Neuron berwarna merah menunjukkan free space bagian free space dari peta, sedangkan *neuron* berwarna hijau menunjukkan tepi dari *obstacle* dalam peta.

Kemudian robot diuji dengan meletakkan pada suatu titik, dan diberi tugas untuk menuju ke titik target. Jalur yang digunakan untuk robot untuk mencapai titik tujuannya ditentukan dengan menggunakan algoritma A* dengan memanfaatkan data hasil pengenalan lingkungan.

Pada ujicoba yang dilakukan, ternyata robot berhasil mencapai titik tujuannya dengan baik dengan jalur yang efektif menuju titik tujuan.



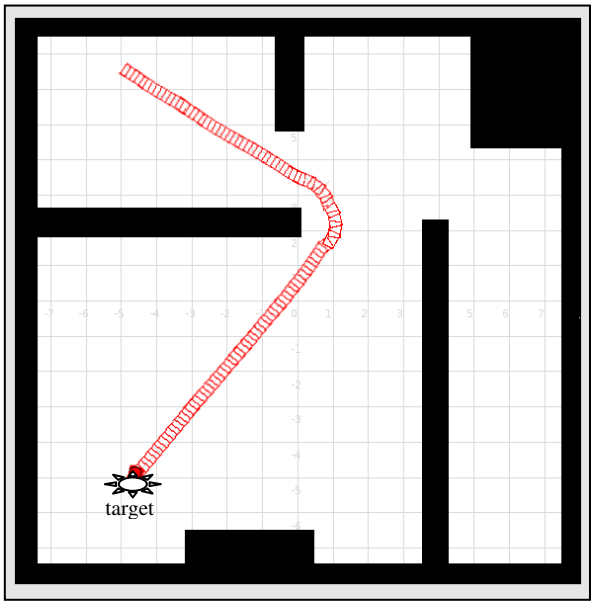
Gambar 7. Robot berhasil mencapai titik target



Gambar 8. Robot gagal mencapai titik target

Gambar 7 menunjukkan gerak robot untuk mencapai titik target dengan memanfaatkan data yang sudah dipelajari sebelumnya. Dari gambar tersebut nampak bahwa robot dapat mencapai titik tujuannya dengan tepat, dari gambar tersebut nampak bahwa robot juga mampu menentukan lintasan yang efektif sampai ke titik tujuan. Sebagai perbandingan, gambar 8 menunjukkan gerak robot ketika berusaha mencapai titik tujuan yang sama namun tidak mempergunakan pembelajaran terhadap lingkungannya, pada gambar tersebut nampak bahwa mula-mula robot mengambil garis lurus menuju titik tujuannya, kemudian

membentur tembok dan akhirnya gagal mencapai titik tujuannya. Gambar 9 menunjukkan percobaan dengan mempergunakan titik tujuan yang lain. Pada percobaan ini robot juga mampu mencapai titik target dengan benar dan dapat menentukan jalur yang efektif hingga mencapai titik tujuan.



Gambar 9. Berhasil mencapai titik target yang lain

5. KESIMPULAN & SARAN

Kesimpulan yang dapat diambil adalah :

a) Metode Kohonen SOM dapat dipakai sebagai algoritma untuk memetakan environment mapping, dan dengan adanya pemetaan tersebut, maka robot dapat mengendalikan diri secara mandiri untuk mencapai titik tujuan.
b) Dalam mencapai target, robot bisa sekaligus untuk mengambil data dan memetakan lingkungannya.
c) Algoritma searching A* dapat digunakan untuk mennentukan jalur yang tepat bagi robot dari titik asal ke titik tujuan dengan cepat, namun algoritma ini baru dapat digunakan jika proses mapping sudah selesai.
d) Proses pembelajaran dengan mempergunakan komputasi *parallel* akan mempercepat proses pengenalan robot terhadap lingkungannya.
e) *Player*/*stage*/gazebo dapat digunakan untukmenguji algoritma-algoritma baru dan mensimulasikannya seperti pada robot yang sebenarnya.

Adapun saran yang dapat digunakan untuk penelitian lebih lanjut adalah :

a. Dalam menggunakan algoritma A* hendaknya mengunakan penggunaan struktur data yang lebih optimal, misalnya menggunakan quick sort atau binary-heap dengan struktur data yang baik, maka kecepatan pencarian menggunakan A* bisa ditingkatkan.
b. Sebaiknya menggunakan algoritma searching yang lain yang berbasis soft computing seperti Genetic Algorithm.
c. Neuron yang digunakan dalam layer SOM, hendaknya memiliki kerapatan yang cukup, sehingga proses mapping bisa menghasilkan data yang lebih baik.
d. Hendaknya dibuat pengaturan data yang baik dari *interface* yang digunakan pada *player*, dengan adanya pengaturan data yang baik, memungkinkan untuk dilakukan pelatihan secara on-line.

6. DAFTAR PUSTAKA

[1] I J Nagrath, L Behera, K Madhava Krishna and K D Rajasekhar, "Real-time Navigation of a Mobile Robot using Kohonen's Topology Conserving Neural Networks", Proc. Eighth International Conference on Advanced Robotics, pp 459-464, Monterey, CA, July 1997.
[2] R. Gutierrez-Osuna, J. A. Janet and R. C. Luo, ”Modeling of Ultrasonic Range Sensors for Localization of Autonomous Mobile Robots”, IEEE Transactions on Industrial Electronics, 45(4), pp. 654-662, 1998.
[3] George Palamas, George Papadaurakis, Manolis Kavaoussanos, “Mobile Robot Position Estimation using unsupervised Neural Networks”,http://www.e-technik.fh-kiel.de/it workshop-aveiro/papers/papadourakis.pdf
[4] Arno J. Knobbe, Joost N. Kok, Mark H. Overmars , “Robot Motion Planning in Unknown Environments using Neural Networks”, Proc. ICANN'95, International Conference on Artificial Neural Networks.
[5] Uwe R. Zimmer & Ewald von Puttkamer, "Realtime-*learning* on an Autonomous Mobile Robot with Neural Networks",IEEE, Real-Time Systems, 1994. Proceedings., Sixth Euromicro Workshop on Volume , Issue , 15-17 Jun 1994 Page(s):40 - 44
[6] C.R. Weibsin, G. de Saussure, and D. Kammeer, Self-Controller: "A real-time expert system for an autonomous mobile robot", Comp. Mech. Eng., pp.12-19, Sept. 1986.
[7] Bernd Fritzke Growing Cell Structures - A Self-organizing Network for Unsupervised and Supervised Learning Technical Report

93-026, International Computer Science Institute, Berkeley, California

[8] J. Borenstein and Y. Koren, "Real-time *obstacle* avoidance for fast mobile robots", IEEE Trans. SMC, vol. 19, no. 5, pp.1179-1186, Sept./Oct., 1989.

[9] J. A. Janet, R. Gutierrez-Osuna, T. A. Chase, M. White and J. C. Sutton, III, ”Autonomous Mobile Robot Global Self-localization using Kohonen and Region-Feature Neural Networks”, Journal of Robotics Systems, 14(4), pp. 263-282, 1997.

[10] Mochamad Hariadi, Muhtadin, Mauridhi Hery, Simulasi Autonomous Mobile Robot Berbasis *Player/Stage* Menggunakan Self-Organizing Feature Maps untuk Pemetaan Lingkungan Global yang Tidak Diketahui, Jurnal Informatika UK Petra, Nopember 2007. .

[11] “Kohonen Networks” , http://www.cs.bham.ac.uk/resources/courses/SEM2A2/Web/Kohonen.htm

[12] “Kohonen's Self Organizing Feature Maps”, http://www.ai-junkie.com/ann/som/som1.html

[13] Jack J, Dongarra, Steve W. Otto, An Introduction to the MPI Standard, -, April 1995