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

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

ABSTRAK: Perkembangan dunia robotika saat ini tidak luput dari teknologi yang mampu beradaptasi dengan lingkungan sekitar robot. *Autonomous mobile robot is the one of robot types which has* 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. Paper ini akan membahas tentang penggunaan Kohonen *Self-Organizing Feature Maps (SOM)* sebagai metode pembelajaran Autonomous mobile robot dalam mengenali lingkungannya. Simulasi dilakukan dengan menggunakan open source software Player/ Stage. Hasil simulasi menunjukkan bahwa SOM menampilkan performa yang baik dalam memetakan lingkungan yang tidak diketahui tanpa supervisi.

Kata kunci: *autonomous mobile robot, self-organizing feature maps (som).*

***ABSTRACT**: Nowadays, the development of robotics require more sophisticated technologies that would be able to adapt with global environment. Autonomous mobile robot is one of the robots which have been developed with environment control even in uncharted environment. Regarding this, an autonomous mobile robot should be capable of learning its environment in unsupervised fashion. This is done by monitoring and capturing the information from employed sensors. This paper describes the implementation of Kohonen Self-Organizing Feature Maps (SOM) as the learning methode for an Autonomous mobile robot for learning and recognizing its uncharted environment. Simulations using open source software called Player/Stage demonstrates good performance, since SOM is capable for mapping the uncharted environment very well in unsupervised fashion.*

***Keywords**: autonomous mobile robot, self-organizing feature maps (som).*

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 [4].

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 [4]. 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 metode 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 [4,8].

Proses pembelajaran robot terhadap lingkungannya ini bisa dilakukan dengan Kohonen **Self-Organizing Feature Maps (SOM)** [4,6]. SOM mempunyai kemampuan untuk melakukan proses klasifikasi dan pengenalan dengan melalui proses pembelajaran secara unsupervised. Metode pembelajarannya dilakukan dengan update weight berdasarkan input sensor yang terdapat pada robot.

DASAR TEORI

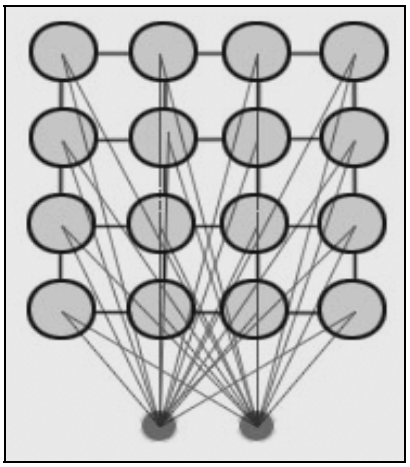
Kohonen Self Organizing Feature Maps

Kohonen Self Organizing Feature Maps, disingkat dengan SOFM atau lebih terkenal dengan istilah SOM ditemukan dan dikembangkan oleh Teuvo Kohonen, seorang profesor di Academy of Finland**.** Metode ini 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 [1].

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 [2]. Hal ini berbeda dengan beberapa metode neural network yang lain seperti back propagation, perceptron, dan sebagainya yang memerlukan adanya target saat proses learning dilaksanakan.

Arsitektur Jaringan SOM

Arsitektur jaringan SOM sederhana bisa digambarkan seperti Gambar 1. Pada Gambar 1, topologi jaringannya terbentuk dari node 4x4 yang masing-masing terhubung dengan layer input yang menggambarkan vektor dua dimensi.



Gambar 1. Jaringan SOM Sederhana

Masing-masing node mempunyai topologi yang spesifik (koordinat x,y dalam pola) dan memiliki vektor weight yang mempunyai dimensi yang sama dengan vektor input. Jika data training terdiri dari vektor V dengan dimensi n maka dapat ditulis:

$$V_1, V_2, V_3 \ldots V_n$$

Kemudian masing-masing vektor memiliki hubungan dengan vektor weight W dalam dimensi n:

$$W_1, W_2, W_3 \ldots W_n$$

Algoritma Pembelajaran SOM

SOM tidak memerlukan target output. Training pada SOM bisa dilakukan dengan langkah-langkah berikut ini:

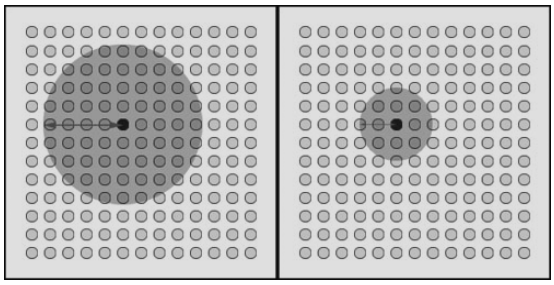
a. Inisialisasi weight pada masing-masing node.
   Sebelum training dimulai, weight masing-masing node diberikan inisialisasi terlebih dahulu. Biasanya diset dengan nilai random yang kecil ($0 < W < 1$).
b. Sebuah vektor dipilih secara random.
c. Setiap node dihitung dan dicari weight yang paling mendekati 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.
   Persamaan mencari jarak Euclidian adalah :

$$Dist = \sqrt{\sum_{i=0}^{i=n} (V_i - W_i)^2}$$

   Dengan V adalah vektor input dan W adalah vektor weight pada node.
d. 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. Proses penyusutan radius ini ditunjukkan dengan persamaan berikut:

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\lambda}\right) \quad t = 1,2,3\ldots$$

Dimana σ_0 adalah lebar pola kohonen saat pertama proses iterasi, λ adalah time constant dan t menunjukkan step saat iterasi berlangsung.



Gambar 2. Penyusutan Radius

e. Update Weight

Setelah node tetangga BMU ditentukan, maka langkah selanjutnya adalah melakukan update weight pada semua node tetangga BMU termasuk juga BMU itu sendiri. Update vektor weight dirumuskan dengan persamaan dibawah ini:

$$W(t+1) = w(t) + \Theta(t)L(t)(V(t) - W(t))$$

Dimana t adalah step iterasi, L adalah learning rate yang berkurang seiring dengan bertambahnya proses iterasi. Penyusutan learning rate dirumuskan dengan persamaan berikut ini:

$$L(t) = L_0 \exp\left(-\frac{t}{\lambda}\right) \quad t = 1,2,3\ldots$$

$\Theta(t)$ didefinisikan sebagai fungsi tetangga (neighborhood function) yang mengambarkan posisi neuron pada output, dirumuskan dengan persamaan berikut:

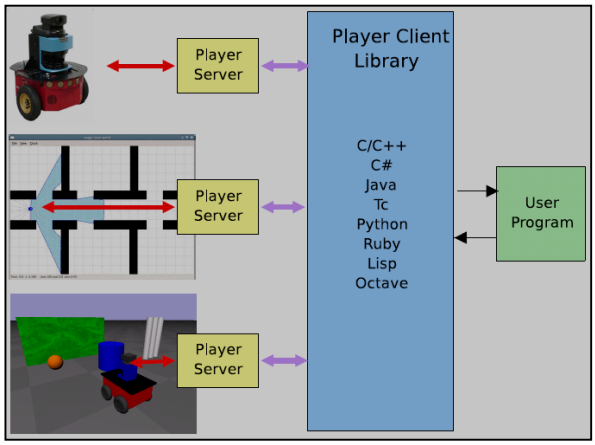
$$\Theta(t) = \exp\left(-\frac{dist^2}{2\sigma^2(t)}\right)$$

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. Ada project lain yang berhubungan dengan Player/Stage yaitu simulator Gazebo yang mampu mensimulasikan robot dalam lingkungan 3 dimensi. Project Player/Stage dimulai di Laboratorium USC Robotics Research pada tahun 1999 yang ditujukan pertama kalinya ditujukan bagi keperluan internal untuk keperluan interfacing dan simulasi. Player/Stage/Gazebo merupakan software yang bersifat Open Source dibawah lisensi GNU General Public Lisence, yang memungkinkan semua kode program dari project Player/Stage bebas untuk digunakan, didistribusikan dan dimodifikasi. Player/Stage dibuat oleh sekelompok peneliti yang tergabung dalam international team of robotics researchers, dan digunakan pada banyak lab diseluruh dunia.

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. Dengan demikian kita dapat membuat program dengan menggunanakan simulasi dalam simulator, kemudian menjalankan program pada real-robot dengan perubahan minimal. Program client dapat ditulis dalam banyak bahasa pemrograman yang mendukung pemrograman socket. saat ini library yang sudah disediakan agar dapat berinteraksi dengan server adalah librari untuk bahasa pemrograman C, C++, TCL, LISP, Java dan Python.



Gambar 3. Model Client Server pada Player/Stage

DESAIN SISTEM

Desain Simulasi

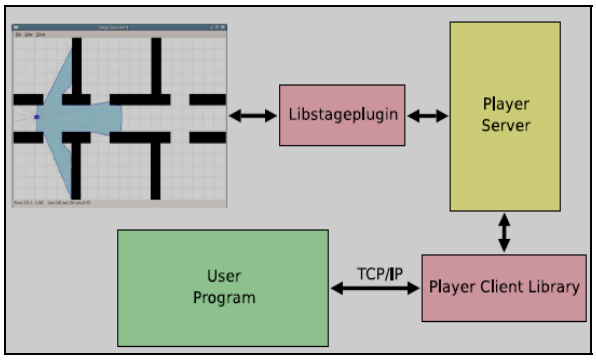
Simulasi yang digunakan adalah Player/Stage dengan menggunakan player sebagai server dan stage sebagai simulatornya. Player menyediakan interface, device dan driver yang akan digunakan untuk membangun sebuah robot beserta lingkungannya. definisi interface, device dan driver ini dituliskan dalam sebuah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

dan driver yang sudah didefiniskan dalam file konfigurasi stage akan didefinisikan pula modelnya yang ditulis dalam file konfigurasi stage.

Libstageplugin adalah plugin untuk player yang memungkinkan player dapat mengakses robot simulasi dalam stage seperti layaknya mengakses robot sebenarnya. Setelah player dapat mengakses robot simulasi tersebut

User program yang digunakan untuk mengontrol robot ditulis dengan menggunakan bahasa C++ memanfaatkan library yang disediakan oleh player, yaitu libplayerc++

Hubungan antara player, stage dan user program ditunjukkan oleh Gambar 4 .



Gambar 4. Desain Simulasi

Desain Robot

Pada penelitian ini, sonar yang dipakai ada 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.

Jika R adalah jarak yang diperoleh dari tranducer, r adalah jarak sonar dari titik pusat robot, Θ adalah sudut hadap robot relatif terhadap lingkungannya, α adalah sudut hadap sonar relatif terhadap robot, maka jarak suatu benda yang ditangkap oleh sonar dapat dihitung dengan persamaan :

$$rx = (R + r) * \cos(\Theta + \alpha)$$
$$ry = (R + r) * \sin(\Theta + \alpha)$$

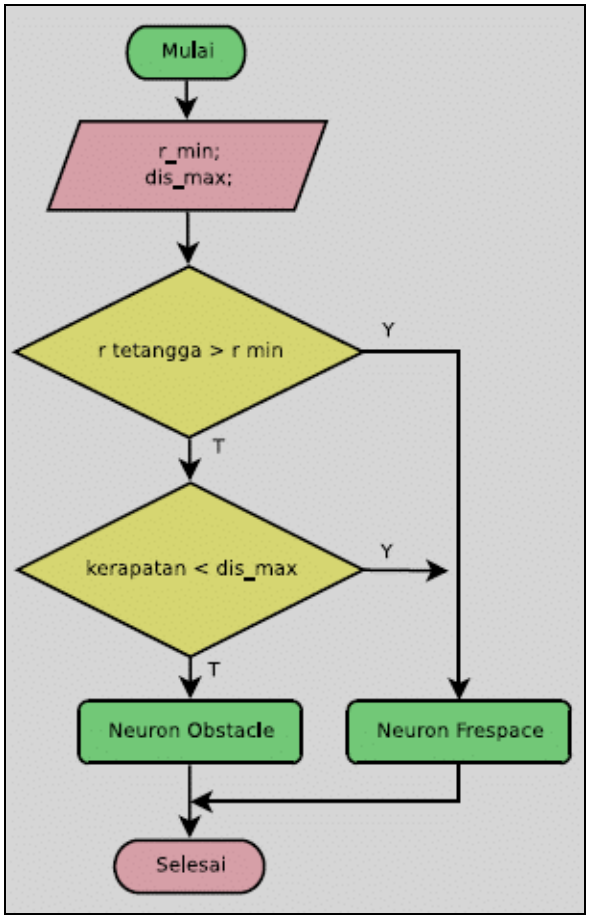
Jika px adalah koordinat x posisi koordinat x robot, dan py adalah koordinat posisi y robot, maka letak benda yang ditangkap oleh sonar bisa dihitung dengan persamaan:

$$x = px + rx$$
$$y = py + ry$$

Pembelajaran (Learning)

Proses learning dilakukan oleh neural pada layer kohonen. Input yang digunakan dari pelatihan tersebut berasal dari data yang dikumpulkan oleh robot dengan menggunakan sonar yang dipunyai oleh robot [7].

Sesudah proses learning selesai, neural pada layer kohonen 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 5.



Gambar 5. Penggolongan Neuron

Setelah semua neuron digolongkan menjadi neuron freespace dan neuron obstacle, proses selanjutnya adalah membuat blok dan mengolongkannya menjadi blok freespace dan blok osbstacle.

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 list.

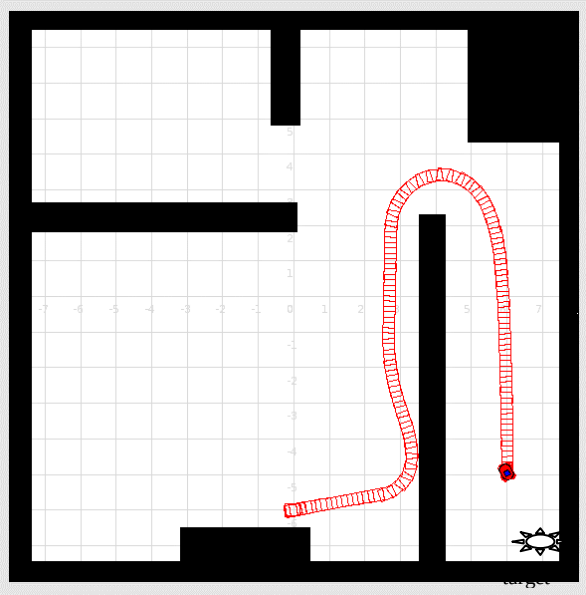
c. Cek blok semua blok yang berhubungan dengan blok awal, abaikan blok tersebut jika blok tersebut berupa blok obstacle. Dan masukkan blok-blok tersebut kedalam open list, untuk masing-masing blok tadi, set parent blok tersebut ke blok asal. Gambar 3.9 menunjukkan proses yang dilakukan terhadap masing-masing blok disekitar blok aktif.
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 :
    1. 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.
    2. 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.
    3. 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.

PENGUJIAN

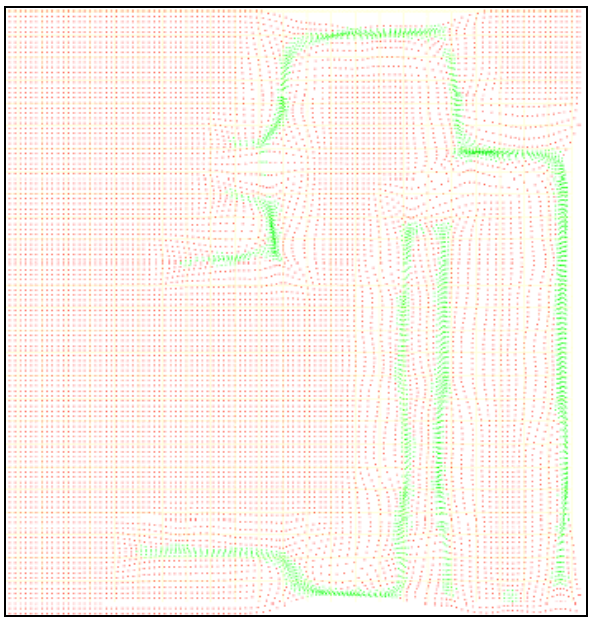
Pengujian dilakukan dengan mengunakan Stage sebagai simulator dengan client deprogram menggunakan C++.

Gambar 6. adalah gambar simulasi ketika robot dijalankan menuju ke target dengan menggunakan localization biasa, yaitu dengan menggunakan VFH localization.

Saat robot dijalankan seperti garis lintasan merah pada Gambar 6, maka sensor akan mengambil data tembok, dengan menggunakan pembelajaran algoritma kohonen, maka lingkungan akan dikenali seperti pada Gambar 7.

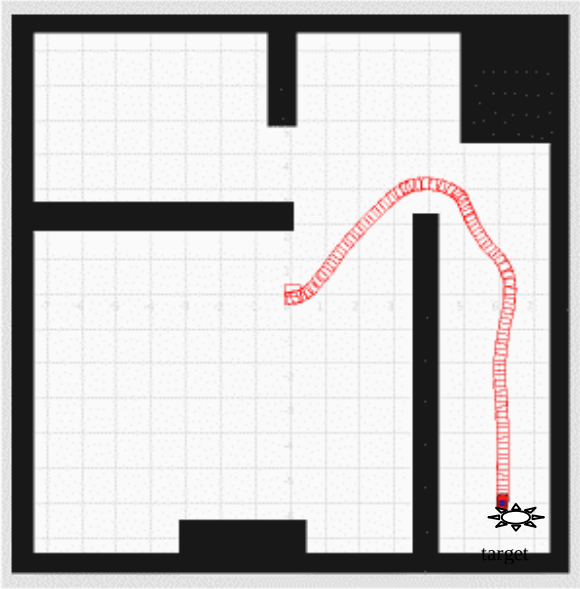


Gambar 6. Robot Dijalankan untuk Mengenali Lingkungannya

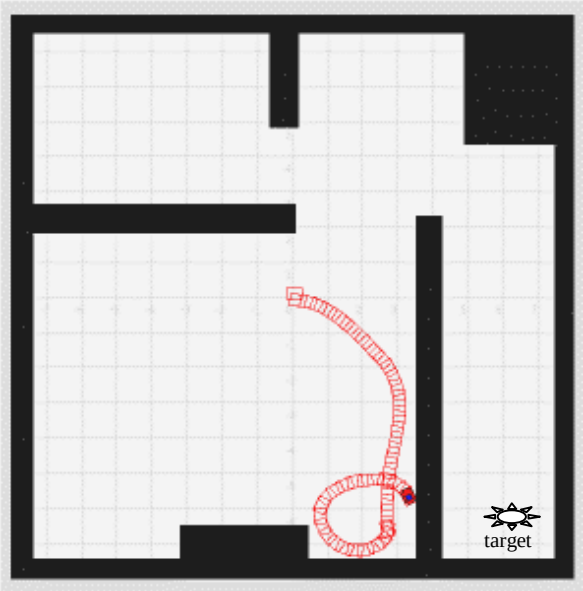


Gambar 7. Hasil Pengenalan Lingkungan.

Kemudian robot diuji dengan meletakkan pada suatu titik, dan diberi tugas untuk menuju ke titik target. Gambar 8 menunjukkan gerak robot untuk mencapai titik target dengan memanfaatkan data yang sudah dipelajari sebelumnya. Sementara gambar 9 menunjukkan gerak robot ketika menuju titik target tanpa menggunakan data yang sudah dipelajari sebelumnya.



Gambar 8. Robot berhasil mencapai titik target



Gambar 9. Robot gagal mencapai titik target

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) Player/stage/gazebo dapat digunakan untuk menguji 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.

DAFTAR PUSTAKA

1. "Kohonen Networks", http://www.cs.bham.ac.uk/resources/courses/SEM2A2/Web/Kohonen.htm
2. "Kohonen's Self Organizing Feature Maps", http://www.ai-junkie.com/ann/som/som1.html
3. George Palamas, George Papadaurakis, Manolis Kavaoussanos, "Mobile Robot Position Estimation using unsupervised Neural Networks", http://www.e-technik.fh-kiel.de/itworkshop-aveiro/papers/papadourakis.pdf
4. 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 Network*", Proc. Eighth International Conference on Advanced Robotics, pp. 459-464, Monterey, CA, July 1997.
5. 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.
6. Kian Hsiang Low, Wee Kheng Leow, Marcelo H. Ang Jr., "Integrated Planning and Control of Mobile Robot with Self-Organizing Neural Network (2002)", Proc. IEEE International Conference on Robotics and Automation (ICRA'02).

7. Mitchell, R.J, "Improved Learning for a Simple Mobile Robot", Proc Control 98, pp 1670-1675, Swansea, Sept 1998.

8. 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.