

AUTONOMOUS MOBILE ROBOT BERBASIS *PLAYER/STAGE* MENGUNAKAN 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 eksplorasi 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 pengetahuan 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 sehingga sulit untuk melakukan eksplorasi secara *online*.

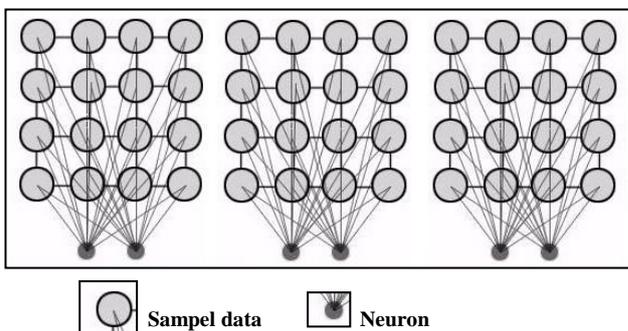
Penelitian di [10] mempunyai kendala kecepatan proses. dalam ini menyajikan bagaimana cara untuk meningkatkan kecepatan proses pembelajaran bagi robot dalam mengenali lingkungannya. Percepatan dilakukan dengan cara menjalankan proses pembelajaran menggunakan komputasi *parallel*. Pemrosesan *parallel* 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 keseluruhan 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 paralel *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 :

- Inisialisasi *weight* pada masing-masing *node*.
Sebelum pembelajaran dimulai, *weight* masing-masing *node* diberikan inisialisasi terlebih dahulu.
- Master* akan membagi lingkungan menjadi area-area yang lebih kecil, masing-masing area akan dikirimkan kepada *slave* untuk dilakukan proses pembelajaran.
- Input berupa sebuah vektor merupakan hasil deteksi dari salah satu sensor pada robot.
- Apabila vektor input masuk pada satu area tertentu, maka proses selanjutnya dikerjakan oleh komputer *slave* yang bersangkutan.
- 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]
- 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

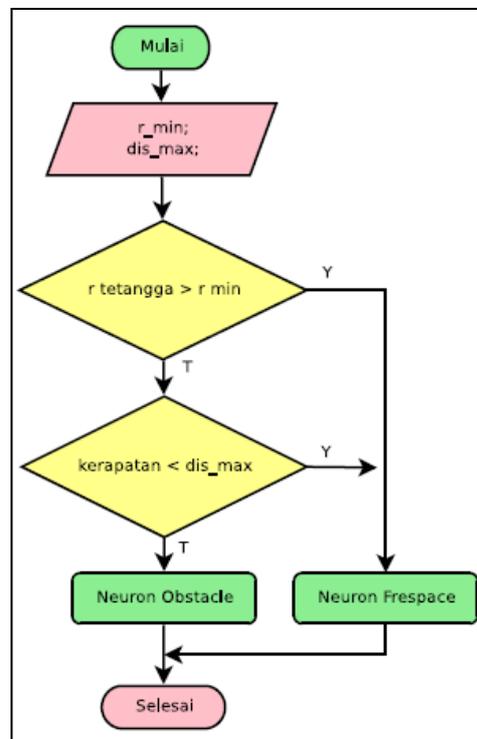
Simulasi yang digunakan adalah *Player/Stage* dengan menggunakan *player* sebagai *server* dan *stage* sebagai simulatornya. *Player* menyediakan *interface*, *device* dan *driver* yang akan digunakan untuk membangun sebuah robot beserta lingkungannya. definisi *interface*, *device* dan *driver* ini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



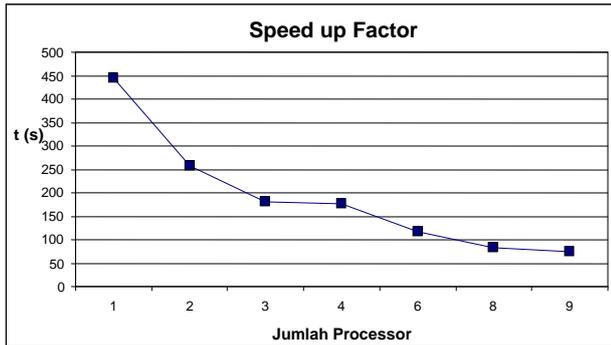
Gambar 2. Penggolongan *neuron*

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*

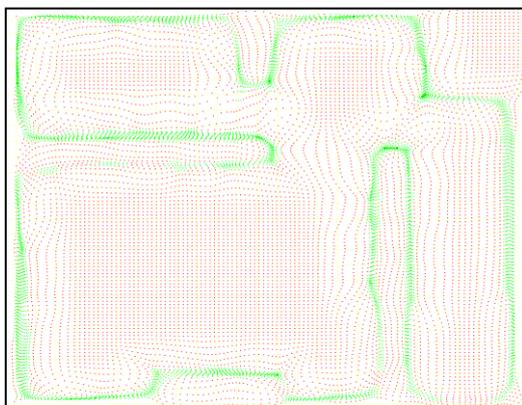
Tabel 1. Tabel Hasil percobaan

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.



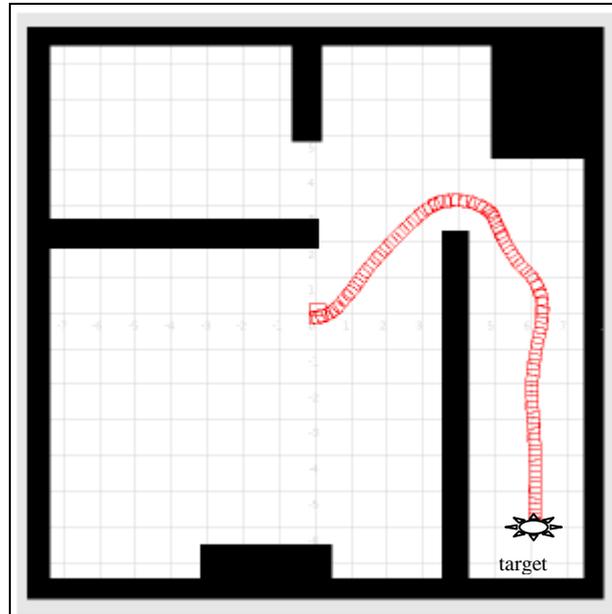
Gambar 6. Hasil pengenalan lingkungan

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.

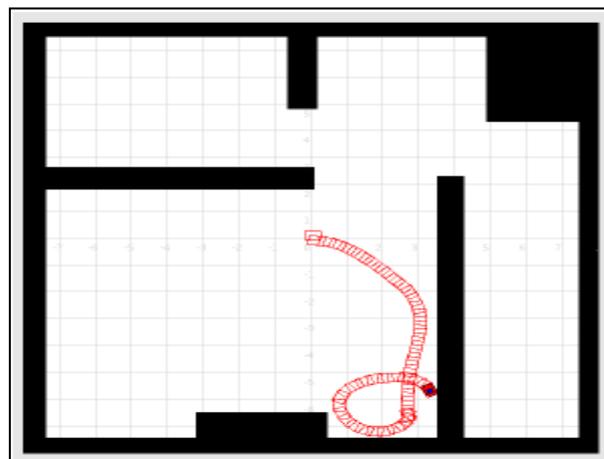
Kemudian robot diuji dengan meletakkan pada suatu titik, dan diberi tugas untuk menuju ke titik target. Jalur yang digunakan untuk robot untuk mencapai titik tujuannya ditentukan dengan

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

- 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