Algoritma Rekursif Untuk Pemetaan Jalur *Autonomous Line Follower*

Sri Wahyuni
Program Studi Mekatronika
Fakultas Teknik Universitas Trunojoyo
Bangkalan, Indonesia
yunhix@yahoo.com

Abstrak—***Autonomus Line Follower (ALF)* sebagai robot mandiri penjejak garis ketika difungsikan untuk mengeksplorasi *line maze*, memerlukan proses pemetaan jalur (*mapping*). Untuk menemukan *goal* ALF harus bisa mengeksplorasi keseluruhan *maze* tidak melewati node secara berulang-ulang. M*aze* berbentuk persegi yang dibangun dengan model konstruksi acak, artinya titik *start* dan *goal* bisa diletakkan dimana saja pada *maze.* Sehingga untuk kebutuhan *maze generated algorithm* yang sesuai dengan jenis *environment* adalah algoritma rekursi. Salah satu keuntungan menggunakan algoritma rekursif untuk permasalahan eksplorasi *maze* memungkinkan *line follower* menemukan posisi *goal* meskipun terletak ditengah *maze*, suatu hal yang tidak dapat dilakukan oleh algoritma eksplorasi lain seperti *left wall follower*. Kesuksesan penelusuran ini sangat dipengaruhi oleh kestabilan ALF untuk mampu berjalan mengikuti garis dengan mulus dan stabil. Untuk itu diperlukan satu metode lagi yang membantu kestabilan yaitu dengan PID. Ternyata tidak seperti simulasi, implementasi riil *autonomous line follower* dengan tiga sensor tidak berjalan sempurna. Bentuk *maze* persegi yang penuh dengan tikungan tajam tidak sebanding dengan jumlah sensor, sehingga pengenalan tikungan menjadi tidak maksimal dan tingkat error mencapai 70%. Dengan penambahan sensor, algoritma rekursif dapat bekerja dengan sangat baik, untuk permasalahan *maze* persegi.**

Kata kunci—*line follower*; rekursif; eksplorasi; *maze; left wall follower;*

I. PENDAHULUAN

Permasalahan utama dalam *mobile robot* adalah masalah pengendalian pergerakan (*motion control)* dan pemetaan jalur (*mapping)*, termasuk mengenai *Autonomous Line Follower* (ALF) penjelajah *maze* yang menjadi topik bahasan dalam penelitian ini. *Autonomous line follower* bergerak dari posisi awal (*start)* menuju *goal* dengan menjelajahi *line maze* yang dibuat dengan model konstruksi acak (*random*). Dalam permasalahan *motion control* pada *mobile robot* data-data input dapat berupa posisi dan orientasi dari robot, yang diperoleh dari sensor sebagai ”indera” atau *interface* dunia nyata dengan robot. Sensor dapat ditambahkan sesuai dengan kebutuhan dan fungsi robot, antara lain sebagai penglihatan, pendengaran maupun perasa.

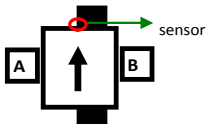
Sebelumnya Thomas Braunl telah menyelesaikan penelitian berupa *software* simulasi *mobile robot Eyesim* untuk aplikasi *maze search* menggunakan algoritma rekursif dan *flood fill*. Tugas *mobile robot* adalah mengeksplorasi *maze* untuk menemukan *goal* dengan jarak minimum mampu diselesaikan dengan sempurna[1]. Tetapi Thomas Braunl juga mempunyai anggapan bahwa keadaan ”sempurna” seperti yang disimulasikan seringkali tidak muncul dalam kondisi yang sebenarnya (*real world implementation*). Pertimbangannya, kondisi riil aktuator tidaklah sesempurna seperti yang diharapkan dalam kajian matematikanya. *Noise*, non linieritas dan keterbatasan jangkauan operasi hampir selalu mengiringi instalasi aktuator dan sensor dalam rangkaian[2]. Sehingga salah satu tujuan penelitian ini adalah untuk menguji kebenaran dari pernyataan Thomas Braunl tersebut, atau disebut sebagai *transformasi computer science problem* menjadi *computer engineering problem* [3].

Untuk merencanakan pemetaan jalur ALF didalam suatu lingkungan tertentu, baik robot maupun lingkungan perlu untuk dimodelkan. Selayaknya manusia yang akan bepergian ke lokasi lain, dia harus tahu posisi sekarang berada untuk memperkirakan berapa lama dia menuju ke lokasi yang diinginkan, dengan apa, dan bagaimana untuk bisa sampai kesana tepat waktu, demikian halnya dengan robot. Pada umumnya meliputi tiga pertanyaan berikut : “Dimana saya ?”, “Kemana tujuan saya ?”, “Bagaimana cara saya menuju ke sana ?” dan untuk menjawab pertanyaan-pertanyaan inilah diperlukan adanya proses *mapping*, *localization* dan sistem navigasi *mobile robot*.

Pada penelitian ini robot membangun sendiri peta lingkungan yang akan di laluinya (*unknown line maze*) dengan sensing. Sejalan dengan proses *localization* pada saat yang sama terjadi proses *mapping* yaitu proses *generate* peta dari lingkungan sesungguhnya. *Map* didapatkan dari hasil rekam ALF mengeksplorasi antar *node*, merekam posisi dan orientasi antar tiap *node* tersebut terhadap titik *start* dan *goal*, proses akan berulang terus sampai ditemukan *goal*. Kegiatan ALF untuk men-*generate maze* sampai menemukan *goal* hingga membentuk peta perjalanan sebagai navigasi diperlukan *maze generated algorithm* [7], yaitu algoritma rekursif.

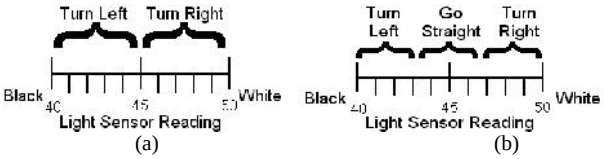
II. *AUTONOMOUS LINE FOLLOWER (ALF)*

Mobile robot yang memadukan *autonomous* dan *line follower robot* di desain untuk menghasilkan *robot* penjejak garis berwarna kontras (contoh hitam dan putih) yang mampu bergerak dengan mengenali dan mengikuti garis tanpa memerlukan kendali langsung dari manusia [5]. Sensor bertugas "membaca" warna garis tersebut dan mengirimkan data untuk menentukan tindakan selanjutnya atau disebut sebagai proses *sensing*. Robot di program mengikuti tepi garis antara hitam dan putih disisi sebelah kiri lintasan (*left hand follower*)[1]. Dengan begitu ketika robot berada di jalur hitam maka dia harus segera berputar (sebesar *error*) ke kiri dan ketika berada dijalur putih ke kanan. Pernjelasan tersebut seperti diperlihatkan pada Gambar 1.



Gambar. 1. Posisi robot di lintasan

Warna yang dibaca sensor pada perjalanannya dikembalikan dalam bentuk nilai, misalkan ketika "melihat putih" diberikan nilai 50 dan "melihat hitam" diberikan nilai 40. *Range* nilai ini dibuat untuk memudahkan menggambarkan, seperti ditunjukkan Gambar 2.

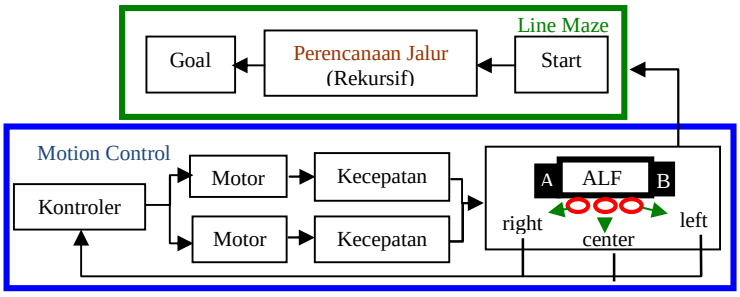


Gambar.2 (a) dan (b) keduanya konsep pemodelan range nilai sensor

Jadi jika level sensor kurang dari 43 maka robot diberi perintah untuk bergerak ke kiri, lurus untuk nilai yang dibaca 44 sampai 47 dan ke kanan untuk nilai lebih besar dari 47[4].

III. PERANCANGAN SISTEM

Pelacakan garis (*motion control*) bertanggung jawab mengatur motor agar tetap aman berjalan dua arah pada berbagai model lintasan *line maze*. Hal ini erat kaitannya dengan mekanisme pembacaan data permukaan *line maze* oleh sensor. Banyak faktor yang mempengaruhi kesalahan pembacaan data dari sensor, misalkan intensitas cahaya dalam ruangan, sensitifitas dari sensor kemampuan lantai memantulkan warna yang diterimanya dan penyebab lainnya. Error ini mempengaruhi kestabilan ALF untuk bisa berjalan mengikuti garis hitam aman dan lancar. Semakin besar error, maka pergerakan ALF semakin tidak stabil dan besar kemungkinan untuk keluar lintasan

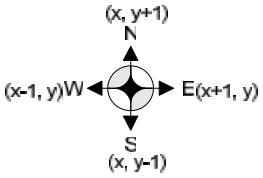


Gambar 3 Diagram blok pembagian modul

Robot yang bergerak pada lingkungan nyata (*real word*) tidak serta merta memodelkan keadaan lingkungan sesungguhnya agar bisa di mengerti oleh robot dan untuk diambil suatu tindakan dalam mencapai tujuan yang diinginkan. Untuk mengetahui posisi dan orientasi terhadap lingkungannya, robot memerlukan bantuan dari sensor yang melakukan penginderaan. Data yang diperoleh dari pembacaan sensor dikontrol dengan PID untuk menyeimbangkan kecepatan motor kanan dan kiri pada saat menemui tikungan dan mendapatkan error yang besar. Dengan begitu pekerjaan selanjutnya untuk perencanaan jalur dapat dimulai, seperti diperlihatkan pada Gambar 3 peran dari masing-masing komponen dalam sistem ini.

A. *Sistem Koordinat*

Robot berkendara dengan jarak tertentu dan berputar dengan sudut tertentu pada sistem koordinat lokal dalam berbagai aplikasi bagaimanapun penting untuk mengawali membangun sebuah peta (pada *unknown environment*) atau merencanakan sebuah jalur (pada *known environment*) [6]. Titik-titik (*node*) jalur ini biasanya dikhususkan pada *global* atau *world coordinates*.



Gambar 4. *Global and local coordinat*

Setiap kali bergerak selangkah robot melakukan pengecekan perubahan posisi dan orientasinya terhadap titik *start*. Sehingga dapat membuat keputusan harus bergerak ke arah mana selanjutnya dan sisi koordinat mana yang harus bertambah atau berkurang nilainya. Seperti sistem koordinat Gambar 4, dari awal ditentukan bahwa utara (*north*) selalu pada posisi lurus (*straight*). Jika arah utara diketahui maka arah mata angin secara otomatis akan diketahui.

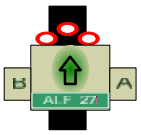
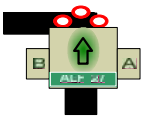
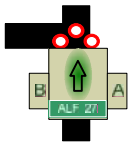
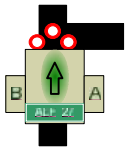
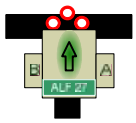
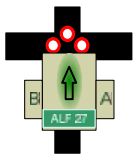
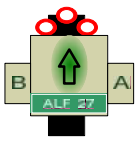
B. *Localization dan Navigation*

Localization diperlukan untuk meyakinkan bahwa jalur yang dilaluinya tidak terulang-ulang atau sebaliknya malah tersesat keluar jalur yang berakibat tidak menemukan *goal* [3]. Berikutnya merancang keperluan navigasi (*navigation*) untuk *mobile robot* ALF. Misalkan ketika sensor membaca

ISBN: 978-602-7776-72-2

hitam dimisalkan “1” dan membaca putih “0” maka Tabel 1. berikut menjelaskan lebih rinci bagaimana cara kerja sensor untuk membedakan tipe percabangan.

Tabel 1. Delapan macam kemungkinan percabangan

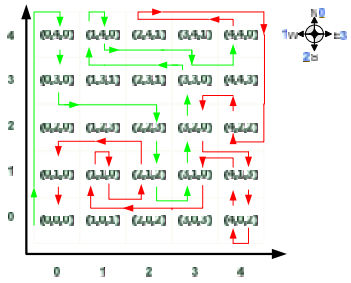
Kemungkinan Percabangan	Tipe Percabangan	Keterangan
	• Jalan lurus • 010 : LeftSensor = 0 CenterSensor = 1 RightSensor = 0	• ALF bergerak lurus • Misal current_dir = 0, maka next_dir =0 • Misal current_dir = 1, maka next_dir =1 • Misal current_dir = 2, maka next_dir =2 • Misal current_dir = 3, maka next_dir =3
	• Belok kiri • 100 : LeftSensor = 1 CenterSensor = 0 RightSensor = 0	• ALF bergerak lurus • Misal current_dir = 0, maka next_dir =1 • Misal current_dir = 1, maka next_dir =2 • Misal current_dir = 2, maka next_dir =3 • Misal current_dir = 3, maka next_dir =0
	• Belok kanan • 001 : LeftSensor = 0 CenterSensor = 0 RightSensor = 1	• ALF bergerak lurus • Misal current_dir = 0, maka next_dir =3 • Misal current_dir = 1, maka next_dir =0 • Misal current_dir = 2, maka next_dir =1 • Misal current_dir = 3, maka next_dir =2
	• Pertigaan lurus-kiri • 110 : LeftSensor = 1 CenterSensor = 1 RightSensor = 0	• ALF bergerak lurus sampai *dead end* kemudian kembali dan belok kanan • Misal urutan dir : 0 – 0 – 2 – 1
	• Pertigaan lurus-kanan • 011 : LeftSensor = 0 CenterSensor = 1 RightSensor = 1	• ALF bergerak lurus sampai *dead end* kemudian kembali dan belok kiri • Misal urutan dir : 0 – 0 – 2 – 3
	• Pertigaan kiri-kanan • 101 : LeftSensor = 1 CenterSensor = 0 RightSensor = 1	• ALF bergerak ke kiri sampai *dead end* kemudian kembali dan bergerak lurus • Misal urutan dir : 0 – 1 – 1 – 3
	• Perempatan • 111 : LeftSensor = 1 CenterSensor = 1 RightSensor = 1	• ALF bergerak lurus sampai *dead end* kemudian kembali dan bergerak ke kiri sampai *dead end* dan kembali baru bergerak ke kanan • Misal urutan dir : 0 – 0 – 2 – 3 – 3 – 1 – 1
	• Putar balik / buntu/ *dead end* • 000 : LeftSensor = 0 CenterSensor= 0 RightSensor = 0	• ALF menemui jalan buntu (*dead end*) • Misal current_dir = 0, maka next_dir =2 • Misal current_dir = 1, maka next_dir =3 • Misal current_dir = 2, maka next_dir =0 • Misal current_dir = 3, maka next_dir =1

Jika current_*dir* adalah arah ALF saat ini maka *next_dir* adalah arah pergerakan selanjutnya. Perubahan arah satu nilai dari nilai arah sebelumnya (*next_dir* – *current_dir* = 1) berarti ALF berputar 90^0 berlawanan arah jarum jam dan -90^0 searah jarum jam. Sedangkan perubahan arah dua nilai dari nilai arah sebelumnya (*next_dir* – *current_dir* = 2) berarti ALF berputar 180^0 berlawanan/searah jarum jam.

Area lingkungan tempat ALF bekerja ditentukan sebagai tipe *unknown environment* yang berbentuk persegi (matrik n x n), terbagi dalam *node-node*. Antar *node* mempunyai jarak sama yang pada lingkungan sebenarnya ditandai oleh *landmark* berwarna hijau, *start* warna kuning, *goal* warna merah. Setiap *node* mempunyai posisi koordinat dan arah terhadap titik *start*. Jadi setiap pergerakan dari ALF arah pergerakannya selalu mengacu pada posisi *start*.

Ditentukan urutan prioritas berjalan ALF adalah lurus (front_open), kiri (left_open) dan kanan (right_open). Urutan ini yang kemudian akan direkursi seperti ditunjukkan *pseudocode* Tabel 2.

ALF ditentukan selalu mulai bergerak dari titik *start* pada (rob_x, rob_y, dir) = (0,0,0) yaitu koordinat (0,0) ke arah utara (dir = 0, north). Koordinat (0,0) terletak di pojok kiri bawah, dan arah 0 adalah kearah atas. Selama ada jalan lurus (front_open) yang belum pernah dikunjungi maka percabangan kiri dan kanan diabaikan hanya dicatat saja alamat koordinatnya. Setelah bertemu dengan jalan buntu (*dead end*) maka ALF kembali ke percabangan terdekat dari posisi *dead end* berjalan seperti itu seterusnya sampai jumlah *node* yang sudah ditandai sama dengan jumlah *cell* yang sudah ditentukan di awal (luas *maze*). Sampai kemudian terbentuk *path mapping* ALF secara lengkap seperti pada Gambar 5, setiap titik *node* arah pergerakan mengacu pada titik *start* dan ketentuan nilai arah mata angin ada pada Gambar 4.



Gambar 5. Jalur perjalanan ALF

Implementasi algoritma rekursif dalam *pseudocode* untuk pemetaan jalur seperti terlihat pada Tabel 2.

Tabel 2. *Pseudocode* pemetaan jalur dengan algoritma rekursif

```
void MoveToFollow()
{
   If (alf.front_open())
   {
      alf.go_one()
      MoveToFollow()    // recurtion base, recursive call
    }
   else  if(alf.left_open())
    {
```

```
        alf.go_one()
        MoveToFollow()    // recurtion base, recursive call
      }
    else  if(alf.right_open())
    {
        alf.go_one()
        MoveToFollow()    // recurtion base, recursive call
      }
    else
    {
        alf.stop()        // base case,  rekursif berhenti
      }
}
```

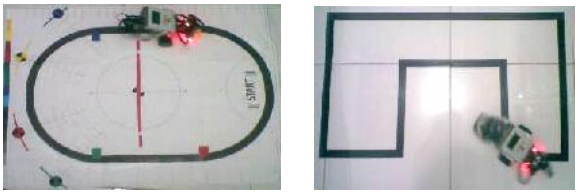
4	15	14	13	12
3	12	11	10	11
2	3	4	9	12
1	8	5	8	9
0	7	6	7	10

Gambar 6. Hasil pemetaan jalur algoritma rekursif

. Nilai-nilai yang tertera pada setiap *node* adalah jarak *node* tersebut dari titik *start.*

IV. PENGUJIAN

Implementasi dilakukan secara riil, termasuk untuk ujicoba pelacakan garis yang dilakukan di atas lantai ubin warna putih dengan garis warna hitam terbuat dari *cloth type* selebar 2,4 cm. Ujicoba menggunakan dua model lintasan, pertama berbentuk *elips* dengan tikungan melingkar (Gambar 7 (a)) dan kedua berbentuk persegi tak beraturan dengan tikungan tajam bersudut lancip (90^0), seperti Gambar 7 (b). Mula-mula menetapkan nilai *setpoint*, tidak ditentukan sebagai konstanta, tetapi didapatkan pada saat awal *line tracer* akan dijalankan dengan meletakkannya ditengah pita hitam. Nilai pertama sesaat setelah *line tracer* dijalankan itulah yang dicatat sebagai nilai *setpoint* dan nilai-nilai selanjutnya disimpan sebagai nilai *actual position.* Selisih antara nilai *setpoint* dan *actual position* ini yang selanjutnya disimpan sebagai error. Secara berturut-turut masing-masing sensor menghasilkan error yaitu, r*ight* sensor menghasilkan error1, *center* sensor error2 dan *left* sensor error3.



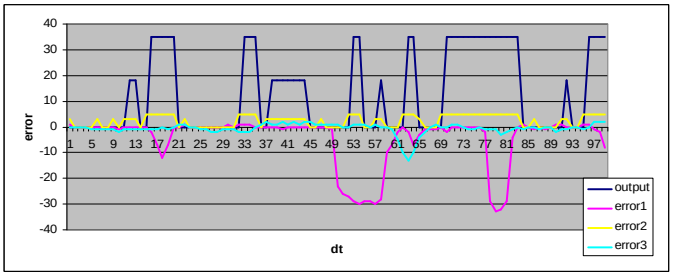
(a) (b)

Gambar 7. (a) Lintasan elips (b) Lintasan persegi

Ada tiga sensor yang digunakan tetapi modifikasi PID hanya diaplikasikan pada *center* sensor (menghasilkan error2), sensor kiri dan kanan menyeimbangkan dan menyesuaikan pergerakan *center* sensor. Kombinasi parameter nilai Kp, Ki, Kd dan kecepatan seperti tertera pada Tabel 3. *Line follower* berada pada posisi stabil jika nilai error2 sebesar 0 atau mendekati 0, artinya robot bertahan pada lintasan garis hitam.

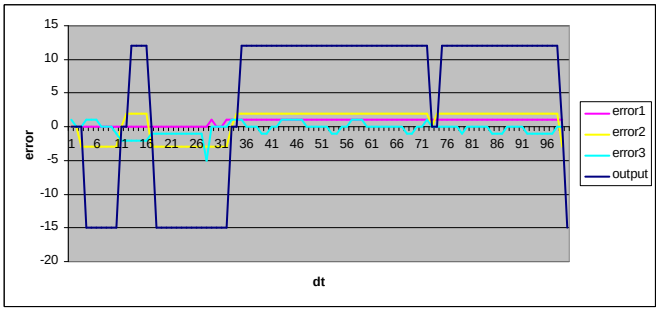
Tabel 3. Kombinasi parameter Kp, Kd, Ki, speed berdasarkan kondisi lapangan

No	Konstanta Kp/Ki/Kd	Tikungan tajam	Tikungan ringan	Jalan lurus
1	Kp	7	6	5.5
2	Ki	2.3	2	1.5
3	Kd	0.0002	0.0002	0.0002
4	Speed	0.5*refspeed	0.25*refspeed	refspeed



Gambar 8. Grafik data error dan output per dt model elips

Pergerakan dari *output* (biru tua) seperti terlihat pada grafik Gambar 8 mengikuti pergerakan error2 (kuning) yaitu error yang dihasilkan oleh *center* sensor. Nilai error tertinggi adalah 5 dan *output* menstabilkannya dengan nilai 35. Tidak ada nilai error2 yang negatif semuanya 0-5 positif, artinya roda robot bergerak antara garis hitam dan putih sebelah kiri. Grafik *output* dan error untuk model elips lebih dinamik dari grafik model persegi tak beraturan ini dikarenakan model persegi (Gambar 9) tikungannya lebih tajam tetapi pendek. Jadi pergerakan di garis lurus lebih panjang sehingga lebih stabil, tidak banyak perubahan kecepatan.



Gambar 9. Grafik data error dan output per dt model persegi tak beraturan

Grafik pada Gambar 9 menunjukkan nilai error2 mencapai titik negatif (-3) sampai positif 2, *output* juga mengikuti pergerakan ini dengan nilai antara negatif (-15) sampai positif 12, sangat stabil karena nilai error yang kecil sedangkan gerakan antara ke kanan dan kiri cukup merata.

ISBN: 978-602-7776-72-2 © Universitas Udayana 2013

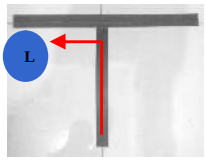
Dari hasil grafik Gambar 8 dan Gambar 9 maka bisa disimpulkan permasalahan mengenai *tracking line* berjalan relatif aman dan lancar tanpa kehilangan jalur pada kedua model lintasan dan dengan kombinasi parameter Kp, Kd, Ki yang digunakan sesuai Tabel 3. Mengurangi kecepatan ketika terjadi *error* dan pada saat sampai di tikungan. Dengan begitu nilai parameter Kp, Kd dan Ki serta pengaturan kecepatan juga sudah cukup memenuhi target. Berbeda model lintasan mempengaruhi tingkat error, menunjukkan tingkat kesulitan jalur yang dilalui.

Pemetaan Jalur

Ujicoba dilakukan secara bertahap mulai dari pengenalan pertigaan, perempatan, *dead end* atau putar balik.

A. *Ujicoba 1*

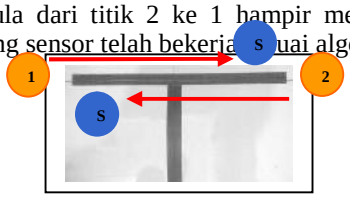
Ujicoba pada pertigaan, dilakukan pada *test pad* model ”T”, seperti pada Gambar 10 berikut. Diinginkan ALF setelah bergerak lurus selalu belok kiri (sesuai prioritas, lurus – kiri – kanan). Hasilnya dari sepuluh kali ujicoba, tujuh kali belok kiri dan tiga kali belok kanan jadi tingkat keberhasilan 70%.



Gambar 10. Model lintasan ”T” untuk ujicoba belok kiri

B. *Ujicoba 2*

Ujicoba ke 2 ini masih berkaitan dengan pertigaan dan menggunakan model yang sama yaitu ”T”, tetapi titik memulai perjalanannya dirubah. jika dari titik 1 pilihannya lurus atau belok kanan sedangkan dari titik 2 pilihannya lurus atau belok kiri. Sesuai aturan prioritas adalah lurus, kiri, kanan, sehingga baik bergerak dari titik 1 maupun 2 seharusnya robot bergerak lurus. Gambar 11 menunjukkan titik mulai pergerakan ALF, Selama bergerak bolak balik dari titik 1 ke 2 dan sebaliknya sama sekali tidak berbelok kanan. Demikian pula dari titik 2 ke 1 hampir mencapai 100%. Masing-masing sensor telah bekerja sesuai algoritma.

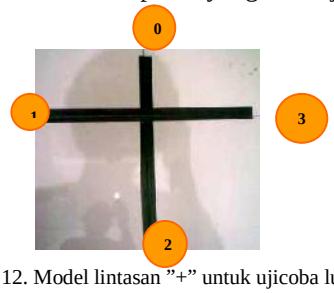


Gambar 11. Model lintasan ”T” untuk ujicoba lurus

Ujicoba belok kanan tidak dilakukan karena merupakan alternatif terakhir jika sudah tidak ada jalan lurus dan belokan ke kiri. Maka pengujian belok kanan bisa melihat hasil ujicoba 1, jika errornya kecil berarti untuk prioritas pilihan juga dianggap mempunyai kemungkinan memilih pergerakan yang benar juga besar.

C. *Ujicoba 3*

Berikut ini ujicoba yang dilakukan pada model lintasan perempatan atau ”+”. Seperti yang ditunjukkan pada Gambar 12.



Gambar 12. Model lintasan ”+” untuk ujicoba lurus

Dari ujicoba ini hasil yang diharapkan, dari titik manapun ALF mulai bergerak pilihannya adalah jalan lurus, sesuai prioritas pertama adalah bergerak lurus. Kenyataannya hasil percobaan seperti ditampilkan pada Tabel 4.

Tabel 4. Tabel ujicoba lurus pada perempatan

Titik mulai	Titik tujuan	Error (%)
0	2	30
1	3	20
2	0	10
3	1	20
	Rata-rata error	20

Pengujian dilakukan sebanyak 10 kali pada masing-masing titik mulai ke titik tujuan. Didapatkan hasil nilai error rata-rata sebesar 20%, error terbesar pada titik mulai 1 ke titik tujuan 2. Sedangkan error terkecil adalah dari titik mulai 2 ke titik tujuan 0 sebesar 10%.

D. *Ujicoba 4 :*

Selanjutnya ujicoba untuk posisi *dead end* atau jalan buntu, ALF harus berputar 180^0 dari orientasi saat itu. Pengujian ini masih dilakukan di model lintasan ”+” atau perempatan yang mempunyai empat titik *dead end*.

Tabel 5. Tabel ujicoba *dead end* pada perempatan

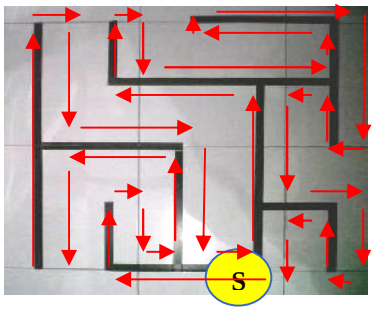
Titik mulai	Titik tujuan	Error (%)
0	2	0
1	3	10
2	0	20
3	1	0
	Rata-rata error	7.5

Pengujian pada *dead end* ini sekaligus dilakukan pada ujicoba 3, hanya saja pengamatan difokuskan pada keberhasilan pada saat ALF bertemu *dead end* dan mengambil keputusan untuk putar balik. Tingkat keberhasilan mencapai lebih dari 90%, artinya hampir setiap

kali bertemu *dead end*, ALF mampu mengenali dan memutar dirinya 180%.

E. Ujicoba 5 :

Pengujian berikutnya dilakukan pada lintasan dengan model *line maze* yang akan diselesaikan sesuai perancangan *maze* di awal, seperti ditunjukkan oleh Gambar 13. Dari 10 kali percobaan pengenalan pertigaan dan perempatan berjalan tapi prioritas jalan yang dipilih sering kali salah, tingkat error mencapai 70%. Tingkat error yang besar ini selain mengakibatkan ALF salah menentukan tipe percabangan juga kesalahan dalam membuat keputusan arah belokan. Faktor utama terjadinya error yang besar adalah jumlah sensor yang kurang, semakin banyak sensor semakin baik hasil pembacaan data, semakin kecil kesalahan error dan kesalahan menentukan percabangan.



Gambar 13. Model *line maze*

F. Ujicoba 6 :

Pengujian terakhir dilakukan pada lintasan yang sama dengan ujicoba 5, tetapi diberikan *landmark* hijau pada *line maze* sebagai penanda *node* dengan jarak yang sama antar *node*. Setiap membaca *landmark* hijau ALF berhenti sejenak dan mulai berjalan pelan beberapa saat sebelum akhirnya berjalan normal kembali. Dalam pemberhentiannya (membaaca *landmark* hijau) tersebut, ALF mengurangi jumlah *node* yang belum dikunjungi sebagai sebuah *node* yang telah dikunjungi. Jika terjadi kesalahan membaca tipe percabangan maka data yang disimpan dalam memori ALF menjadi tidak valid dan menyebabkan pengulangan eksplorasi dititik-titik yang pernah dikunjungi. Sebab setiap *node* mempunyai identitas yang membedakan *node* satu dengan lainnya, masing-masing *node* menyimpan data koordinat dan *direction* terhadap titik *start*. Dari pengujian sebanyak 10 kali putaran (10 x 24 node) ada total 20 kali ALF gagal mengenali *landmark* hijau, ditandai dengan ALF berjalan terus tanpa berhenti sejenak. Tingkat error sekitar 10%. Sedangkan kegagalan menyimpan identitas *node* mencapai 70% sama dengan ujicoba 5, akibat kekurangan sensor mengantisipasi kesalahan pembacaan data seperti yang telah diuraikan.

Gambar 14. Model *line maze* untuk pengenalan ujicoba *landmark*

Berdasarkan 6 model ujicoba, masing-masing dilakukan sebanyak 10 kali akhirnya dapat dianalisa bahwa kegagalan implementasi ini bukan disebabkan logika pemrograman ALF. Kurangnya perencanaan navigasi yang tidak mengantisipasi masalah pengaruh jumlah sensor yang digunakan dengan kemungkinan memperkecil error pembacaan data merupakan penyebab utama kegagalan. Penggunaan sensor dari tiga buah menjadi lima atau tujuh sensor cahaya atau warna akan dapat meminimalisasi kesalahan.

KESIMPULAN DAN SARAN

Setelah melakukan ujicoba berulang-ulang dan menganalisa hasil dari ujicoba, penulis mendapatkan beberapa kesimpulan. Pelacakan garis menggunakan 3 sensor (1 sensor warna ditengah, dan 2 sensor cahaya di sisi samping) dengan metode PID mampu berjalan dengan sangat aman untuk target pelacakan garis tapi tidak untuk memenuhi navigasi pemetaan jalur. Kekurangannya ada pada saat pengenalan percabangan, sedikit saja ALF tidak berada pada posisi yang seharusnya maka gagal sudah sensor menentukan tipe percabangan yang ditemui. Solusinya dapat ditambahkan jumlah sensor menjadi 5 atau 7 untuk hasil yang lebih baik lagi. Tetapi penambahan sensor artinya juga merubah desain algoritma rekursif hampir 50%, karena masing-masing data yang diperoleh dari sensor menentukan pergerakan selanjutnya. Disinilah letak kelemahan algoritma rekursif untuk pemetaan jalur.

Pelacakan garis sudah mampu berjalan secara *left/right hand follower* secara baik. Ini penting karena pada saat perjalanan menelusuri *maze* ALF tidak terus berjalan secara *left* atau *right hand follower*, namun akan berlaku bolak-balik. Contohnya setelah bertemu *dead end*, ALF yang sebelumnya berjalan *left follower* maka setelah putar balik dari *dead end* maka ALF akan berjalan secara *right hand follower*.

Penulis setuju dengan pernyataan Thomas Braunl mengenai kendala yang ditemui untuk membuat implementasi riil seringkali tidak terjadi pada simulasi. Seperti yang terjadi disini, penelitian Thomas Braunl secara simulasi sukses menerapkan kombinasi algoritma rekursif dan *flood fill* dengan 3 sensor, tidak pada penelitian implementasi riil yang penulis lakukan. Biasanya terjadi pada ketidaksesuaian desain mekanik dan elektronik dengan kondisi riil lapangan. Algoritma flood fill disini tidak ada pengaruh ketika dihilangkan, karena baru

ISBN: 978-602-7776-72-2

diimplementasikan untuk shortest path, yaitu setelah proses pemetaan jalur.

Sebagai saran, penelitian ini bisa dilanjutkan dengan menggunakan 5 atau 7 sensor warna atau cahaya tetap menggunakan algoritma rekursif. Termasuk juga memperlebar pita dan arena *maze* agar pergerakan ALF juga lebih efektif.

Sebagai pembanding algoritma bisa menggabungkan algoritma rekursif dengan algoritma A* untuk pencarian jarak minimumnya atau melanjutkan dengan menggunakan algoritma *flood fill*.

Untuk lebih menguatkan hasil kerja algoritma ini perlu adanya penelitian lanjutan yang memperhatikan waktu dan jarak minimum robot line maze.

[1] Braunl, T., dkk.. " Fault-Tolerant robot programming through simulation with realistic sensor models". Advance Robotic Systems International, 2006

[2] Pitowarno, E. "Robotika : Desain, Kontrol, dan Kecerdasan Buatan". Andi Offset. Yogyakarta, 2006.

[3] Braunl, T. "Embedded Robotic (Mobile Robot Design and Application with Embedded System". Verlag. Springer, 2008

[4] Wahyuni. S., "Analisis Penggunaan Sensor Cahaya dan Sensor Warna Untuk Kestabilan Lego NXT *Line Follower"*. Prosiding EECCIS Vol. II Control Informatics., 2012

[5] Meystel, A. "Autonomous Mobile Robots : Vehicle With Cognitive Control".World Scientific. Singapore, 1991

[6] Van den Berg, Jur P., Overmars, M. H."Roadmap-Based Motion Planning in Dynamic Environments". IEEE.Transactions on Robotic. Vol. 21. No. 5, 2005

[7] Borenstein, J. dkk."Navigating Mobile Robots – Systems and Techniquet". Wellesley. A.K.Peters, 1996

Halaman Ini Sengaja Dikosongkan

ISBN: 978-602-7776-72-2