TELEKONTRAN, VOL. 7, NO. 2, OKTOBER 2019
DOI 10.34010/telekontran.v7i2.2701
p-ISSN : 2303 – 2901
e-ISSN : 2654 – 7384

Studi Performansi Algoritma Perencanaan Jalur diantara PRM, RRT, RRT* dan *Informed*-RRT*

Performance Study of Path Planning Algorithm between PRM, RRT, RRT and Informed-RRT**

Nelci Dessy Rumlaklak[1,*], Yelly Y. Nabuasa[2], Tiwuk Widiastuti[3]
[1,2,3] Program Studi Ilmu Komputer Fakultas Sains dan Teknik Universitas Nusa Cendana
Jl. Adisucipto, Lasiana, Kelapa Lima, Kupang, Nusa Tenggara Timur 85228.
[*]Email: dessyrumlaklak@gmail.com

Abstrak – Tujuan dari penelitian ini adalah melakukan studi perbandingan performansi dari beberapa algoritma perencanaan jalur. Penelitian ini membandingkan empat algoritma perencanaan jalur yang terkenal, yaitu algoritma *Probabilistic Roadmap (PRM)*, *Rapidly-exploring Random Tree (RRT), RRT** dan *Informed-RRT**. Pengujian dilakukan melalui eksperimen berbasis simulasi menggunakan python. Pengujian dilakukan menggunakan beberapa kasus *benchmark* yang ada, yaitu lingkungan *narrow*, *maze*, *trap* dan *clutter*. Kriteria optimalitas yang dibandingkan adalah biaya jalur, waktu komputasi dan jumlah total node pada pohon yang dibutuhkan. Dari hasil pengujian, terlihat bahwa algoritma RRT memiliki waktu komputasi yang tercepat tetapi memiliki kualitas jalur yang kurang baik. Algoritma RRT*, *informed*-RRT* dan PRM memiliki kualitas jalur yang mirip. Algoritma PRM memiliki waktu komputasi tertinggi dibandingkan algoritma RRT, RRT* maupun *informed*-RRT*. Hasil penelitian ini akan memberikan informasi kepada pembaca mengenai algoritma mana yang paling cocok untuk digunakan pada aplikasi pengguna dimana terdapat beberapa parameter kerja yang hendak dioptimalkan.

Kata Kunci : Perencanaan jalur, PRM, RRT, RRT*, Informed-RRT*

Abstract *- This paper will discuss a comparative performance review of several path planning algorithms. This study compares five well-known path planning algorithms, namely the Probabilistic Roadmap (PRM), Rapidly-exploring Random Tree (RRT), RRT* and Informed-RRT* algorithm. Testing is done through simulation based experiments using python. The test was conducted using several existing benchmark cases, namely narrow, maze, trap and clutter environment. The optimality criteria compared are path costs, computational time and the total number of nodes in the tree needed. From the test results, it appears that the RRT algorithm has the fastest computing time but has a poor path quality. RRT *, informed-RRT * and PRM algorithms have similar path quality. PRM algorithm has the highest computational time compared to the RRT, RRT * and informed-RRT * algorithms. The results of this study will provide information to readers about which algorithm is most suitable for use in user applications where there are several working parameters to be optimized.*

Keywords *: Motion planning, PRM, RRT, RRT*, Informed-RRT**

I. PENDAHULUAN

Perencanaan jalur (*path planning*) adalah salah satu tugas penting dalam proses pengendalian robot otonom. Tujuan dari perencanaan jalur adalah menghasilkan jalur bebas tabrakan dari posisi awal ke posisi tujuan dengan pada lingkungan yang memiliki hambatan statis maupun dinamis. Algoritma perencanaan jalur akan mencari jalur dengan biaya jalur optimal atau mendekati optimal. Aplikasi perencanaan jalur pada robot otonom memiliki banyak kegunaan seperti pada mobil otonom (*self driving car*) [1], *Unmanned Aerial Vehicle* [2], forklift pada industry [3], ataupun pada operasi pengawasan dan misi luar angkasa [4,5]. Keterbatasan dari algoritma perencanaan jalur klasik adalah dibutuhkannya model lingkungan yang akurat sebelum algoritma perencana jalur dapat diterapkan. Agar tidak perlu membuat model lingkungan yang akurat, salah satu

solusinya adalah menggunakan metode perencanaan jalur berbasis sensor. Metode perencanaan jalur berbasis sensor memungkinkan robot untuk bekerja secara mandiri di lingkungan yang belum dipetakan. Gerakan robot dihasilkan langkah demi langkah, dengan mengumpulkan semakin banyak informasi mengenai lingkungan. Salah satu metode lainnya adalah dengan menggunakan metode perencanaan jalur berbasis graph. Algoritma pencarian jalur berbasis graph yang sering digunakan adalah Djikstra [6], A* [7], D* [8] ataupun AD* [9]. Metode perencanaan jalur lainnya yang sering digunakan adalah algoritma *Sampling Based Planning* (SBP). Algoritma SBP dapat memberikan solusi yang cepat untuk masalah yang kompleks dan memiliki dimensi yang tinggi. Algoritma SBP menggunakan pengambilan sampel acak pada ruang pencarian [3, 10]. Mungkin algoritma SBP yang paling terkenal adalah *Probabilistic Roadmap Method* (PRM) [11] dan *Rapidly-exploring Random Tree* (RRT) [12]. Algoritma RRT diusulkan oleh Lavelle [12] dimana Lavelle berhasil menerapkannya pada masalah komplek hingga dua belas derjat kebebasan. Tetapi kekurangan dari algoritma RRT adalah jalur yang dihasilkan tidak optimal. Karaman dan Frazzoli [5] mengusulkan variasi dari algoritma RRT yang dinamakan RRT*. Menggunakan dua fitur tambahan yaitu *ChooseParent* dan *Rewire*, algoritma RRT* memiliki kualitas keluaran yang bersifat asimptotik optimal. Gammell dkk. [13] lalu mengembangkan algoritma RRT* menjadi algoritma yang dinamakan *Informed*-RRT*. Algoritma *informed*-RRT* membatasi ruang pengambilan sample pada algoritma RRT yaitu hanya pada suatu daerah elips yang melingkupi titik awal dan titik akhir. Teknik pengambilan sample pada algoritma *informed*-RRT* ini diklaim dapat mempercepat laju konvergensi dari algoritma RRT*. Penelitian ini bertujuan untuk membandingkan performansi beberapa algoritma perencanaan jalur yang sering digunakan. Algoritma yang hendak dibandingkan performansinya adalah PRM, RRT, RRT* dan *Informed*-RRT*.

Memang telah ada beberapa penelitian yang melakukan perbandingan performansi dari beberapa algoritma perencanaan jalur. Karaman [5] melakukan perbandingan total jarak tempuh dan tingkat konvergensi antara algoritma RRT dan RRT*. Zaheer [14] melakukan perbandingan performansi antara algoritma RRT, PRM, *Artificial Potential Field* (APF), A* dan *Free Configuration Eigen-spaces* (FCE). Noreen [15] melakukan perbandingan performansi antara RRT, RRT* dan RRT*-Smart. Elbanhawi [10] melakukan perbandingan performansi antara RRT, biased RRT dan Bidirectional RRT. Gamell [13] melakukan perbandingan performansi antara RRT* dan informed-RRT*. Nasir [14] melakukan perbandingan performansi antara algoritma RRT* dan RRT*-Smart. Noreen [17] melakukan perbandingan performansi antara RRT, RRT* dan RRT*-AB. Tetapi sepanjang yang penulis ketahui, belum ada penelitian yang membandingkan performansi algoritma PRM, RRT, RRT* dan Informed-RRT* secara bersamaan pada kasus-kasus lingkungan *narrow*, *maze*, *trap* dan *clutter.*

Makalah ini akan membahas studi performansi dari empat algoritma perencanaan jalur yang sering digunakan, yaitu PRM, RRT, RRT* dan Informed-RRT*. Pengujian dilakukan secara simulasi menggunakan program python. Pengujian dilakukan menggunakan beberapa kasus *benchmark* pada permasalahan perencanaan jalur, yaitu lingkungan *narrow*, *maze*, *trap* dan *clutter*. Kriteria optimalitas yang dibandingkan pada pengujian adalah biaya jalur, waktu komputasi dan jumlah total node pada pohon yang dibutuhkan. Hasil penelitian ini akan memberikan informasi kepada pembaca mengenai keunggulan dan keterbatasan dari setiap algoritma. Serta akan memberikan informasi mengenai algoritma mana yang paling cocok untuk kebutuhan aplikasi pengguna.

II. METODE

Penelitian ini akan membandingkan performansi dari empat algoritma perencanaan jalur yang sering digunakan. Algoritma yang dibandingkan adalah PRM, RRT, RRT* dan Informed-RRT*.

Algoritma PRM yang digunakan terdiri dari dua proses. Proses pertama adalah pembangkitan sample point dan membangun roadmap, selanjutnya adalah membangun jalur berdasarkan roadmap yang ada menggunakan algoritma Djikstra. Algoritma proses yang pertama ini ditunjukkan pada **Gambar 1**, adapun algoritma proses yang kedua ditunjukkan pada **Gambar 2**. Ilustrasi dari proses pengambilan sample dan pembangunan roadmap ditunjukkan pada **Gambar 3**. Ilustrasi proses yang pembangunan jalur menggunakan algoritma Djikstra ditunjukkan pada **Gambar 4**.

Algorithm 1 : Generate Sample Point and Roadmap Construction Algorithm
Input :
n : number of nodes to put
k : number of closest neighbors to examine for each configuration
Output : A roadmap $T = (V, E)$

1. $V \leftarrow \emptyset$
2. $E \leftarrow \emptyset$
3. **while** $|V| < n$ **do**
4. **repeat**
5. $q \leftarrow$ a random configuration in Q
6. **Until** q is collision-free
7. $V \leftarrow V \cup \{q\}$
8. **end while**
9. **for all** $q \in V$ **do**
10. $N_q \leftarrow$ the k closest negihbors of q chosen from V according to $dist$
11. **for all** $q' \in N_q$ **do**
12. **if** (q, q') is collision-free **then**
13. $E \leftarrow E \cup \{(q, q')\}$
14. **end if**
15. **end for**
16. **end**

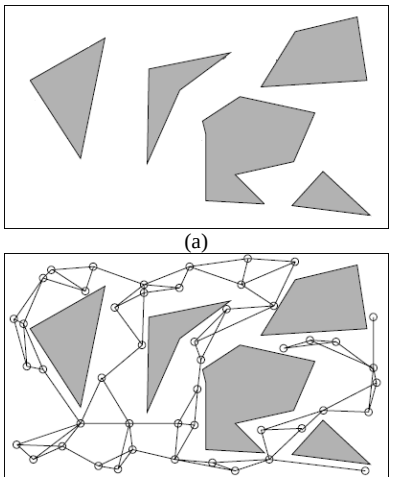
Gambar 1. Algoritma PRM untuk membangkitkan sampel dan membuat roadmap [diadaptasi dari (18)]

Algorithm 2 : Solve Query Algorithm
Input :
q_{init} : the initial configuration
q_{goal} : the goal configuration
k : the number of closest neighbors to examine for each configuration
$T = (V, E)$: the roadmap computed by algorithm 1
Output : A path from q_{init} to q_{goal} or failure

1. $N_{q_{init}} \leftarrow$ the k closest neighbors of q_{init} from V according to $dist$
2. $N_{q_{goal}} \leftarrow$ the k closest neighbors of q_{goal} from V according to $dist$
3. $V \leftarrow \{q_{init}\} \cup \{q_{goal}\} \cup V$
4. set q' to be the closest neighbor of q_{init} in $N_{q_{init}}$
5. **repeat**
6. **if** $\Delta(q_{init}, q') \neq$ NIL **then**
7. $E \leftarrow (q_{init}, q') \cup E$
8. **else**
9. set q' to be the next closest neighbor of q_{init} in $N_{q_{init}}$
10. **end if**
11. **until** a connection was successful or the set $N_{q_{init}}$ is empty
12. set q' to be the closest neighbor of q_{goal} in $N_{q_{goal}}$
13. **repeat**
14. **if** $\Delta(q_{goal}, q') \neq$ NIL **then**
15. $E \leftarrow (q_{goal}, q') \cup E$
16. **else**
17. set q' to be the next closest neighbor of q_{goal} in $N_{q_{goal}}$
18. **end if**
19. **until** a connection was successful or the set $N_{q_{goal}}$ is empty
20. $P \leftarrow shortest\ path(q_{init}, q_{goal}, T)$ using Djikstra Algorithm
21. **if** P is not empty **then**
22. **return** P
23. **else**
24. **return** failure
25. **end if**

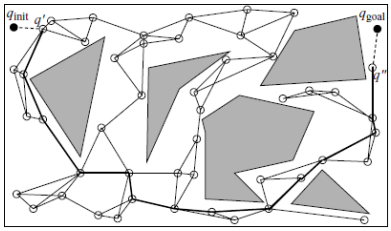
Gambar. 2. Algoritma PRM untuk membangkitkan jalur final [diadaptasi dari [18])

Algoritma RRT yang digunakan ditunjukkan pada **Gambar 5**. Pada setiap iterasi, node q_{rand} dipilih secara acak dari ruang konfigurasi Q_{free}. Node terdekat ke q_{rand} dari pohon pencarian T lalu dipilih dan dinamakan $q_{nearest}$. Selanjutnya akan dibuat cabang baru dari pohon pencarian T diantara $q_{nearest}$ dan q_{rand}. Cabang baru ini berjarak Δq dari $q_{nearest}$ dan menuju arah q_{rand}. Cabang baru ini dengan node baru bernama q_{new} akan ditambahkan ke pohon pencarian T jika diantara q_{new} dan $q_{nearest}$ tidak ada hambatan.



(a) (b)

Gambar 3. Ilustrasi pembangkitan sampel node dan roadmap oleh algoritma PRM : (a) peta lingkungan sebelum pembangkitan sampel dan roadmap, (b) peta lingkungan setelah pembangkitan sampel dan roadmap [diadaptasi dari [18])



Gambar 4. Ilustrasi pembangkitan jalur final oleh algoritma PRM

Algorithm 3 : $T = (V, E) \leftarrow RRT(q_{init})$
Input :
q_{start} : the configuration where the tree is rooted
n : maximum iteration
Output : A roadmap $T = (V, E)$

19. $T \leftarrow InitializeTree()$
20. $T \leftarrow InsertNode(\emptyset, q_{init}, T)$
21. **for** $k \leftarrow 1$ **to** N **do**
22. $\quad q_{rand} \leftarrow RandomSample(k)$
23. $\quad q_{nearest} \leftarrow NearestNeighbor(q_{rand}, Q_{near}, T)$
24. $\quad q_{new} \leftarrow Steer\ (q_{nearest}, q_{rand}, \Delta q)$
25. $\quad$ **if** $Obstaclefree(q_{new}, q_{nearest})$ **then**
26. $\quad\quad T \leftarrow InsertNode(q_{min,} q_{new}, T)$
27. **end**

Gambar 5. Algoritma RRT

Algoritma RRT* yang digunakan ditunjukkan pada **Gambar 4**. Pengembangan fitur RRT* adalah adnaya operasi ChooseParent **(Gambar 5**) dan operasi Rewire (**Gambar 6**). Seperti pada algoritma RRT, algoritma RRT secara bertahap membangun pohon pencarian T dengan mengambil sampel q_{rand} secara acak pada ruang Q_{free} (baris 4 Algorithm 3). Kemudian menambahkan lintasan baru q_{new} yang memperluas node terdekat dari pohon pencarian T dengan node q_{rand} (baris 5 – 6). Jika node baru q_{new} tersebut tidak bertabrakan dengan hambatan, maka algoritma RRT akan menambahkan node baru q_{new} kepada pohon pencarian T lalu melanjutkan iterasi selanjutnya.

Algorithm 4 : $T = (V, E) \leftarrow RRT^*(q_{init})$
Input :
q_{start} : the configuration where the tree is rooted
n : maximum iteration
Output : A roadmap $T = (V, E)$

25. $T \leftarrow InitializeTree()$
26. $T \leftarrow InsertNode(\emptyset, q_{init}, T)$
27. **for** $k \leftarrow 1$ **to** N **do**
28. $\quad q_{rand} \leftarrow RandomSample(k)$
29. $\quad q_{nearest} \leftarrow NearestNeighbor(q_{rand}, Q_{near}, T)$
30. $\quad q_{new} \leftarrow Steer\ (q_{nearest}, q_{rand}, \Delta q)$
31. $\quad$ **if** $Obstaclefree(q_{new}, q_{nearest})$ **then**
32. $\quad\quad Q_{\boldsymbol{near}} \leftarrow Near(T, z_{new})$
33. $\quad q_{min} \leftarrow ChooseParent(q_{new}, Q_{near}, q_{nearest})$
34. $\quad\quad T \leftarrow InsertNode(q_{min,} q_{new}, T)$
35. $\quad\quad T \leftarrow Rewire(T, Q_{near}, q_{min}, q_{new})$
36. **end**

Gambar 6. Algoritma RRT*

Algorithm 5:
$q_{min} \leftarrow ChooseParent(q_{rand}, Q_{near}, q_{nearest}, \Delta q)$

27. $q_{min} \leftarrow q_{nearest}$
28. $c_{min} \leftarrow Cost(q_{nearest}) + c(q_{rand})$
29. **for** $q_{near} \in Q_{near}$ **do**
30. $\quad q_{path} \leftarrow Steer(q_{near}, q_{rand}, \Delta q)$
31. $\quad$ **if** $ObstacleFree(q_{path})$ **then**
32. $\quad\quad c_{new} \leftarrow Cost(q_{near}) + c(q_{rand})$
33. $\quad\quad$ **if** $c_{min} < c_{new}$ **then**
34. $\quad\quad\quad c_{min} \leftarrow c_{new}$
35. $\quad\quad\quad q_{min} \leftarrow q_{new}$
36. $\quad\quad$ **end**
37. $\quad$ **end**
38. **end**
39. **return** q_{min}

Gambar 7. Operasi ChooseParent pada algoritma RRT*

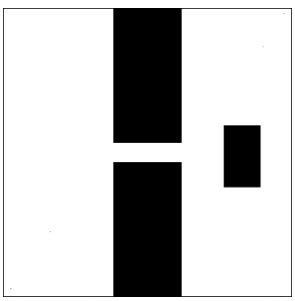
Algorithm 6: $T \leftarrow Rewire(T, Q_{near}, q_{min}, q_{rand})$

13. **for** $q_{near} \in Q_{near}$ **do**
14. $q_{path} \leftarrow Steer(q_{near}, q_{rand}, \Delta q)$
15. **if** $ObstacleFree(q_{path})$ **and** $Cost(q_{rand}) + c(q_{path}) < Cost(q_{near})$ **then**
16. $T \leftarrow ReConnect(q_{rand}, q_{near}, T)$
17. **end**
18. **return** T

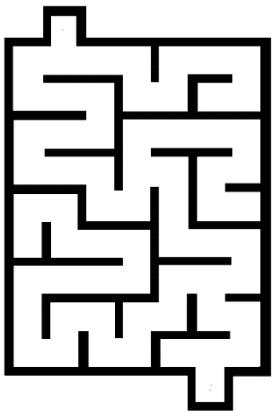
Gambar 8. Operasi Rewire pada algoritma RRT*

Perbedaan algoritma RRT* dengan RRT adalah pada saat memilih node yang akan dijadikan induk dari node baru. Algoritma RRT* mempertimbangkan semua simpul terdekat dari node q_{new}(baris 8 algorithm 4) dan mengecek biaya jalur dari masing-masing node terdekat tersebut. Node terdekat yang akan menghasilkan jalur terpendek diantara q_{new} dan q_{Start} akan dijadikan induk bagi node q_{new} (algorithm 5). Proses Rewire, yang dijelaskan pada algorithm 6, akan memeriksa setiap node Q_{near} untuk mengetahui apakah menjangkau q_{near} melalui q_{new} akan menghasilkan jarak yang lebih dekat ke q_{Start}. Ketika koneksi q_{near} ke q_{new} menuju q_{Start} diperoleh jalur yang lebih pendek dari sebelumnya, maka program akan melakukan proses rewire dengan mengubah induk q_{near} menjadi q_{new}. Lalu algoritma akan berlanjut ke iterasi selanjutnya.

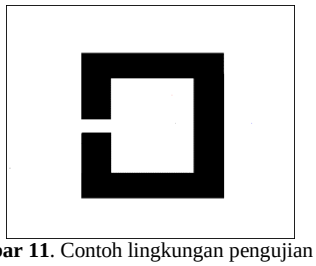
Pengujian dilakukan menggunakan simulasi berbasis python. Kami menggunakan toolbox PythonRobotics yang dikembangkan oleh Sakai [19]. Kami mengambil beberapa lingkungan khusus yang sering digunakan pada pengujian algoritma perencanaan jalur, yaitu *narrow* (celah sempit, seperti **Gambar 9**), *maze* (labirin, seperti **Gambar 10**), *trap* (ruangan dengan celah sempit untuk masuk atau keluar, seperti **Gambar 11**) serta *clutter* (lingkungan dengan banyak rintangan yang berantakan, seperti **Gambar 12**). Kriteria optimalitas yang diukur dan dibandingkan adalah adalah biaya jalur, waktu komputasi dan jumlah total node pada pohon yang dibutuhkan.



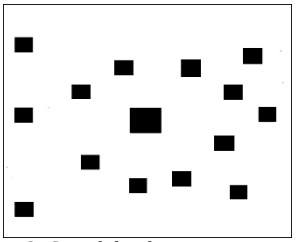
Gambar 9. Contoh lingkungan pengujian *narrow*



Gambar 10. Contoh lingkungan pengujian *maze*



Gambar 11. Contoh lingkungan pengujian *trap*



Gambar 12. Contoh lingkungan pengujian *clutter*

III. HASIL DAN DISKUSI

Mengikuti contoh yang dilakukan oleh Karaman [5], untuk membandingkan kinerja algoritma RRT dan RRT*, maka pada penelitian ini keempat algoritma akan diuji pengujian dengan cara dijalankan sebanyak 500 kali. Lalu dirata-ratakan jarak jalur yang dihasilkan, waktu komputasi dan jumlah node dari setiap iterasi. Hasilnya ditunjukkan pada **Gambar 10 - 13**.

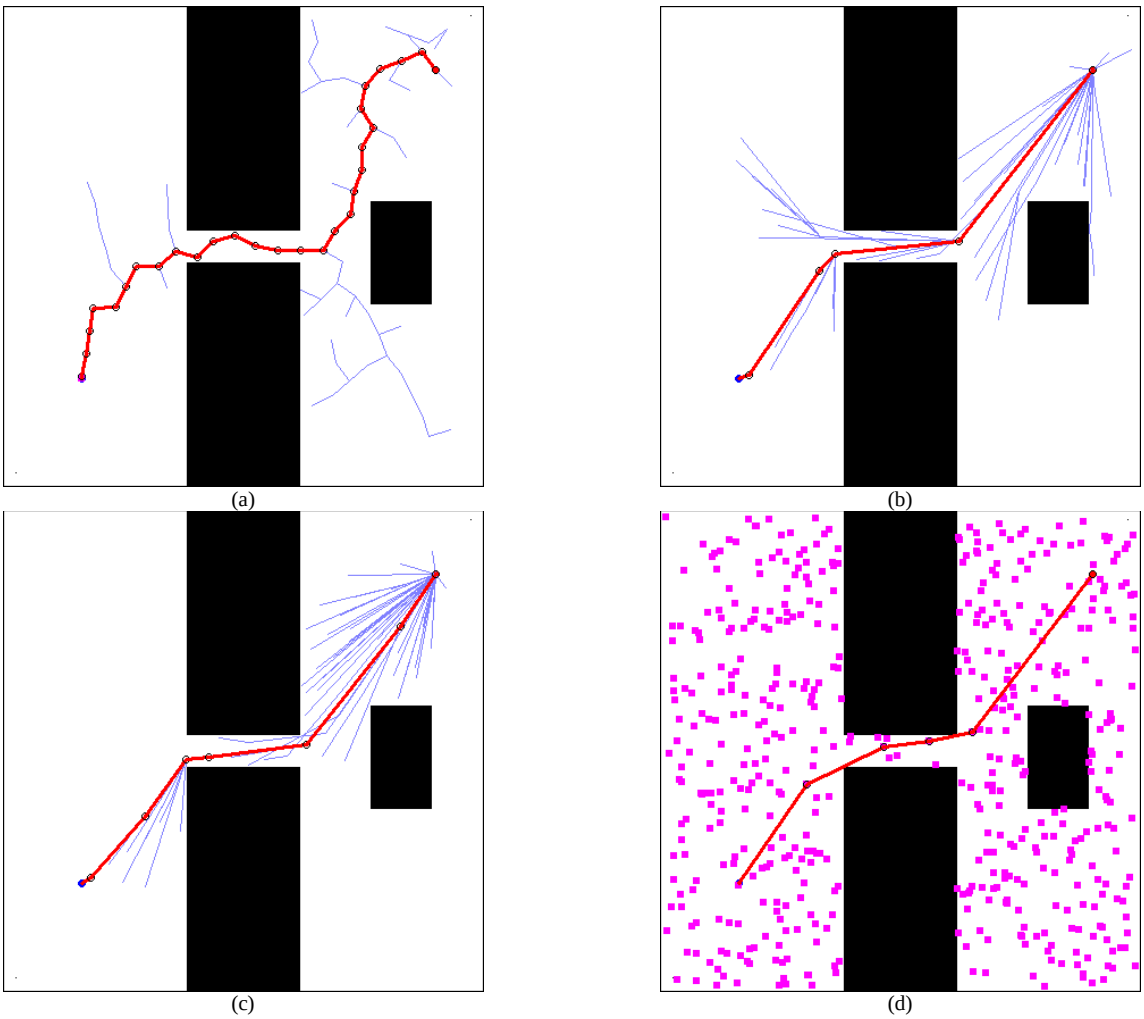
Terlihat bahwa algoritma dasar RRT cenderung menghasilkan jalur yang berliku-liku (**Gambar 10(a), 11(a), 12(a)** dan **13(a)**). Hal ini salah satu yang mengakibatkan algoritma RRT bersifat sub-optimal. Adapun algoritma RRT* maupun informed-RRT* mampu membangun jalur yang tidak berliku-liku (**Gambar 11(a)** dan **(b)**, **12(a)** dan **(b)**, **13(a)** dan **(b)**, serta **14(a)** dan **(b)**). Hal ini karena RRT* menggunakan fitur ChooseParent dan Rewire yang cenderung mencari rute dengan jalur yang terpendek, dan jalur terpendek adalah jalur yang tidak berbelok-belok.

Tetapi keunggulan algoritma RRT adalah mampu membangun jalur dengan sangat cepat.

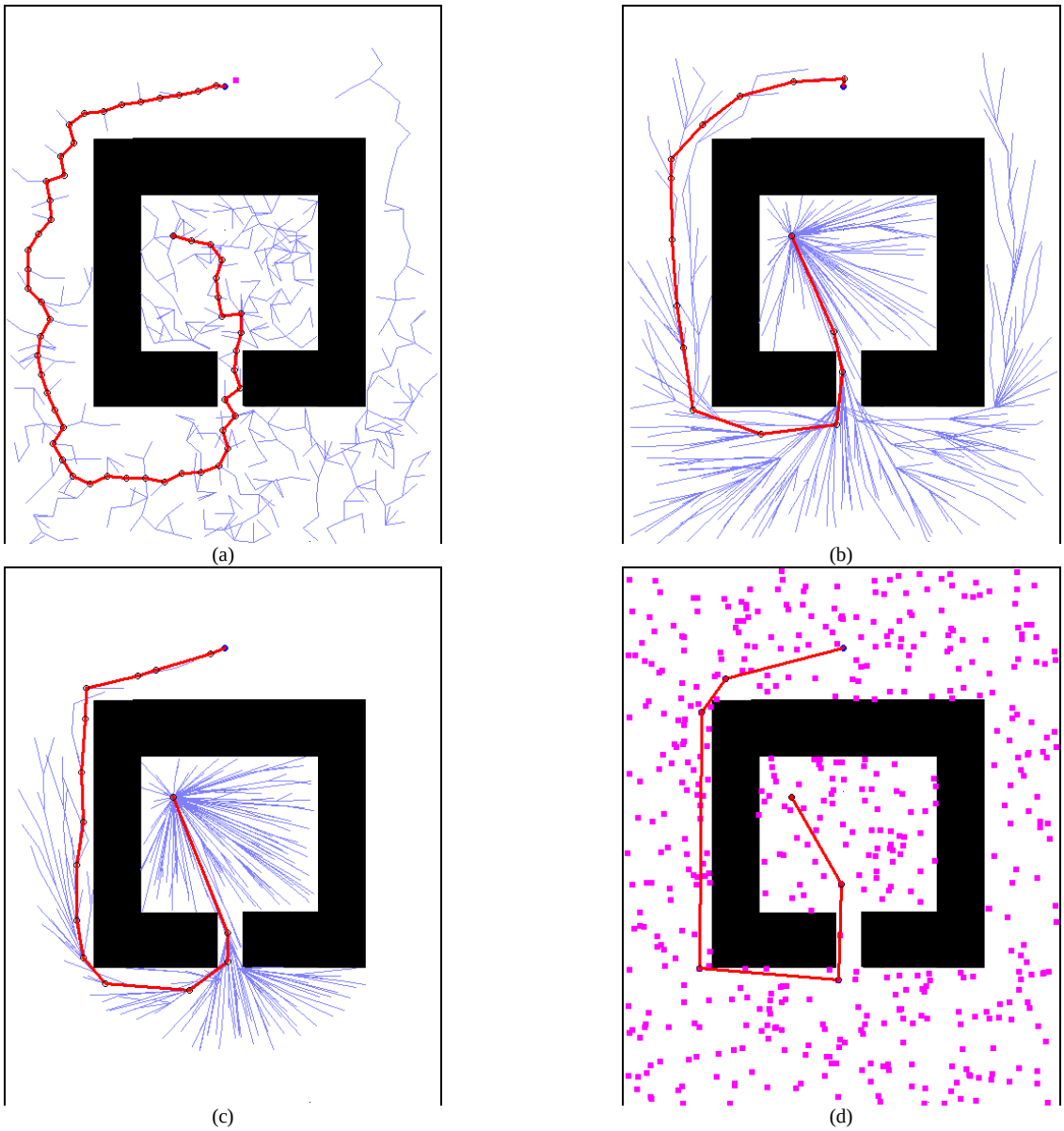
Untuk kasus lingkungan *narrow*, algoritma RRT dapat membangun jalur sekitar tiga kali lebih cepat daripada RRT* maupun informed-RRT*. Walaupun algoritma PRM dapat menghasilkan jalur yang terpendek, tetapi waktu komputasinya sangat tinggi. Untuk kasus lingkungan *narrow*, waktu komputasi algoritma PRM hingga 6000 kali lebih lambat daripada algoritma RRT. Tentu saja waktu komputasi dari algoritma PRM ini akan sangat dipengaruhi oleh jumlah sampel yang digunakan. Pada penelitian ini, digunakan 500 sampel node pada algoritma PRM. Penelitian Zaheer pada [14] juga menunjukkan tingginya waktu komputasi PRM dibandingkan RRT ini.

Prinsip dari algoritma *Informed*-RRT* adalah membatasi ruang sampling, yaitu hanya disekitar daerah elips yang melingkupi node awal dan node akhir. Maka dapat terlihat pada **Gambar 10(c)**, **11(c), 12(c)** dan **13(c)**, bahwa arah penyebaran pohon pencarian tidak terlalu melebar kemana-mana, tetapi seolah-olah dibatasi oleh lingkaran elips tak terlihat yang titik pusatnya di node awal dan node akhir. Dibandingkan dengan algoritma RRT* seperti pada **Gambar 10(b), 11(b), 12(b)** dan **13(b)**, pohon pencarian menyebar kemana-mana.

Strategi membatasi ruang pencarian cukup membantu algoritma *informed*-RRT* untuk memangkas waktu komputasinya. Dapat terlihat pada **Gambar 10**, **12** dan **13**, bahwa waktu komputasi algoritma *informed*-RRT* lebih kecil daripada waktu komputasi algoritma RRT* untuk lingkungan *narrow*, *clutter* dan *maze*. Adapun untuk kasus lingkungan *trap*, rata-rata waktu komputasi algoritma *informed*-RRT* memang masih lebih lama daripada waktu komputasi algoritma RRT*. Hal ini dapat dipahami karena tingkat kesulitan lingkungan tersebut, dimana membatasi ruang pencarian sampel dalam bentuk elips tidak terlalu memungkinakan dalam kasus tersebut.



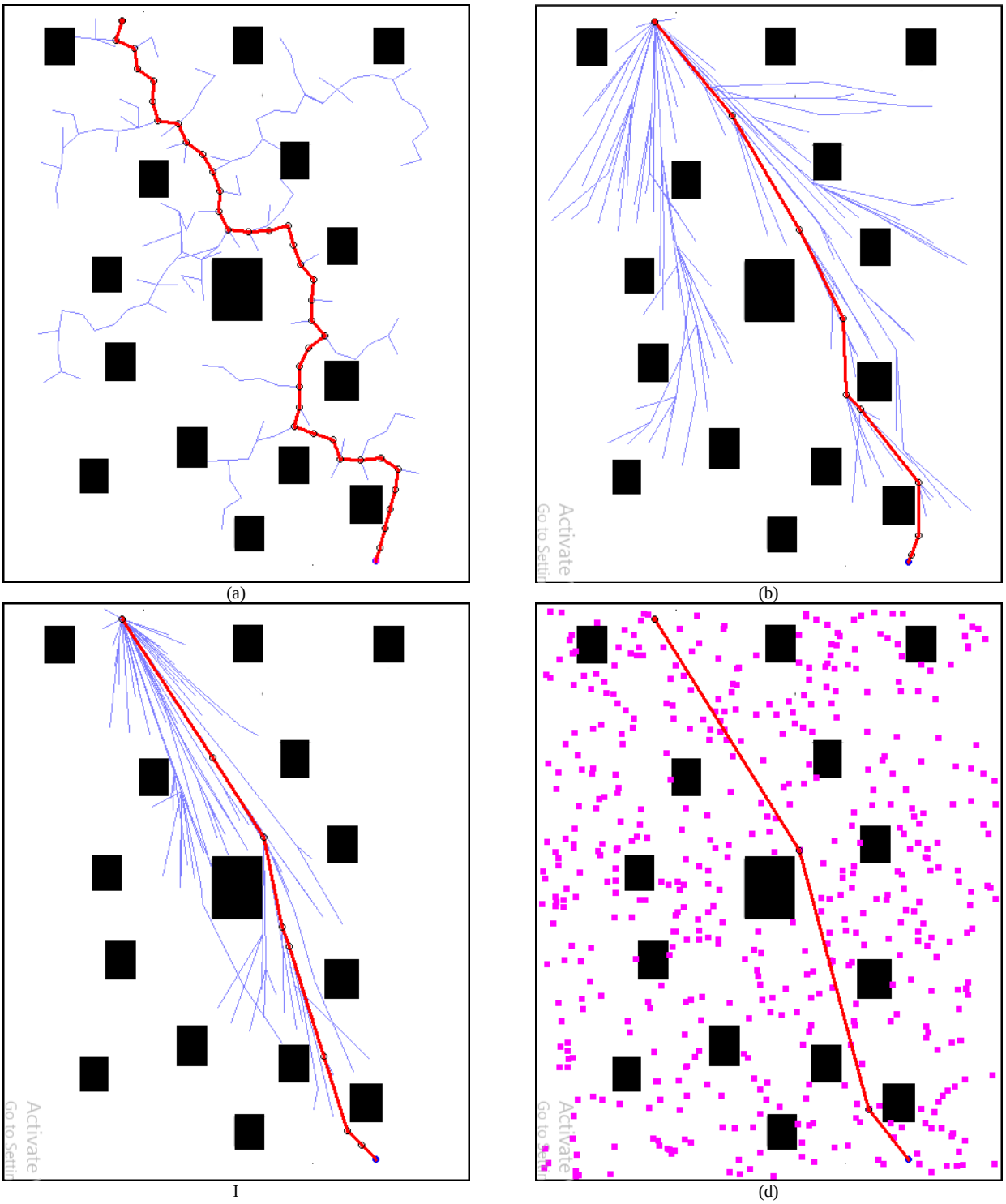
Gambar 10. Perbandingan performansi pada lingkungan dengan celah sempit (*narrow*) :
(a) Algoritma RRT : rata-rata jarak jalur = 542,485; rata-rata jumlah node = 185; rata-rata waktu komputasi = 5,36 ms
(b) Algoritma RRT* : rata-rata jarak jalur = 441,238; rata-rata jumlah node = 158; rata-rata waktu komputasi = 17,44 ms
(c) Algoritma Informed-RRT* : rata-rata jarak jalur = 538,573; rata-rata jumlah node = 147; rata-rata waktu komputasi = 13,98 ms
(d) Algoritma PRM : rata-rata jarak jalur = 535,309; rata-rata jumlah node = 500; rata-rata waktu komputasi = 33.324,3 ms



Gambar 11. Perbandingan performansi pada ruangan dengan celah sempit untuk masuk atau keluar (*trap*) :
(a) Algoritma RRT : rata-rata jarak jalur = 1069,79; rata-rata jumlah node = 1013; rata-rata waktu komputasi = 32,2 ms
(b) Algoritma RRT* : rata-rata jarak jalur = 840,667; rata-rata jumlah node = 1074; rata-rata waktu komputasi = 121,9 ms
(c) Algoritma Informed-RRT* : rata-rata jarak jalur = 790,84; rata-rata jumlah node = 1190; rata-rata waktu komputasi = 145,55 ms
(d) Algoritma PRM : rata-rata jarak jalur = 788,524; rata-rata jumlah node = 500; rata-rata waktu komputasi = 18.398,5 ms

Hal lain yang menarik untuk diperhatikan juga adalah implementasi keempat algoritma ini pada kasus lingkungan *trap*. Pada kasus ini, algoritma RRT, RRT*, dan *informed*-RRT* menunjukkan kenaikan waktu komputasi dibandingkan pengujian dalam lingkungan yang lain. Tetapi algoritma PRM justru menunjukkan penurunan waktu komputasi dibandingkan pengujian dalam lingkungan yang lain. Hal ini menunjukkan bahwa pada algoritma pengambilan sampel (seperti RRT, RRT*, dan *informend* RRT*) akan mengalami peningkatan waktu komputasi jika diterapkan pada kasus ruangan dengan celah sempit untuk masuk dan keluar. Tetapi mungkin algoritma PRM cukup tepat untuk diterapkan pada lingkungan *trap* ini. Walau waktu komputasi algoritma PRM ini masih tinggi, tetapi dengan menurunkan jumlah sampel yang digunakan pada algoritma PRM, maka waktu komputasi dari algoritma PRM ini dapat dikurangi.

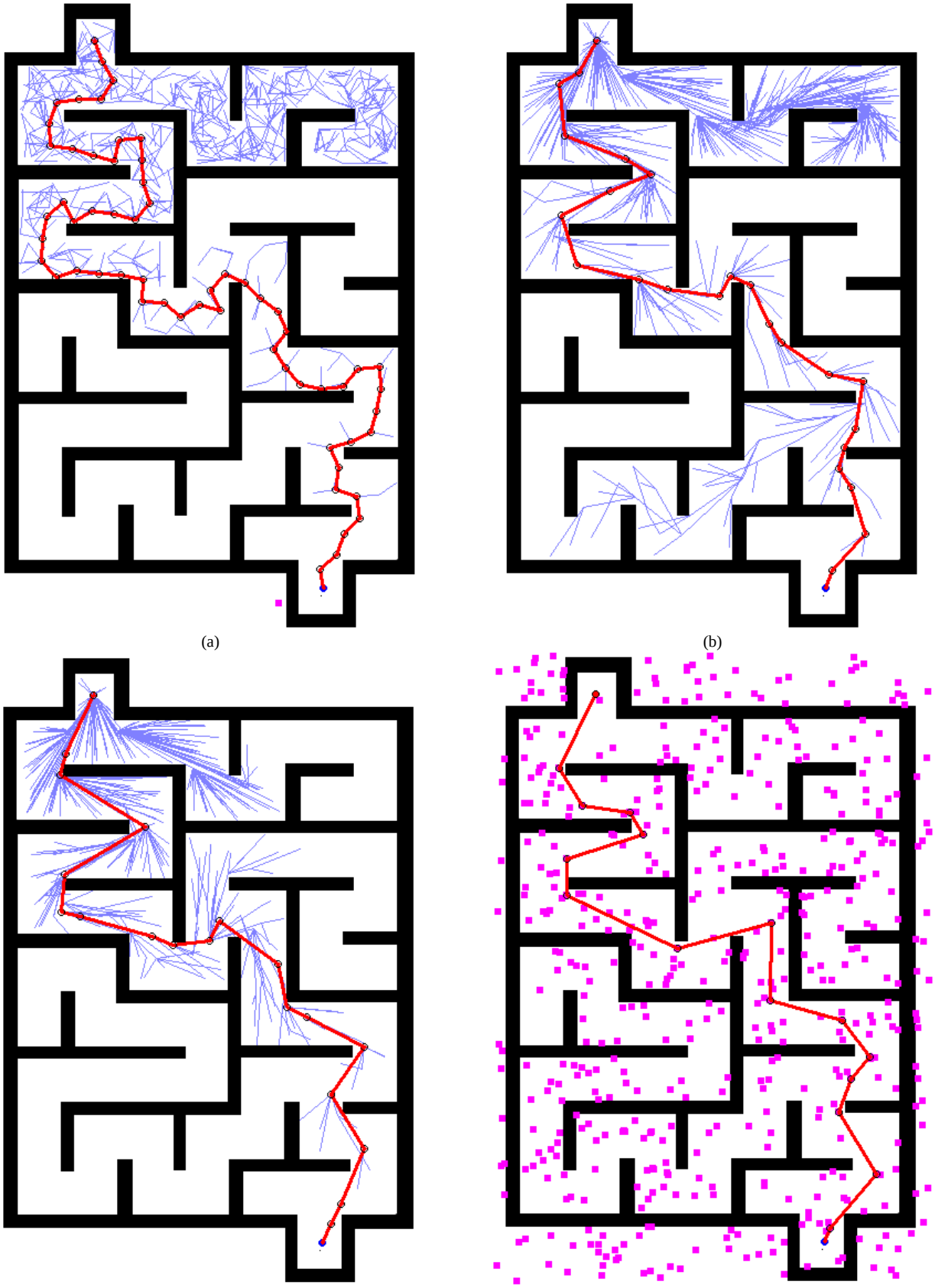
Untuk memberikan pemahaman lebih lanjut kepada pembaca mengenai karakteristik performansi dari masing-masing algoritma, maka pada **Tabel I** – **Tabel XII** disajikan rincian performansi keluaran rata-rata, minimal dan maksimal dari setiap algoritma.



(a) (b)

I (d)

Gambar 12. Perbandingan performansi pada lingkungan dengan banyak halangan berantakan (*clutter*) :
(a) Algoritma RRT : rata-rata jarak jalur = 755,501; rata-rata jumlah node = 245; rata-rata waktu komputasi = 14,7 ms
(b) Algoritma RRT* : rata-rata jarak jalur = 612,358; rata-rata jumlah node = 198; rata-rata waktu komputasi = 33,4 ms
I Algoritma Informed-RRT* : rata-rata jarak jalur = 594,469; rata-rata jumlah node = 143; rata-rata waktu komputasi = 24,95 ms
(d) Algoritma PRM : rata-rata jarak jalur = 593,192; rata-rata jumlah node = 500; rata-rata waktu komputasi = 27.240,5 ms



(a) (b)

I (d)

Gambar 13. Perbandingan performansi pada lingkungan labirin :
(a) Algoritma RRT : rata-rata jarak jalur = 1180,79; rata-rata jumlah node = 3584; rata-rata waktu komputasi = 51,05 ms
(b) Algoritma RRT* : rata-rata jarak jalur = 843,805; rata-rata jumlah node = 3483; rata-rata waktu komputasi = 248,6 ms
I Algoritma Informed-RRT* : rata-rata jarak jalur = 831,799; rata-rata jumlah node = 2189; rata-rata waktu komputasi = 189,6 ms
(d) Algoritma PRM : rata-rata jarak jalur = 859,724; rata-rata jumlah node = 500; rata-rata waktu komputasi = 1,579.8 ms

Tabel I. Perbandingan jarak jalur pada lingkungan *narrow*

	Algoritma			
	RRT	**RRT***	**Informed-RRT***	**PRM**
Minimum	480	430.55	429.89	429.64
Maksimum	642.24	469.07	454.54	444.91
Rata-rata	542.49	441.24	438.57	435.31

Tabel II. Perbandingan jumlah node pada lingkungan *narrow*

	Algoritma		
	RRT	**RRT***	**Informed-RRT***
Minimum	68	56	55
Maksimum	661	868	576
Rata-rata	185	158	147

Tabel III. Perbandingan waktu komputasi pada lingkungan *narrow* (dalam milidetik)

	Algoritma			
	RRT	**RRT***	**Informed-RRT***	**PRM**
Minimum	3	7	6	30194
Maksimum	16	84	55	37633
Rata-rata	5.36	17.44	13.98	33724.28

Tabel IV. Perbandingan jarak jalur pada lingkungan *trap*

	Algoritma			
	RRT	**RRT***	**Informed-RRT***	**PRM**
Minimum	957.03	794.49	766.61	765.49
Maksimum	1235.81	911.39	817.14	805.26
Rata-rata	1069.79	840.67	790.84	788.52

Tabel V. Perbandingan jumlah node pada lingkungan *trap*

	Algoritma		
	RRT	**RRT***	**Informed-RRT***
Minimum	640	626	811
Maksimum	1767	1788	2145
Rata-rata	1013	1074	1190

Tabel VI. Perbandingan waktu komputasi pada lingkungan *trap* (dalam milidetik)

	Algoritma			
	RRT	**RRT***	**Informed-RRT***	**PRM**
Minimum	22	84	99	16851
Maksimum	43	191	261	19916
Rata-rata	32.20	121.90	145.55	18398.55

Tabel VII. Perbandingan jarak jalur pada lingkungan *clutter*

	Algoritma			
	RRT	**RRT***	**Informed-RRT***	**PRM**
Minimum	683.16	591.93	589.32	590.85
Maksimum	836.64	636.27	598.56	597.40
Rata-rata	755.50	612.36	594.47	593.19

Tabel VII. Perbandingan jumlah node pada lingkungan *clutter*

	Algoritma		
	RRT	**RRT***	**Informed-RRT***
Minimum	123	123	71
Maksimum	490	359	282
Rata-rata	244.55	198.35	143.15

Tabel IX. Perbandingan waktu komputasi pada lingkungan *clutter* (dalam milidetik)

	Algoritma			
	RRT	**RRT***	**Informed-RRT***	**PRM**
Minimum	8	20	13	24645
Maksimum	32	55	50	29516
Rata-rata	14.7	33.4	24.95	27240.55

Tabel X. Perbandingan jarak jalur pada lingkungan *maze*

	Algoritma			
	RRT	**RRT***	**Informed-RRT***	**PRM**
Minimum	1074.87	802.15	806.19	821.95
Maksimum	1297.46	890.28	894.70	898.69
Rata-rata	1180.79	843.80	831.80	859.72

Tabel XI. Perbandingan jumlah node pada lingkungan *maze*

	Algoritma		
	RRT	**RRT***	**Informed-RRT***
Minimum	2640	1823	1484
Maksimum	5326	6303	3514
Rata-rata	3583.95	3482.7	2189.4

Tabel XII. Perbandingan waktu komputasi pada lingkungan *maze* (dalam milidetik)

	Algoritma			
	RRT	**RRT***	**Informed-RRT***	**PRM**
Minimum	34	121	122	1516
Maksimum	81	484	310	1684
Rata-rata	51.05	248.6	189.6	1579.8

Pada **Tabel I** dapat terlihat bahwa jalur terbaik yang dihasilkan oleh algoritma RRT (nilai minimum), masih belum sebaik jalur terjauh yang dihasilkan oleh algoritma RRT* (nilai maksimum). Mengenai rentang kualitas/panjang jalur yang dihasilkan oleh algoritma RRT*, *Informed-RRT** maupun PRM memang masih saling mendekati. Tetapi kekurang besar dari algoritma PRM memang adalah waktu komputasinya seperti terlihat pada **Tabel III, VI, IX**, dan **XII**. Dapat terlihat bahwa pada perbandingan jumlah node di **table II, V, VII**, maupun **XI** tidak menyertakan algoritma PRM. Hal ini dikarenakan jumlah node pada algoritma PRM merupakan setingan parameter awal.

IV. KESIMPULAN

Dari hasil pengujian pada lingkungan *trap*, *narrow*, *maze* maupun *clutter*, dapat terlihat beberapa karakteristik dari algoritma RRT, RRT*, *Informed*-RRT* dan PRM. Algoritma RRT memiliki waktu komputasi yang tercepat, tetapi memiliki kualitas jalur yang kurang baik. Bahkan berdasarkan pengujian di lingkungan *trap*, kualitas jalur terbaik algoritma RRT masih kalah dibandingkan kualitas jalur terburuk algoritma RRT*. Algoritma RRT*, *informed*-RRT* dan PRM memiliki kualitas jalur yang mirip. Pada kasus lingkungan *trap*, menunjukkan peningkatan waktu komputasi yang tinggi pada algoritma RRT, RRT* dan *informed*-RRT*, tetapi tidak pada algoritma PRM. Tetapi algoritma PRM tetap memiliki waktu komputasi tertinggi dibandingkan algoritma RRT, RRT*, maupun *informed*-RRT*. Algoritma RRT dapat memiliki waktu komputasi yang cepat karena algoritma-nya yang sederhana. Adapun algoritma PRM dapat memiliki waktu komputasi yang tinggi karena melakukan pengecekan kepada seluruh node yang ada untuk memperoleh jalur yang terpendek.

DAFTAR PUSTAKA

[1] X. Lan, dan S. D. Cairano, "Continuous curvature path planning for semi-autonomous vehicle maneuvers using RRT," *2015 IEEE European Control Conference (ECC)*, pp. 2360-2365, 2015

[2] D. Alejo, J. A. Cobano, G. Heredia, J. R. Martínez-de Dios, dan A. Ollero, "Efficient trajectory planning for WSN data collection with multiple UAVs," *Cooperative Robots and Sensor Networks 2015 Springer, Cham*, pp. 53 – 75, 2015

[3] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, dan S. Teller, "Anytime motion planning using the RRT," *2011 IEEE International Conference on Robotics and Automation*, pp. 1478-1483, 2011

[4] S. M. LaValle, "Planning algorithms," *Cambridge university press*, 2006

[5] S. Karaman, dan E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no 7, pp. 846-894, 2011

[6] E. W. Dijkstra, E. W. "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269-271, 1959

[7] P. E. Hart, N. J. Nilsson, dan B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100-107, 1968

[8] A. Stentz, "Optimal and efficient path planning for unknown and dynamic environments, " *Carnegie-Mellom Univ Pittsburh PA Robotics Inst*, 1993

[9] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, dan S. Thrun, "Anytime search in dynamic graphs," *Artificial Intelligence*, vol. 172, no. 14, pp. 1613-1643, 2008

[10] M. Elbanhawi, dan M. Simic, "Sampling-Based Robot Motion Planning: A Review survey", *IEEE Access*, vol. 2, pp. 56-77, 2014.

[11] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, M. H. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566-580, 1996

[12] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Tech. Rep., Computer Science Dept., Iowa State University*, pp. 1-4, 1998

[13] J. D. Gammell, S. S. Srinivasa, dan T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2997-3004, 2014

[14] S. Zaheer, M. Jayaraju, dan T. Gulrez, "Performance analysis of path planning techniques for autonomous mobile robots," *2015 IEEE international conference on electrical, computer and communication technologies (ICECCT)*, pp. 1-5, 2015

[15] I. Noreen, A. Khan, dan Z. Habib, "A comparison of RRT, RRT* and RRT*-smart path planning algorithms," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 16, no. 10, pp. 20-27, 2016

[16] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, dan M. S. Muhammad, "RRT*-SMART: A rapid convergence implementation of RRT," *International Journal of Advanced Robotic Systems*, vol. 10, no. 7, pp. 299, 2013

[17] I. Noreen, A. Khan, K. Asghar, dan Z. Habib, "A Path-Planning Performance Comparison of RRT*-AB with MEA* in a 2-Dimensional Environment," *Symmetry*, vol. 11, no. 7, pp. 1-16, 2019

[18] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, dan S. Thrun, "Principles of robot motion: theory, algorithms, and implementation," *MIT press*, 2005.

[19] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, dan A. Paques, "PythonRobotics : a Python code collection of robotics algorithms," *arXiv preprint arXiv:1808.10703*, 2018