TELEKONTRAN, VOL. 7, NO. 2, OKTOBER 2019
DOI 10.34010/telekontran.v7i2.2651
p-ISSN : 2303 – 2901
e-ISSN : 2654 – 7384

Simulator Edukatif untuk Pembelajaran Algoritma *Rapidly-exploring Random Tree (RRT)*

Educational Simulator for Teaching of Rapidly-exploring Random Tree (RRT) Algorithms

Eka Setia Nugraha*
Teknik Telekomunikasi, Institut Teknologi Telkom Purwokerto
Jl. D.I Panjaitan No.128 Purwokerto, Indonesia
*Email : eka_nugraha@ittelkom-pwt.ac.id

Abstrak – Paper ini mempresentasikan perancangan dan penggunaan simulator pendidikan untuk membantu pengajaran materi perencanaan gerak menggunakan algoritma *Rapidly-exploring Random Tree* (RRT). Simulator ini dibuat menggunakan Laboratory Virtual Instrumentation Engineering Workbench (LabVIEW). Pada simulator pendidikan ini dibuat juga beberapa variasi dari RRT, yaitu RRT*, bi-RRT dan RRT biased. Keunggulan dari simulator ini adalah setiap subproses dari algoritma RRT dapat divisualisasikan sehingga akan membantu meningkatkan pemahaman dalam pembelajaran algoritma ini. Parameter-parameter dari algoritma RRT juga dapat dengan mudah dimodifikasi sehingga dapat diamati pengaruhnya pada performansi keluaran dari RRT. Pengujian performansi simulator dilakukan dengan membandingkan implementasinya pada beberapa kasus *benchmark* yang ada. Karena salah satu hal penting dalam algoritma perencanaan jalur adalah mengenai waktu komputasi, maka pembuatan software simulator ini telah diefisiensikan agar diperoleh waktu komputasi sistem yang minimal. Salah satu yang dilakukan agar meminimalkan waktu komputasi ini adalah dengan cara meminimalkan *overhead* pada program subVI. *Overhead* pada subVI diminimalkan dengan menjadikannya sebagai *subroutine*. Karena itu, pada perancangan simulator ini juga dihindari penggunaan fungsi-fungsi pada LabVIEW yang tidak bisa dijadikan sebagai *subroutine*. Menggunakan salah satu kasus *benchmark* pada perencanaan jalur, algoritma RRT yang dirancang mampu menyusun jalur dalam waktu 185 milidetik adapun jika menggukan algoritma RRT*, perencanaan jalur dapat selesai dirancang dalam waktu 1273 milidetik. Dengan memanfaatkan simulator ini diharapkan dapat membantu mahasiswa untuk mendapatkan pemahaman yang lebih baik mengenai algorima perencanaan gerak menggunakan RRT.

Kata Kunci : Perencanaan gerak, *Rapidly-exploring Random Tree*, simulator, variasi RRT, visualisasi proses

Abstract *- This paper presents the design and use of educational simulators to assist in teaching motion planning material using the Rapidly-exploring Random Tree (RRT) algorithm. This simulator is made using Laboratory Virtual Instrumentation Engineering Workbench (LabVIEW). In this educational simulator also made several variations of the RRC, namely RRT*, bi-RRT and RRT biased. The advantage of this simulator is that each subprocess of the RRT algorithm can be visualized so that it will help improve understanding in learning this algorithm. The parameters of the RRT algorithm can also be easily modified so that they can be observed for the output performance of the RRT. Simulator performance testing is done by comparing its implementation in several existing benchmark cases. Because one of the important things in the path planning algorithm is about computational time, making this simulator software has been streamlined in order to obtain minimal system computing time. Using one of the benchmark cases in path planning, the RRT algorithm is designed to be able to arrange a path in 185 milliseconds, whereas if using the RRT* algorithm, the path planning can be completed in 1273 milliseconds. One of the ways to minimize this computation time is by minimizing overhead on subVI programs. Subhead overheads are minimized by making them subroutines. Therefore, the design of this simulator also avoids the use of functions in LabVIEW which cannot be used as subroutines. Using this simulator is expected to help students to get a better understanding of the motion planning algorithm using RRT.*

Keywords *: Motion planning, Rapidly-exploring Random Tree, simulators, RRT variations, process visualization*

I. PENDAHULUAN

Algoritma perencanaan gerak atau jalur (*motion planning*) adalah salah satu bidang penelitian mendasar dalam robotika [1]. Telah banyak metode yang sering digunakan untuk melakukan perencanaan jalur ini. Algoritma pencarian seperti AD* [2] dapat menemukan solusi optimal dalam kasus graph dinamis. Namun, algoritma AD* hanya bias diterapkan pada ruang kerja yang telah didiskritisasi. Metode pencarian jalur yang melibatkan diskritisasi ruang kerja ini memiliki kekurangan yaitu dapat mengalami penurunan kinerjanya dalam kasus dimensi yang tinggi. Penelitian Pivtoraiko [3] menggunakan perencanaan gerak konvensional yang digabungkan dengan algoritma pencarian graph. Tetapi metode ini masih tetap membutuhkan diskritisasi ruang kerja. Algoritma *Ant Colony Optimization* [4] juga pernah diterapkan pada pencarian jalur robot. Tetapi metode seperti ACO dapat terjebak pada solusi minimum local dan dapat berkinerja buruk pada kasus lingkungan yang memiliki celah sempit. Metode perencanaan reaktif berbasis sensor juga pernah diteliti [5, 6], tetapi metode reaktif ini tidak dapat digunakan sebagai perencanaan jalur global atau jarak jauh. Metode berbasis kontrol konvensional membutuhkan formulasi model yang tepat baik untuk robot maupun untuk lingkungan [7]. Pembuatan formulasi model yang akurat ini terkadang sulit untuk dilakukan. Adapun algoritma perencanaan jalur berbasis pengambilan sampel atau *Sampling Based Planning* (SBP) memiliki keunggulan yaitu dapat memberikan solusi yang cepat pada masalah yang sulit [8]. Sejauh yang penulis ketahui, algoritma SBP yang paling umum digunakan adalah algoritma *Probabilistic Roadmap Method* (PRM) [9] dan algoritma *Rapidly-exploring Random Tree* (RRT) [10]. Keunggulan dari algoritma PRM adalah dapat memberikan solusi yang optimal asimptotik, tetapi memiliki kekurangan yaitu waktu komputasi yang tinggi. Adapun keunggulan algoritma RRT adalah memiliki waktu komputasi yang kecil, tetapi kekurangannya adalah hanya menyediakan solusi yang bersifat sub-optimal [11]. Algoritma RRT dikembangkan oleh Karaman menjadi RRT* [12]. Keunggulan RRT* adalah dapat menghasilkan solusi jalur yang bersifat optimal asimptotik [13]. Tetapi, waktu komputasi dari algoritma RRT* cukup tinggi. Saat ini telah banyak algoritma variasi dari RRT* ini. Akgun dan Stilman [13] mengusulkan algoritma B-RRT *. Algoritma B-RRT* ini meningkatkan optimasi dari jalur yang dihasilkan dengan cara menggunakan pencarian dua arah yaitu pencarian yang bermula dari titik awal dan titik akhir. Kumar et al. [14] mengajukan algoritma RRT-biased. Gammell et al. [15] mengajukan algoritma Informed RRT*. Algoritma Informed-RRT* mengusulkan metode pengambilan sampel yang dilakukan hanya di daerah elips yang mengelilingi simpul awal dengan simpul akhir. Nasir et al. [16] memperkenalkan algoritma RRT*-Smart. RRT*-Smart menggunakan dua fitur yang dinamakan, *intelligent sampling* dan *route optimization*. Fitur dari RRT*-Smart dapat mempercepat laju konvergensi. Beberapa peneliti telah berhasil menerapkan algoritma RRT dan variasinya pada perencanaan gerak kendaraan otonom. Tim MIT pada *Darpa Urban Challenge* menggunakan algoritma RRT-*Closed Loop* [17]. Karaman [18] menerapkan algoritma RRT pada robot forklift. Adapun mobil nuTonomy paa *Singapore's One-North technology business district* menggunakan algoritma RRT* [19].

Terdapat beberapa publikasi mengenai pembuatan toolbox dari algoritma RRT ini. Sakai [20] telah mempublikasikan toolbox robotic, diantaranya untuk perencanaan jalur menggunakan A*, PRM, RRT, RRT*, dan masih ada lagi. Tetapi Sakai menggunakan program python. Vahrenkamp juga telah mempublikasikan toolbox perencanaan jalur, tetapi dibuat menggunakan program C+ [21]. Corke [22] mempublikasikan toolbox robotic tetapi dibuat menggunakan program MATLAB. Sejauh yang diketahui oleh peneliti, belum ada penelitian yang membuat toolbox perencanaan jalur menggunakan LabVIEW.

Tujuan dari penelitian ini adalah untuk merancang perangkat lunak simulator sebagai alat bantu pengajaran materi perencanaan jalur (*motion planning*) mengenai algoritma RRT. Pada simulator ini juga dibuat beberapa variasi dari algoritma RRT, yaitu RRT*, RRT biased dan algoritma bi-RRT. Keunggulan dari simulator ini adalah dapat divisualisasikannya setiap subproses dari algoritma RRT sehingga dapat membatu meningkatkan pemahaman mahasiswa dalam mempelajari karakteristik algoritma RRT. Pengguna juga dapat dengan mudah memodifikasi parameter-parameter dari RRT sehingga dapat diamati pengaruhnya pada performansi perencanaan jalur ini. Agar waktu komputasi dari program yang dibuat dapat diminimalkan, maka salah satu strategi yang dilakukan adalah dengan meminimalkan *overhead* pada subprogram LabVIEW [23,24]. *Overhead* pada subprogram LabVIEW diminimalkan dengan mengubahnya

menjadi *subroutine*. Karena itu, pada perancangan simulator ini, dihindari penggunaan fungsi-fungsi dan subprogram pada LabVIEW yang tidak bisa dijadikan sebagai *subroutine*.

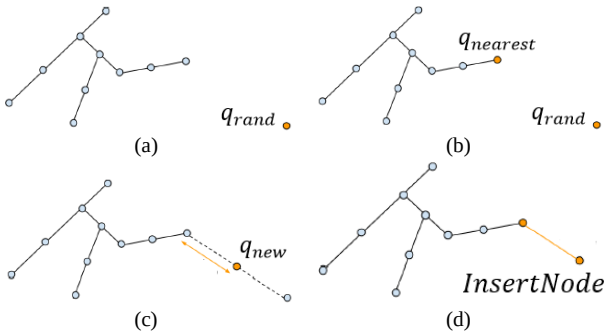
II. METODE

Simulator pengajaran algoritma RRT ini dibuat menggunakan LabVIEW. Algoritma dasar RRT yang digunakan ditunjukkan pada **Gambar 1**. Terlihat bahwa terdapat beberapa proses yang harus dilakukan, yaitu *RandomSample*, *NearestNeighbor*, *Steer* dan *InsertNode*. Proses *RandomSample* bertujuan mengambil sampel pada ruang pencarian, yang disebut sebagai q_{rand} (baris 4). *NearestNeighbor* mencari node pada pohon pencarian yang terdekat ke q_{rand}. Node terdekat ini dinamakan sebagai $q_{nearest}$ (baris 5). Selanjutnya akan dibangun node baru diantara $q_{nearest}$ dan q_{rand}. Node baru ini dinamakan q_{new}. Node q_{new} akan berjarak Δq dari $q_{nearest}$ (baris 6). Jika diantara q_{new} dan $q_{nearest}$ tidak ada hambatan, maka node q_{new} akan ditambahkan ke pohon pencarian (baris 7 dan 8). Lalu iterasi akan berulang sebanyak N kali (baris 3 dan 9). Ilustrasi dari algoritma RRT ini ditunjukkan pada **Gambar 2**.

Algorithm 1 : $T = (V, E) \leftarrow RRT(q_{init})$

1. $T \leftarrow InitializeTree()$
2. $T \leftarrow InsertNode(\emptyset, q_{init}, T)$
3. **for** $k \leftarrow 1$ **to** N **do**
4. $q_{rand} \leftarrow RandomSample(k)$
5. $q_{nearest} \leftarrow NearestNeighbor(q_{rand}, Q_{near}, T)$
6. $q_{new} \leftarrow Steer\ (q_{nearest}, q_{rand}, \Delta q)$
7. **if** $Obstaclefree(q_{new}, q_{nearest})$ **then**
8. $T \leftarrow InsertNode(q_{parent}, q_{new}, T)$
9. **end**

Gambar 1. Algoritma dasar dari RRT



Gambar 2. Ilustrasi proses algoritma RRT : (a) baris 4, (b) baris 5, (c) baris 6, (d) baris 8 pada **Algorithm 1** (diadaptasi dari [1])

Adapun algoritma RRT* ditunjukkan pada **Gambar 3**. Algoritma RRT* ini hampir mirip dengan algoritma RRT (baris 1 – 6 algoritma pada **Gambar 1** sama dengan baris 1 – 6 pada **Gambar 3**). Perbedaannya adalah adanya proses ChooseParent (baris 9) dan Rewire (baris 11).

Detil algoritma ChooseParent ditunjukkan pada **Gambar 4**. Pada proses ChooseParent, akan dicari node-node yang berjarak terdekat dari q_{new}. Node-node yang terdekat dari q_{new} ini dinamakan Q_{near} (baris 8 pada **Algorithm 2**). Diantara Q_{near} ini akan dicari node yang bila dihubungkan ke q_{new} akan diperoleh jalur antara q_{new} dan node awal yang paling minimal. Node dari Q_{near} yang akan membuat jarak antara q_{new} dan node awal minimal, akan dijadikan node parent dari q_{new} (**Algorithm 3**). Ilustrasi dari proses ChooseParent ini ditunjukkan pada **Gambar 5**.

Algorithm 2 : $T = (V, E) \leftarrow RRT^*(q_{init})$

1. $T \leftarrow InitializeTree()$
2. $T \leftarrow InsertNode(\emptyset, q_{init}, T)$
3. **for** $k \leftarrow 1$ **to** N **do**
4. $q_{rand} \leftarrow RandomSample(k)$
5. $q_{nearest} \leftarrow NearestNeighbor(q_{rand}, Q_{near}, T)$
6. $q_{new} \leftarrow Steer\ (q_{nearest}, q_{rand}, \Delta q)$
7. **if** $Obstaclefree(q_{new}, q_{nearest})$ **then**
8. $Q_{near} \leftarrow Near(T, q_{new})$
9. $q_{parent} \leftarrow ChooseParent(q_{new}, Q_{near}, q_{nearest})$
10. $T \leftarrow InsertNode(q_{parent}, q_{new}, T)$
11. $T \leftarrow Rewire(T, Q_{near}, q_{parent}, q_{new})$
12. **end**

Gambar 3. Algoritma RRT* (diadaptasi dari [12])

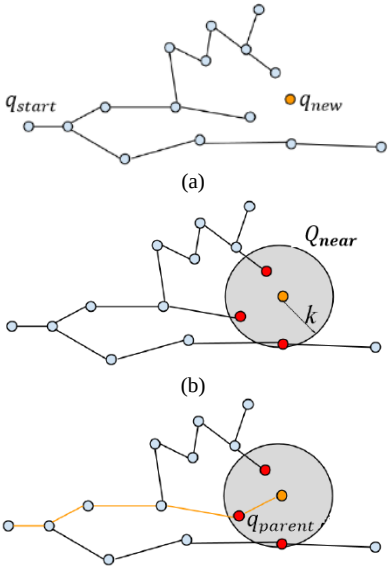
Algorithm 3 :
$q_{min} \leftarrow ChooseParent(q_{rand}, Q_{near}, q_{nearest}, \Delta q)$

1. $q_{min} \leftarrow q_{nearest}$
2. $c_{min} \leftarrow Cost(q_{nearest}) + c(q_{rand})$
3. **for** $q_{near} \in Q_{near}$ **do**
4. $q_{path} \leftarrow Steer(q_{near}, q_{rand}, \Delta q)$
5. **if** $ObstacleFree(q_{path})$ **then**
6. $c_{new} \leftarrow Cost(q_{near}) + c(q_{rand})$
7. **if** $c_{min} < c_{new}$ **then**
8. $c_{min} \leftarrow c_{new}$
9. $q_{min} \leftarrow q_{new}$
10. **end**
11. **end**
12. **end**
13. **return** q_{min}

Gambar 4. Operasi *chooseparent* pada algoritma RRT* (diadaptasi dari [12])

Adapun detil dari algoritma Rewire ditunjukkan pada **Gambar 6**. Pada proses Rewire, akan dicari node-node yang terdejat dengan q_{new} dimana jika node tersebut dihubungkan ke node awal melalui q_{new} akan diperoleh jalur yang lebih dekat. Maka parent baru dari node-node tersebut akan diubah menjadi q_{new}. Ilustrasi dari

proses Rewire ini ditunjukkan pada **Gambar 7**. Sebelum proses rewire, node q_a terhubung dengan node q_b (jalur menukju node awal cukup jauh karena melalui jalur beliku-liku, lalu setelah proses rewire, node q_a terhubung dengan q_{new} menuju node awal, dimana jalur baru ini lebih minimal jaraknya daripada jarak sebelum proses rewire.

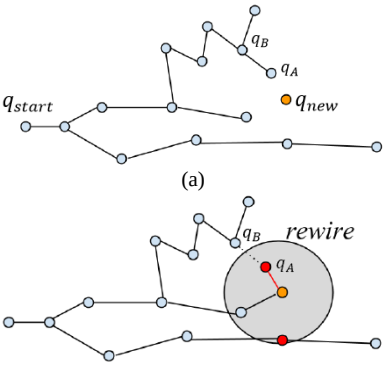


Gambar 5. Ilustrasi proses dari *chooseparent* RRT* : (a) dan (b) baris 8, (c) baris 9 pada **Algorithm 2** (diadaptasi dari [1])

Algorithm 4 : $T \leftarrow Rewire(T, Q_{near}, q_{min}, q_{rand})$

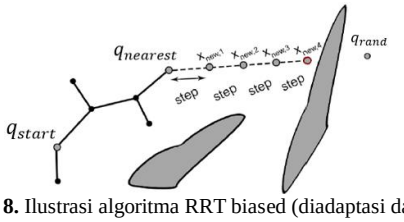
1. **for** $q_{near} \in Q_{near}$ **do**
2. $q_{path} \leftarrow Steer(q_{near}, q_{rand}, \Delta q)$
3. **if** $ObstacleFree(q_{path})$ **and** $Cost(q_{rand}) + c(q_{path}) < Cost(q_{near})$ **then**
4. $T \leftarrow ReConnect(q_{rand}, q_{near}, T)$
5. **end**
6. **return** T

Gambar 6. Operasi *rewire* pada algoritma RRT* (diadaptasi dari [12])



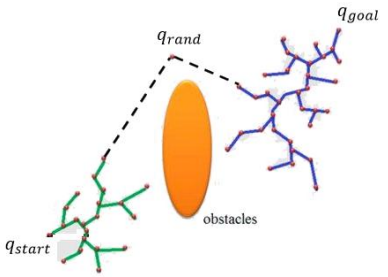
Gambar 7. Ilustrasi dari proses rewire RRT* : (a) pada keadaan awal node q_a terhubung dengan node q_b, (b) setelah proses rewire, node q_a terhubung dengan q_{new}

Terdapat beberapa variasi dari algoritma RRT ini. Diantaranya adalah algoritma RRT biased [14] dan Bi-RRT [13]. Pada algoritma RRT biased, laju konvergensi ditingkatkan dengan cara membiaskan pertumbuhan pohon pencarian. Metode yang digunakan adalah dengan cara saat $q_{nearest}$ hendak membangun node baru yang mengarah ke q_{rand}, maka node baru tidak dibuat hanya satu kali langkah saja, tetapi dibuat beberapa langkah hingga q_{rand} tercapai atau jalur yang dibuat terhalangi oleh hambatan. Ilustrasi dari algoritma RRT biased ini ditunjukkan pada **Gambar 8**.



Gambar 8. Ilustrasi algoritma RRT biased (diadaptasi dari [25])

Adapun algoritma Bi-RRT meningkatkan efisiensi dan optimalisasi jalur yang dihasilkan oleh algoritma dengan cara menggunakan versi dua arah pencarian dari algoritma RRT. Pada algoritma Bi-RRT, salah satu pohon pencarian dimulai dari node awal dan secara bersamaan dibuat pohon pencarian lainnya yang dimulai dari node tujuan. Ilustrasi dari proses algoritma Bi-RRT ini ditunjukkan pada **Gambar 9**.



Gambar 9. Ilustrasi algoritma Bi-RRT. Terdapat dua pohon pencarian yang berawal dari node start dan node goal

III. HASIL DAN DISKUSI

Simulator pembelajaran algoritma RRT ini dapat memvisualisasikan proses pencarian jalur yang dilakukan oleh algoritma RRT. Simulator pembelajaran algoritma RRT ini terdiri dari beberapa sub-program, yaitu :

1. Subprogram “Inisialisasi Data.vi”
2. Subprogram “Random Sampling.vi”
3. Subprogram “Get Nearest List Index.vi”
4. Subprogram “Steer.vi”
5. Subprogram “Collision Check.vi”
6. Subprogram “Insert Node.vi”
7. Subprogram “Draw.vi”

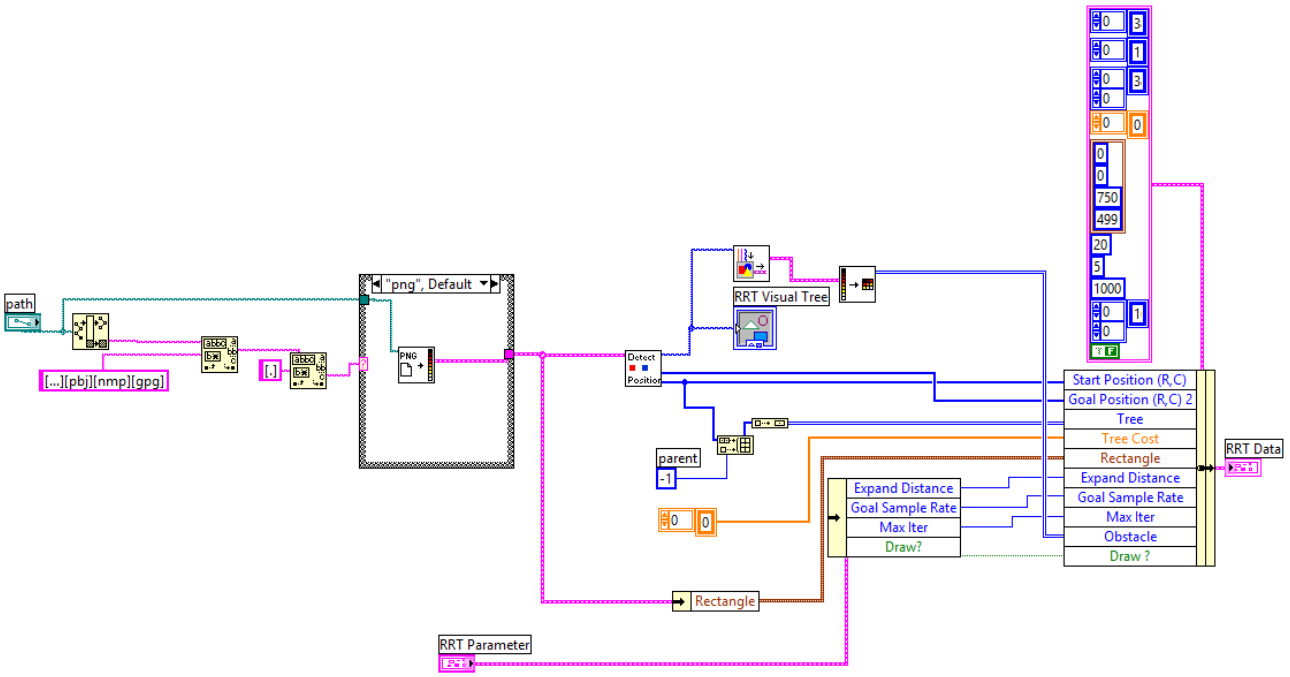
8. Subprogram “Check Goal.vi”
9. Subprogram “Make and Calculate Path.vi”
10. Subprogram “Draw Complete Path.vi”

Subprogram “Inisialisasi Data.vi” berfungsi untuk menerima gambar peta lingkungan yang akan di terapkan algoritma RRT. Pengguna cukup membuat peta lingkungan pada software seperti MS paint. Daerah yang boleh dilewati oleh algoritma RRT diwarnai putih, sedangkan hambatan yang tidak boleh dilewati algoritma RRT diwarnai hitam. Titik Start dibuat dengan membuat titik warna merah (dengan nilai RGB : R = 255, G = 0, B = 0). Titik tujuan dibuat dengan membuat titik warna biru (dengan nilai RGB : R = 0, G = 0, B = 255). Program “Inisialisasi Data.vi” akan mencari kedua titik merah dan biru tersebut pada peta lingkungan yang diberikan, kemudian mengeluarkan koordinatnya untuk digunakan pada subprogram selanjutnya. Block diagram dari “Inisialisasi Data.vi” ditunjukkan pada **Gambar 10**.

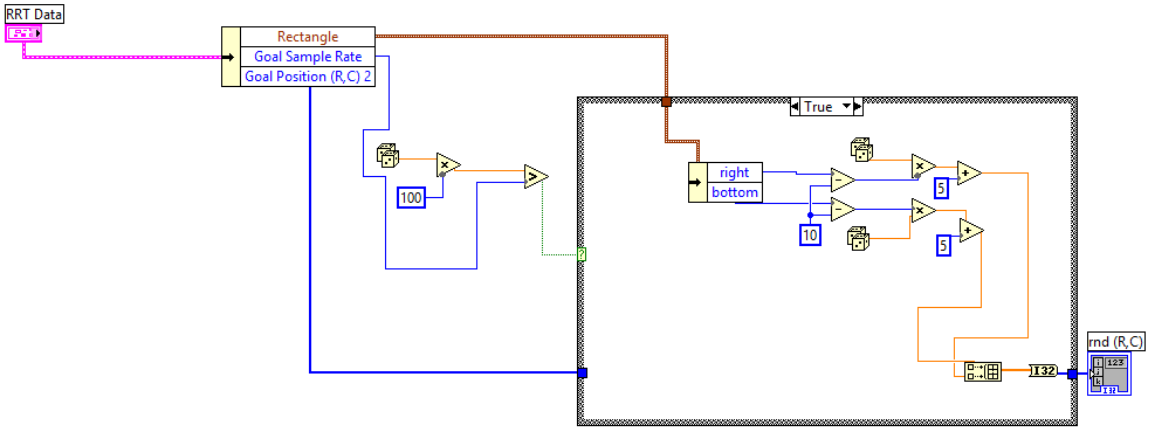
Subprogram “Random Sampling.vi” melakukan perintah baris 4 pada Algorithm 1. Block diagram dari “Random Sampling.vi” ditunjukkan pada **Gambar 11**. Subprogram “Get Nearest List Index.vi” melakukan perintah baris 5 pada Algorithm 1. Block diagram dari “Get Nearest List Index.vi” ditunjukkan pada **Gambar 12**. Subprogram “Steer.vi” melakukan perintah baris 6 pada Algorithm 1. Block diagram dari “Steer.vi”” ditunjukkan pada **Gambar 13**. Subprogram “Collision Check.vi” melakukan perintah baris 7 pada Algorithm 1. Subprogram “Insert Node.vi” melakukan perintah baris 8 pada Algorithm 1. Block diagram dari “Insert Node.vi” ditunjukkan pada **Gambar 14**.

Agar algoritma RRT dapat divisualisasikan prosesnya, maka dibuat subprogram “Draw.vi” yang akan mengvisualisasikan proses yang dilakukan oleh algoritma RRT untuk setiap iterasinya. Block diagram dari “Draw.vi” ditunjukkan pada **Gambar 15**.

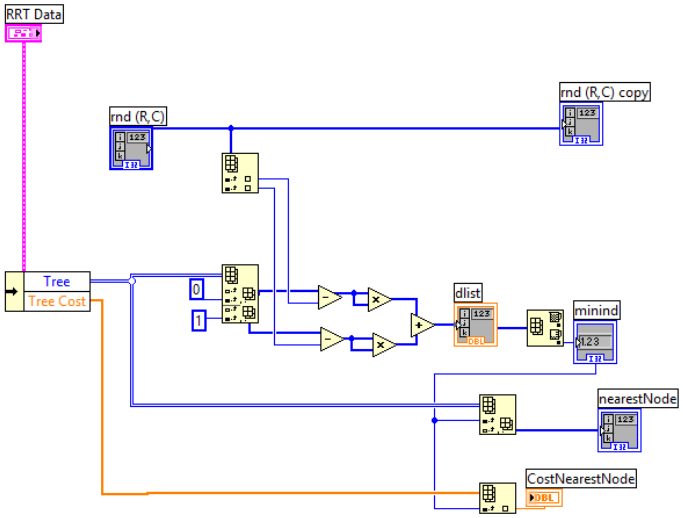
Agar dapat diamati apakah solusi jalur yang dibuat oleh RRT bersifat konvergen atau tidak, maka akan dicatat histori solusi jalur lengkap yang telah menghubungkan node awal dengan node tujuan akan disimpan. Pertama-tama subprogram “Check Goal.vi” akan mengecek apakah pada pohon pencarian yang dikembangkan oleh RRT sudah mencapai node tujuan. Jika telah ada node baru pada pohon pencarian yang berhasil mencapai node tujuan, maka oleh subprogram “Make and Calculate Path.vi” akan dibuat jalur lengkap yang menghubungkan node tersebut dengan node awal berdasarkan data dari pohon pencarian yang ada. Lalu pada subprogram “Draw Complete Path.vi”, jalur lengkap yang sudah dibuat akan divisualisasikan menggunakan garis yang lebih tebal. Block diagram dari “Check Goal.vi” ditunjukkan pada **Gambar 16**, block diagram dari “Make and Calculate Path.vi” ditunjukkan pada **Gambar 17** dan block diagram dari “Draw Complete Path.vi” ditunjukkan pada **Gambar 18**.



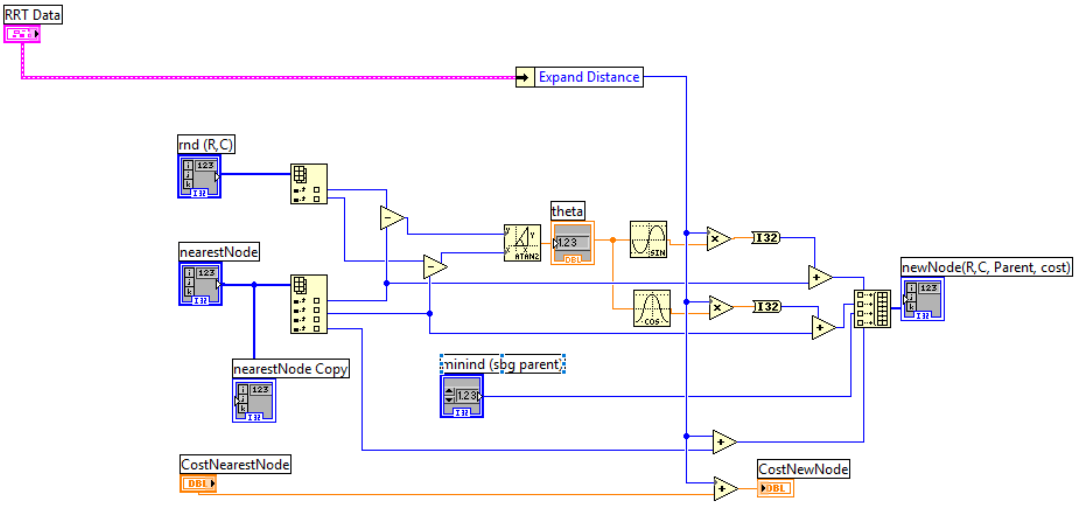
Gambar 10. Block diagram “Inisialisasi Data.vi”



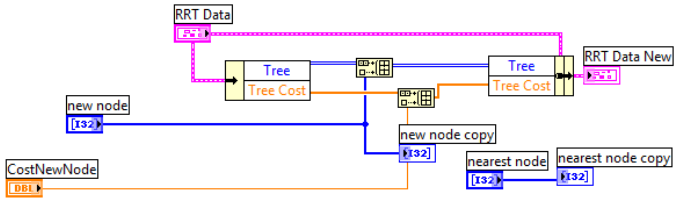
Gambar 11. Block diagram "Random Sampling.vi"



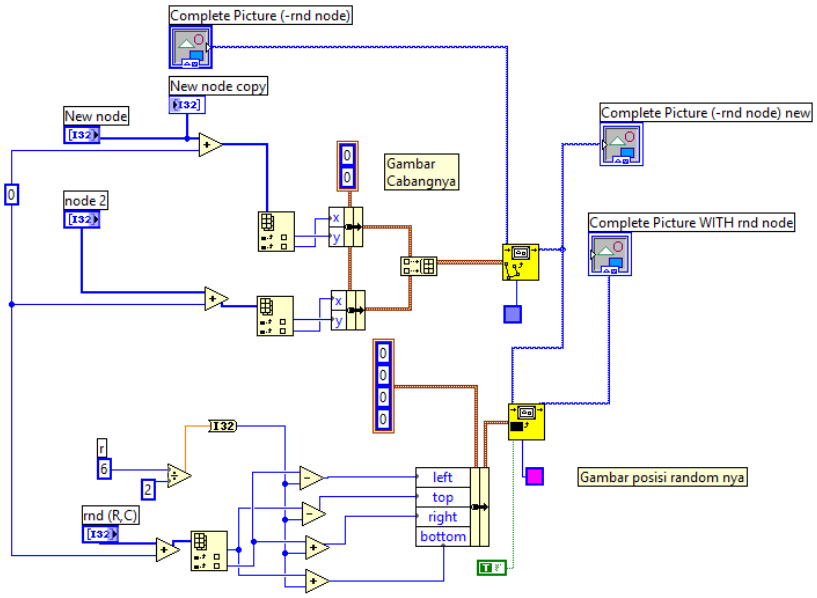
Gambar 12. Block diagram "Get Nearest List Index.vi"



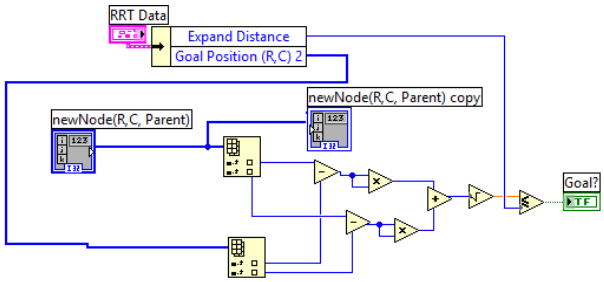
Gambar 13. Block diagram "Steer.vi"



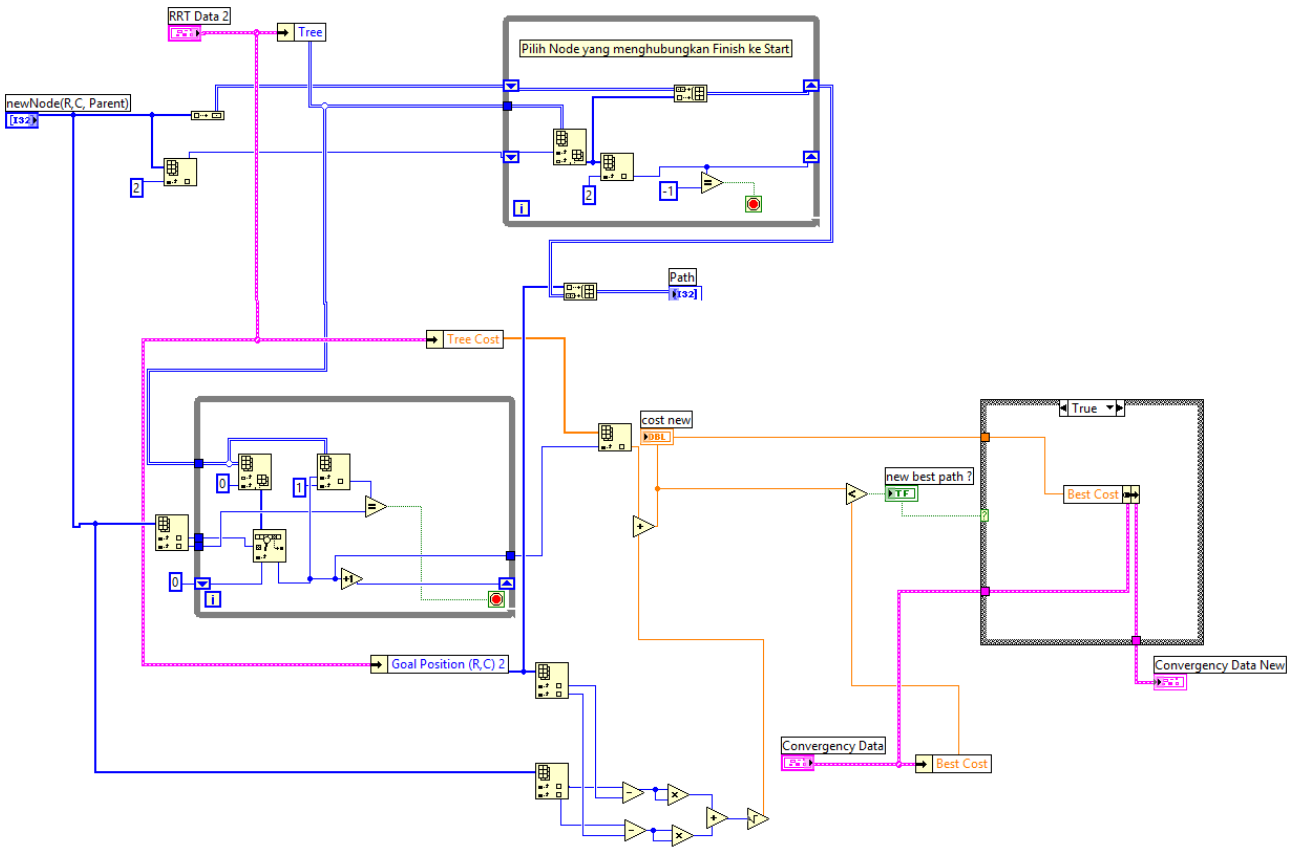
Gambar 14. Block diagram "Insert Node.vi"



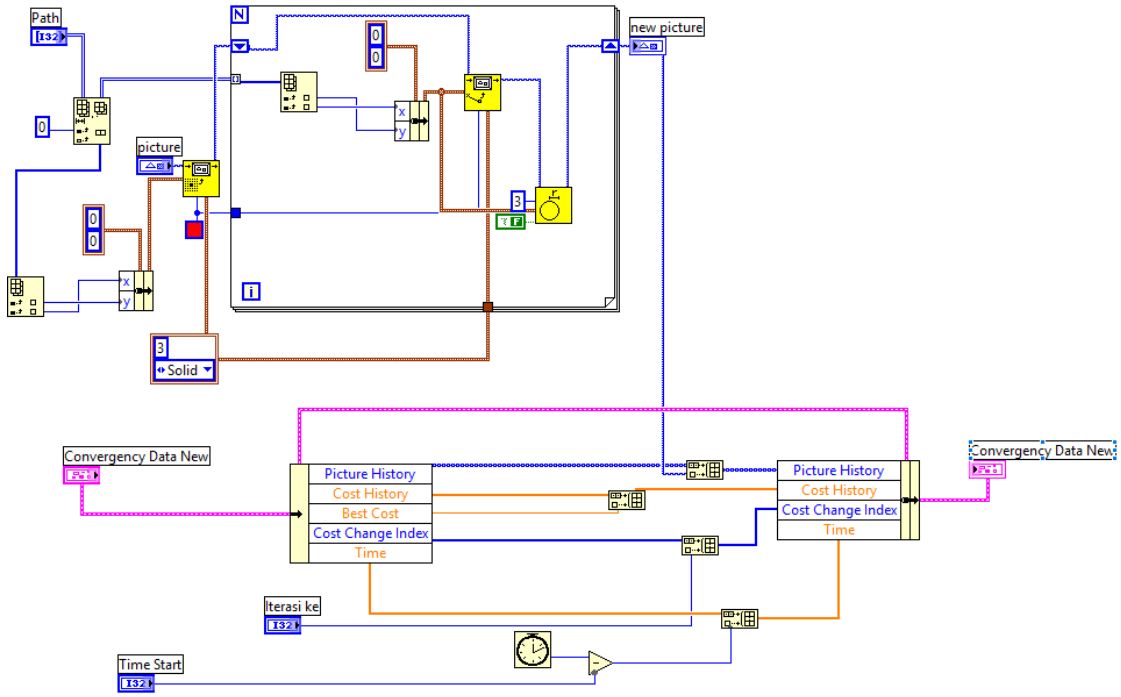
Gambar 15. Block diagram "Draw.vi"



Gambar 16. Block diagram "Check Goal.vi"



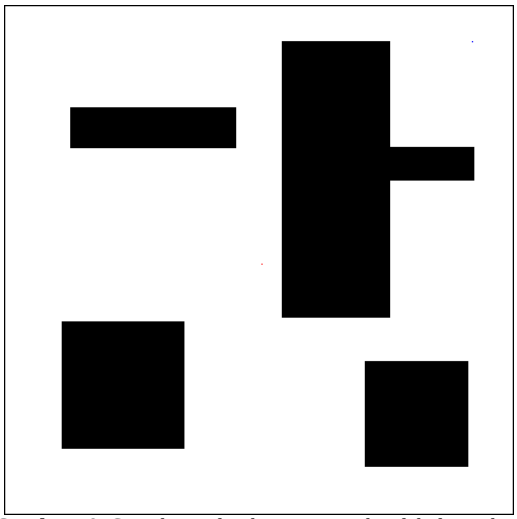
Gambar 17. Block diagram "Make and Calculate Path.vi"



Gambar 18. Block diagram "Draw Complete Path.vi"

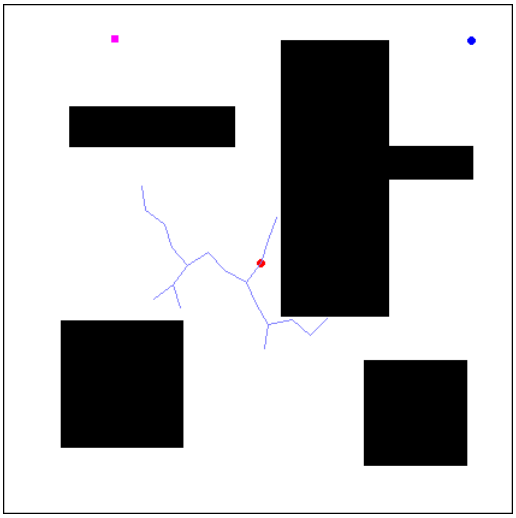
Agar waktu komputasi dapat minimal, maka overhead subVI diminimalkan dengan menjadikan subVI sebagai subroutine [20]. Program juga tidak menggunakan fungsi-fungsi bawaan LabVIEW yang tidak bisa dijadikan sebagai subroutine. Hal ini berhasil terujikan dimana dalam menyelesaikan suatu kasus *benchmark*, program RRT yang dibuat dapat menyelesaikannya dalam waktu 185 milidetik.

Salah satu pengujian dilakukan dengan menggunakan peta *benchmark* yang diusulkan oleh Karaman pada [11]. Peta lingkungan yang digunakan untuk pengujian ditunjukkan pada **Gambar 19**. Gambar peta tersebut dibuat pada program MS Paint.



Gambar 19. Contoh peta lingkungan yang hendak digunakan untuk pengujian algoritma RRT. Peta dibuat menggunakan program MS Paint.

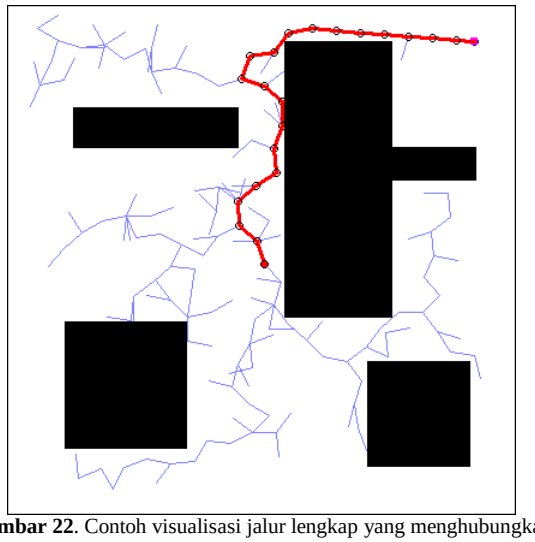
Saat program dijalankan dapat diperoleh visualisasi seperti pada **Gambar 20**. Pada visualisasi ini diperlihatkan posisi node random pada setiap iterasinya (kotak pink). Dan akan terlihat juga adanya cabang baru yang menuju kearah node random tersebut. Posisi node start diperlihatkan sebagai lingkaran berwarna merah. Posisi node goal diperlihatkan sebagai lingkaran berwarna biru.



Gambar 21. Contoh Visualisasi saat node baru dibangun. Terlihat terdapat cabang baru menuju nore random (kotak pink)

Saat algoritma telah berhasil menemukan node yang berdekatan dengan node goal, maka akan dibangun jalur lengkap yang menghubungkan node awal dengan node tujuan. Contoh jalur lengkap yang berhasil dibangun divisualisasikan dalam garis tebal seperti pada **Gambar 22**.

Selama iterasi berlangsung akan ada kemungkinan algoritma RRT menemukan alternatif jalur lengkap lainnya yang memiliki jarak tempuh lebih pendek daripada jalur lengkap yang sudah ada. Maka jalur lengkap terpendeklah yang akan ditampilkan.

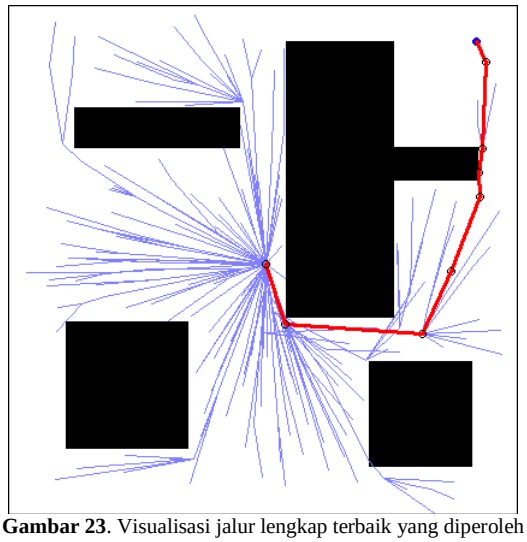


Gambar 22. Contoh visualisasi jalur lengkap yang menghubungkan node awal dan node tujuan. Jalur diperoleh dalam waktu 185 milidetik

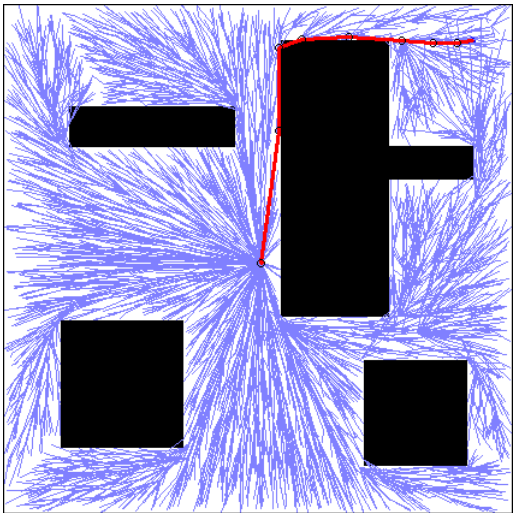
Untuk pengujian menggunakan algoritma RRT*, dapat terlihat hasilnya seperti pada **Gambar 23** dan **Gambar 24**. **Gambar 23** adalah jalur yang dihasilkan oleh algoritma RRT* pada iterasi ke 366. Adapun **Gambar 24** adalah jalur yang dihasilkan oleh algoritma RRT* pada iterasi ke 9271. Kedua gambar ini mencontohkan bahwa solusi jalur dari algoritma RRT* bersifat asimptotik optimal, yaitu solusi dari algoritma RRT* akan menuju solusi yang optimal jika waktu iterasi yang diberikan memungkinkan.

Hasil ini mendukung laporan yang dihasilkan oleh Karaman pada [11]. Karaman melaporkan contoh hasil algoritma RRT seperti pada **Gambar 25** dan **Gambar 26**. **Gambar 25** menunjukkan hasil yang diperoleh Karaman pada iterasi ke 1500, dan **Gambar 26** menunjukkan hasil yang diperolehnya pada iterasi ke 15000.

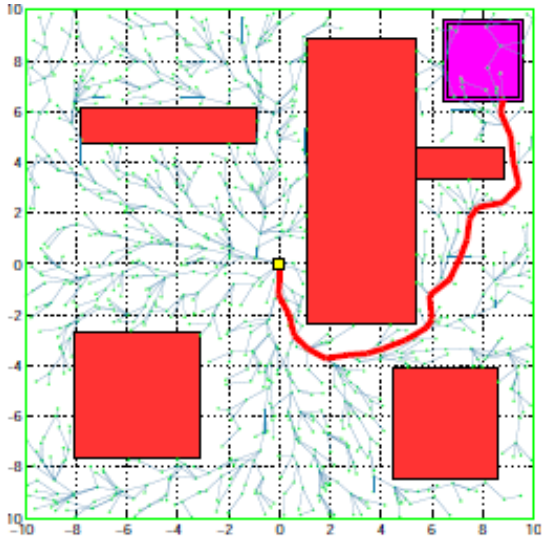
Untuk pengujian menggunakan algoritma RRT-biased, diperoleh hasil seperti pada **Gambar 27**. Sesuai dengan prinsipnya bahwa node baru tidak dibuat satu kali, tetapi dibuat beberapa kali hingga node goal tercapai atau terhalang oleh hambatan, maka pada **Gambar 27** dapat terlihat bahwa arah cabang yang sama terdapat garis lurus dimana diantara garis lurus tersebut terdapat node-node antaranya. Hal ini membuat lebih sedikti jalur berliku yang ada.



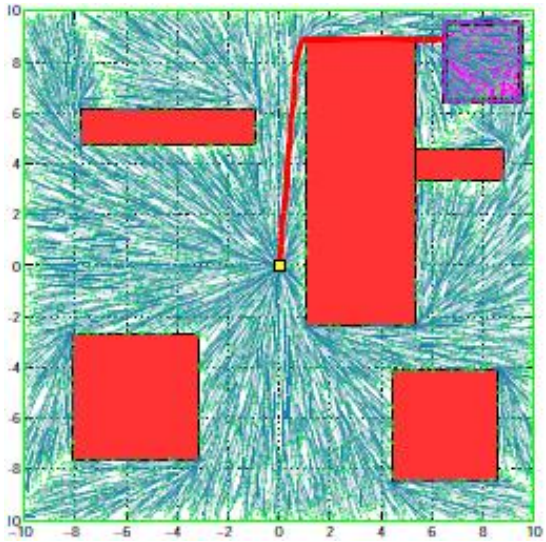
Gambar 23. Visualisasi jalur lengkap terbaik yang diperoleh algoritma RRT setelah iterasi ke 366. Jalur diperoleh dalam waktu 1273 milidetik



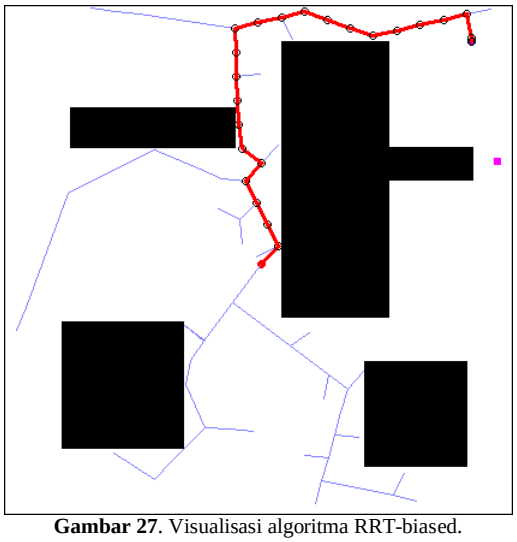
Gambar 24. Visualisasi jalur lengkap terbaik yang diperoleh algoritma RRT setelah iterasi ke 9271. Jalur diperoleh dalam waktu 10644 milidetik



Gambar 25. Visualisasi jalur lengkap terbaik yang dilaporkan oleh Karaman pada [11] setelah iterasi ke 1500



Gambar 26. Visualisasi jalur lengkap terbaik yang dilaporkan oleh Karaman pada [11] setelah iterasi ke 15000

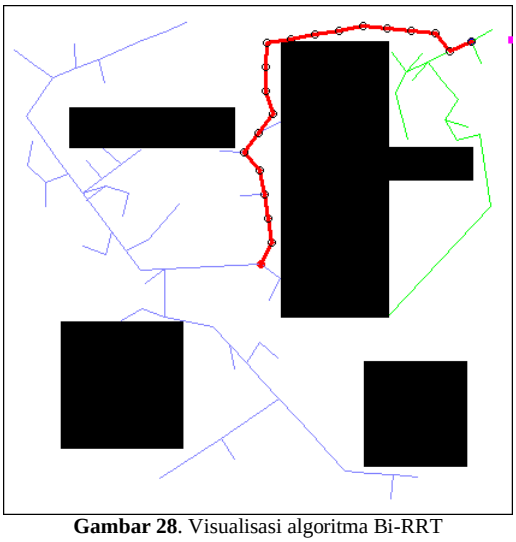


Gambar 27. Visualisasi algoritma RRT-biased.

Adapun untuk pengujian algoritma Bi-RRT, ditunjukkan pada **Gambar 28**. Terlihat terdapat dua pohon pencarian yang ada, yaitu pohon pencarian yang berawal dari node start (berwarna biru) dan pohon pencarian yang berawal dari node tujuan (berwarna hijau).

IV. KESIMPULAN

Dari penelitian ini telah dipresentasikan simulator untuk pembelajaran algoritma RRT. Simulator ini juga menyajikan beberapa variasi dari algoritma RRT, yaitu RRT*, RRT-biased dan Bi-RRT. Simulator dapat menyajikan proses yang dilakukan algoritma RRT pada setiap iterasinya. Waktu komputasi dari program juga sudah diminimalkan. Untuk suatu kasus *benchmark*, algoritma RRT dapat menyusun jalur dalam waktu 185 milidetik sedangkan algoritma RRT* dapat menyusun jalur dalam waktu 1273 milidetik. Dengan memanfaatkan simulator ini diharapkan dapat membantu mahasiswa untuk mendapatkan pemahaman yang lebih baik mengenai algorima perencanaan gerak menggunakan RRT.



Gambar 28. Visualisasi algoritma Bi-RRT

DAFTAR PUSTAKA

[1] M. Elbanhawi, and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56-77, 2014

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

[3] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308-333, 2009

[4] M. P. Garcia, O. Montiel, O. Castillo, R. Sepúlveda, and P. Melin, P. "Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation," *Applied Soft Computing*, vol. 9, no. 3, pp. 1102-1110, 2009

[5] F. Belkhouche, and B. Bendjilali, "Reactive path planning for 3-D autonomous vehicles," *IEEE transactions on control systems technology*, vol. 20, no. 1, pp. 249-256, 2011

[6] B. Manup, and P. Raja, "Collision-avoidance for mobile robots using region of certainty: A predictive approach," *Journal of Engineering Science and Technology*, vol. 11, no. 1, pp. 18-28, 2016

[7] M. Werling, S. Kammel, J. Ziegler, and L. Gröll, "Optimal trajectories for time-critical street scenarios using discretized terminal manifolds," *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 346-359, 2012

[8] M. Elbanhawi, and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56-77, 2014

[9] 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

[10] 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

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

[12] S. Karaman, and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," *49th IEEE conference on decision and control (CDC)*, pp. 7681-7687, 2010

[13] B. Akgun, and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2640-2645, 2011

[14] B. S. Kumar, P. S. Abhimanyu, B. V. V. P Bharagav, P. Agarwal, and K. M. Krishna, *RoboCup SSL Team Description, IRL RC*, pp. 1-4, 2010
[15] J. D. Gammell, S. S. Srinivasa, and 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
[16] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, and M. S. Muhammad, "RRT*-SMART: A rapid convergence implementation of RRT," *International Journal of Advanced Robotic Systems*, vol. 10, no. 7, pp. 1-12, 2013
[17] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, dan J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105-1118, 2009
[18] 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
[19] E. Ackerman, "Hail, robo-taxi!" *IEEE Spectrum*, vol. 54, no. 1, pp. 26-29, 2017
[20] 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
[21] N. Vahrenkamp, T. Asfour, dan R. Dillmann, "Simo: A simulation and motion planning toolbox for c+", 2010
[22] P. Corke, "Robotics toolbox for MATLAB", 2011
[23] Aria, M. Educational Simulator for Teaching of Particle Swarm Optimization in Lab VIEW, 2013.
[24] LabVIEW Basics, "Development course manual," *National Instruments Corporation, Austin, Texas*, 2003
[25] B. C. Guevara, "An Overview of the Class of Rapidly-Exploring Random Trees," *Utrecht University*, 2018