TELEKONTRAN, VOL. 9, NO. 2, OKTOBER 2021
DOI : 10.34010/telekontran.v9i2.5750
p-ISSN : 2303 – 2901
e-ISSN : 2654 – 7384

Analisa Performansi Metode *Sampling Goal Biasing* dan Metode *Sampling Gaussian* pada Algoritma *Rapidly-exploring Random Tree**

Performance Analysis of Goal Biasing Sampling Method and Gaussian Sampling Method on a Rapidly-exploring Random Tree Algorithm*

Tri Rahajoeningroem*, Dea Ferida
Program Studi Teknik Elektro, Fakultas Teknik dan Ilmu Komputer
Universitas Komputer Indonesia Jl. Dipati ukur No 112, Bandung
*Email : tri.rahajoeningroem@email.unikom.ac.id

Abstrak - Algoritma perencanaan gerak atau jalur merupakan salah satu bidang penelitian mendasar dalam robotika. Algoritma *Sampling Based Planning* (SBP) yang paling umum digunakan yaitu algoritma *Rapidly-exploring Random Tree* (RRT). Algoritma yang digunakan pada penelitian ini adalah algoritma RRT* *goal biasing* dan algoritma RRT* g*aussian sampling*. Tujuan penelitian ini adalah melakukan analisa performansi perencanaan jalur algoritma RRT* *goal biasing* dan algoritma RRT* g*aussian sampling*. Pengujian dilakukan menggunakan beberapa kasus *benchmark* yang ada, yaitu lingkungan *narrow*, *trap*, dan *clutter*. Parameter yang dibandingkan adalah biaya jalur, waktu komputasi, dan total node. Menggunakan kasus *benchmark* lingkungan *narrow, clutter,* dan *trap* algoritma RRT* *goal biasing* memperoleh nilai rata-rata untuk biaya jarak, waktu, dan jumlah node yaitu; 8,3, 219 node, 30,065 detik (lingkungan *narrow*), 11,6, 222,1 node, 30,05 detik (lingkungan *clutter),* dan 14,3, 199,45 node, 30,045 detik (lingkungan *trap).* Kemudian untuk kasus *benchmark* lingkungan *narrow, clutter,* dan *trap* yang menggunakan algoritma RRT* g*aussian* memperoleh nilai rata-rata untuk biaya jarak, waktu, dan jumlah node yaitu; 8,1, 372 node, 32,955 detik (lingkungan *narrow*), 11,6, 642,85 node, 33,295 detik (lingkungan *clutter),* dan 13,5, 345 node, 30,45 detik (lingkungan *trap)*. Berdasarkan hasil pengujian algoritma RRT* *goal biasing* memiliki keunggulan untuk waktu dan jumlah node untuk mencapai titik goal, tetapi biaya jarak yang dihasilkan kurang optimal. Algoritma *goal biasing* akan mengeluarkan node random secara acak dan sesekali akan memunculkan node random ke titik tujuan yang dapat membuat pencarian awal node tujuan menjadi lebih cepat. Implikasi dari penelitian ini dapat menggunakan algoritma *goal biasing* dalam pencarian jalur yang mengoptimalkan waktu dan jumlah node.

Kata kunci *:* *goal biasing sampling*, *gaussian sampling*, perencanaan jalur, *rapidly-exploring random tree**

Abstract *- Motion or path planning algorithms are one of the fundamental research areas in robotics. The most commonly used Sampling Based Planning (SBP) algorithm is the Rapidly-exploring Random Tree (RRT) algorithm. The algorithm used in this research is the RRT* goal biasing algorithm and the RRT* gaussian sampling algorithm. The goal of this study is to analyse the performance of the RRT* goal biasing and RRT* gaussian sampling algorithms in path planning. The test is performed using several existing benchmark cases, including narrow, trap, and clutter environments. Path costs, computation time, and the total node are the parameters compared. The RRT* goal biasing algorithm obtains the average values for the cost of distance, time, and number of nodes using the cases of narrow, clutter, and trap environmental benchmarks, namely; 11,6, 222,1 nodes, 30,05 second (in narrow), 11,6, 642,85 nodes, 33,295 second (in clutter), and 13,5, 345 nodes, 30,45 second (in trap). The average values for the cost of distance, time, and number of nodes for the case of narrow, clutter, and trap environmental benchmarks using the Gaussian RRT* algorithm are; 8,1, 372 nodes, 32,955 second (in narrow), 11,6, 642,85 nodes, 33,295 second (in clutter), and 13,5, 345 nodes, 30,45 second (in trap). According to the test results, the RRT* goal biasing algorithm has advantages in terms of time and number of nodes required to reach the goal point, but the resulting distance costs are less than optimal. The goal biasing algorithm will issue random nodes at random and occasionally will bring up random nodes to the destination point which can make the initial search for the destination node faster. The implication of this research is to use an objective bias algorithm in finding a path that optimizes the time and number of nodes.*

Keywords *: goal biasing sampling, gaussian sampling, path planning, rapidly-exploring random tree**

I. PENDAHULUAN

A. Latar Belakang

Algoritma perencanaan gerak atau jalur (*motion planning*) merupakan salah satu bidang penelitian mendasar dalam robotika [1]. Algoritma perencanaan jalur (*path planning)* telah banyak diteliti dan dikembangkan pada program kendaraan maupun robotika. Tujuan dari perencanaan jalur adalah merancang jalur dari posisi awal ke posisi tujuan pada lingkungan yang memiliki hambatan statis maupun dinamis [2]. Perencanaan jalur juga dapat digunakan di industri, seperti pada tangan robot untuk memindahkan barang dari satu lokasi ke lokasi lain [3].

Algoritma perencanaan jalur memungkinkan robot atau kendaraan menemukan jalur terpendek atau jalur optimal antara dua titik. Jalur optimal dapat berupa jalur yang meminimalkan jumlah belokan, jumlah pengereman, jumlah tabrakan, atau apa pun yang diperlukan dalam program tersebut [4]. *Path planning* juga dapat diaplikasikan pada robot otonom. Robot otonom memiliki memampuan untuk bekerja secara otomatis tanpa campur tangan manusia. Robot otonom dapat bekerja untuk mencari informasi, dapat bekerja untuk waktu yang lama, serta dapat bekerja pada tempat yang berbahaya. Robot otonom dengan kecerdasan dapat menyelesaikan tugas biarpun lingkungannya telah berubah, karena robot otonom dengan kecerdasan dapat belajar atau menemukan pengetahuan baru.

Keamanan untuk tidak menabrak kendaraan lain merupakan hal penting dalam perencanaan algoritma *path planning* [5]. Pada proses pencarian jalur harus dapat menghindari setiap *rintangan atau halangan* yang ada [6][7]. Pada algoritma *path planning* haruslah mampu mengoptimalkan kualitas jalur dan cukup cepat untuk digunakan dalam aplikasi kompleks dunia nyata [8]. Robot dapat bekerja dengan baik apabila kualitas jalur yang tersusun dengan optimal.

Setiap metode pencarian jalur bergantung pada keadaan lingkungan [9]. Keadaan lingkungan mewakili semua posisi yang mungkin untuk orientasi robot.

Pada perencanaan jalur terdapat beberapa algoritma yang dapat digunakan. Algoritma *Rapidly-exploring Random Tree* (RRT) merupakan algoritma yang dapat digunakan untuk perencanaan jalur pada robotika. Algoritma RRT dikembangkan oleh Karaman menjadi *Rapidly-exploring Random Tree** (RRT*) [10] merupakan salah satu algoritma yang dapat digunakan pada algoritma *path planning*.

B. Tinjauan *State of Art*

Algoritma Sampling Based Planning (SBP) yang paling umum digunakan adalah algoritma *Probabilistic Roadmap Method* (PRM) [11] dan algoritma *Rapidly-exploring Random Tree* (RRT) [12]. Keunggulan dari algoritma PRM adalah dapat memberikan solusi yang optimal *asimptotik*, tetapi memiliki kekurangan yaitu waktu komputasi yang tinggi. Adapun keunggulan algoritma Rapidly-exploring Random Tree (RRT) adalah memiliki waktu komputasi yang kecil, tetapi kekurangannya adalah hanya menyediakan solusi yang bersifat sub-optimal [13].

Algoritma RRT* memiliki keunggulan yaitu dapat menghasilkan solusi jalur yang bersifat optimal asimptotik [14]. Terdapat beberapa publikasi mengenai pembuatan program dari algoritma RRT ini. Sakai [15] telah mempublikasikan program robotic, diantaranya untuk perencanaan jalur menggunakan A*, PRM, RRT, RRT*, dan masih ada lagi. Tetapi Sakai menggunakan program python. Vahrenkamp juga telah mempublikasikan programas perencanaan jalur, tetapi dibuat menggunakan program C+ [16]. Sepengetahuan penulis, belum ada penelitian sebelumnya yang menggunakan metode *goal biasing* dan *gaussian sampling* pada algoritma RRT* pada bahasa pemrograman LabVIEW. Oleh karena itu, pada penelitian ini dibuat algoritma perencanaan jalur RRT* *goal biasing* dan g*aussian sampling* menggunakan Bahasa pemrograman LabVIEW.

C. Tujuan

Tujuan dari penelitian ini adalah untuk menganalisa performansi algoritma perencanaan jalur *Rapidly Exploring Random Tree** (RRT*) dengan metode *goal biasing*, dan algoritma RRT* dengan metode *sampling gaussian.*

D. Sistematika Pembahasan

Pada bagian kedua yaitu metodologi akan menjelaskan inti algoritma yang digunakan serta *inti block diagram* yang buat pada LabVIEW. Algoritma yang digunakan dalam *path planning* yaitu Algoritma RRT* *goal biasing* dan RRT* *gaussian sampling.* Pada bagian 3 yaitu hasil dan pembahasan akan menjelaskan performansi dengan parameter waktu eksekusi tiap algoritma, biaya jarak, dan jumlah *node* yang dibutuhkan untuk mencapai titik goal. Pengujian dilakukan pada lingkungan *narrow*, *clutter*, serta *trap*. Hasil

dari pengujian pada ketiga lingkungan yang telah dilakukan akan disimpulkan pada bagian 4.

II. METODOLOGI

Pada bab ini akan membahas mengenai perancangan sistem. Perancangan ini terdiri dari perancangan algoritma RRT* *goal biasing* dan RRT* *gaussian sampling* pada bahasa pemograman LabVIEW. Algoritma RRT* memiliki penambahan dua algoritma diantaranya algoritma *choose parent* dan algoritma *rewire*, kedua algoritma itu yang membedekan dengan algoritma RRT. Seperti yang dijelaskan sebelumnya algoritma RRT* sangat mempertimbangkan semua node terdekat dalam proses pencarian jalur, proses pemilihan node yang lebih optimal ini berada pada program *choose parent*. Kemudian algoritma RRT* juga akan ditambahkan subprogram *goal biasing* dan *gaussian sampling* untuk mengambil nilai *random number*. Detail Algoritma RRT* terdapat pada **Gambar 1.**

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 1. Algoritma RRT* [17]

Dilihat dari **Gambar 1.** yang merupakan Algoritma RRT* yang akan melakukan beberapa proses, RandomSample, NearestNeighbor, Steer dan InsertNode. Proses RandomSample bertujuan mengambil sampel pada ruang pencarian, yang disebut sebagai qrand (baris 4). NearestNeighbor mencari node pada pohon pencarian yang terdekat ke *qrand*. Node terdekat ini dinamakan sebagai qnearest (baris 5).

Selanjutnya akan dibangun node baru diantara qnearest dan qrand. Node baru ini dinamakan qnew. Node qnew akan berjarak Δq dari qnearest (baris 6). Jika diantara qnew dan qnearest tidak ada hambatan, maka node baru akan ditambahkan ke pohon pencarian (baris 7 dan 8). Lalu iterasi akan berulang sebanyak N kali (baris 3 dan 9) [17]. Kemudian pada penelitian ini *random sample* yang terdapat pada baris 4 akan menggunakan metode *goal biasing sampling* dan g*aussian sampling.* Program algoritma RRT* g*oal biasing, dan* RRT* *gaussian sampling* terdiri dari 14 sub-program inti diantarannya:

1. Subprogram "*Goal Biasing sampling*.vi"
2. Subprogram *"Gaussian Sampling.vi"*
3. Subprogram "*Choose parent*.vi"
4. Subprogram "*Rewire*.vi"
5. Subprogram *"Get Nearest Node.vi"*
6. Subprogram *"Steer.vi"*
7. Subprogram *"Draw Path.vi"*
8. Subprogram *"Draw Graph.vi"*
9. Subprogram *"Calc Dist to Goal.vi"*
10. Subprogram *"Calc New Cost.vi"*
11. Subprogram *"Check Obstacle.vi"*
12. Subprogram *"Check Collision.vi"*
13. Subprogram *"Generate Final Course.vi"*
14. Subprogram *"Get Nearest Node.vi"*

Berikut **Gambar 2.** dan **Gambar 3.** algoritma dari *choose parent* dan algoritma *rewire*.

Algorithm 4 : $q_{min} \leftarrow ChooseParent\ (q_{gauss}, Q_{near}, q_{nearest}, \Delta q)$

33. $q_{min} \leftarrow q_{nearest}$
34. $c_{min} \leftarrow Cost\ (q_{nearest}) + c\ (q_{gauss})$
35. **for** $q_{near} \in Q_{near}$ **do**
36. $q_{path} \leftarrow Streer\ (q_{near}, q_{gauss}, \Delta q)$
37. **if** *ObstacleFree* (q_{path}) *then*
38. $c_{new} \leftarrow Cost\ (q_{near}) + c\ (q_{gauss})$
39. **If** $c_{min} < c_{new}$ **then**
40. $c_{min} \leftarrow c_{new}$
41. $q_{min} \leftarrow q_{new}$
42. **end**
43. **end**
44. **end**
45. **Return** q_{min}

Gambar 2. Algoritma *choose parent* [17]

Algorithm 5: $T \leftarrow Rewire\ (T, Q_{near}, q_{min}, q_{gauss})$

46. **for** $q_{near} \in Q_{near}$ **do**
46. $q_{path} \leftarrow Steer\ (q_{near}, q_{gauss}, \Delta q)$
47. **if** *ObstacleFree* (q_{path}) *then*
47. $Cost\ (q_{gauss}) + c\ (q_{path}) < Cost\ (q_{near})$ **then**
48. $T \leftarrow ReConnect\ (q_{gauss}, q_{near}, T)$
49. **end**
50. **return** T

Gambar 3. Algoritma *rewire* [17]

Berdasarkan **Gambar 2.** jika pemilihan *parent* dilakukan, akan dicari *node-node* terdekat dengan q_{new} lalu dihubungkan antara *node* awal hingga q_{new} akan diperoleh jalur yang lebih dekat. Program perubahan *path planning* ini terjadi pada proses algoritma *rewire* (algoritma 5) yang dapat dilihat

pada **Gambar 3.** terjadi proses pencarian *node-node* terdekat dengan *node* q_{new} yang menghasilkan jalur yang lebih singkat.

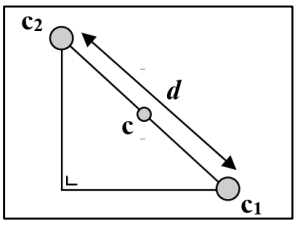
Subprogram "*Gaussian sampling*.vi" pada algorima RRT* berfungsi untuk mencari nilai *gaussian random sampling* pada ruang. Nilai *gaussian random sampling* didapatkan dengan menggunakan metode transformasi *box muller*. Persamaan *box muller* secara umum terdapat dua parameter dapat dilihat pada persamaan (1) dan persamaan (2) berikut ini.

$$x_1 = \sqrt{-2ln(R_1)cos(2\pi R_2)} \quad (1)$$

dan

$$x_2 = \sqrt{-2ln(R_1)sin(2\pi R_2)} \quad (2)$$

Sebuah *sampling* dapat disebut metode *gaussian* apabila posisi c_1 dan c_2 yang terhubung sejauh d, salat satu titik nya berada didalam *obstacle* dan titik lainnya berada diruang bebas *obstacle*. Ketika titik c_1 yang berada di luar bebas *obstacle* maka c_1 dapat dimasukan ke graph. Begitupun sebaliknya jika c_2 yang berada di ruang bebas *obstacle* maka titik c_2 yang akan ditambahkan ke graph. Pada metode *gaussian* titik *sampling* yang dapat diambil sebagai *node random* terdapat pada titik-titik antara jarak c_1 dengan c_2. Nilai titik *random* diantara garis c_1 dan c_2 dinotasikan sebagai c. Berikut **Gambar 4.** ilustrasi pengambilan titik sampling *Gaussian.*



Gambar 4. Ilustrasi Pengambilan Titik *Sampling Gaussian*

Pada **Gambar 4.** dapat terlihat ilustrasi proses pengambilan *sampling* dengan metode *gaussian*. Mula-mula pada subprogram ini diinisialisakan sebuah *random node* yang diberi nama c_1 pada C_{free}. Selanjutnya peneliti menentukan nilai jarak yang akan digunakan pada distribusi normal dengan nama d. Dengan rumus phytagoras ditariklah garis dari titik c_1 sejauh d yang akan menghasilkan titik baru yang dinamakan titik c_2. Begitupun sebaliknya jika c_2 yang berada di ruang bebas *obstacle* maka titik c_2 yang akan ditambahkan ke graph. Pada metode *gaussian* titik *sampling* yang dapat diambil sebagai *node random* terdapat pada titik-titik antara jarak c_1 dengan c_2. Nilai titik *random* diantara garis c_1 dan c_2 dinotasikan sebagai c.

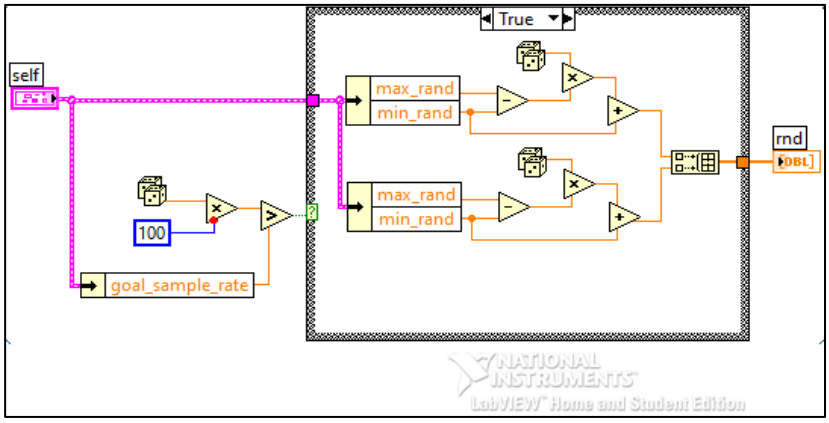
Block diagram subprogram "*Goal Biasing sampling*.vi" pada **Gambar 5.** berfungsi untuk mengambil sampel acak pada ruang pencarian, kemudian sesekali mengambil akan mengambil titik sampel langsung pada titik tujuan. Berdasarkan **Gambar 5.** yang merupakan Blok Digram program *goal biasing*, pada program tersebut akan mengambil konfigurasi tujuan qgoal dan dipilih bilangan acak yang dibangkitkan dari nilai random antara 0 – 100. Jika bilangan acak kurang dari persentase R(goal_sample_rate), maka konfigurasi acak *qrand* akan menjadi konfigurasi tujuan *qgoal*. Dan jika bilangan acak lebih dari persentase R(goal_sample_rate), maka konfigurasi acak qrand dipilih seperti metode pengambilan sampel seragam

Block diagram Subprogram "*Choose parent.vi"* pada **Gambar 6.** berfungsi untuk memilih *random node* yang dapat dijadikan *node parent* q_{parent} yang selanjutnya akan ditumbuhkan cabang baru. Dalam menampilkan hasil pencarian jalur RRT* dengan metode *gaussian sampling* divisualisasikan dengan pada subprogram "*Draw graph.vi".*

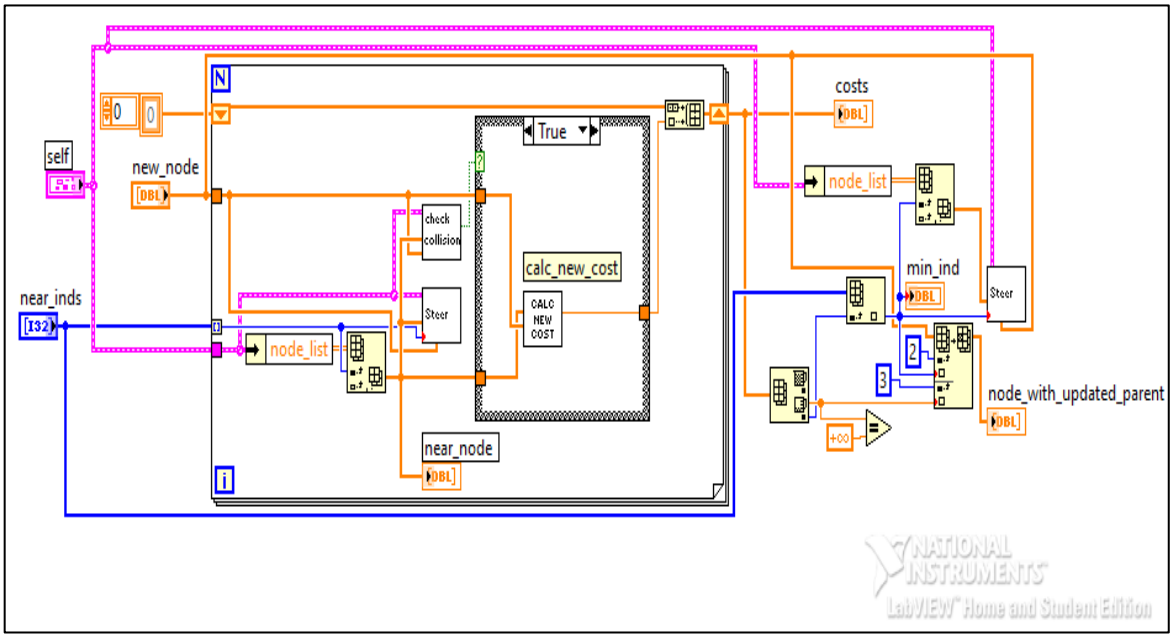
Block diagram subprogram "*get nearest node.vi"* pada **Gambar 7.** melakukan pemilihan node yang dapat dijadikan q_{parent}. Dalam mencari node yang terdekat dengan node *random* dibutuhkan subprogram "*find near node*.*vi"* pada **Gambar 8.**.

Block Diagram subprogram "*Check Collision.vi"* pada **Gambar 9.** berfungsi untuk mengecek ada atau tidaknya *obstacle* pada saat membuat *new node*. Proses ini berguna ketika pada saat pencarian jalur agar tidak terjadi tumbukan atau jalur menabrak *obstacle.*

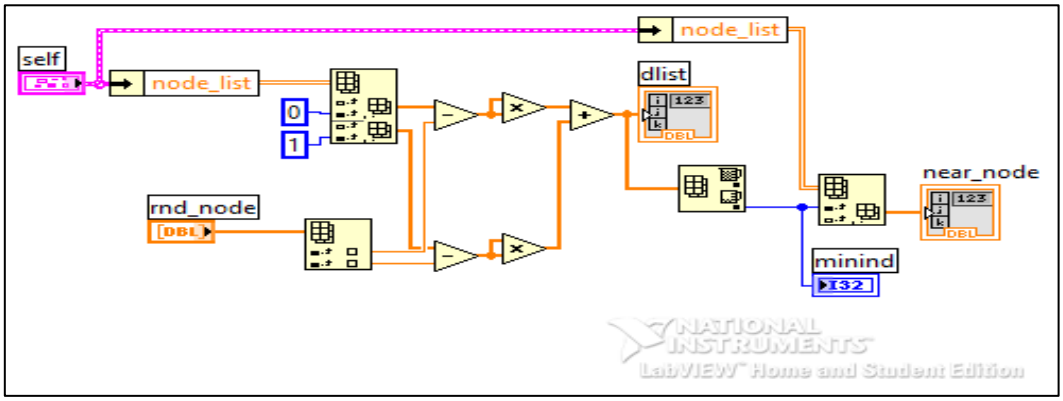
Peneliti dilakukan dengan menggunakan bahasa pemograman LabVIEW. Peneliti mengambil beberapa jenis lingkungan yang sering diuji pada pengujian algoritma pencarian jalur yaitu lingkungan *clutter* (banyak rintangan yang beragam), lingkungan *narrow passages* (celah sempit) dan lingkungan *obstacle trap*. Kriteria yang akan diujikan yaitu *path costs* (nilai jalur antara titik mulai menuju titik goal), waktu komputasi yang dibuthkan untuk mencapai titik *goal*, dan jumlah *node* yang dibutuhkan untuk mencapai titik *goal.*



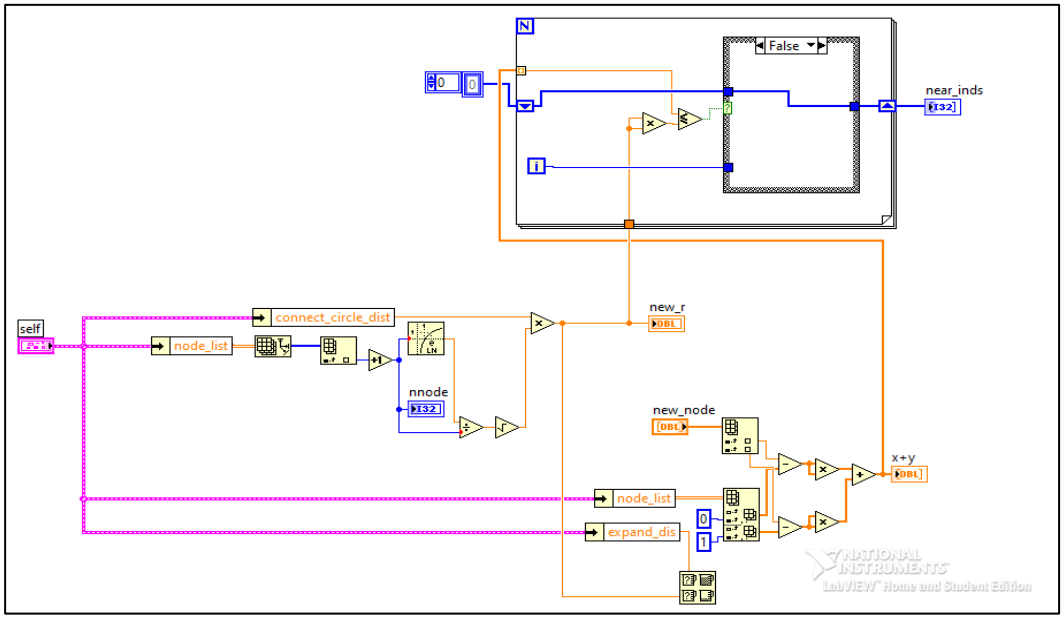
Gambar 5. Block diagram "*Goal Biasing sampling*.vi"



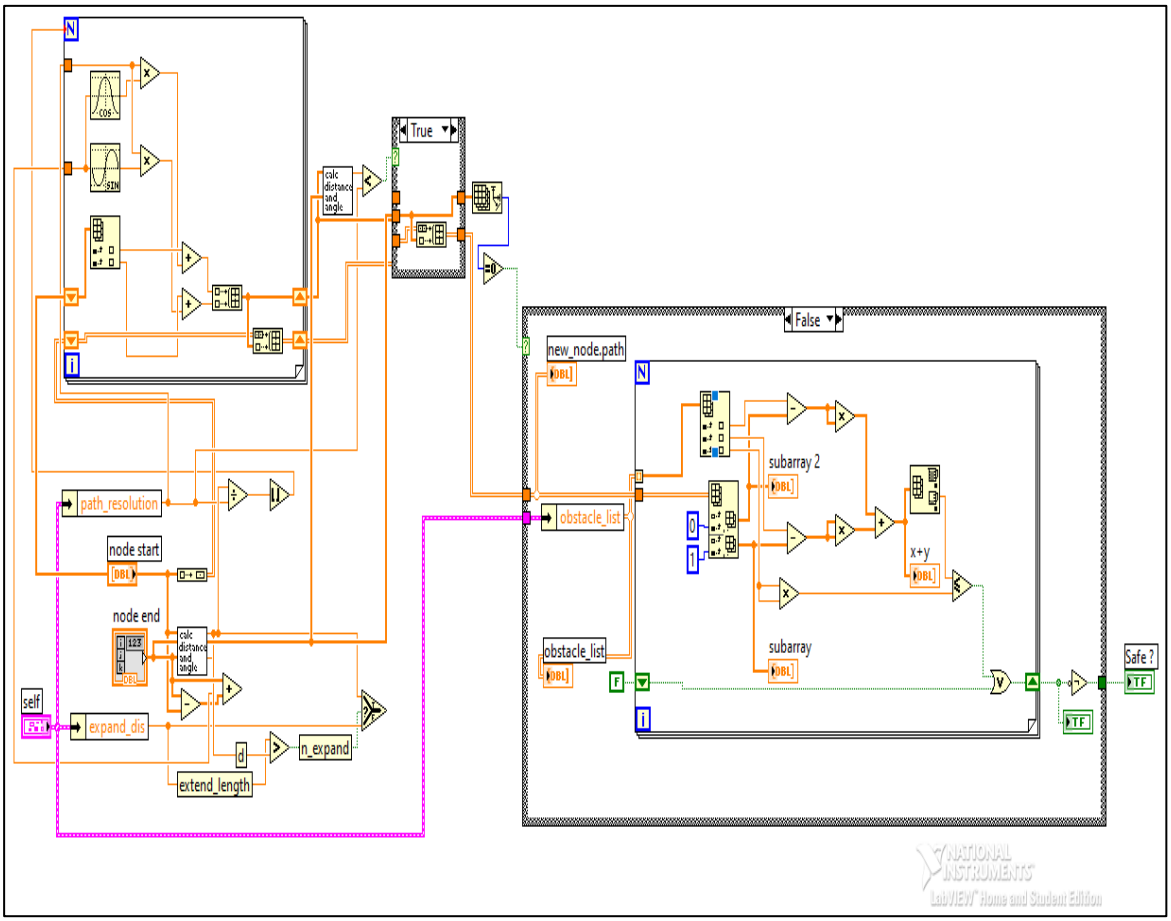
Gambar 6. Block diagram "*Choose parent.vi*"



Gambar 7. Block diagram "*get nearest node.vi"*



Gambar 8. "*find near node.vi*"



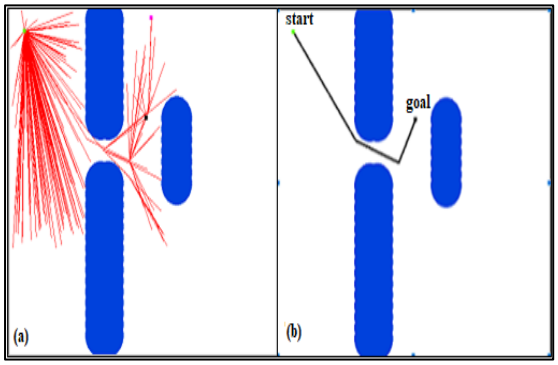
Gambar 9. "*Check Collision.vi*"

III. HASIL DAN PEMBAHASAN

Pengujian dan analisis pada tahap ini merupakan tahapan yang dilakukan setelah dirancang algoritma RRT* metode *goal biasing* dan RRT* g*aussian sampling* pada bahasa pemograman LabVIEW. Pengujian algoritma dilakukan dengan membandingkan performasi algoritma RRT* metode *goal biasing* dan RRT* g*aussian sampling*. Metode *goal biasing* pada perancangan algoritma RRT* pada bahasa pemograman LabVIEW.

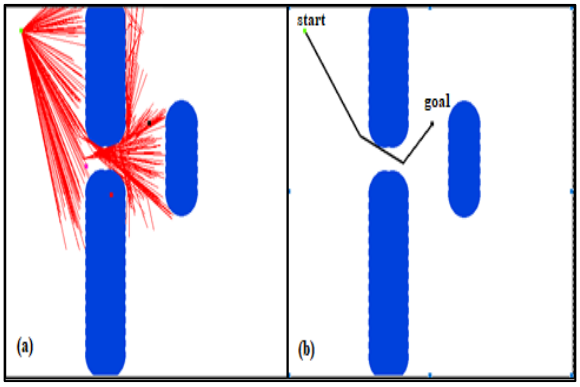
Metode tersebut sesuai dengan prinsip kerja nya pada algoritma RRT* pohon pencarian akan mencari jalur dengan menempatkan titik *random* secara acak pada ruang konfigurasi, kemudian sesekali akan menempatkan titik *random* yang searah dengan titik *goal*. Pengujian algoritma RRT* metode *goal biasing* dan RRT* g*aussian sampling* diukur dari parameter biaya jarak, waktu komputasi, dan jumlah node yang dibutuhkan dalam mencapai titik goal pad lingkungan *narrow*, *clutter*, dan *trap*. Hasil dari seluruh pengujian tersebut, kemudian dianalisa untuk mengetahui performansi dari RRT* *goal biasing* dan RRT* g*aussian sampling*.

Pada pengujian ini, algoritma RRT* *goal biasing* dan RRT* g*aussian sampling* masing-masing dijalankan sebanyak 20 kali pada kasus lingkungan obstacle trap, narrow, dan clutter dengan 300 iterasi. Hal ini bertujuan utnuk membandingkan performansi RRT* *goal biasing* dan RRT* g*aussian sampling*. Data yang diambil dalam pengujian tersebut yaitu waktu komputasi, jumlah node, dan jarak jalur. Berikut **Gambar 8.** ilustrasi pada saat proses pencarian jalur menggunakan *goal biasing sampling* pada lingkungan *narrow*.



Gambar 8. (a) Proses Pencarian Jalur Goal Biasing Lingkungan Narrow (b) *Final path* Goal Biasing Lingkungan Narrow

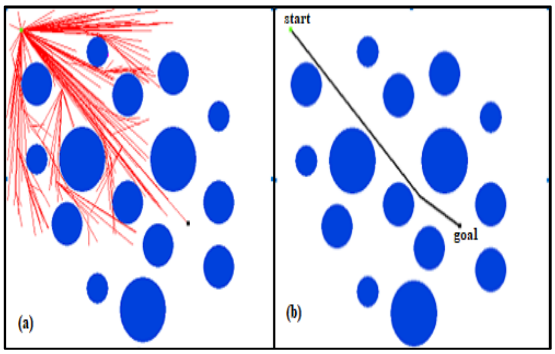
Berdasarkan **Gambar 8.** pada gambar (a) merupakan algoritma RRT* *goal biasing* yang sedang menyusun pencarian jalur pada lingkungan *narrow*, karakteristik dari metode *goal biasing* yaitu akan mengeluarkan node random secara acak dan sesekali akan memunculkan node random ke titik tujuan. Dilihat dari gambar tersebut pohon pecarian menyebar ke segala arah. Pada **Gambar 8.** (b) merupakan *final path* yang tercipta dari simulasi percobaan yang telah dilakukan pada lingkungan *narrow* selama 300 iterasi. **Gambar 8.** (a) terlihat bahwa cabang pada pohon pencarian menyebar kesegala arah. Hal ini disebabkan karena metoda *goal biasing sampling* akan mengambil sampel acak pada ruang pencarian dan sesekali mengambil random ke titik tujuan sehingga beberapa cabang bergerak ke segala arah secara random. Hal ini menyebabkan waktu komputasi awal menjadi lebih lama. Berikut **Gambar 9.** merupakan gambar proses pencarian node goal dan gambar *final path* algoritma RRT* *gaussian* di lingkungan *narrow*.



Gambar 9. (a) Proses Pencarian Jalur *Gaussian* Lingkungan Narrow (b) *Final path Gaussian* Lingkungan Narrow

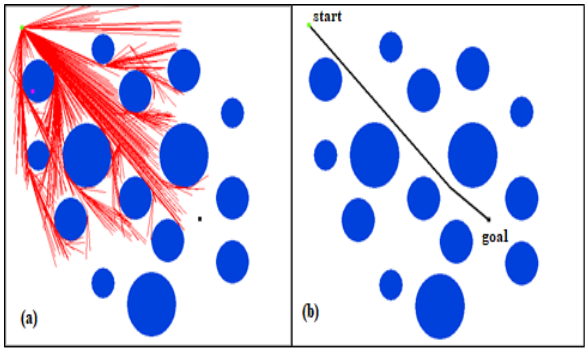
Berdasarkan **Gambar 9.** pada gambar (a) merupakan algoritma RRT* *gaussian* yang sedang menyusun pencarian jalur pada lingkungan *narrow*, karakteristik dari metode *gaussian* yaitu akan mengeluarkan node random yang mengarah mendekati *obstacle*. Terlihat pada gambar (a) pohon pencarian menyebar ke arah *obstacle*. Pada **Gambar 9.** (b) merupakan *final path* yang tercipta dari simulasi percobaan yang telah dilakukan pada lingkungan *narrow* selama 300 *iterasi*. *Final path* akan tercipta dari node start sampai ke node tujuan. proses pencarian jalur g*aussian sampling* pada lingkungan *narrow* memiliki karakteristik melakukan *random* ke arah *obstacle*. Sehingga dapat menemukan jalur yang optimal dengan lebih baik. Berikut **Gambar 10.** merupakan gambar proses pencarian node goal dan gambar *final path*

algoritma RRT* *goal biasing* di lingkungan *clutter*.



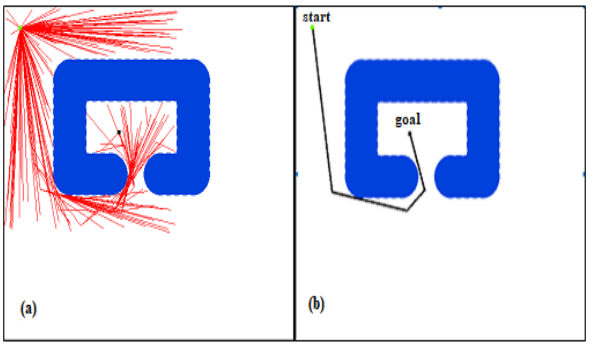
Gambar 10. (a) Proses Pencarian Jalur Goal Biasing Lingkungan Clutter (b) *Final path* Goal Biasing Lingkungan Clutter

Berdasarkan **Gambar 10.** pada gambar (a) merupakan algoritma RRT* *goal biasing* yang sedang menyusun pencarian jalur pada lingkungan *clutter*, karakteristik dari metode *goal biasing* yaitu akan mengeluarkan node random secara acak dan sesekali akan memunculkan node random ke titik tujuan. Pada **Gambar 10.** (b) merupakan *final path* yang tercipta dari simulasi percobaan yang telah dilakukan pada lingkungan *narrow* selama 300 *iterasi*. Metode *goal biasing* akan membutuhkan jumlah node yang lebih sedikit dibandingkan **Gambar 11.** yang merupakan metode g*aussian sampling*. Karena pada lingkungan *clutter* metode g*aussian sampling* akan menumbuhkan banyak node mendekati *obstacle* sampai ditemukannya node tujuan.



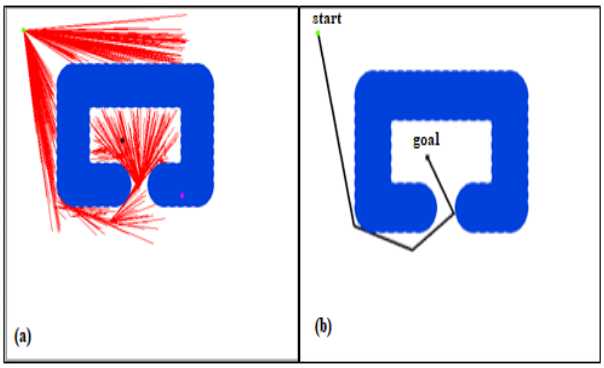
Gambar 11. (a) Proses Pencarian Jalur *Gaussian* Lingkungan Clutter (b) *Final path* *Gaussian* Lingkungan Clutter

Berdasarkan **Gambar 11.** pada gambar (a) merupakan algoritma RRT* *gaussian* yang sedang menyusun pencarian jalur pada lingkungan *clutter*, karakteristik dari metode *gaussian* yaitu akan mengeluarkan node random yang mengarah mendekati *obstacle*. Terlihat pada gambar (a) pohon pencarian menyebar ke arah *obstacle*, karena node random yang harus ditumbuhkan mendekati *obstacle* maka jumlah node yang dibutuhkan pada lingkungan *clutter* mencapai 799 node. Pada **Gambar 11.** (b) merupakan *final path* yang tercipta dari simulasi percobaan yang telah dilakukan pada lingkungan *narrow* selama 300 *iterasi*. *Final path* akan tercipta dari node start sampai ke node tujuan. Berikut **Gambar 12.** merupakan gambar proses pencarian node goal dan gambar *final path* algoritma RRT* g*oal biasing* di lingkungan *trap*.



Gambar 12. (a) Proses Pencarian Jalur *Goal Biasing* di Lingkungan *Trap* (b) *Final path* *Gaussian* di Lingkungan *Trap*

Berdasarkan **Gambar 12.** pada gambar (a) merupakan algoritma RRT* *goal biasing* yang sedang menyusun pencarian jalur pada lingkungan *trap*, karakteristik dari metode *goal biasing* yaitu akan mengeluarkan node random secara acak dan sesekali akan memunculkan node random ke titik tujuan. Pada **Gambar 12.** (b) merupakan *final path* yang tercipta dari simulasi percobaan yang telah dilakukan pada lingkungan *trap* selama 300 *iterasi*. *Final path* akan tercipta dari node start sampai ke node tujuan. Berikut **Gambar 13.** merupakan gambar proses pencarian node goal dan gambar *final path* algoritma RRT* *gaussian* di lingkungan *trap*.



Gambar 13. (a) Proses Pencarian Jalur *Gaussian* Lingkungan *Trap* (b) *Final path* *Gaussian* Lingkungan *Trap*

Berdasarkan **Gambar 13.** pada gambar (a) merupakan algoritma RRT* *gaussian* yang sedang menyusun pencarian jalur pada lingkungan *trap*, karakteristik dari metode *gaussian* yaitu akan mengeluarkan node random

yang mengarah mendekati *obstacle*. Terlihat pada gambar (a) pohon pencarian menyebar ke arah *obstacle* sampai pohon pencarian tersebut menemukan node goal atau titik tujuan. Pada **Gambar 13.** (b) merupakan *final path* yang tercipta dari simulasi percobaan yang telah dilakukan pada lingkungan *trap* selama 300 *iterasi*. *Final path* akan tercipta dari node start sampai ke node tujuan. Terlihat karakteristik dari metode g*aussian sampling* yang akan menumbuhkan random node di dekat *obstacle*. Pada pohon pencarian terlihat langsung mengarah pada obstacle, node random akan terus ditumbuhkan dekat dengan *obstacle* sampai menemukan node tujian dan iterasi selesai. Metode goal biasing dapat meningkatkan efisiensi waktu pada saat melakukan pencarian jalur untuk menemukan jalur awal [14]. Metode gaussian sampling dapat meningkatkan tingkat konvergensi, untuk pengambilan sampel praktis yang lebih baik [18]. Untuk memberikan pemahaman lebih lanjut kepada pembaca mengenai karakteristik performansi dari masing-masing algoritma, maka pada **Tabel I – Tabel IX** disajikan rincian performansi keluaran rata-rata, minimal (terbaik) dan maksimal (terburuk) dari setiap algoritma yang diujikan pada lingkungan *narrow*, *clutter*, dan *trap*. Parameter yang diambil dari pengujian ini adalaha biaya jarak, waktu, dan jumlah node yang dibutuhkan untuk mencapai titik goal.

Tabel X. Perbandingan biaya jarak pada lingkungan *narrow*

	Path Cost	
	RRT* *Goal Biasing*	RRT* *Gaussian*
Maksimum	9,5	9,3
Minimum	7,9	7,5
Rata-rata	8,3	8,1

Tabel XI. Perbandingan biaya jarak pada lingkungan *clutter*

	Path Cost	
	RRT* *Goal Biasing*	RRT* *Gaussian*
Maksimum	12,2	12,2
Minimum	11,3	11,3
Rata-rata	11,6	11,6

Tabel XII. Perbandingan biaya jarak pada lingkungan *trap*

	Path Cost	
	RRT* *Goal Biasing*	RRT* *Gaussian*
Maksimum	17,1	15,3
Minimum	13	13
Rata-rata	14,3	13,5

Tabel XIII. Perbandingan jumlah node pada lingkungan *narrow*

	Jumlah Node	
	RRT* *Goal Biasing*	RRT* *Gaussian*
Maksimum	242	531
Minimum	192	192
Rata-rata	219,7	372

Tabel XIV. Perbandingan jumlah node pada lingkungan *clutter*

	Jumlah Node	
	RRT* *Goal Biasing*	RRT* *Gaussian*
Maksimum	233	799
Minimum	202	507
Rata-rata	222,1	642,85

Tabel XV. Perbandingan jumlah node pada lingkungan *trap*

	Jumlah Node	
	RRT* *Goal Biasing*	RRT* *Gaussian*
Maksimum	222	573
Minimum	165	139
Rata-rata	199,45	345

Tabel XVI. Perbandingan waktu (detik) capai pada lingkungan *narrow*

	Waktu (detik)	
	RRT* *Goal Biasing*	RRT* *Gaussian*
Maksimum	30,2	37,2
Minimum	30	30
Rata-rata	30,065	32,955

Tabel XVII. Perbandingan waktu (detik) capai pada lingkungan *clutter*

	Waktu (detik)	
	RRT* *Goal Biasing*	RRT* *Gaussian*
Maksimum	30,1	37,8
Minimum	30	30
Rata-rata	30,05	33,295

Tabel XVIII. Perbandingan waktu (detik) capai pada lingkungan *trap*

	Waktu (detik)	
	RRT* *Goal Biasing*	RRT* *Gaussian*
Maksimum	30,1	32,5
Minimum	30	30
Rata-rata	30,045	30,49

Pada **Tabel I, II,** dan **III** dapat terlihat bahwa jalur terbaik dihasilkan oleh algoritma RRT* g*aussian sampling* (nilai minimum), mengenai rentang kualitas/panjang jalur yang dihasilkan oleh algoritma RRT* *goal biasing* dan RRT* g*aussian* memang masih saling mendekati. Tetapi kekurangan dari algoritma RRT* g*aussian sampling* terdapat pada jumlah node (nilai maksimum) seperti terlihat pada **Tabel IV**, **V**, dan **XI**. Dapat terlihat bahwa pada perbandingan waktu (nilai maksimum) di **Tabel VII**, **VIII**, dan **IX** RRT* *goal biasing* sedikit lebih unggul. Hal ini dikarenakan RRT* *goal biasing* dapat langsung menumbuhkan node random yang mengarah langsung ke node tujuan. Manfaat dari penelitia ini yaitu dapat menentukan algoritma pencarian jalur salah satunya untuk bidang robotika dengan menerapkan algoritma RRT* *goal biasing* dapat membuat pencarian awal node tujuan menjadi lebih cepat.

IV. KESIMPULAN

Penggunaan metoda *gaussian sampling* pada algoritma RRT* dapat memberikan solusi yang lebih optimal untuk jarak yang dibutuhkan sampai ke titik tujuan, hal ini diperkuat dengan adanya informasi jarak (nilai minimum) 7,2 untuk lingkungan *narrow*, dan 11,2 untuk lingkungan *trap.* Tetapi kekurangan dari algoritma RRT* g*aussian sampling* terdapat pada jumlah node (nilai maksimum) pada lingkungan *narrow, clatter*, dan *trap* dengan nilai ; 531, 799, 573. Hal ini dikarenakan metode g*aussian sampling* akan menumbuhkan banyak node random di dekat *obstacle,* terlihat pada pengujian di lingkungan *clutter*. Jumlah node (maksimum) yang dibutuhkan metode g*aussian sampling* sebesar 799**.**Dari segi program, dapat dikembangkan kembali untuk metode sampling RRT* pada pemrograman LabVIEW. Sehingga dapat menciptakan metode sampling yang lebih optimal dari metode goal biasing dan gaussian.

DAFTAR PUSTAKA

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

[2] G. Carbone dan F. Gomez-Bravo, "Erratum to: Motion and Operation Planning of Robotic Systems," hal. E1–E1, 2015

[3] B. Fu *et al.*, "An improved A* algorithm for the industrial robot path planning with high success rate and short length," *Rob. Auton. Syst.*, vol. 106, hal. 26–37, 2018

[4] J. D. Gammell, S. S. Srinivasa, dan T. D. Barfoot, "Informed RRT*," *2014 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, hal. 2997–3004, 2014.

[5] M. Aria, "Algoritma Perencanaan Jalur Kendaraan Otonom berbasis Hibridisasi Algoritma BFS dan Path Smoothing," *Telekontran J. Ilm. Telekomun. Kendali dan Elektron. Terap.*, vol. 8, no. 1, hal. 13–22, 2020

[6] J. Wang, W. Chi, C. Li, C. Wang, dan M. Q. H. Meng, "Neural RRT∗: Learning-Based Optimal Path Planning," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 4, hal. 1748–1758, 2020

[7] Y. Gao, J. Liu, M. Q. Hu, H. Xu, K. P. Li, dan H. Hu, "A New Path Evaluation Method for Path Planning with Localizability," *IEEE Access*, vol. 7, hal. 162583–162597, 2019

[8] T. Oral dan F. Polat, "MOD∗ lite: An incremental path planning algorithm taking care of multiple objectives," *IEEE Trans. Cybern.*, vol. 46, no. 1, hal. 245–257, 2016

[9] F. Duchon *et al.*, "Path planning with modified A star algorithm for a mobile robot," *Procedia Eng.*, vol. 96, hal. 59–69, 2014

[10] S. Karaman dan E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," *Proc. IEEE Conf. Decis. Control*, hal. 7681–7687, 2010

[11] M. Lewis-Beck, A. Bryman, dan T. Futing Liao, "Probabilistic," *The SAGE Encyclopedia of Social Science Research Methods*. 2012

[12] L. G. D. O. Véras, F. L. L. Medeiros, dan L. N. F. Guimarães, "Rapidly exploring Random Tree* with a sampling method based on Sukharev grids and convex vertices of safety hulls of obstacles," *Int. J. Adv. Robot. Syst.*, vol. 16, no. 1, hal. 1–19, 2019

[13] S. Karaman dan E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Rob. Res.*, vol. 30, no. 7, hal. 846–894, 2011

[14] B. Akgun dan M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," *IEEE Int. Conf. Intell. Robot. Syst.*, hal. 2640–2645, 2011

[15] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, dan A. Paques, "PythonRobotics: a Python code collection of robotics algorithms," hal. 1–8, 2018

[16] N. Vahrenkamp dan T. Asfour, "Simox : A Simulation and Motion Planning Toolbox for C ++," hal. 1–34, 2011.

[17] E. S. Nugraha, "Simulator Edukatif untuk Pembelajaran Algoritma Rapidly-exploring Random Tree (RRT)," *Telekontran J. Ilm. Telekomun. Kendali dan Elektron. Terap.*, vol. 7, no. 2, 2020

[18] Li Yuncheng dan Shao Jie, "A revised Gaussian distribution sampling scheme based on RRT* algorithms in robot motion planning," 2017 3rd International Conference on Control, Automation and Robotics (ICCAR), hal. 22-26, 2017