TELEKONTRAN, VOL. 10, NO. 1, APRIL 2022
DOI : 10.34010/telekontran.v10i1.7796
p-ISSN : 2303 – 2901
e-ISSN : 2654 – 7384

LabVIEW Libraries untuk Algoritma Perencanaan Jalur Robotik

LabVIEW Libraries for Robotic Path Planning Algorithms

Muhammad Aria Rajasa Pohan
Teknik Elektro, Universitas Komputer Indonesia
Jalan Dipati Ukur No 112 – 116 Bandung, Indonesia
Email : muhammad.aria@email.unikom.ac.id

Abstrak – Makalah ini mempresentasikan pengembangan *library* yang dapat digunakan untuk membuat algoritma perencanaan jalur untuk sistem robotik. *Laboratory Virtual Instrumentation Engineering Workbench* (LabVIEW) digunakan untuk membuat *library* algoritma perencanaan jalur ini. Algoritma perencanaan jalur yang dapat dibuat menggunakan *library* ini adalah *Rapidly-exploring Random Tree* (RRT), RRT*, RRT*-*Connect*, *Informed* RRT*, *Informed* RRT*-*Connect*, *Probabilistic Roadmap* (PRM), A*, Dijkstra, *Particle Swarm Optimization* (PSO), dan RRT-*Ant Colony System* (ACS). Kelebihan dari *library* ini adalah setiap subproses dapat divisualisasikan, waktu komputasi yang cepat, tersedia *tools* untuk analisis, skenario pengujian dapat dibuat secara fleksibel, serta parameter algoritma dapat dengan mudah diubah untuk melihat bagaimana pengaruhnya terhadap kinerja keluaran algoritma. Kinerja dari *library* ini dievaluasi dengan membandingkan implementasi *library* pada skenario benchmark dari publikasi peneliti lain. Karena waktu komputasi merupakan faktor penting dalam algoritma perencanaan jalur, pengembangan perangkat lunak *library* ini telah disederhanakan untuk mencapai waktu komputasi sistem sesingkat mungkin. Dengan menjadikan setiap SubVI di *library* ini sebagai *subroutine*, *overhead* program subVI telah berhasil dikurangi. Akibatnya, waktu komputasi *library* ini berkurang. Hasil pengujian meggunakan beberapa kasus benchmark menunjukkan bahwa *library* yang diusulkan memiliki waktu komputasi lebih cepat daripada *library* pembanding. Implementasi *library* pada sistem robotik diuji menggunakan LabVIEW Robotics Simulator. Kami juga memberikan contoh beberapa pengembangan yang dapat dilakukan dengan menggunakan *library* ini. Dengan menggunakan *library* ini, diharapkan mahasiswa lebih memahami dan mempermudah dalam pengembangan algoritma perencanaan jalur untuk sistem robotik.

Kata Kunci : Perencanaan jalur, LabVIEW *library*, *rapidly-exploring random* tree, waktu komputasi, *analysis tools*

Abstract *- This paper presents the development of a library that can be used to create path planning algorithms for robotic systems. Laboratory Virtual Instrumentation Engineering Workbench (LabVIEW) was used to create this library of path planning algorithms. Path planning algorithms that can be created using this library are Rapidly-exploring Random Tree (RRT), RRT*, RRT*-Connect, Informed RRT*, Informed RRT*-Connect, Probabilistic Roadmap (PRM), A*, Dijkstra, Particle Swarm Optimization (PSO), and the RRT-Ant Colony System (ACS). The performance of this library is evaluated by comparing the implementation of the library in benchmark scenarios from other researchers' publications. The advantages of this library are that each subprocess can be visualized, computation time is fast, tools are available for analysis, test scenarios can be created flexibly, and algorithm parameters can be easily changed to see how it affects the algorithm's output performance. Since computation time is an essential factor in path planning algorithms, the software development of this library has been streamlined to achieve the shortest possible system compute time. By making every SubVI in this library a subroutine, the overhead of subVI programming has been reduced. As a result, the computation time of this library is reduced.The test results show that the proposed library has a faster computation time than the comparison library. The library implementation on the robotic system was tested using the LabVIEW Robotics Simulator. We also provide examples of some developments that can be done using this library. By using this library, it is hoped that students will understand and make it easier to develop path planning algorithms for robotic systems.*

Keywords *: Path planning, LabVIEW library, Rapidly-exploring Random Tree, computation time, analysis tools*

I. PENDAHULUAN

Algoritma *path planning* atau perencanaan jalur adalah bidang penelitian mendasar pada robotika [1]. Algoritma perencanaan jalur dapat dibagi menjadi beberapa kelas, diantaranya: metode *graph search* dan metode *incremental search* [2, 3]. Contoh metode *graph search* adalah Dijkstra [4] dan A* [5]. Kelemahan metode *graph search* adalah optimalitas algoritma hanya dapat dijamin jika resolusi diskritisasi ruang keadaan cukup baik. Namun, resolusi yang baik mungkin sulit dicapai dalam ruang berdimensi tinggi [6]. Algoritma *Probabilistic Roadmaps* (PRM) [7] juga dikategorikan sebagai metode *graph search* oleh Paden [2]. Metode *incremental search* menggunakan metode berbasis sampel agar tidak memerlukan diskritisasi ruang keadaan. Hal ini membuat metode *incremental search* lebih efektif pada ruang berdimensi tinggi [8]. Contoh metode *incremental search* adalah *Rapidly-exploring Random Tree* (RRT) [9] dan RRT* [10]. Algoritma RRT dapat membuat jalur dengan cepat, tetapi solusi yang dihasilkan suboptimal [10]. Algoritma RRT* dapat membuat jalur yang optimal asimtotik [11], tetapi tingkat konvergensi RRT* lambat [12]. Banyak metode telah dikembangkan untuk meningkatkan kualitas algoritma RRT dan RRT*. Kuffner dan Lavalle memperkenalkan versi dual-tree dari RRT, yaitu RRT-*Connect* [13]. Klem dkk. memperkenalkan algoritma RRT*-*Connect* [14], yang menggabungkan RRT-*Connect* dengan RRT*. Gammel dkk memperkenalkan algoritma *Informed* RRT* [8], yang menggunakan sampling terinformasi pada RRT* setelah solusi pertama ditemukan. Prosedur pengambilan sampel dilakukan di area elips yang mengelilingi node awal dan tujuan. Area sampling akan semakin kecil dan lebih fokus jika jalur yang menghubungkan node awal ke node tujuan menjadi lebih pendek. Mashayekhi dkk. [15] memperkenalkan algoritma *Informed* RRT*-*Connect* yang berperilaku seperti RRT*-*Connect* hingga jalur pertama ditemukan. Setelah solusi pertama ditemukan, area pengambilan sampel dari *Informed* RRT*-*Connect* dibatasi seperti pada *Informed* RRT*. Beberapa peneliti juga mengembangkan algoritma perencanaan jalur berbasis *swarm intelligence. Pohan dkk. [16] memperkenalkan algoritma perencanaan jalur menggunakan hibridisasi algoritma RRT dan Ant Colony System (ACS). Algoritma tersebut dinamakan RRT-ACS. Li dkk. [17] menggunakan Particle Swarm Optimization (PSO) sebagai dasar untuk melakukan perencanaan jalur.*

Berbagai peneliti telah melakukan publikasi dengan subyek pembuatan toolbox algoritma perencanaan jalur. Sakai mempublikasikan toolbox perencanaan jalur yang diberi nama PythonRobotics [18]. Beberapa algoritma perencanaan jalur yang dapat dibuat menggunakan PythonRobotics adalah Dijkstra, PRM, A*, RRT, RRT*, dan *Informed* RRT*. Namun, PythonRobotics yang dibuat Sakai hanya dapat digunakan pada program python. Sedangkan program python memiliki waktu komputasi yang lama [19]. Lingkungan pengujian yang dapat digunakan untuk PythonRobotics juga hanya *obstacle* berbentuk lingkaran. Vahrenkamp mempublikasikan toolbox perencanaan jalur yang ditulis dalam C+ [20]. Tetapi algoritma perencanaan jalur yang tersedia pada toolbox Vahrenkamp sebagian besar hanya didasarkan pada algoritma RRT. Corke mempublikasikan toolbox robotic yang ditulis dalam bahasa pemrograman MATLAB [21]. Beberapa algoritma perencanaan jalur yang dapat dibuat menggunakan toolbox robotik Corke adalah A*, D*, PRM, dan RRT. Variasi dari algoritma RRT (seperti RRT*, *Informed* RRT*, maupun RRT-ACS) tidak terdapat pada toolbox robotik Corke. Padahal, salah satu algoritma yang paling banyak digunakan untuk perencanaan jalur adalah algoritma RRT* [22]. Nugraha telah membuat simulator edukatif untuk pembelajaran algoritma RRT menggunakan LabVIEW [23]. Kumar dan Goswami [24] melaporkan bahwa LabVIEW memiliki waktu komputasi yang cepat. Waktu komputasi toolbox RRT yang dibuat Nugraha juga telah diminimalkan dengan mengurangi *overhead* program subVI [25]. Tetapi, algoritma perencanaan jalur yang dapat dibuat menggunakan toolbox Nugraha hanya algoritma RRT dan RRT* saja.

Berdasarkan studi *state of art* diatas, terdapat beberapa kekurangan dalam mengembangkan algoritma *library* path planning. Beberapa kekurangan ini diantaranya toolbox dengan waktu komputasi yang lambat, tidak ada alat untuk membantu analisis, skenario pengujian yang tidak fleksibel (hanya dapat melibatkan rintangan berbentuk lingkaran atau persegi), dan kurangnya variasi dari algoritma RRT* yang ada. Sedangkan, algoritma RRT* tergolong algoritma perencanaan jalur yang paling populer saat ini [22]. Penelitian dan makalah ini bertujuan untuk mengembangkan *library* algoritma perencanaan jalur untuk sistem robotik. *Library* algoritma perencanaan jalur ini dibuat menggunakan *Laboratory Virtual Instrumentation Engineering Workbench* (LabVIEW). Dengan menggunakan *library* ini,

dapat dibuat algoritma perencanaan jalur seperti RRT, RRT*, RRT*-*Connect*, *Informed* RRT*, *Informed* RRT*-*Connect*, PRM, A*, Dijkstra, Particle Swarm Optimization PSO, dan RRT-ACS. Keunggulan dari *library* ini antara lain adalah kemampuan untuk memvisualisasikan setiap subproses, waktu komputasi yang cepat, tersedianya alat untuk membantu analisis, kemampuan untuk membuat skenario pengujian yang fleksibel, dan kemampuan untuk dengan mudah mengubah parameter algoritma untuk melihat bagaimana pengaruhnya terhadap kinerja keluaran algoritma. Kinerja *library* ini diukur dengan membandingkan implementasinya dalam skenario benchmark dari publikasi peneliti lain. Hasil pengujian menggunakan beberapa kasus benchmark menunjukkan bahwa *library* yang diusulkan memiliki komputasi yang lebih cepat daripada *library* pembanding. Simulator Robotika LabVIEW digunakan untuk menguji implementasi *library* pada sistem robotik. Kami juga menunjukkan beberapa perkembangan yang dapat dilakukan dengan *library* ini. Diharapkan dengan menggunakan *library* ini, mahasiswa akan lebih memahami dan lebih mudah dalam mengembangkan algoritma perencanaan jalur untuk sistem robotik.

II. METODE

Algoritma perencanaan jalur yang dapat dibuat menggunakan *library* ini adalah berbagai variasi algoritma RRT, yaitu RRT*, RRT*-*Connect*, *Informed* RRT*, *Informed* RRT*-*Connect*, dan RRT-ACS. Karena algoritma RRT* dan variannya merupakan algoritma perencanaan jalur yang paling banyak digunakan saat ini [22], maka penjelasan tentang algoritma perencanaan jalur pada Bab II dibatasi pada algoritma RRT dan variannya. *Library* ini masih berisi algoritma PRM, A*, Dijkstra, dan PSO, namun penjelasannya tidak disertakan dalam tulisan ini. Pembaca dapat merujuk ke referensi yang terkait langsung dengan algoritma tersebut sebagaimana yang dijelaskan dalam Bab I.

Algoritma RRT yang dapat dibuat menggunakan *library* ini ditunjukkan pada **Gambar 1**. Pertama-tama dilakukan inisialisasi pohon pencarian yang disimbolkan sebagai T. Lalu pada T ditambahkan posisi node awal, yaitu q_{init}. Hal ini akan membuat proses pencarian jalur akan berawal dari posisi q_{init}. Selanjutnya algoritma akan melakukan *looping* untuk mengembangkan pohon pencarian T.

Pada proses *looping*, fungsi pertama yang dilakukan adalah RandomSample. Pada proses RandomSample, dilakukan pengambilan sampel acak, yang dinamakan q_{rand} (baris 5). Setelah diperoleh nilai q_{rand}, proses selanjutnya adalah mencari node-node pada T yang berjarak kurang dari suatu nilai threshold ke q_{rand}. Node-node tersebut dinamakan sebagai Q_{near} (baris 6). Lalu dipilih node pada Q_{near} yang berjarak terdekat dengan q_{rand}. Node dengan jarak terdekat tersebut dinamakan sebagai $q_{nearest}$ (baris 7). Selanjutnya akan dibuat suatu cabang baru dari node $q_{nearest}$ yang menuju ke q_{rand} dengan jarak Δq dari $q_{nearest}$. Node baru tersebut dinamakan sebagai q_{new} (baris 8). Selanjutnya akan dilakukan pemeriksaan apakah ada hambatan diantara $q_{nearest}$ dengan q_{new} (baris 9). Jika diantara $q_{nearest}$ dengan q_{new} tidak terdapat hambatan, maka node q_{new} akan dimasukkan ke T (baris 10). Selanjutnya akan diperiksa apakah ada node pada T saat ini yang berjarak cukup dekat dengan node tujuan atau q_{goal} (baris 12). Jika terdapat node yang berjarak cukup dekat dengan node tujuan, maka akan dibangun rute X^{bs}, yaitu rute yang menghubungkan node awal dengan node tujuan (baris 13). Proses ini akan berulang sehingga dipenuhi kondisi berhenti. Proses *looping* dapat memiliki beberapa syarat yang bervariasi untuk berhenti, misalnya jika node tujuan sudah tercapai atau jika jumlah iterasi maksimal sudah terpenuhi.

Algoritma 1: $X^{bs} \leftarrow \mathrm{RRT}(q_{init}, q_{goal}, map)$

```
1:  % ========== Initialization
2:  T ← InitializeTree()
3:  T ← InsertNode(∅, q_init, T)
4:  while termination condition not met do
5:      q_rand ← RandomSample(k)
6:      Q_near ← NearNodes(T, q_rand)
7:      q_nearest ← NearestNeighbor(q_rand, Q_near, T)
8:      q_new ← Steer (q_nearest, q_rand, Δq)
9:      if Obstaclefree(q_nearest, q_new) then
10:         T ← InsertNode(q_min, q_new, T)
11:     end if
12:     if CanConnected(q_goal, T) then
13:         X^bs ← UpdateBestPath(T, q_goal)
14:     end if
15: end while
```

Gambar 1. Algoritma RRT. Nomor baris akan digunakan untuk menjelaskan *library* LabVIEW yang dibuat.

Algoritma perencanaan jalur RRT* diperlihatkan pada **Gambar 2**. Algoritma RRT* merupakan algoritma RRT dengan penambahan proses *choose parent* (baris 10-11) dan *rewire* (baris 13) [10]. Pada proses *choose parent*, pertama-tama akan dipilih node-node pada T yang berjarak kurang dari suatu nilai threshold ke q_{rand}. Node-node tersebut dinamakan sebagai Q_{near}

(baris 10). Lalu pada node-node di Q_{near} akan dicari node yang jika terhubung dengan q_{new} akan berakibat jalur diantara node awal dan q_{new} akan menjadi lebih pendek. Node tersebut dinamakan sebagai q_{min} (baris 11), dan akan dijadikan node parent dari q_{new}. Adapun untuk proses Rewire, akan dilakukan pencarian node-node pada Q_{near} yang jika node tersebut dihubungkan ke q_{new}, maka akan diperoleh jalur yang lebih dekat dari node awal ke node tersebut.. Maka node q_{new} akan dijadikan sebagai parent baru dari node tersebut.

Algoritma 2: $X^{bs} \leftarrow$RRT*$(q_{init}, q_{goal}, map)$

```
1:  % ========== Initialization
2:  T ← InitializeTree()
3:  T ← InsertNode(∅, q_init, T)
4:  while termination condition not met do
5:      q_rand ← RandomSample(k)
6:      Q_near ← NearNodes(T, q_rand)
7:      q_nearest ← NearestNeighbor(q_rand, Q_near, T)
8:      q_new ← Steer (q_nearest, q_rand, Δq)
9:      if Obstaclefree(q_nearest, q_new) then
10:         Q_near ← Near(T, q_new)
11:         q_min ← ChooseParent(q_new, Q_near, q_nearest)
12:         T ← InsertNode(q_min, q_new, T)
13:         T ← Rewire(T, Q_near, q_min)
14:     end if
15:     if CanConnected(q_goal, T) then
16:         X^bs ← UpdateBestPath(T, q_goal)
17:     end if
18: end while
```

Gambar 2. Algoritma RRT*. Nomor baris akan digunakan untuk menjelaskan *library* LabVIEW yang dibuat.

Algoritma perencanaan jalur RRT*-Connect diperlihatkan pada **Gambar 3.** Algoritma ini adalah algoritma RRT* dengan dua pohon pencarian. Berbeda dengan algoritma RRT dan RRT* yang hanya memiliki satu pohon pencarian yaitu T, maka pada algoritma RRT*-Connect memiliki dua pohon pencarian yaitu T_A dan T_B. Pada **Gambar 3** dapat terlihat bahwa proses dari algoritma RRT*-Connect ini mirip dengan proses pada algoritma RRT* yang ditunjukkan pada **Gambar 2**, kecuali terdapat dua pohon pencarian yang digunakan yaitu T_A dan T_B.

Algoritma *Informed* RRT* ditunjukkan pada **Gambar 4.** Setelah memperoleh solusi pertama, algoritma ini melakukan pengampilan sample berdasarkan informasi pada area berbentuk elips yang dibentuk diantara node awal dan node akhir. Proses *informed sampling* ini pertama-tama dilakukan dengan menghitung jarak terbaik antara node awal dengan node tujuan saat ini (baris 5), yaitu disebut c_{best}. Kemudian berdasarkan nilai c_{best} tersebut, akan dibuat suatu area elipsoid yang melingkupi node tujuan dan node awal. Luas area elipsoid akan dipengaruhi oleh nilai c_{best}. Maka proses sampling nilai q_{rand} hanya dilakukan pada area elipsoide tersebut saja (baris 6).

Adapun **Gambar 5** menunjukkan algoritma *Informed RRT*-Connect*. Algoritma *Informed RRT*-Connect* adalah versi dua pohon pencarian dari *Informed RRT**. Proses *InformedSample dan CalculateShortestPathLength* pada **Gambar 5** adalah sama dengan proses pada **Gambar 4** yang telah dijelaskan sebelumnya. Adapun proses *PruneTree* adalah proses penghilangan cabang-cabang pada T yang berada diluar area elipsoid pada proses informed sampling. Proses *PruneTree* akan meningkatkan kecepatan komputasi algoritma dikarenakan node-node yang perlu diperiksa pada T akan dibatasi hanya pada area ellipsoid yang akan semakin mengecil [8, 15].

Algoritma 3: $X^{bs} \leftarrow$RRT*-*Connect*$(q_{init}, q_{goal}, map)$

```
1:  % ========== Initialization
2:  T_A ← InitializeTree()
3:  T_B ← InitializeTree()
4:  T_A ← InsertNode(∅, q_init, T_A)
5:  T_B ← InsertNode(∅, q_goal, T_B)
6:  while termination condition not met do
7:      q_rand ← RandomSample(k)
8:      Q_near ← Near(T_A, q_rand)
9:      q_nearest ← NearestNeighbr(q_rand, Q_near, T_A)
10:     q_new ← Steer (q_nearest, q_rand, Δq)
11:     if Obstaclefree(q_nearest, q_new) then
12:         Q_near ← Near(T_A, q_new)
13:         q_min ← ChoosPrnt(q_new, Q_near, q_nearest)
14:         T_A ← InsertNode(q_min, q_new, T_A)
15:         T_A ← Rewire(T, Q_near, q_min)
16:         if CanConnected(q_new, T_B) then
17:             X^bs ← UpdateBestPath(T_A, T_B)
18:         end if
19:     end if
20:     Q_near ← Near(T_A, q_rand)
21:     q_nearest ← NearestNeighbr(q_rand, Q_near, T_A)
22:     q_new ← Steer (q_nearest, q_rand, Δq)
23:     if Obstaclefree(q_nearest, q_new) then
24:         Q_near ← Near(T_B, q_new)
25:         q_min ← ChoosPrnt(q_new, Q_near, q_nearest)
26:         T_B ← InsertNode(q_min, q_new, T_B)
27:         T_B ← Rewire(T, Q_near, q_min)
28:         if CanConnected(q_new, T_A) then
29:             X^bs ← UpdateBestPath(T_A, T_B)
30:         end if
31:     end if
32: end while
```

Gambar 3. Algoritma RRT*-*Connect*. Nomor baris akan digunakan untuk menjelaskan *library* LabVIEW yang dibuat.

Algorithm 4 : $X^{bs} \leftarrow$*Informed* RRT*$(q_{init}, q_{goal}, map)$

```
1:  % ========== Initialization
2:  T ← InitializeTree()
3:  T ← InsertNode(∅, q_init, T)
4:  while termination condition not met do
5:      c_best ← CalculateShortestPathLength(X^bs)
6:      q_rand ← InformedSample(k)
7:      Q_near ← NearNodes(T, q_rand)
8:      q_nearest ← NearestNeighbor(q_rand, Q_near, T)
9:      q_new ← Steer (q_nearest, q_rand, Δq)
10:     if Obstaclefree(q_nearest, q_new) then
11:         Q_near ← Near(T, q_new)
12:         q_min ← ChoosePrnt(q_new, Q_near, q_nearest)
13:         T ← InsertNode(q_min,q_new, T)
14:         T ← Rewire(T, Q_near, q_min)
15:     end if
16:     if CanConnected(q_goal, T) then
17:         X^bs ← UpdateBestPath(T, q_goal)
18:         PruneTree(T)
19:     end if
20: end while
```

Gambar 4. Algoritma *Informed* RRT*. Nomor baris akan digunakan untuk menjelaskan *library* LabVIEW yang dibuat.

Algoritma 5: $X^{bs} \leftarrow$ *Informed* RRT*-*Connect*$(q_{init}, q_{goal}, map)$

```
1:  % ========== Initialization
2:  T_A ← InitializeTree()
3:  T_B ← InitializeTree()
4:  T_A ← InsertNode(∅, q_init, T_A)
5:  T_B ← InsertNode(∅, q_goal, T_B)
6:  while termination condition not met do
7:      c_best ← CalculateShortestPathLength(X^bs)
8:      q_rand ← InformedSample(k)
9:      Q_near ← Near(T_A, q_rand)
10:     q_nearest ← NearestNeighbr(q_rand, Q_near, T_A)
11:     q_new ← Steer (q_nearest, q_rand, Δq)
12:     if Obstaclefree(q_nearest, q_new) then
13:         Q_near ← Near(T_A, q_new)
14:         q_min ← ChoosPrnt(q_new, Q_near, q_nearest)
15:         T_A ← InsertNode(q_min,q_new, T_A)
16:         T_A ← Rewire(T, Q_near, q_min)
17:         if CanConnected(q_new, T_B) then
18:             X^bs ← UpdateBestPath(T_A, T_B)
19:             PruneTree(T)
20:         end if
21:     end if
22:     Q_near ← Near(T_A, q_rand)
23:     q_nearest ← NearestNeighbr(q_rand, Q_near, T_A)
24:     q_new ← Steer (q_nearest, q_rand, Δq)
25:     if Obstaclefree(q_nearest, q_new) then
26:         Q_near ← Near(T_B, q_new)
27:         q_min ← ChoosPrnt(q_new, Q_near, q_nearest)
28:         T_B ← InsertNode(q_min,q_new, T_B)
29:         T_B ← Rewire(T, Q_near, q_min)
30:         if CanConnected(q_new, T_A) then
31:             X^bs ← UpdateBestPath(T_A, T_B)
32:             PruneTree(T)
33:         end if
34:     end if
35: end while
```

Gambar 5. Algoritma *Informed* RRT*-*Connect*. Nomor baris akan digunakan untuk menjelaskan *library* LabVIEW yang dibuat.

Meskipun algoritma RRT* pada **Gambar 2** mirip dengan yang dibahas oleh Karaman pada [10], perubahan telah dilakukan untuk memungkinkan setiap algoritma yang dibahas dalam bagian metode ini dapat dibandingkan dengan mudah. Demikian pula, algoritma RRT*-*Connect* pada **Gambar 3**, algoritma *Informed* RRT*-*Connect* pada **Gambar 4**, dan algoritma *Informed* RRT*-*Connect* pada **Gambar 5** telah dimodifikasi dibandingkan dengan algoritma terkait yang dibahas oleh Klem [14], Gammel [8,] dan Mashayekhi [15]. Nomor baris pada **Gambar 1** - **Gambar 5** akan digunakan untuk menjelaskan *library* LabVIEW yang dibuat pada Bab III.

Adapun algoritma RRT-ACS dapat dilihat pada paper Pohan [16]. Nomor barus yang digunakan pada algoritma RRT-ACS di paper Pohan akan digunakan untuk menjelaskan *library* LabVIEW yang dibuat pada Bab III.

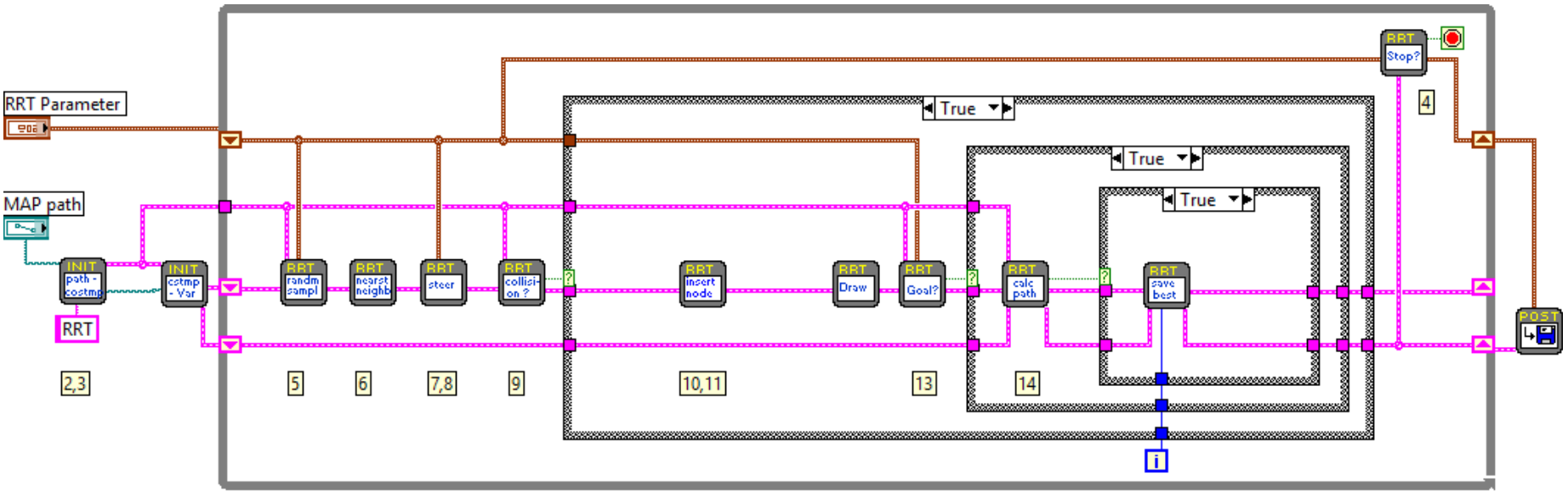
III. HASIL DAN DISKUSI

LabVIEW library untuk perencanaan jalur ini memiliki beberapa keunggulan, yaitu dapat memvisualisasikan setiap subproses, waktu komputasi yang cepat, tersedia *tools* untuk analisis, skenario pengujian dapat dibuat secara fleksibel, serta parameter algoritma dapat dengan mudah diubah untuk melihat bagaimana pengaruhnya terhadap kinerja keluaran algoritma. Maka pembahasan selanjutnya adalah mengenai contoh pembuatan algoritma perencanaan jalur yang dijelaskan pada Bab II, serta mendemonstrasikan keunggulan dari *Library* ini.

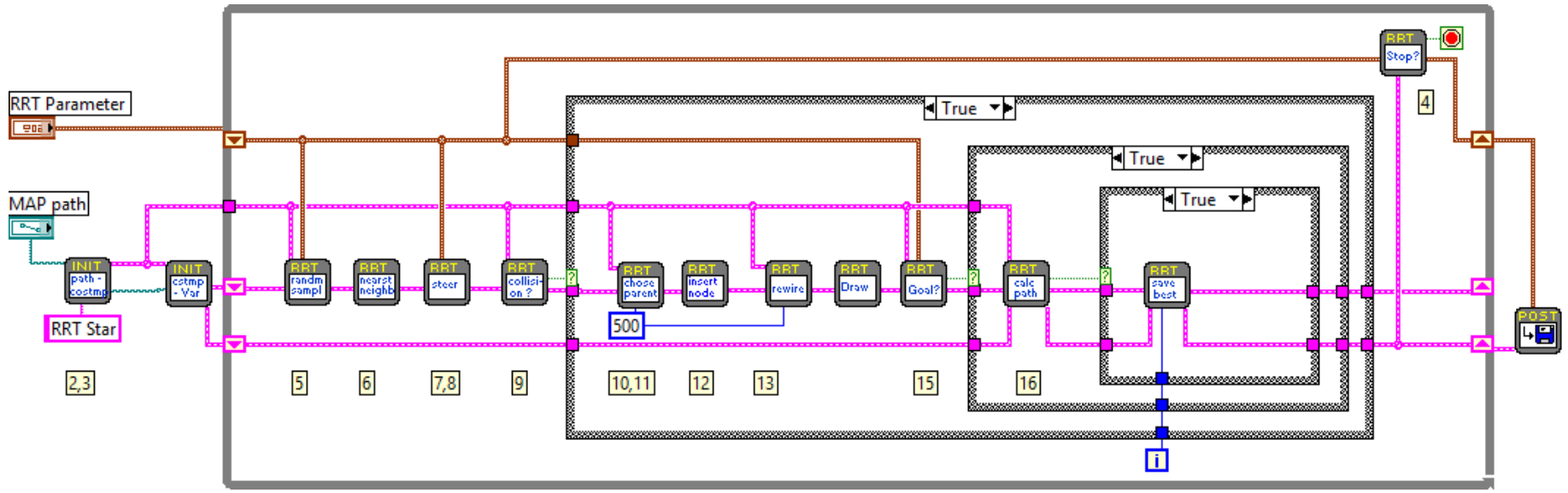
A. Block Diagram LabVIEW

Block diagram LabVIEW dari algoritma RRT yang dibuat dengan *library* ini ditunjukkan pada **Gambar 6**. Nomor dalam kotak kuning menunjukkan proses yang dijelaskan pada Algoritma 1 **Gambar 1**. Pada **Gambar 6**, terdapat beberapa sub-program yang digunakan pada *library* ini, yaitu:

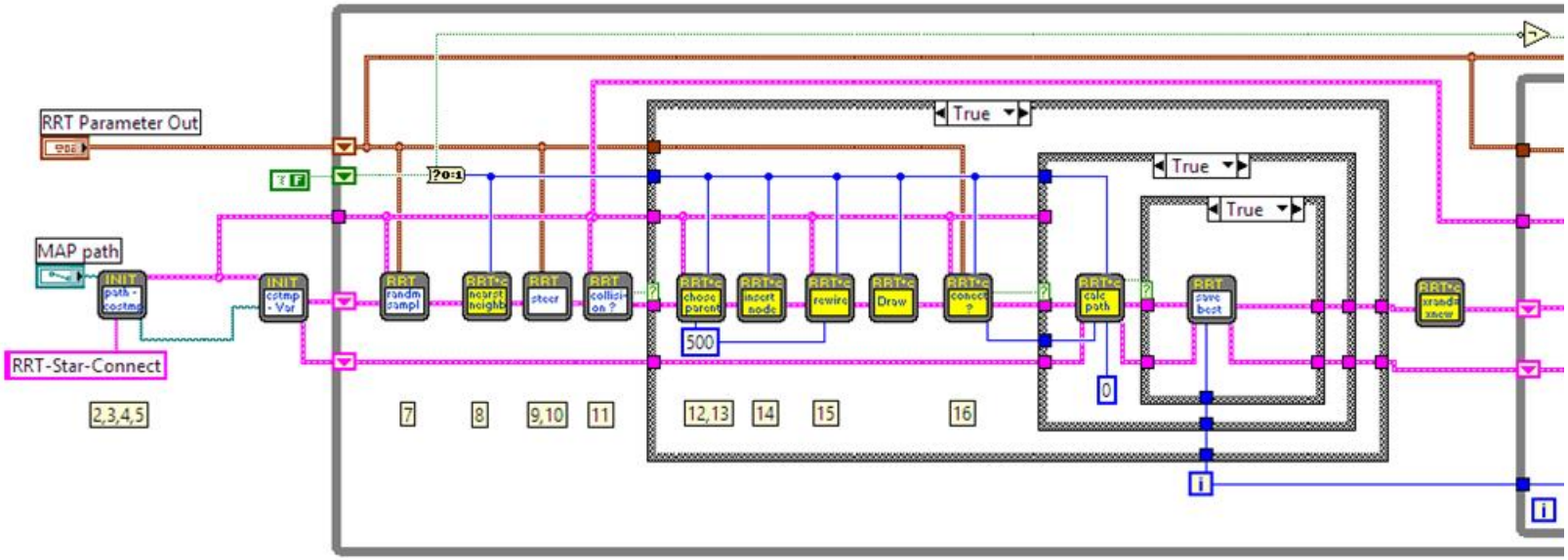
1. “Data Initialization.vi” [nomor 2, 3]
2. “Sampling.vi” [nomor 5]
3. “Nearest List Index.vi” [nomor 6]
4. “Steer.vi” [nomor 7, 8]
5. “Check Collision.vi” [nomor 9]
6. “Insert.vi” [nomor 10, 11]
7. “Check Goal.vi” [nomor 13]
8. “Make Final Path.vi” [nomor 14]



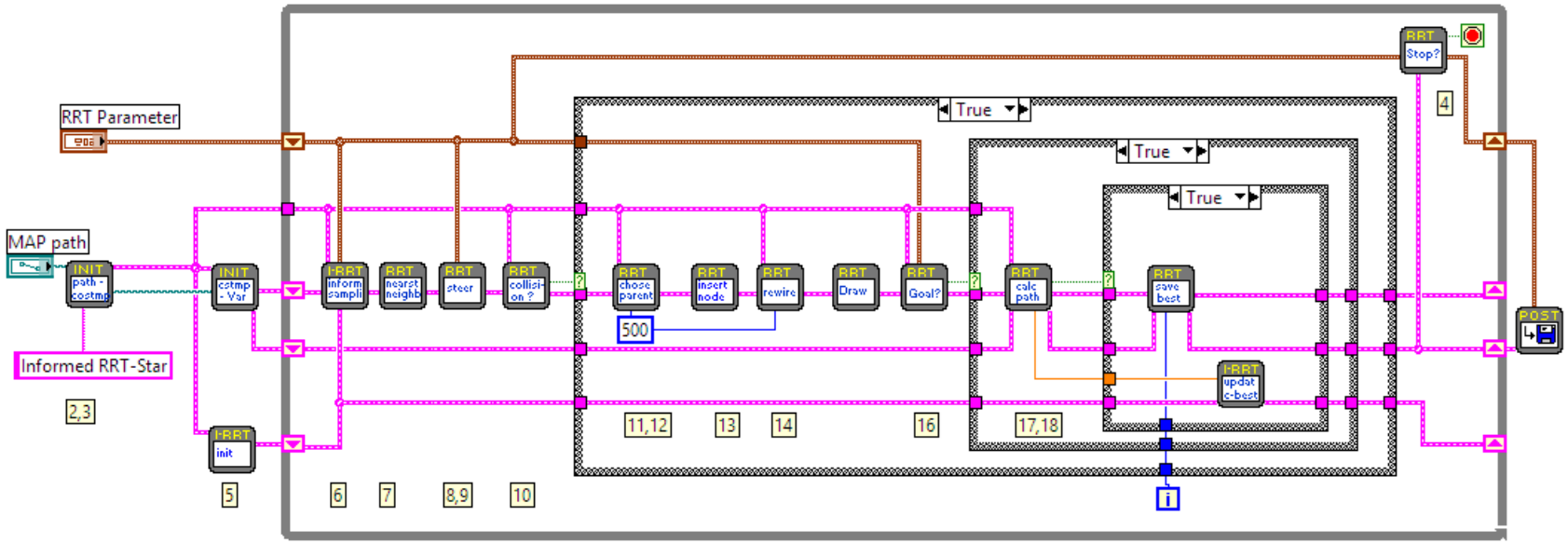
Gambar 6. Block diagram algoritma RRT yang dibuat dengan *library* ini. Nomor dalam kotak kuning menunjukkan proses yang dijelaskan pada Algoritma 1 Gambar 1.



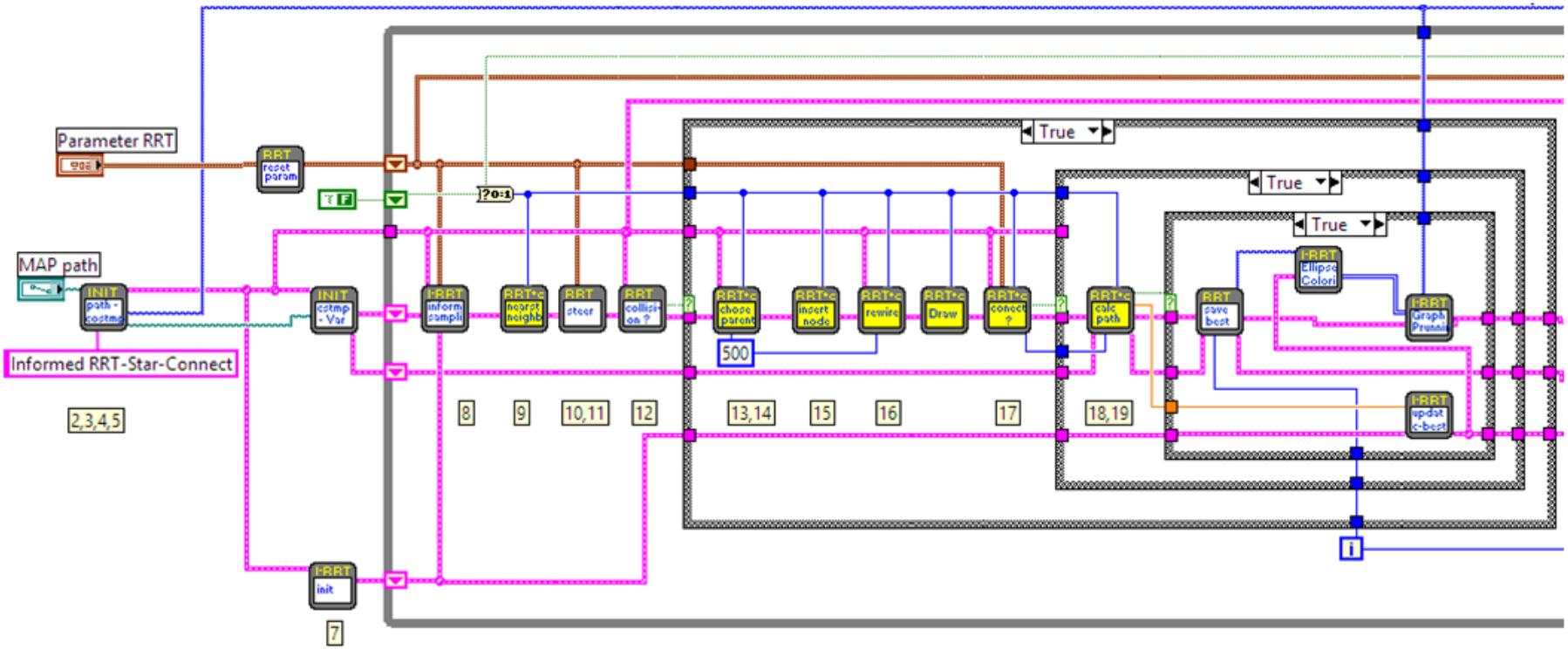
Gambar 7. Block diagram algoritma RRT* yang dibuat dengan *library* ini. Nomor dalam kotak kuning menunjukkan proses yang dijelaskan pada Algoritma 2 Gambar 2.



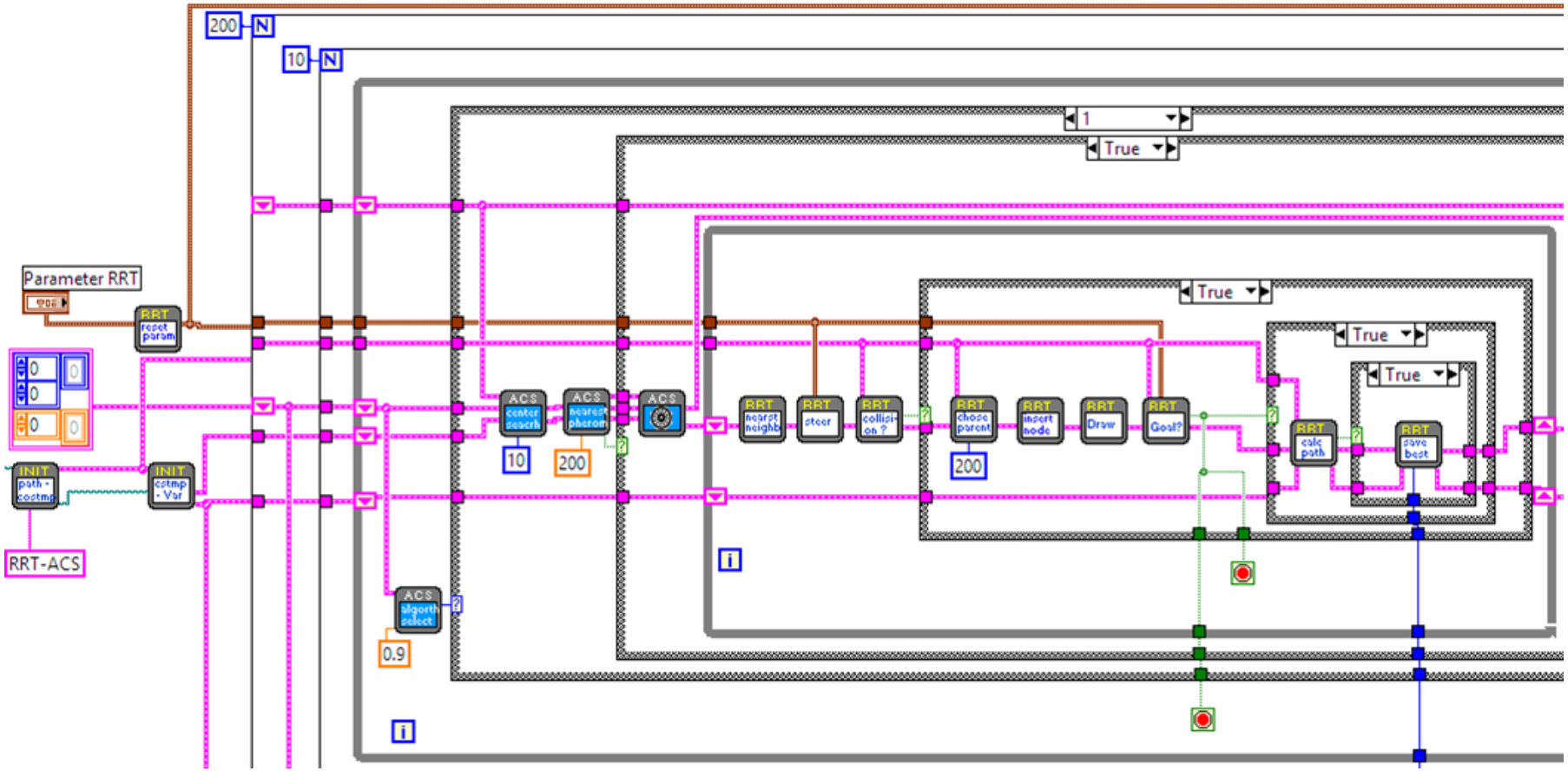
Gambar 8. Sebagian block diagram algoritma RRT*-*Connect* yang dibuat dengan *library* ini. Nomor dalam kotak kuning menunjukkan proses yang dijelaskan pada Algoritma 3 Gambar 3.



Gambar 9. Block diagram algoritma *Informed* RRT* yang dibuat dengan *library* ini. Nomor dalam kotak kuning menunjukkan proses yang dijelaskan pada Algoritma 4 Gambar 4.



Gambar 10. Sebagian block diagram algoritma *Informed* RRT*-*Connect* yang dibuat dengan *library* ini. Nomor dalam kotak kuning menunjukkan proses yang dijelaskan pada Algoritma 5 Gambar 5.



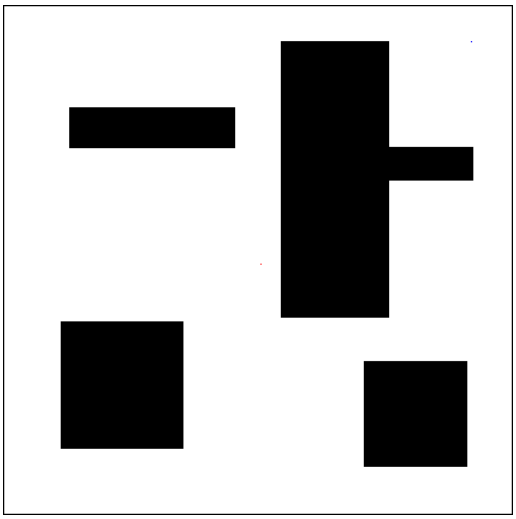
Gambar 11. Sebagian block diagram algoritma RRT-ACS yang dibuat dengan *library* ini. Nomor dalam kotak kuning menunjukkan proses yang dijelaskan makalah Pohan [16].

Subprogram “Data Initialization.vi” berfungsi untuk mengolah data environment yang akan di terapkan pada algoritma RRT. Proses ini meniru algoritma RRT pada **Gambar 1** baris 2 dan 3. Subprogram “Sampling.vi” melakukan perintah pada algoritma RRT pada **Gambar 1** baris 5. Subprogram “Nearest List Index.vi” melakukan perintah pada algoritma RRT pada **Gambar 1** baris 6. Subprogram “Steer.vi” melakukan perintah pada algoritma RRT pada **Gambar 1** baris 7, 8. Subprogram “Check Collision.vi” melakukan perintah pada algoritma RRT pada **Gambar 1** baris 9. Subprogram “Check Collision.vi” melakukan perintah pada algoritma RRT pada **Gambar 1** baris 9. Subprogram “Insert.vi” melakukan perintah pada algoritma RRT pada **Gambar 1** baris 10, 11. Subprogram “Check Goal.vi” melakukan perintah pada algoritma RRT pada **Gambar 1** baris 13. Subprogram “Make Final Path.vi” melakukan perintah pada algoritma RRT pada **Gambar 1** baris 14.

Adapun **Gambar 7** menunjukkan Block diagram LabVIEW dari algoritma RRT* yang dibuat dengan *library* ini. **Gambar 8** menunjukkan Block diagram LabVIEW dari algoritma RRT*-*Connect* yang dibuat dengan *library* ini. **Gambar 9** menunjukkan Block diagram LabVIEW dari algoritma *Informed* RRT* yang dibuat dengan *library* ini. **Gambar 10** menunjukkan Block diagram LabVIEW dari algoritma *Informed* RRT*-*Connect* yang dibuat dengan *library* ini. Sedangkan **Gambar 11** menunjukkan Block diagram LabVIEW dari algoritma RRT-ACS yang dibuat dengan *library* ini. Setiap nomor dengan kotak kuning pada **Gambar 7 - 11**, melakukan perintah pada algoritma bersangkutan pada **Gambar 2 – 5.**

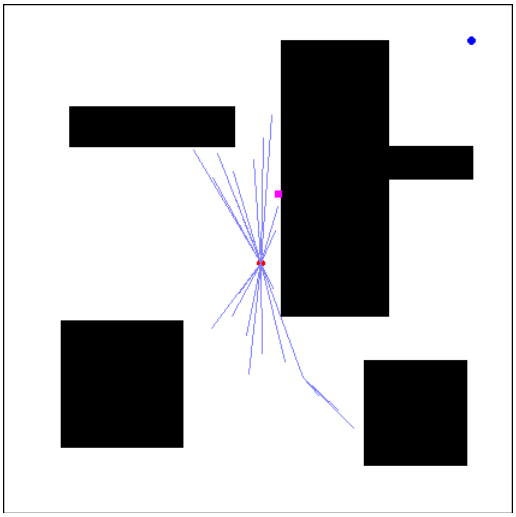
B. Visualisasi Proses Perencanaan Jalur

Salah satu keunggulan dari LabVIEW *library* yang diusulkan ini adalah dapat memvisualisasikan setiap subproses. Berikut ini adalah contoh visualisasi proses perencanaan jalur algoritma RRT* menggunakan skenario kasus yang diusulkan Karaman [10]. **Gambar 12** menunjukkan skenario lingkungan yang digunakan.



Gambar 12. Skenario lingkungan yang untuk pengujian algoritma RRT*, seperti yang diusulkan Karaman [10]..

Saat program dijalankan, dapat diperoleh visualisasi pohon pencarian seperti **Gambar 13**. Visualisasi ini memperlihatkan posisi node acak pada setiap iterasinya. Posisi node acak ditunjukkan oleh kotak merah muda. Cabang baru dari pohon pencarian akan menuju ke arah node acak. Lingkaran berwarna merah adalah node awal. Lingkaran berwarna biru adalah node tujuan.

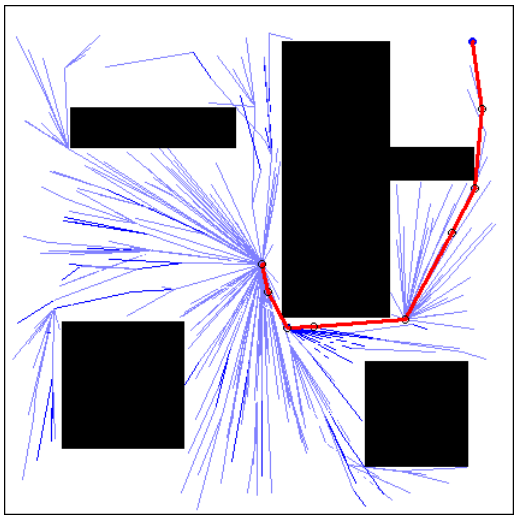


Gambar 13. Contoh Visualisasi dari pohon pencarian RRT*. Cabang baru dari pohon pencarian menuju node acak berwarna pink

Jika node baru dari pohon pencarian berada yang berdekatan dengan node tujuan, maka akan dibangun jalur lengkap yang menghubungkan node awal dan node tujuan. Contoh dari suatu jalur lengkap ditunjukkan pada **Gambar 14.**

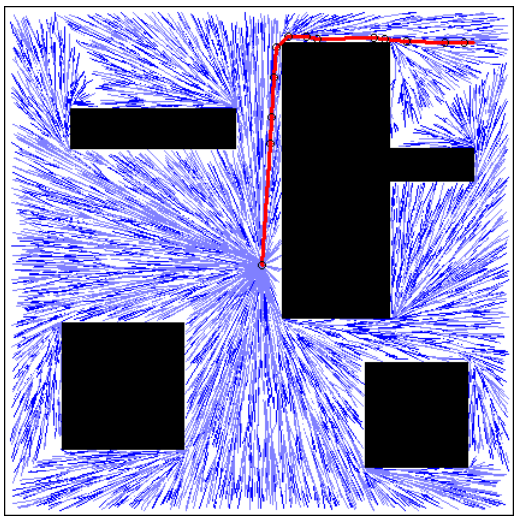
Selama iterasi berlangsung, algoritma RRT* dapat membangun jalur baru yang dapat menghubungkan node awal dan node akhir dengan

jarak lebih pendek. Jalur baru tersebut kemudian akan ditampilkan.



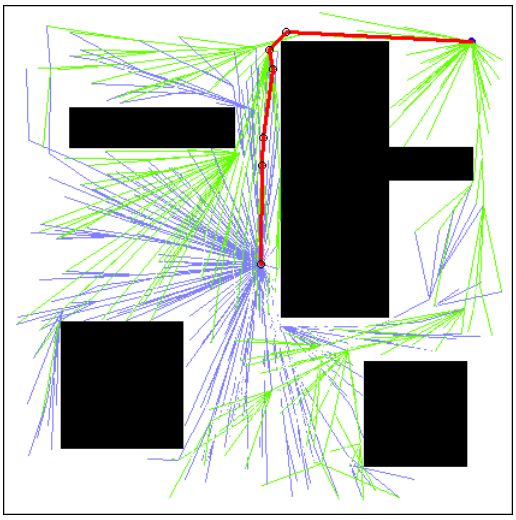
Gambar 14. Garis berwarna merah tebal adalah contoh jalur lengkap dari node awal ke node tujuan.

Gambar 15 menyajikan hasil algoritma RRT* setelah iterasi ke-9721. **Gambar 14** dan **Gambar 15** menunjukkan bahwa algoritma RRT* memiliki sifat optimal asimtotik. Sifat ini berarti algoritma RRT* memiliki jaminan dapat menghasilkan solusi yang optimal jika dan hanya jika diberikan waktu yang cukup. Karaman juga melaporkan sifat asimptotik optimal dari algoritma RRT* [11].



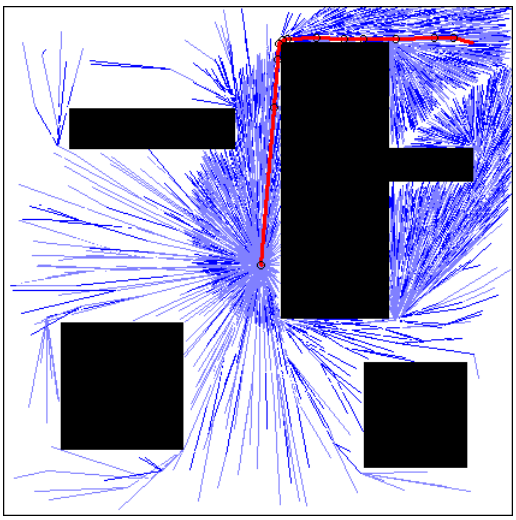
Gambar 15. Contoh jalur lengkap dari algoritma RRT* setelah melakukan iterasi sebanyak 9721.

Sebagai contoh visualisasi dari algoritma-algoritma lainnya yang ada di *library* ini, maka pada **Gambar 16** menunjukkan visualisasi proses dan hasil perencanaan jalur dari algoritma RRT*-*Connect*. Garis biru pada **Gambar 16** adalah pohon pencarian pertama, sedangkan garis hijau adalah pohon pencarian kedua. Kedua pohon pencarian akan berusaha untuk saling terhubung.



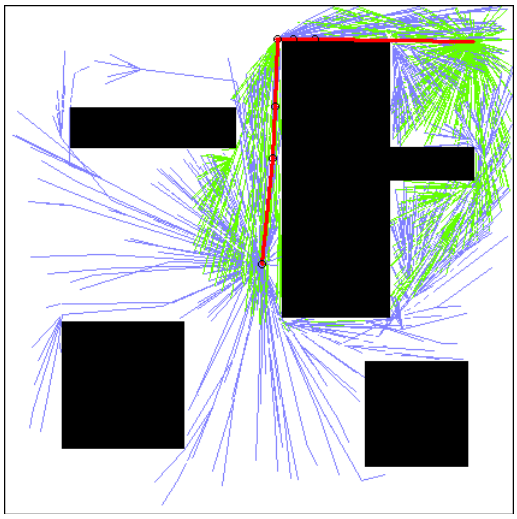
Gambar 16. Visualisasi jalur lengkap yang diperoleh algoritma RRT*-*Connect*

Gambar 17 menunjukkan visualisasi proses dan hasil perencanaan jalur dari algoritma *Informed* RRT*. Pada **Gambar 17** terlihat bahwa area pencarian difokuskan pada area ellipsoid yang ukurannya sebanding dengan rute terbaik saat ini.



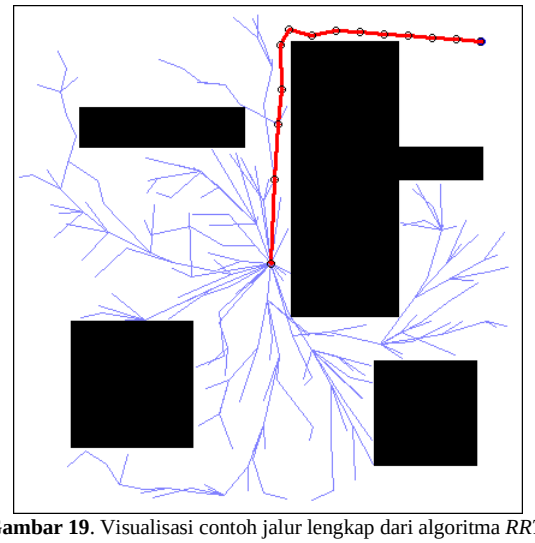
Gambar 17. Visualisasi contoh jalur lengkap dari algoritma RRT*-*Connect*

Gambar 18 menunjukkan visualisasi proses dan hasil perencanaan jalur dari algoritma *Informed* RRT*-*Connect*. Seperti pada algoritma RRT*-*Connect*, Garis biru pada **Gambar 18** adalah pohon pencarian pertama, sedangkan garis hijau adalah pohon pencarian kedua. Kedua pohon pencarian akan berusaha untuk saling terhubung. Dan seperti pada algoritma *Informed* RRT*, Pada **Gambar 18** terlihat bahwa area pencarian difokuskan pada area ellipsoid yang ukurannya sebanding dengan rute terbaik saat ini.

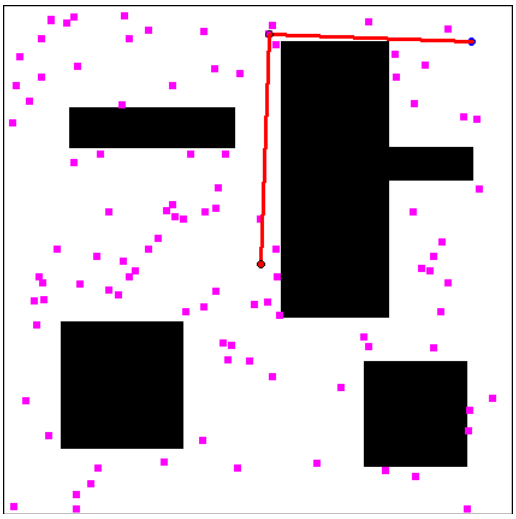


Gambar 18. Visualisasi contoh jalur lengkap dari algoritma *Informed* RRT*-*Connect*

Gambar 19 menunjukkan visualisasi proses dan hasil perencanaan jalur dari algoritma RRT-ACS. **Gambar 20** menunjukkan visualisasi proses dan hasil perencanaan jalur dari algoritma PRM. Seperti yang ditunjukkan pada **Gambar 20**, algoritma PRM pertama-tama membangkitkan sample point (kotak merah muda). Lalu algoritma PRM menggunakan Dijkstra untuk mencari rute terbaik yang menghubungkan node awal dengan tujuan melalui *sample point* tersebut.



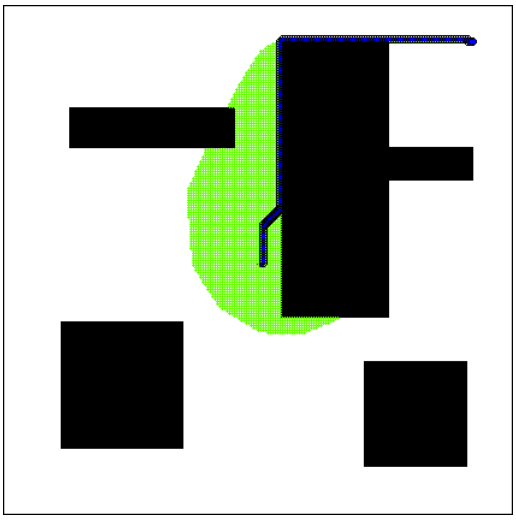
Gambar 19. Visualisasi contoh jalur lengkap dari algoritma *RRT-ACS*



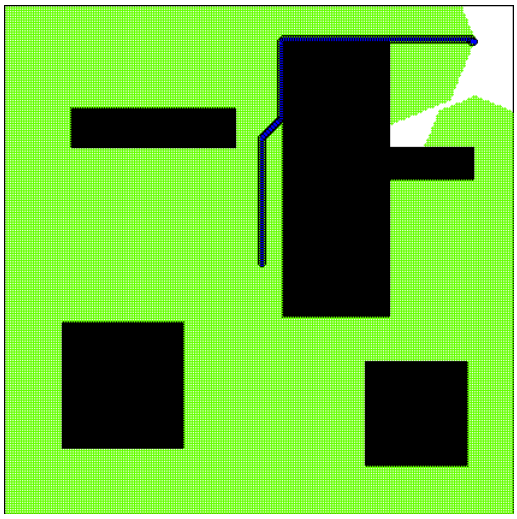
Gambar 20. Visualisasi contoh jalur lengkap dari algoritma PRM

Gambar 21 menunjukkan visualisasi proses dan hasil perencanaan jalur dari algoritma A*. **Gambar 22** menunjukkan visualisasi proses dan hasil perencanaan jalur dari algoritma Dijkstra. Kotak-kotak pada **Gambar 21** dan **Gambar 22** adalah hasil diskritisasi ruang pencarian, karena algoritma A* dan Dijkstra memerlukan proses diskritisasi ini. Lingkaran hijau menunjukkan node-node yang telah diperiksa oleh algoritma A* dan Dijkstra. Garis biru tebal adalah jalur terbaik yang dihasilkan oleh kedua algoritma tersebut.

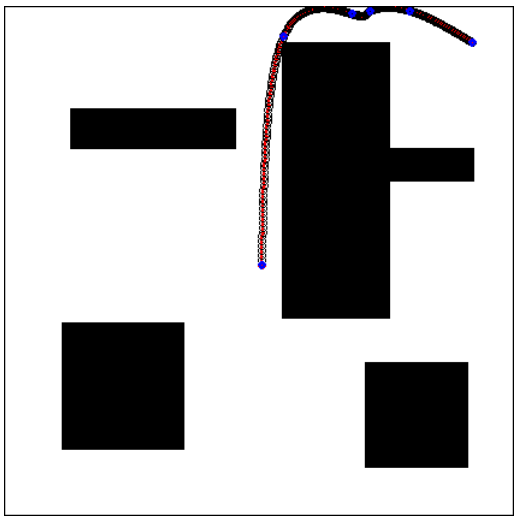
Adapun **Gambar 23** menunjukkan visualisasi proses dan hasil perencanaan jalur dari algoritma perencanaan jalur berbasis PSO. Lingkaran biru yang ada pada **Gambar 23** adalah partikel-partikel PSO yang bergerak sesuai peramaan umum PSO.



Gambar 21. Visualisasi contoh jalur lengkap dari algoritma algoritma A*



Gambar 22. Visualisasi contoh jalur lengkap dari algoritma Dijkstra



Gambar 23. Visualisasi jalur lengkap yang diperoleh algoritma perencanaan jalur berbasis PSO

C. Analisa Waktu Komputasi

Untuk mendemonstrasikan keunggulan waktu komputasi dari *library* yang diusulkan, kami membandingkan waktu yang komputasi dari *library* ini dengan PythonRobotics yang dibuat oleh Sakai menggunakan python [18] dan program RRT* yang dibuat oleh Vemprala menggunakan matlab [26]. Pengujian dilakukan menggunakan algoritma RRT*, dikarenakan algoritma tersebutlah yang ada pada setiap *library* yang akan diuji. Lamanya waktu yang dibutuhkan oleh algoritma RRT* untuk menyelesaikan satu iterasi diukur. Pengujian dilakukan sebanyak 50 kali untuk setiap *library*. **Tabel I** menunjukkan hasil perbandingan ini.

Table I. Perbandingan waktu komputasi beberapa *library* perencanaan jalur

Bahasa	Versi/ Compiler	Pembuat	Rata-rata Waktu Satu Iterasi
Python	3.7	[18]	227.00
Matlab	2018a	[26]	145.3
LabVIEW	7.1	Penulis	1.71

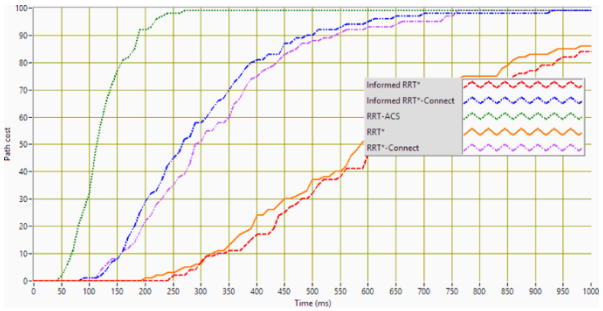
Tabel I menunjukkan keunggulan dari LabVIEW *library* yang diusulkan ini dalam hal waktu komputasi. Waktu komputasi dari LabVIEW *library* dapat diminimalkan dengan cara menjadikan semua subVI pada *library* ini sebagai *subroutine* [20]. Dengan menjadikan semua subVI sebagai *subroutine*, maka *overhead* subVI dapat diminimalkan. *Library* ini juga hanya fungsi-fungsi (subVI) bawaan LabVIEW yang dapat diubah menjadi *subroutine*. Perbandingan waktu komputasi jika suatu subVI dijadikan *subroutine* dan jika tidak dijadikan *subroutine* ditunjukkan pada **Tabel II**.

Table II. Perbandingan waktu komputasi jika suatu subVI dijadikan *subroutine* dan jika tidak dijadikan *subroutine*

Kondisi subVI	Waktu rata-rata algoritma untuk melakukan 20,000 iterasi
subVI dijadikan sebagai *subroutine*	66,4 milidetik
subVI tidak dijadikan sebagai *subroutine*	95,8 milidetik

D. Tools Untuk Analisis

LabVIEW *library* untuk perencanaan jalur ini, dibuat untuk membantu penelitian perencanaan jalur. Karena itu, *library* ini dilengkapi dengan berbagai *tools* yang dapat digunakan untuk menganalisis kinerja algoritma perencanaan jalur. Menggunakan *library* ini, setiap pertumbuhan pohon pencarian dan jalur terbaik yang dihasilkan dapat disimpan otomatis selama iterasi. Data-data dari proses perencanaan jalur juga dapat tersimpan dalam format file excel. Data iterasi, jalur terbaik, dan waktu untuk mencapai iterasi semuanya disimpan dalam file excel tersebut. *Library* ini juga dilengkapi *tools* untuk membuat grafik dari data performansi suatu perencanaan jalur, seperti yang ditunjukkan pada **Gambar 24**. **Gambar 24** menunjukkan grafik kecepatan konvergensi suatu algoritma untuk mencapai nilai optimal.



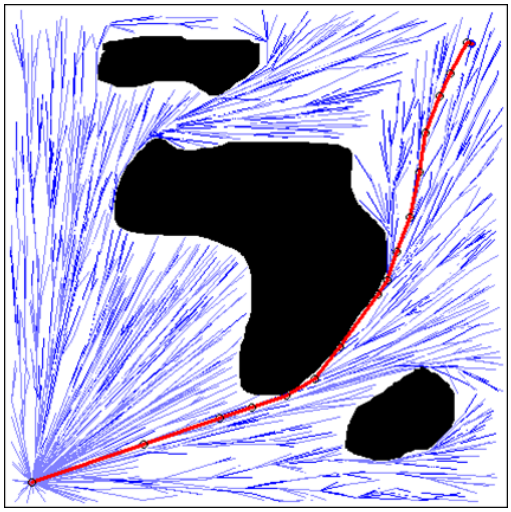
Gambar 24. Tools untuk analisis III: performansi dari suatu algoritma dapat divisualisasikan dalam bentuk grafik kecepatan konvergensi

E. Peta Skenario Pengujian

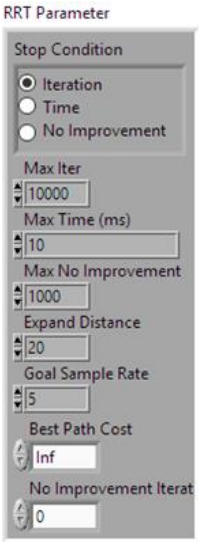
Salah satu kekurangan dari PythonRobotics yang dibuat oleh Sakai [18] adalah hanya bisa diaplikasikan pada lingkungan dengan *obstacle* berbentuk lingkaran saja. LabVIEW *library* ini dapat diaplikasikan pada lingkungan dengan berbagai bentuk *obstacle*. Bahkan *library* ini dapat diaplikasikan pada lingkungan dengan *obstacle* berbentuk kurva, seperti yang ditunjukkan pada **Gambar 25**.

F. Parameter Algoritma

Pada *library* ini, pengubahan parameter-parameter dari suatu algoritma dapat dilakukan dengan mudah. Dengan mengubah-ubah parameter ini, maka dapat diamati bagaimana pengaruhnya terhadap performansi dari algoritma perencanaan jalur tersebut. Contoh front panel yang disediakan untuk mengubah parameter algoritma RRT* ditunjukkan pada **Gambar 26**.



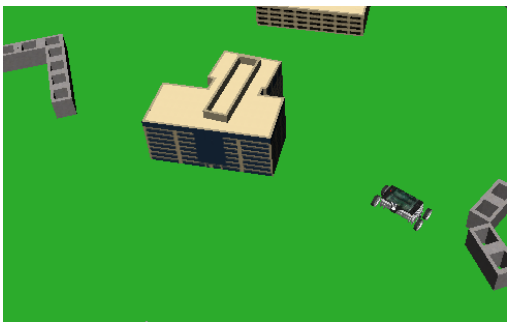
Gambar 25. Algoritma perencanaan jalur yang dibuat menggunakan *library* ini, bahkan dapat diimplementasikan pada lingkungan dengan *obstacle* berbentuk kurva



Gambar 26. Contoh *front panel* LabVIEW yang disediakan untuk mengubah parameter dari algoritma RRT*.

G. Contoh Implementasi pada LabVIEW Robotics Simulator.

LabVIEW *library* untuk perencanaan jalur ini telah diuji pada LabVIEW Robotic Simulator seperti terlihat pada **Gambar 27**. Pertama-tama, robot memindai rintangan untuk membuat peta lingkungan. Berdasarkan peta lingkungan tersebut, lalu algoritma perencanaan jalur diterapkan. Robot kemudian digerakkan untuk mengikuti jalur yang telah dibuat.



Gambar 27. Contoh implementasi pada LabVIEW Robotic Simulator

H. Beberapa Pengembangan yang Dapat Dilakukan

Dengan menggunakan *LabVIEW library* untuk perencanaan jalur diharapkan dapat membantu pengguna dalam mengembangkan algoritma perencanaan jalur. Terdapat beberapa topik pengembangan perencanaan jalur yang potensial untuk dikembangkan menggunakan *library* ini.

Yang pertama adalah pengembangan proses sampling dari algoritma RRT. Veras dkk. [27], dalam makalah mereka, telah membahas tinjauan literatur tentang metode pengambilan sampel dalam algoritma RRT. Pembaca dapat

menggunakan referensi tersebut untuk mengembangkan metode pengambilan sampel algoritma RRT. Sopa dkk. [28] serta Pakaya dan Pohan [29], dalam makalah mereka, telah mengembangkan metode pengambilan sampel dari algoritma RRT* menggunakan *library* ini.

Yang kedua adalah melakukan hibridisasi algoritma RRT dengan algoritma lainnya. Hibridisasi adalah menggabungkan dua algoritma berbeda untuk membuat algoritma baru yang dapat mencapai hasil yang lebih baik [30]. Pohan dkk. [11], dalam papernya telah melakukan hibridisasi algoritma RRT dengan ACS, dan menamakannya sebagai RRT-ACS. Pohan melaporkan bahwa performansi RRT-ACS lebih unggul daripada algoritma RRT maupun RRT*.

IV. KESIMPULAN

Dari penelitian ini telah dipresentasikan *library* perencanaan jalur berbasis LabVIEW. Menggunakan *library* ini, dapat dibuat beberapa algoritma perencanaan jalur algoritma RRT, RRT*, RRT*-*Connect*, *Informed* RRT*, *Informed* RRT*-*Connect*, RRT-ACS, PRM, A*, Dijkstra, dan PSO. Telah ditunjukkan beberapa keunggulan dari *library* ini, yaitu: dapat memvisualisasikan setiap subproses, waktu komputasi yang cepat, tersedia *tools* untuk analisis, skenario pengujian dapat dibuat secara fleksibel, serta parameter algoritma dapat dengan mudah diubah untuk melihat bagaimana pengaruhnya terhadap kinerja keluaran algoritma. Paper ini juga membahas beberapa pengembangan potensial dari algoritma RRT yang dapat dilakukan menggunakan *library* ini. Dengan menggunakan *library* ini, diharapkan mahasiswa dapat memperoleh pemahaman yang lebih baik mengenai algoritma perencanaan jalur serta membantu dalam melakukan penelitian terkait algoritma perencanaan jalur.

DAFTAR PUSTAKA

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

[2] B. Paden, M. Čáp, S. Z. Yong, D. Yershov dan E. Frazzoli, “A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles,” *IEEE Transactions Intelligence Vehicles*, vol. 1, no. 1, hlm. 33-55, 2016.

[3] M. F. Abdelwahed, A. E. Mohamed dan M. A. Saleh, “Improving Solution Quality for Experience-Based Framework Through Clustering Algorithms,” *IEEE Access*, vol. 7, hlm. 106720-106725, 2019.

[4] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematical*, vol. 1, no. 1, pp. 269–271, 1959.

[5] P. Hart, N. Nilsson, dan B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions Systems Science Cybernetics*, vol. SSC-4, no. 2, hlm. 100–107, 1968

[6] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, dan M.H. Ang, “Perception, Planning, Control, and Coordination for Autonomous Vehicles,” *machines*, vol. 5, no. 6, hlm. 1-54, 2017.

[7] L. E. Kavraki, M. N. Kolountzakis, dan J.-C. Latombe, “Analysis of probabilistic roadmaps for path planning,” *Transactions on Robotics and Automation*, vol. 14, hlm. 166–171, 1998.

[8] 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,” *IEEE International Conference on Intelligent Robots and Systems*, hlm. 2997-3004. 2014.

[9] S.M. Lavalle dan S. Lavalle, “Randomized Kinodynamic Planning,” *The international journal of robotics research*, vol. 20, no. 5, hlm. 378-400, 2001.

[10] S. Karaman, dan E. Frazzoli, “Sampling-based Algorithms for Optimal Motion Planning,” *The international journal of robotics research*, vol. 30, no. 7, hlm. 846-894, 2011.

[11] M. Aria, "Path Planning Algorithm Using Informed Rapidly exploring random tree*-*Connect* with local search," *Journal of Engineering Science and Technology Special Issue on INCITEST*, 50-57, 2020.

[12] W. Wang, H. Gao, Q. Yi, K. Zheng dan T. Gu, “An Improved RRT* Path Planning Algorithm for Service Robot,” *IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference*, hlm. 1824-1828, 2020.

[13] J. J. Kuffner dan S. M. LaValle, “RRT-*Connect*: An efficient approach to single-query path planning,” *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, vol. 2, hlm. 995-1001, 2000.

[14] S. Klemm, A. Hermann, J. Oberlander, dan A. Roennau, “RRT*-Connect: Faster, asymptotically optimal motion planning,” *IEEE international conference on robotics and biomimetics*, hlm. 1670-1677, 2015.

[15] R. Mashayekhi, M. Y. I. Idris, M. H. Anisi, I. Ahmedy, dan I. Ali, “Informed RRT*-Connect: An Asymptotically Optimal Single-Query Path Planning Method,” *IEEE Access*, 2020, vol. 8, hlm. 19842-19852, 2020.

[16] M. A. R. Pohan, B. R. Trilaksono, S. P. Santosa and A. S. Rohman, "Path Planning Algorithm Using the Hybridization of the Rapidly-Exploring Random Tree and Ant Colony Systems," *in IEEE Access*, vol. 9, hlm. 153599-153615, 2021.

[17] X. Li, D. Wu, J. He, M. Bashir, and M. Liping, "An improved method of particle swarm optimization for path planning of mobile robot," *Journal of Control Science and Engineering*, vol. 2020, hlm. 1-12, 2020.

[18] 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 (https://github.com/AtsushiSakai/PythonRobotics)

[19] S. B. Aruoba dan J. F. Villaverde “A comparison of programming languages in macroeconomics,” *Journal of Economic Dynamics and Control*, 58, 265-273, 2105

[20] N. Vahrenkamp, T. Asfour, dan R. Dillmann, “Simo: A simulation and motion planning toolbox for c+”, 2010

[21] P. Corke, “Robotics toolbox for MATLAB”, 2011, (https://github.com/petercorke/robotics-toolbox-matlab)

[22] L. G. Véras, F. L. Medeiros, dan L. N. Guimaraes, L. N, “Rapidly exploring Random Tree* with a sampling method based on Sukharev grids and convex vertices of safety hulls of *obstacles*,” *International Journal of Advanced Robotic Systems*, vol. 16, no. 1, hlm. 1-19, 2019

[23] E. S. Nugraha, “Simulator Edukatif untuk Pembelajaran Algoritma Rapidly-exploring Random Tree (RRT),” *Telekontran: Jurnal Ilmiah Telekomunikasi, Kendali dan Elektronika Terapan*, vol. 7, no. 2, hlm. 125-135, 2019

[24] A. Kumar, dan M. Goswami, M. “LabVIEW is faster and C is economical interfacing tool for UCT automation,” *arXiv preprint arXiv:2205.08260*, 2022.

[25] LabVIEW Basics, “Development course manual,” *National Instruments Corporation, Austin, Texas*, 2003

[26] S. Vemprala, "2D/3D RRT* algorithm (https://www.mathworks.com/matlabcentral/fileexchange/609

93-2d-3d-rrt-algorithm)," MATLAB Central File Exchange, 2022
[27] L. G. D. O. Véras, F. L. L. Medeiros dan L. N. F. Guimaráes, "Systematic Literature Review of Sampling Process in Rapidly-Exploring Random Trees," IEEE Access, vol. 7, hlm. 50933-50953, 2019.
[28] A. Sopa, T. Rahajoeningroem, dan R. Hartono. "Algoritma Rapidly Exploring Random Tree Star Dengan Integrasi Metode Sampling Goal Biassing, Gaussian, Dan Boundary." Telekontran: Jurnal Ilmiah Telekomunikasi, Kendali dan Elektronika Terapan, Vol. 9, no. 2, hlm. 129-138, 2021.
[29] A. Pakaya, dan M.A.R. Pohan, "Metode Improved Gaussian Sampling Pada Algoritma Rapidly Exploring Random Tree," Telekontran: Jurnal Ilmiah Telekomunikasi, Kendali dan Elektronika Terapan, Vol. 9, no. 2, hlm. 106-118, 2021.
[30] T. M. Shami, A. A. El-Saleh, M. Alswaitti, Q. Al-Tashi, M. A. Summakieh dan S. Mirjalili, "Particle Swarm Optimization: A Comprehensive Survey," *IEEE Access*, vol. 10, hlm. 10031-10061, 2022.