

BAB II

LANDASAN TEORI

2.1 *Rapidly Exploring Random Tree*

Rapidly exploring random tree (RRT) dikembangkan untuk memecahkan masalah dalam pembuatan perencanaan jalur, algoritma *Rapidly Exploring Random Tree* adalah algoritma berdasarkan struktur pohon pencarian. Algoritma ini sangat cocok untuk menyelesaikan masalah kinematika dan perencanaan jalur, karena kurva yang dapat dieksekusi disimulasikan melalui kinematika dan pasangan node. Algoritma *Rapidly Exploring Random Tree* dapat dilihat pada **Gambar 2.1** di bawah ini.

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 2.1 Algoritma *Rapidly Exploring Random Tree* [12]

Berdasarkan **Gambar 2.1** di atas 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 baru akan ditambahkan ke pohon pencarian (baris 7 dan 8). Lalu iterasi akan berulang sebanyak N kali (baris 3 dan 9) [12].

2.2 *Rapidly Exploring Random Tree**

Algoritma *Rapidly Exploring Random Tree** merupakan algoritma berbasis pada *sampling* tambahan yang akan menemukan jalur awal dengan cepat, kemudian mengoptimalkan jalur saat eksekusi berlangsung. **Gambar 2.2** di bawah merupakan algoritma dari *Rapidly Exploring Random Tree**

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 2.2 Algoritma *Rapidly Exploring Random Tree** [12]

Pada **Gambar 2.2** di atas menunjukkan *pseudo code* algoritma *Rapidly Exploring Random Tree**. Algoritma *Rapidly Exploring Random Tree** ini hampir mirip dengan algoritma *Rapidly Exploring Random Tree* (baris 1 – 6 algoritma pada **Gambar 2.1** sama dengan baris 1 – 6 pada **Gambar 2.2**). Perbedaannya terdapat adanya proses *ChooseParent* (baris 9) dan *Rewire* (baris 11). Untuk penjelasan lebih lanjut akan dijelaskan pada Bab 3.

2.3 Goal Biasing Sampling

Metode ini berawal dari ide dasar sederhana, yakni goal *biasing* bekerja dengan cara mengambil sampel acak pada ruang pencarian, kemudian sesekali mengambil akan mengambil titik sampel langsung pada titik tujuan. Apabila sampel yang di ambil pada titik tujuan tidak menabrak *obstacle* maka *random node* tersebut akan dimasukan ke *node list*. **Gambar 2.3** di bawah merupakan algoritma dari *goal biasing sampling*, algoritma tersebut akan digunakan sebagai salah satu dari metode sampling dalam pengujian di algoritma *rapidly exploring eandom tree** yang akan dirancang pada BAB III.

Algorithm 3 : GoalBiasSAMPLE(q_{goal})	
1: $rand \leftarrow \text{RandomNumber}$	▷ between 0 and 100
2: if $rand < k$ then	
3: $q_{rand} \leftarrow q_{goal}$	
4: else	
5: $q_{rand} \leftarrow \text{RandomConfig}();$	
6: end if	
7: return q_{rand}	

Gambar 2.3 Algoritma Goal Biasing Sampling [13]

Pada **Gambar 2.3** merupakan *pseudo code* dari *goal biasing sampling* yang bertujuan untuk pengambilan *random* sampel secara acak pada ruang pencarian, kemudia akan mengambil sampel pada ruang tujuan. Algoritma mengambil konfigurasi tujuan q_{goal} dan bilangan acak dalam algoritma yang dipilih dari 0 hingga 100. Jika bilangan acak kurang dari persentase R, konfigurasi acak q_{rand} akan menjadi konfigurasi tujuan q_{goal} . Jika bilangan acak lebih dari persentase R, konfigurasi acak q_{rand} dipilih seperti metode pengambilan sampel seragam [13].

2.4 Gaussian Sampling

Gaussian sampling, yaitu sebuah teknik pengambilan *random* sampel di dekat rintangan. *Gaussian sampling* akan menumbuhkan c_1 dan c_2 secara bersama dengan probabilitas yang lebih tinggi di dekat *obstacle*. Berikut **Gambar**

2.4 Algoritma dari *gaussian sampling*

```

1.  loop
2.     $c_1 \leftarrow$  a random configuration
3.     $d \leftarrow$  a distance chosen according to
        a normal distribution
4.     $c_2 \leftarrow$  a random conf. at distance  $d$  from  $c_1$ 
5.    if  $c_1 \in C_{\text{free}}$  and  $c_2 \notin C_{\text{free}}$  then
6.      add  $c_1$  to the graph
7.    else if  $c_2 \in C_{\text{free}}$  and  $c_1 \notin C_{\text{free}}$  then
8.      add  $c_2$  to the graph
9.    else
10.     discard both

```

Gambar 2.4 Algoritma *Gaussian Sampling* [14]

Pada **Gambar 4** di atas menunjukkan algoritma *pseudo code gaussian sampling*. *Random* konfigurasi di *inisialisasi* dengan mengeluarkan c_1 dan c_2 dengan jarak d yang ditetapkan (baris 2 - 3) kemudian dirandom menggunakan *gaussian sampling*. Jika c_1 bebas (berada di luar *obstacle*), dan c_2 berada dalam *obstacle* maka c_1 digambar dimasukan ke *node_list*. Sebaliknya jika c_2 bebas (berada di luar *obstacle*), dan c_1 berada dalam *obstacle* maka c_2 digambar dimasukan ke *node_list* (baris 5 – 8). Selain itu tidak akan di eksekusi (baris 9 – 10).

2.5 LabVIEW

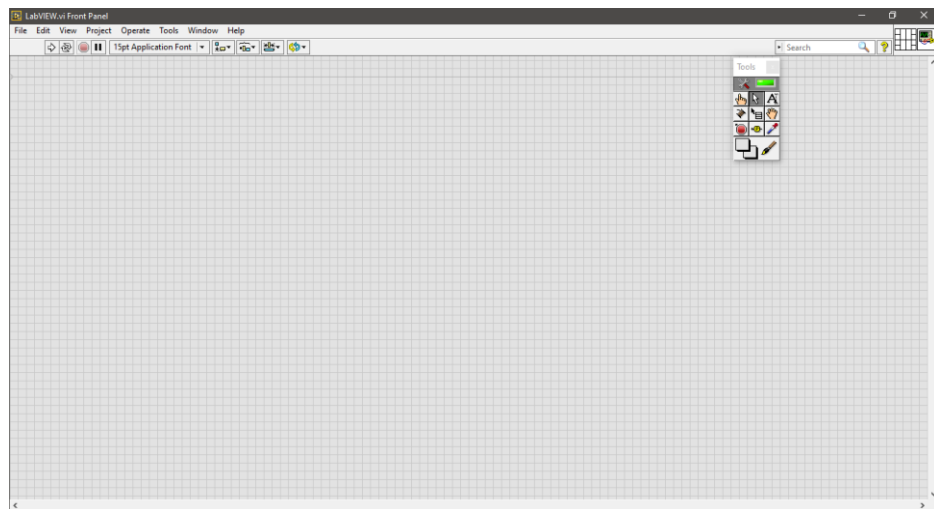
LabVIEW (*Laboratory Virtual Instrumentation Engineering Workbench*) merupakan perangkat lunak komputer untuk pemrosesan data dalam bidang akuisisi data serta merupakan kendali instrumentasi dan automasi industri. LabVIEW itu sendiri menggunakan gambar atau *grafis* sebagai fungsi *Graphical Programming*

Language atau *Visual Programming Language*. LabVIEW menggunakan bahasa pemrograman yang dapat menginterpretasikan data dengan sebuah *grafis* sebagai suatu fungsi.

LabVIEW disebut juga dengan *Virtual Instrumen* (VI) karena terdapat beberapa tampilan dan operasi pada program LabVIEW yang menyerupai *instrument* seperti osiloskop dan multimeter. Pada setiap *Virtual Instrumen* (VI) banyak menggunakan fungsi – fungsi yang memanipulasi input dari *user interface* atau sumber lainnya dan menampilkan sebuah informasi atau memindahkan informasi tersebut ke file atau *computer* lainnya.

Terdapat tiga desain antarmuka yang digunakan oleh LabVIEW:

- *Front panel* adalah desain antarmuka yang dapat digunakan untuk memberi masukan atau sebagai keluaran. Berikut **Gambar 2.5** *Front panel*



Gambar 2.5 Front Panel LabVIEW

Pada **Gambar 2.5** di atas merupakan *front panel* LabVIEW, contoh fungsi dari *front panel* LabVIEW yaitu *numeric control*. *numeric control* adalah sebuah kendali yang dapat memberi masukan berupa angka atau *image display* yang dapat menampilkan gambar.

- *Block Diagram* adalah desain antarmuka yang berisi *grafis* fungsi – fungsi perhitungan yang digunakan dalam program LabVIEW. *Grafis* fungsi yang tersedia akan dihubungkan untuk membuat persamaan atau untuk menghasilkan nilai – nilai yang kita inginkan
- *Control and Functions Palettes* pada *functions* dan *controls palettes* terdapat semua *grafis* fungsi yang dimiliki oleh LabVIEW yang dapat digunakan pada *front panel* ataupun pada *block diagram*.