BAB II

DASAR TEORI

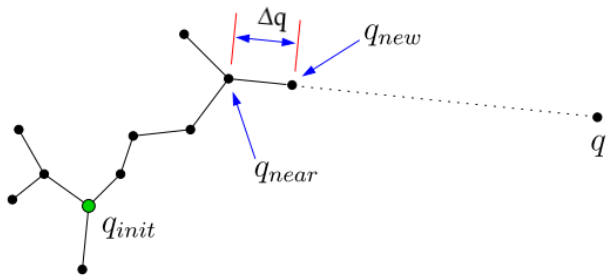
2.1 Algoritma *Rapidly Exploring Random Tree*

Rapidly Exploring Random Tree (RRT) merupakan algoritma yang dikembangkan oleh Steven M. LaValle dan James pada tahun 1998 [10]. RRT sudah pernah dikembangkan pada program *path planning* algoritma *Ariadne's clew* [13] dan algoritma *expansive configuration spaces* [14].

Algoritma RRT adalah struktur data yang dirancang secara efisien untuk mencari ruang bebas tabrakan. RRT seperti cabang pada pohon, dimana algoritma ini menumbuhkan setiap cabang yang akan berakar pada konfigurasi awal dengan menggunakan sampel acak pada ruang pencarian. Jika koneksi antara cabang dengan sampel acak tersebut melewati ruang bebas dan bebas hambatan, maka cabang ini dapat ditambahkan ke cabang pohon baru. Cabang pohon algoritma RRT akan terus bertumbuh ke segala arah yang tidak terhalang oleh collsion hingga menemukan titik tujuan. Jalur yang dilewati cabang pohon pertama merupakan jalur terpendek untuk mencapai titik tujuan tersebut.

Algortima RRT dimulai dengan inisialisasi *node start* (q_{start}) pada daerah ruang konfigurasi bebas (C_{free}). Setiap iterasi terdapat *node random* (q_{rand}) yang akan berpindah-pindah pada seluruh daerah konfigurasi C_{free}. Setiap perpindahan q_{rand} akan dilakukan pengecekan *collison* disekitar q_{rand}. Jika q_{rand} terdapat pada titik bebas *obstacle* maka akan dipilih *node* terdekat dari pohon pencarian yang

dinamakan $q_{nearest}$. Selanjutnya dibuat cabang baru dari q_{rand} menuju $q_{nearest}$ sejauh *expand distance* (Δq). Cabang baru yang telah dihubungkan tersebut dinamakan *new node* (q_{new}). Lalu algoritma RRT akan terus berlanjut ke iterasi selanjutnya hingga menemukan titik *goal node*. Ilustrasi algoritma pohon pencarian RRT ditunjukkan pada **Gambar 2.1.**



Gambar 2.1 Ilustrasi pohon pencarian RRT [7]

Pohon pencarian pada algoritma RRT lebih cenderung mengeksplore wilayah konfigurasi yang belum pernah dijelajahi. Waktu komputasi yang dibutuhkan pohon pencarian RRT juga lebih singkat dikarenakan algoritma RRT tidak mempertimbangkan jalur terpendek menuju ke *goal node*. Algoritma dasar RRT dalam pseudo code ditunjukkan pada **Gambar 2.2.**

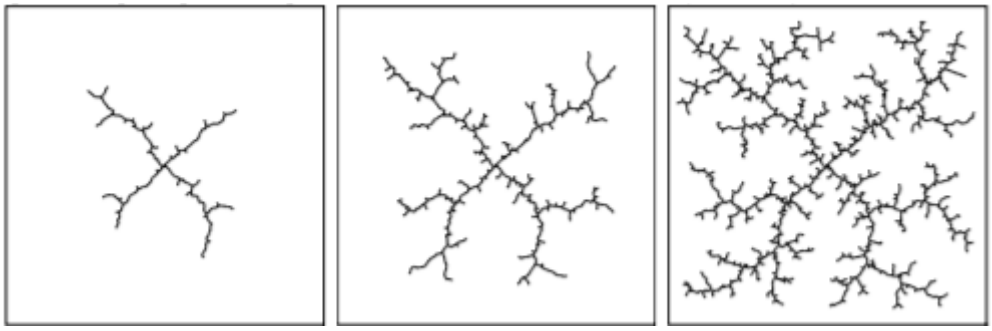
Algorithm 1 : $T = (V, E) \leftarrow RRT(q_{init})$

```
10. T ← InitializeTree()
11. T ← InsertNode(∅, q_init, T)
12. for k ← 1 to N do
13.     q_rand ← RandomSample(k)
14.     q_nearest ← NearestNeighbor(q_rand, Q_near, T)
15.     q_new ← Steer (q_nearest, q_rand, Δq)
16.     if Obstaclefree(q_new, q_nearest) then
17.         T ← InsertNode(q_min, q_new, T)
18. end
```

Gambar 2.2 Algoritma dasar RRT [15]

Lavelle menjelaskan bahwa proses perkembangan cabang pohon algoritma RRT, cabang baru cenderung mengekplorasi daerah yang belum terjelajahi [16].

Ilustrasi jalur pohon pencarian RRT setiap iterasi untuk mencapai *goal* ditunjukkan pada **Gambar 2.3.**



Gambar 2.3 Ilustrasi jalur pohon pencarian RRT setiap iterasi [16]

2.2 Algoritma *Rapidly Exploring Random Tree**

Rapidly Exploring Random Tree Star (RRT*) ialah algoritma hasil pengembangan yang lebih optimal dari algoritma *Rapidly Exploring Random Tree* (RRT). RRT* pertama kali diperkenalkan oleh Dr, Karaman dan Dr. Frazolli pada tahun 2010 yang bertujuan untuk mencapai jalur terpendek dan waktu komputasi mencapai *goal* tercepat [8]. Berbeda dengan algoritma RRT, algoritma RRT* lebih mempertimbangkan semua *node* terdekat dari q_{new} dan selalu mengecek *cost* terkecil dari masing-masing q_{near} tersebut.

Algoritma RRT* memiliki penambahan dua subprogram yang berbeda dengan algoritma RRT diantaranya subprogram *choose parent* dan subprogram *rewire*. Seperti yang dijelaskan sebelumnya algoritma RRT* sangat mempertimbangkan semua *node* terdekat, proses pemilihan *node* yang lebih optimal ini berada pada program *choose parent*. Proses *choose parent* akan mencari *node-node* terdekat dari q_{new}, lalu *node* terdekat yang ini dinamakan *node* q_{near}. Jalur yang memiliki cost terkecil antara q_{near} dengan q_{new} memiliki kemungkinan untuk dijadikannya *node parent* dari q_{new}. Algoritma dasar RRT* (algoritma 2) dapat

dilihat pada **Gambar 2.4.** Pseudo code *choose parent* (algoritma 3) ditunjukkan pada **Gambar 2.5.**

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

13. $T \leftarrow InitializeTree()$
14. $T \leftarrow InsertNode(\emptyset, q_{init}, T)$
15. **for** $k \leftarrow 1$ **to** N **do**
16. $q_{rand} \leftarrow RandomSample(k)$
17. $q_{nearest} \leftarrow NearestNeighbor(q_{rand}, Q_{near}, T)$
18. $q_{new} \leftarrow Steer\ (q_{nearest}, q_{rand}, \Delta q)$
19. **if** $Obstaclefree(q_{new}, q_{nearest})$ **then**
20. $Q_{near} \leftarrow Near(T, z_{new})$
21. $q_{parent} \leftarrow ChooseParent(q_{new}, Q_{near}, q_{nearest})$
22. $T \leftarrow InsertNode(q_{parent}, q_{new}, T)$
23. $T \leftarrow Rewire(T, Q_{near}, q_{parent}, q_{new})$
24. **end**

Gambar 2.4 Algoritma dasar RRT* [15]

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

14. $q_{min} \leftarrow q_{nearest}$
15. $c_{min} \leftarrow Cost(q_{nearest}) + c(q_{rand})$
16. **for** $q_{near} \in Q_{near}$ **do**
17. $q_{path} \leftarrow Steer(q_{near}, q_{rand}, \Delta q)$
18. **if** $ObstacleFree(q_{path})$ **then**
19. $c_{new} \leftarrow Cost(q_{near}) + c(q_{rand})$
20. **if** $c_{min} < c_{new}$ **then**
21. $c_{min} \leftarrow c_{new}$
22. $q_{min} \leftarrow q_{new}$
23. **end**
24. **end**
25. **end**
26. **return** q_{min}

Gambar 2.5 Operasi *choose parent* pada algoritma RRT* [10]

Jika pemilihan *parent* dilakukan, maka 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 4) yang dapat dilihat pada **Gambar 2.6.**

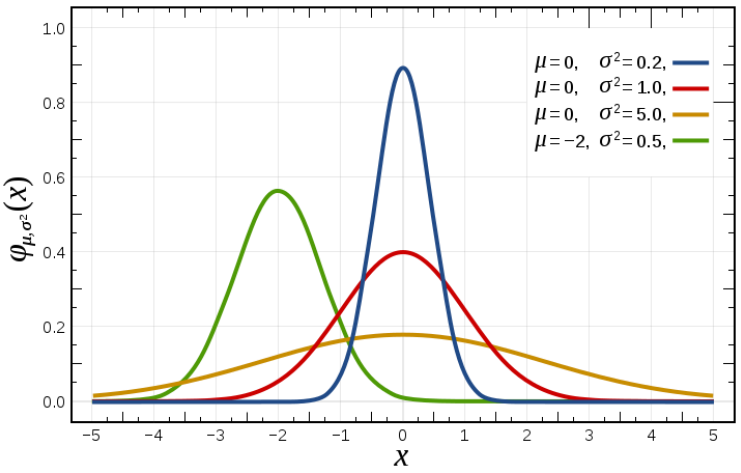
Algorithm 4 : $T \leftarrow Rewire(T, Q_{near}, q_{min}, q_{rand})$

7. **for** $q_{near} \in Q_{near}$ **do**
8. $q_{path} \leftarrow Steer(q_{near}, q_{rand}, \Delta q)$
9. **if** $ObstacleFree(q_{path})$ **and** $Cost(q_{rand}) + c(q_{path}) < Cost(q_{near})$ **then**
10. $T \leftarrow ReConnect(q_{rand}, q_{near}, T)$
11. **end**
12. **return** T

Gambar 2.6 Operasi *rewire* pada algoritma RRT* [10]

2.3 *Gaussian Sampling*

Distribusi *gaussian* atau bisa dikenal dengan distribusi normal merupakan fungsi probabilitas yang menunjukan penyebaran suatu variable acak yang diindekskan berdasarkan nilai waktu dan ruang konfigurasi. Distribusi *gaussian* diambil dari nama penemunya yaitu Carl Friederich Gauss seorang matematikan asal Jerman yang mengembangkan teori distribusi pada periode 1794 – 1809 [11]. Kurva distribusi normal/ditribusi *gaussian* dapat dilihat pada **Gambar 2.7.**



Gambar 2.7 Kurva distribusi normal/*gauss* [17]

Distribusi normal memiliki dua parameter yang dijadikan acuan kurva yaitu *mean* (nilai rata-rata) dan standar deviasi (simpangan baku). Nilai rata-rata digunakan sebagai pusat distribusi pada kurva distribusi normal dan yang menentukan lokasi titik puncak kurva lonceng. Sedangkan standar deviasi

merupakan perhitungan variabilitas yang menentukan lebar sebuah kurva distribusi normal. Semakin kecil nilai standar deviasi maka kurva yang dihasilkan akan semakin runcing ke atas. Secara matematis fungsi nilai *gaussian sampling* dapat dilihat pada persamaan di bawah ini:

$$f\ (x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2\sigma^2}(x-\mu)^2} \tag{2.1}$$

Dengan sebagai f*(x)* merupakan fungsi kerapatan (*density function*) dari x. Nilai rata – rata didefinisikan dengan μ, sedangan σ adalah simpangan baku (*standart deviation*). Sebuah distribusi dapat disebut dengan distribusi normal apabila nilai μ=0 dan σ=1. Sehingga persamaan di atas akan berubah menjadi persamaan berikut ini:

$$f\ (x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-1}{2}x^2} \tag{2.2}$$

Berdasarkan Teori Limit Pusat (*Central Limit Theorem*) yang mengatakan bahwa jika sampel cukup besar maka distribusi nilai rata-rata akan mengikuti distribusi *gaussian*[18]. Algoritma dasar *gaussian sampling* pada dapat dilihat pada **Gambar 2.8.**

```
1.  loop
2.      c1 ← a random configuration
3.      d ← a distance chosen according to
                a normal distribution
4.      c2 ← a random conf. at distance d from c1
5.      if c1 ∈ Cfree and c2 ∉ Cfree then
6.          add c1 to the graph
7.      else if c2 ∈ Cfree and c1 ∉ Cfree then
8.                  add c2 to the graph
9.          else
10.                 discard both
```

Gambar 2.8 Algoritma dasar *gaussian sampling* [8]

Gaussian sampling pertama kali dilakukan dengan *rand*om konfigurasi titik c_1 pada seluruh ruang konfigurasi terlihat pada baris 2. Ketika *node* c_1 telah berada pada ruang konfigurasi maka ditarik garis sejauh panjang nilai *gaussian* distribusi d terlihat pada baris 4. Dengan menggetahui nilai posisi titik c_1 dan nilai d maka dapat digunakan persamaan phytagoras untuk mengetahui posisi titik c_2. Ketika letak posisi titik c_1 dan c_2 telah diketahui dengan jarak sejauh d, lalu dapat ditentukan nilai *rand*om *gaussian* diantara dua titik tersebut dengan metode transformasi *box muller.*

2.4 *Box muller of Gaussian Distribution*

Metode transformasi *box muller* dikemukakan oleh George E. P dan Mervin E. Muller pada tahun 1958 [12]. Metode *box muller* ini menentukan nilai rata-rata sampel acak dari distribusi *gaussian* standar. Persamaan *box muller* secara umum terdapat dua parameter dapat dilihat pada persamaan berikut ini.

$$x_1 = \sqrt{-2ln(R_1)cos(2\pi R_2)} \tag{2.3}$$

dan

$$y_2 = \sqrt{-2ln(R_1)sin(2\pi R_2)} \tag{2.4}$$

Dengan R_1 dan R_2 adalah dua bilangan acak seragam dari interval (0,1) dan X adalah hasil sampel yang diinginkan dari distribusi *gaussian* standar. Algoritma dasar *box muller gaussian sampling* dapat dilihat pada **Gambar 2.9** berikut ini.

Algorithm 1. Box-Muller

1: $a \leftarrow \sqrt{-2 \ln U_1}, b \leftarrow 2\pi U_2$
2: **return** $(a \sin b, a \cos b)$ {Return pair of independent numbers}

Gambar 2.9 Algoritma dasar *box muller* [19]