BAB II
LANDASAN TEORI

2.1 *Path Planning* (Perencanaan Jalur)

Path planning merupakan salah satu masalah mendasar dalam robotika, dan menjadi permasalahan yang banyak dipelajari. Definisi dari *path planning* sendiri yaitu menemukan gerakan bebas tumbukan antara awal (mulai) dan konfigurasi akhir (tujuan) dalam lingkungan tertentu. Situasi paling sederhana adalah ketika jalur direncanakan dalam lingkungan yang statis, tetapi secara lebih umum *path planning* juga dapat dirumuskan untuk sistem robot pada kendala kinematik, dalam lingkungan yang dinamis dan tidak diketahui (random).

2.2 *Rapidly-exploring Random Tree*

Terdapat banyak algoritma yang dapat digunakan pada *path planning*, salah satunya adalah algoritma *Rapidly-exploring Random Tree* atau RRT. RRT adalah algoritma pencarian jalur yang menggunakan *sampling*. Algoritma RRT sendiri telah banyak dikembangkan menjadi beberapa macam algoritma, yaitu algoritma RRT*, *Informed-RRT*, *biased* RRT, *bidirectional* RRT, dan lain sebagainya.

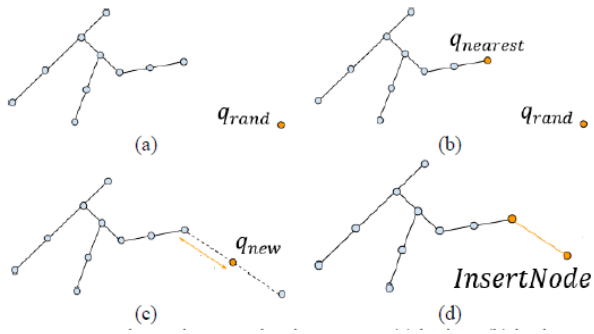
Pada penelitian ini digunakan 2 jenis algoritma RRT, yaitu RRT* dan *Informed-RRT*. Analisis performansi algoritma *Informed-RRT** dan RRT* ini dibuat menggunakan LabVIEW. Pada **Gambar 2.1** terdapat algoritma dasar RRT. Dari gambar tersebut terdapat beberapa proses yang dilakukan oleh algoritma RRT, 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_{rand} dan $q_{nearest}$. Node baru ini dinamakan q_{new}. Node q_{new}. akan berjarak Δq dari $q_{nearest}$ (baris 6). Jika diantara $q_{nearest}$ dan q_{new} tidak ada hambatan, maka node q_{new} akan ditambahkan ke

pohon pencarian (baris 7 dan 8). Lalu iterasi akan berulang sebanyak *N* kali (baris 3 dan 9).

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. $\quad q_{rand} \leftarrow RandomSample(k)$
5. $\quad q_{nearest} \leftarrow NearestNeighbor(q_{rand}, Q_{near}, T)$
6. $\quad q_{new} \leftarrow Steer\ (q_{nearest}, q_{rand}, \Delta q)$
7. $\quad$ **if** $Obstaclefree(q_{new}, q_{nearest})$ **then**
8. $\quad\quad T \leftarrow InsertNode(q_{parent}, q_{new}, T)$
9. **end**

Gambar 2.1 Algoritma dasar RRT [14]



Gambar 2.2 Ilustrasi proses algoritma RRT [13]

Dari **Gambar 2.1** dan **Gambar 2.3** dapat dilihat bahwa algoritma RRT dan RRT* memiliki kemiripan, yaitu pada baris 1-6 memiliki isi yang sama. Perbedaan dari algoritma ini yaitu pada algoritma RRT* terdapat proses *Chooseparent* (baris 9 **Gambar 2.3**) dan *Rewire* (baris 11 **Gambar 2.3**).

Pengembangan fitur RRT* adalah adanya operasi *Chooseparent* (**Gambar 2.4**) dan operasi *Rewire* (**Gambar 2.5**). Seperti pada algoritma RRT, algoritma RRT* secara bertahap membangun pohon pencarian dengan mengambil sampel secara acak pada ruang Kemudian menambahkan lintasan baru yang memperluas node terdekat dari pohon pencarian dengan node. Jika node baru tersebut tidak bertabrakan dengan hambatan, maka algoritma RRT akan menambahkan node baru kepada pohon pencarian lalu melanjutkan iterasi selanjutnya.

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. $\quad q_{rand} \leftarrow RandomSample(k)$
5. $\quad q_{nearest} \leftarrow NearestNeighbor(q_{rand}, Q_{near}, T)$
6. $\quad q_{new} \leftarrow Steer\ (q_{nearest}, q_{rand}, \Delta q)$
7. $\quad$ **if** $Obstaclefree(q_{new}, q_{nearest})$ **then**
8. $\quad\quad Q_{near} \leftarrow Near(T, q_{new})$
9. $\quad\quad q_{parent} \leftarrow ChooseParent(q_{new}, Q_{near}, q_{nearest})$
10. $\quad\quad T \leftarrow InsertNode(q_{parent}, q_{new}, T)$
11. $\quad\quad T \leftarrow Rewire(T, Q_{near}, q_{parent}, q_{new})$
12. **end**

Gambar 2.3 Algoritma RRT* [14]

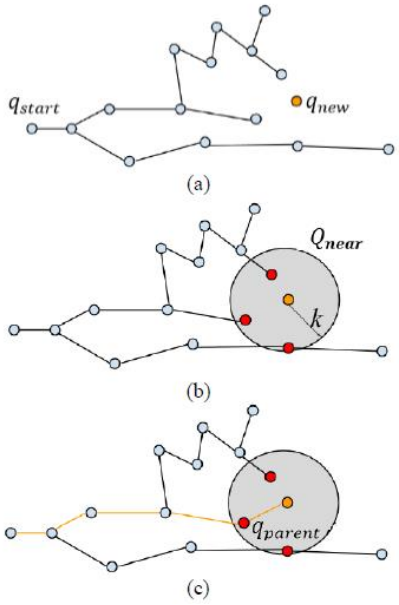
Pada proses *Chooseparent*, akan dicari node-node yang berjarak terdekat dari q_{new}. Node-node yang terdekat dari q_{new} ini dinamakan Q_{near} (baris 8 **Gambar 2.3**). Diantara Q_{near} ini akan dicari node yang bila dihubungkan ke *qnew* akan diperoleh jalur antara q_{new} dan node awal yang paling minimal. Node dari Q_{near} yang akan membuat jarak antara q_{new} dan node awal minimal, akan dijadikan node *parent* dari q_{new} (**Gambar 2.4**).

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

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

Gambar 2.4 *Chooseparent* pada algoritma RRT* [14]

Pada **Gambar 2.5** terdapat proses *rewire*. Pada proses *rewire* akan dicari node-node yang terdekat dengan q_{new} dimana jika node tersebut dihubungkan ke node awal melalui q_{new} akan diperoleh jalur yang lebih dekat. Maka *parent* baru dari node-node tersebut akan diubah menjadi q_{new}. Sebelum proses *rewire*, node q_a terhubung dengan node q_b. Jalur menuju node awal cukup jauh karena melalui jalur beliku-liku, lalu setelah proses *rewire*, node q_a terhubung dengan q_{new} menuju node awal, dimana jalur baru ini lebih minimal jaraknya daripada jarak sebelum proses *rewire*.

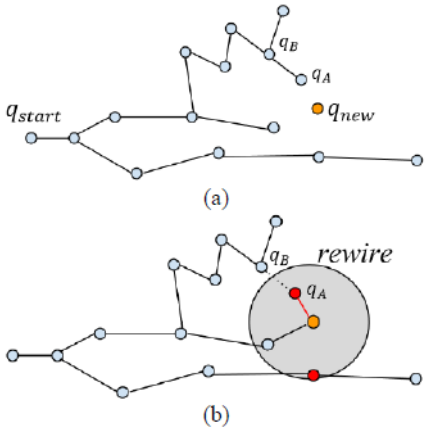


Gambar 2.5 Proses dari *choose parent* pada RRT* [13]

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

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

Gambar 2.6 *Rewire* pada algoritma RRT* [14]



Gambar 2.7 Proses dari *rewire* pada RRT* [13]

Algorithm 1: Informed RRT*$(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}})$

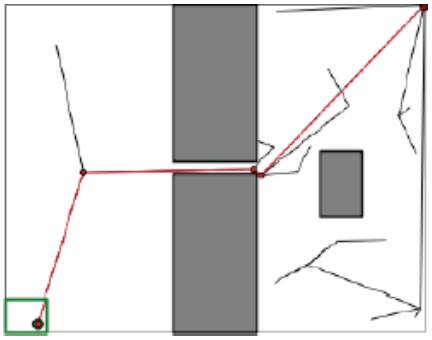
$V \leftarrow \{\mathbf{x}_{\text{start}}\}$;
$E \leftarrow \emptyset$;
$X_{\text{soln}} \leftarrow \emptyset$;
$\mathcal{T} = (V, E)$;
for iteration $= 1 \dots N$ **do**
  $c_{\text{best}} \leftarrow \min_{\mathbf{x}_{\text{soln}} \in X_{\text{soln}}} \{\texttt{Cost}(\mathbf{x}_{\text{soln}})\}$;
  $\mathbf{x}_{\text{rand}} \leftarrow \texttt{Sample}(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}, c_{\text{best}})$;
  $\mathbf{x}_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, \mathbf{x}_{\text{rand}})$;
  $\mathbf{x}_{\text{new}} \leftarrow \texttt{Steer}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{rand}})$;
  if $\texttt{CollisionFree}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}})$ **then**
    $V \leftarrow \cup \{\mathbf{x}_{\text{new}}\}$;
    $X_{\text{near}} \leftarrow \texttt{Near}(\mathcal{T}, \mathbf{x}_{\text{new}}, r_{\text{RRT}*})$;
    $\mathbf{x}_{\min} \leftarrow \mathbf{x}_{\text{nearest}}$;
    $c_{\min} \leftarrow \texttt{Cost}(\mathbf{x}_{\min}) + c \cdot \texttt{Line}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}})$;
    for $\forall \mathbf{x}_{\text{near}} \in X_{\text{near}}$ **do**
      $c_{\text{new}} \leftarrow \texttt{Cost}(\mathbf{x}_{\text{near}}) + c \cdot \texttt{Line}(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}})$;
      if $c_{\text{new}} < c_{\min}$ **then**
        if $\texttt{CollisionFree}(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}})$ **then**
          $\mathbf{x}_{\min} \leftarrow \mathbf{x}_{\text{near}}$;
          $c_{\min} \leftarrow c_{\text{new}}$;
    $E \leftarrow E \cup \{(\mathbf{x}_{\min}, \mathbf{x}_{\text{new}})\}$;
    for $\forall \mathbf{x}_{\text{near}} \in X_{\text{near}}$ **do**
      $c_{\text{near}} \leftarrow \texttt{Cost}(\mathbf{x}_{\text{near}})$;
      $c_{\text{new}} \leftarrow \texttt{Cost}(\mathbf{x}_{\text{new}}) + c \cdot \texttt{Line}(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{near}})$;
      if $c_{\text{new}} < c_{\text{near}}$ **then**
        if $\texttt{CollisionFree}(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{near}})$ **then**
          $\mathbf{x}_{\text{parent}} \leftarrow \texttt{Parent}(\mathbf{x}_{\text{near}})$;
          $E \leftarrow E \setminus \{(\mathbf{x}_{\text{parent}}, \mathbf{x}_{\text{near}})\}$;
          $E \leftarrow E \cup \{(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{near}})\}$;
    if $\texttt{InGoalRegion}(\mathbf{x}_{\text{new}})$ **then**
      $X_{\text{soln}} \leftarrow X_{\text{soln}} \cup \{\mathbf{x}_{\text{new}}\}$;
return $\mathcal{T}$;

Gambar 2.8 Algoritma *Informed-RRT** [2]

Algoritma *Informed-RRT** merupakan pengembangan dari algoritma RRT*. *Informed-RRT** memiliki prinsip kerja yang sama dengan algoritma RRT*, namun untuk *Informed-RRT** terdapat beberapa perbedaan. Pada *Informed-RRT** ketika telah ditemukan solusi, maka pencarian difokuskan disekitar goal yang membentuk daerah ellips, dengan demikian daerah pencarian pun dipersempit.

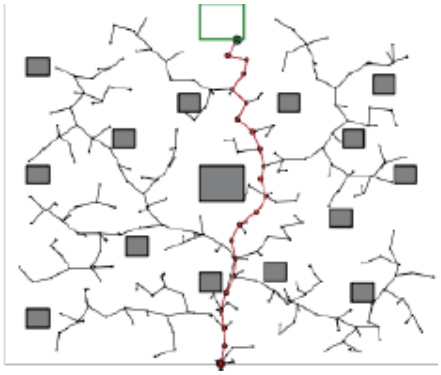
2.3 Lingkungan *Obstacle*

Lingkungan *obstacle* yang digunakan dalam pengujian algoritma RRT* dan *Informed-RRT** memiliki kasus tertentu. Terdapat beberapa lingkungan khusus yang menjadi *benchmark* pengujian algoritma perencanaan jalur. Lingkungan khusus tersebut yaitu lingkungan *obstacle narrow*, *trap*, serta *clutter*. Lingkungan *obstacle narrow* dapat dilihat pada **Gambar 2.9**. Lingkungan *narrow* merupakan lingkungan dimana *obstacle* terbentuk menjadi celah sempit, dan jalur yang dibangkitkan nantinya akan melewati celah tersebut untuk mecapai goal node.

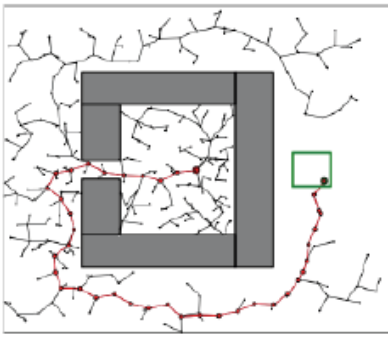


Gambar 2.9 Lingkungan *Obstacle Narrow* [13]

Lingkungan *obstacle* selanjutnya yaitu lingkungan *clutter*. Lingkungan *clutter* terdapat pada **Gambar 2.10**. Lingkungan *clutter* terbentuk dari beberapa *obstacle* yang berantakan dan tidak menentu. Sedangkan untuk lingkungan *obstacle trap* terbentuk menjadi sebuah perangkap. Goal node diposisikan didalam perangkap yang berbentuk kotak tersebut, dan nantinya program akan mencari jalan untuk masuk kedalam perangkap dan menemukan goal node. Lingkungan *obstacle trap* dapat dilihat pada **Gambar 2.11**.



Gambar 2.10 Lingkungan *Obstacle Clutter* [13]



Gambar 2.11 Lingkungan *Obstacle Trap*[13]