BAB 2

LANDASAN TEORI

Penelitian ini disusun dengan melakukan kajian literatur yang berkaitan untuk mendukung perancangan pengembangan algoritma perencanaan jalur dengan menggunakan hibridisasi algoritma *Informed* PRM dan algoritma BIT*.

2.1 Algoritma Perencanaan Jalur

Algoritma perencanaan jalur (*path planning*) adalah salah satu bidang penelitian dasar robotika dan telah banyak diteliti serta dikembangkan untuk algoritma pemrograman kendaraan atau robotika [5]. Masalah perencanaan jalur bertujuan untuk menemukan lintasan robot dari keadaan awal (*start*) sampai keadaan tujuan (*goal*) [6], dengan menjauhi rintangan maupun batas jalur yang ada [7]. Algoritma perencanaan jalur banyak digunakan dalam berbagai macam aplikasi seperti otomatisasi [8], navigasi robot [9], mobil tanpa pengemudi [10], animasi kepribadian digital [11], bedah robotik [12], serta dalam bidang kimia yaitu proses pelipatan protein [13]. Algoritma perencanaan jalur yang berbeda memiliki kinerja atau kualitas yang berbeda juga. Waktu perencanaan dan biaya atau optimasi jalur dapat menjadi beberapa parameter untuk mengukur kinerja algoritma perencanaan jalur [14], [15]. Oleh karena itu, algoritma perencanaan jalur diharapkan dapat memberikan solusi jalur yang optimal dalam waktu singkat [16]. Beberapa metode yang digunakan dalam perencanaan jalur ini antara lain metode perencanaan jalur berbasis sensor, perencanaan jalur berbasis grafik, serta perencanaan jalur berbasis pengambilan sampel acak [17].

Dalam metode perencanaan jalur berbasis sensor, robot beroperasi di lingkungan yang belum dipetakan. Robot mengumpulkan berbagai informasi tentang lingkungan atau area yang dilaluinya, berdasarkan informasi tersebut robot dapat bergerak selangkah demi selangkah. Kemudian terdapat metode perencanaan jalur berbasis grafik yang terbagi menjadi beberapa bagian, yaitu algoritma perencanaan jalur Djikstra [28], A* [29], D* [30], dan AD* [18]. Algoritma-algoritma tersebut termasuk dalam perencanaan jalur berbasis grafik yang paling umum digunakan. Namun, algoritma-algoritma tersebut memiliki beberapa kelemahan. Seperti algoritma perencanaan jalur Djikstra yang hanya dapat digunakan untuk mencari jalur terpendek dalam graf dengan bobot non-negatif, yang berarti bahwa setiap jalur pada graf memiliki nilai yang positif atau sama dengan nol. Dalam algoritma Djikstra bobot ini digunakan untuk menghitung jarak terpendek antara dua titik pada graf. Jika ada jalur dengan bobot negatif pada graf, maka algoritma Djikstra tidak dapat memberikan hasil yang benar, karena dapat terjadi siklus negatif yang menyebabkan nilai jarak terpendek tidak konvergen. Selain itu, algoritma Djikstra tidak efisien dalam graf yang sangat besar atau kompleks, karena memerlukan waktu dan memori yang besar untuk melakukan pencarian. Serta tidak dapat menangani perubahan dinamis pada graf, sehingga jika ada perubahan pada graf, algoritma harus dijalankan kembali dari awal [28].

Metode perencanaan jalur lain yang umum digunakan yaitu algoritma *Sampling Based Planning* (SBP). Algoritma ini menggunakan pengambilan sampel acak pada ruang pencarian serta memiliki kemampuan memberikan

solusi yang cepat untuk masalah yang kompleks dan multidimensi [6]. Contoh metode SBP yang paling umum digunakan adalah algoritma *Rapidly-exploring Random Tree* (RRT) [18] dan algoritma *Probabilistic Road-Map* (PRM) [19]. Algoritma RRT yang dikemukakan oleh S. M. LaValle berhasil diterapkan untuk memecahkan masalah yang kompleks dalam waktu yang singkat, namun memiliki kelemahan yaitu memberikan solusi jalur yang sub-optimal. Kemudian dikembangkan oleh Karaman dan Frazzoli yang diberi nama algoritma *Rapidly-exploring Random Tree Star* (RRT*) [19] memiliki kemampuan memberikan solusi yang asimtotik optimal, namun memiliki kelemahan yaitu waktu komputasi yang dihasilkan lambat. J. D. Gammell, dkk.23 [20], [21] memperkenalkan algoritma *informed* RRT* yang menggunakan pengambilan sampel berdasarkan informasi dari RRT* yang melewati sebagian ruang konfigurasi. Algoritma ini menghasilkan solusi yang lebih baik baik, namun memiliki kelemahan yaitu hanya dapat mempercepat proses pengoptimalan perencanaan jalur. Selain itu, ada masalah lain pada metode perencanaan jalur satu arah yang membutuhkan waktu untuk mencapai konfigurasi target, terutama ketika target tersembunyi di balik lorong sempit. Ada beberapa metode perencanaan jalur berbasis RRT lainnya yang dapat menemukan solusi lebih baik daripada versi RRT*, yaitu algoritma PRM [19] yang membangun peta probabilitas rintangan menggunakan sampel acak. Algoritma ini dapat menghasilkan solusi jalur yang optimal jika waktu yang diberikan cukup, namun kelemahan dari algoritma PRM tidak dapat bersifat asimtotik optimal. Selain itu, ada algoritma *Batch Informed Trees* (BIT*) [22]

yang merupakan hibridisasi antara algoritma PRM dan algoritma *informed* RRT*, yang menggunakan pendekatan *batch processing* dan *informed search*. Sejumlah titik yang dihasilkan secara bersamaan dan digunakan untuk memperbaiki jalur yang ada. Sehingga BIT* memiliki kemampuan menghasilkan solusi jalur yang lebih optimal secara asimtotik, meskipun memerlukan waktu komputasi yang lambat.

Algoritma *informed* PRM termasuk ke dalam bagian dari algoritma SBP, sedangkan algoritma BIT* merupakan bagian dari metode perencanaan jalur berbasis grafik. Sepengetahuan penulis, saat ini belum ada penelitian yang mencoba untuk meningkatkan kualitas dari algoritma SBP dengan menggabungkan algoritma *Informed* PRM yang merupakan bagian dari SBP dan algoritma BIT* yang merupakan bagiann dari metode perencanaan jalur berbasis grafik untuk meningkatkan performansi algoritma perencanaan jalur.

2.2 Algoritma *Informed Probabilistic Road-Map* (*Informed* PRM)

Informed Probabilistic Road-Map (*Informed* PRM) adalah algoritma perencanaan jalur yang menggabungkan algoritma *Probabilistic Road-Map* (PRM) dengan algoritma *Informed* RRT* [26]. Algoritma ini bertujuan untuk menghasilkan jalur yang hampir optimal dengan memanfaatkan informasi yang diperoleh dari iterasi sebelumnya.

PRM adalah algoritma perencanaan jalur yang menggunakan sampel acak untuk membangun grafik jalur probabilistik. Sampel acak ini kemudian dihubungkan dengan titik sampel yang ada untuk membentuk grafik sebuah

jalur. Jalur yang optimal kemudian dapat dicari dengan menggunakan algoritma perencanaan jalur seperti Djikstra atau A*.

Informed PRM menggunakan dua metode pencarian informasi untuk meningkatkan kinerja algoritma PRM. Pertama, metode sampling yang diinformasikan digunakan untuk menghasilkan sampel acak yang lebih cerdas. Sampel acak ini dihasilkan dengan membatasi area pencarian hanya pada area yang memiliki peluang tinggi untuk menghasilkan jalur optimal. Metode ini memanfaatkan informasi dari jalur terpendek yang berhasil direncanakan pada iterasi sebelumnya.

Kedua, metode pencarian lokal digunakan untuk memperbaiki jalur yang dihasilkan oleh PRM. Metode ini mencari jalur lokal yang lebih baik dengan mempertimbangkan lingkungan sekitarnya. Metode ini membantu meningkatkan kualitas jalur yang dihasilkan oleh algoritma PRM.

Dalam penelitian yang dilakukan, algoritma *Informed* PRM berhasil menghasilkan jalur yang hampir optimal untuk semua kasus yang diberikan. Algoritma ini juga berhasil mengungguli algoritma RRT* dan algoritma *Informed* RRT* dalam hal waktu komputasi dan biaya jalur. Dengan demikian, algoritma *Informed* PRM menawarkan solusi yang lebih efisien dan dapat diimplementasikan dalam berbagai sistem yang membutuhkan algoritma perencanaan jalur optimal, seperti pada sistem bedah robotik medis atau sistem kendaraan otonom.

Algoritma *Informed* PRM adalah algoritma perencanaan jalur yang digunakan untuk menemukan jalur minimum antara dua titik dalam lingkungan yang penuh dengan rintangan atau hambatan. Adapun *pseudocode* algoritma *Informed* PRM ditunjukkan pada **Gambar 2.1** berikut ini.

Algorithm 1. $X_{sol} = (n, map)$
1. **while** *termination condition not met* **do**
2. $V \leftarrow \emptyset$
3. $E \leftarrow \emptyset$
4. $X_{soln} \leftarrow \emptyset$
5. **while** $|V| < n$ **do**
6. **repeat**
7. $c_{best} \leftarrow min(x_{soln} \in X_{soln})\{Cost(x_{soln})\}$
8. $x_{rand} \leftarrow$ Sample $(x_{start}, x_{goal}, c_{best})$
9. **until** q is collision-free
10. $V \leftarrow V \cup \{q\}$
11. **end while**
12. **for all** $q \in V$ **do**
13. $N_q \leftarrow$ the neighbors of q chosen from V according to $dist$
14. **for all** $q' \in N_q$ **do**
15. **if** (q, q') is collision-free **then**
16. $E \leftarrow E \cup \{(q, q')\}$
17. **end if**
18. **end for**
19. **end for**
20. $T = (V, E)$
21. $X_{sol} \leftarrow shortest\ path(q_{init}, q_{goal}, T)$ using Djikstra Algorithm
22. **End while**

Gambar 2. 1 *Pseudocode* Algoritma *Informed* PRM

Pseudocode algoritma *Informed* PRM yang ditunjukkan pada **Gambar 2.1** terdiri dari beberapa langkah utama. Pertama, algoritma dimulai dengan inisialisasi titik awal, titik akhir, dan lingkungan yang diberikan. Selanjutnya, algoritma akan membuat sebuah *roadmap* kosong yang akan digunakan untuk merepresentasikan lingkungan. Selama proses iterasi, algoritma akan secara acak membuat sampel di lingkungan. Kemudian, algoritma akan mencoba menghubungkan sampel tersebut dengan titik-titik lain yang sudah ada dalam *roadmap*. Jika koneksi antara sampel dan titik lain berhasil dibuat, maka *edge*

akan ditambahkan ke *roadmap*. Langkah ini akan terus diulang hingga *roadmap* terbentuk dengan baik.

Setelah *roadmap* terbentuk, algoritma *Informed* PRM akan menggunakan algoritma Djikstra untuk mencari jalur minimum dari titik awal ke titik akhir. Algoritma Djikstra akan menentukan jalur optimal dengan mempertimbangkan bobot dari setiap *edge* yang terhubung dalam *roadmap*. Algoritma *Informed* PRM pada **Gambar 2.1** tidak dibatasi oleh daerah pencarian tertentu, sehingga menghasilkan *roadmap* yang lebih luas. Namun, hal ini juga dapat mempengaruhi waktu yang dibutuhkan untuk membangun *roadmap*. Algoritma ini cocok digunakan dalam lingkungan dengan rintangan yang tidak terlalu padat, namun mungkin tidak efisien dalam lingkungan yang sangat kompleks atau penuh dengan rintangan.

Pada *pseudocode* algoritma *Informed* PRM yang ditunjukkan oleh **Gambar 2.2** berikut ini adalah metode perencanaan jalur yang menggabungkan algoritma PRM dengan algoritma *Informed* RRT*.

Algorithm 2. Sample $(x_{start}, x_{goal}, c_{best})$

1. **if** $c_{max} < \infty$ **then**
2. **if** $|V| < n/2$ **then**
3. $c_{min} \leftarrow \|x_{goal} - x_{start}\|_2$
4. $x_{centre} \leftarrow (x_{goal} + x_{start})/2$
5. $C \leftarrow$RotationToWorldFrame(x_{start}, x_{goal})
6. $r_1 \leftarrow c_{max}/2$
7. $\{r_i\}_{i=2,\dots,n} \leftarrow \left(\sqrt{c_{max}^2 - c_{min}^2}\right)/2$
8. $L \leftarrow$diag$\{r_1, r_2, \dots, r_n\}$
9. $x_{ball} \leftarrow$SampleUnitBall
10. $x_{rand} \leftarrow (CLx_{ball} + x_{centre}) \cap X$
11. **else**
12. $x_{rand} \leftarrow$ SamplingNearBestPath(X_{sol}, d)
13. **end if**
14. **else**
15. $x_{rand} \leftarrow$RandomSampling(map)
16. **end if**
17. **return** x_{rand}

Gambar 2. 2 *Pesudocode* Sample Algoritma *Informed* PRM

Langkah-langkahnya meliputi inisialisasi titik awal, titik akhir, dan lingkungan yang diberikan. Selanjutnya, algoritma akan membuat sebuah *roadmap* kosong yang akan digunakan untuk merepresentasikan lingkungan. Selama melakukan proses iterasi untuk membentuk *roadmap*, dimulai dengan menciptakan sampel acak di dalam daerah pencarian yang diinformasikan (*ellipsoid subset*). Daerah pencarian yang diinformasikan ini membantu membatasi sampel acak agar lebih fokus pada area yang berpotensi menghasilkan jalur optimal. Setelah itu, dilakukan pengecekan untuk memastikan sampel terhubung dengan titik lain pada *roadmap*. Jika terdapat koneksi yang memungkinkan antara sampel-sampel tersebut, maka sebuah *edge* ditambahkan ke dalam *roadmap*.

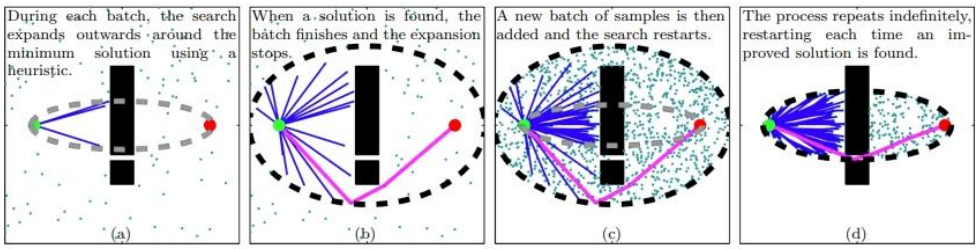
Selama proses iterasi, algoritma akan terus memperbarui *eccentricity ellipsoid* berdasarkan panjang jalur terpendek saat ini. Hal ini memungkinkan algoritma untuk menyesuaikan daerah pencarian yang diinformasikan sesuai

dengan perkembangan *roadmap* dan jalur terbaik yang telah ditemukan. Setelah *roadmap* telah terbentuk, algoritma akan melakukan perhitungan panjang jalur terpendek saat ini dari titik awal ke titik akhir menggunakan algoritma Djikstra. Hasil dari perhitungan ini memberikan gambaran tentang jalur optimal pada *roadmap* tersebut. Setelah panjang jalur terpendek teridentifikasi, langkah selanjutnya adalah menyesuaikan *eccentricity ellipsoid* berdasarkan panjang jalur terpendek yang telah dihitung. Proses ini membantu dalam meningkatkan akurasi *roadmap* dengan mempertimbangkan jalur optimal yang telah ditemukan. Proses iterasi ini berlanjut untuk mencapai *roadmap* yang semakin optimal dan sesuai dengan kondisi terkini. Dengan menggunakan daerah pencarian yang diinformasikan dan mengadaptasi *eccentricity ellipsoid*, *pseudocode* algoritma *Informed* PRM pada **Gambar 2.2** tersebut dapat membantu mempercepat pencarian jalur optimal dan menghasilkan solusi yang lebih efisien dalam lingkungan yang kompleks dengan rintangan.

2.3 Algoritma *Batch Informed Trees* (BIT*)

Algoritma *Batch Informed Trees* (BIT*) adalah salah satu algoritma yang digunakan dalam masalah perencanaan jalur untuk mencari jalur terpendek antara dua titik dalam ruang pencarian. Algoritma ini merupakan pengembangan dari algoritma A* yang dapat diurutkan berdasarkan solusi minimum yang diusulkan oleh heuristik untuk pencarian jalur terpendek [31]. Algoritma BIT* adalah algoritma perencanaan jalur yang menggabungkan teknik perencanaan berbasis graf dan teknik perencanaan berbasis sampel dengan menggunakan teori *Random Geometric Graphs* (RGG) dan teknik

pencarian bertahap [32]. Algoritma ini mengintegrasikan efisiensi teknik berbasis graf seperti A* dengan skabilitas dari algoritma berbasis sampel seperti *Rapidly-exploring Random Tree* (RRT). Algoritma BIT* menggunakan heuristik untuk semua aspek biaya jalur, untuk memprioritaskan pencarian jalur berkualitas tinggi dan memfokuskan pencarian untuk perbaikan, menghasilkan solusi yang lebih baik dan konvergensi lebih cepat menuju optimal dibandingkan dengan algoritma perencanaan optimal berbasis sampel lainnya dan RRT [22]. Algoritma ini konvergen secara asimtotik ke solusi optimal dengan probabilitas yang hampir pasti, berarti bahwa algoritma BIT* dapat mencapai solusi yang optimal dalam jangka panjang. Algoritma BIT* juga dapat menghasilkan solusi yang memadai dalam waktu terbatas dan terus meningkatkan solusinya seiring waktu yang lebih lama [16]. Dengan menggunakan teknik ini, algoritma BIT* berhasil menggabungkan keuntungan perencanaan berbasis graf dan sampel, menciptakan pendekatan yang menjanjikan untuk memecahkan masalah perencanaan kontinu. Ilustrasi dari prosedur pencarian yag diinformasikan oleh algoritma BIT* ditunjukkan oleh **Gambar 2.3**.



Gambar 2. 3 Ilustrasi pencarian algoritma BIT*

Ilustrasi pada **Gambar 2.3** tersebut menunjukkan langkah-langkah pencarian yang dilakukan oleh algoritma BIT* dalam menemukan solusi optimal. Berikut adalah penjelasan mengenai setiap bagian dari ilustrasi tersebut:

a. Pada gambar (a) menunjukkan pencarian yang sedang berkembang dari *batch* pertama sampel. Pencarian dimulai dari titik awal (ditandai dengan warna hijau) dan bergerak menuju tujuan (ditandai dengan warna merah) melalui serangkaian sampel yang dihasilkan.
b. Pada gambar (b), pencarian pertama berakhir ketika solusi ditemukan. Solusi saat ini ditandai dengan warna magenta, dan pencarian dihentikan setelah solusi ditemukan.
c. Setelah pemangkasan dan penambahan *batch* kedua sampel, pada gambar (c) pencarian dimulai kembali pada graf yang lebih padat. Proses ini memungkinkan algoritma untuk fokus pada submasalah yang mungkin mengandung solusi yang lebih baik.
d. Gambar (d) ini menunjukkan bahwa pencarian kedua berakhir ketika solusi yang lebih baik ditemukan. Hal ini menunjukkan kemampuan algoritma BIT* untuk terus meningkatkan solusi yang ditemukan melalui iterasi pencarian yang berulang.

Ilustrasi ini memberikan gambaran visual tentang bagaimana algoritma BIT* melakukan pencarian yang diinformasikan untuk menemukan solusi optimal, serta bagaimana algoritma tersebut terus meningkatkan solusi yang

ditemukan melalui penggunaan *batch* sampel yang berulang. Adapun *pseudocode* dari algoritma BIT* dapat dilihat pada **Gambar 2.4** berikut ini.

Algorithm 1: BIT*$(\mathbf{x}_{\text{start}} \in X_{\text{free}}, \mathbf{x}_{\text{goal}} \in X_{\text{goal}})$

$V \leftarrow \{\mathbf{x}_{\text{start}}\}$; $E \leftarrow \emptyset$; $X_{\text{samples}} \leftarrow \{\mathbf{x}_{\text{goal}}\}$;
$\mathcal{Q}_E \leftarrow \emptyset$; $\mathcal{Q}_V \leftarrow \emptyset$; $r \leftarrow \infty$;
repeat
  if $\mathcal{Q}_E \equiv \emptyset$ **and** $\mathcal{Q}_V \equiv \emptyset$ **then**
    `Prune` $\left(g_{\mathcal{T}}\left(\mathbf{x}_{\text{goal}}\right)\right)$;
    $X_{\text{samples}} \xleftarrow{+}$ `Sample` $\left(m, g_{\mathcal{T}}\left(\mathbf{x}_{\text{goal}}\right)\right)$;
    $V_{\text{old}} \leftarrow V$;
    $\mathcal{Q}_V \leftarrow V$;
    $r \leftarrow$ `radius` $\left(|V| + |X_{\text{samples}}|\right)$;
  while `BestQueueValue` $(\mathcal{Q}_V) \leq$ `BestQueueValue` $(\mathcal{Q}_E)$ **do**
    `ExpandVertex` (`BestInQueue` $(\mathcal{Q}_V)$);
  $(\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}}) \leftarrow$ `BestInQueue` $(\mathcal{Q}_E)$;
  $\mathcal{Q}_E \xleftarrow{-} \{(\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}})\}$;
  if $g_{\mathcal{T}}(\mathbf{v}_{\text{m}}) + \widehat{c}(\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}}) + \widehat{h}(\mathbf{x}_{\text{m}}) < g_{\mathcal{T}}\left(\mathbf{x}_{\text{goal}}\right)$ **then**
    if $\widehat{g}(\mathbf{v}_{\text{m}}) + c(\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}}) + \widehat{h}(\mathbf{x}_{\text{m}}) < g_{\mathcal{T}}\left(\mathbf{x}_{\text{goal}}\right)$ **then**
      if $g_{\mathcal{T}}(\mathbf{v}_{\text{m}}) + c(\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}}) < g_{\mathcal{T}}(\mathbf{x}_{\text{m}})$ **then**
        if $\mathbf{x}_{\text{m}} \in V$ **then**
          $E \xleftarrow{-} \{(\mathbf{v}, \mathbf{x}_{\text{m}}) \in E\}$;
        else
          $X_{\text{samples}} \xleftarrow{-} \{\mathbf{x}_{\text{m}}\}$;
          $V \xleftarrow{+} \{\mathbf{x}_{\text{m}}\}$; $\mathcal{Q}_V \xleftarrow{+} \{\mathbf{x}_{\text{m}}\}$;
        $E \xleftarrow{+} \{(\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}})\}$;
        $\mathcal{Q}_E \xleftarrow{-} \{(\mathbf{v}, \mathbf{x}_{\text{m}}) \in \mathcal{Q}_E \mid g_{\mathcal{T}}(\mathbf{v}) + \widehat{c}(\mathbf{v}, \mathbf{x}_{\text{m}}) \geq g_{\mathcal{T}}(\mathbf{x}_{\text{m}})\}$;
  else
    $\mathcal{Q}_E \leftarrow \emptyset$; $\mathcal{Q}_V \leftarrow \emptyset$;
until `STOP`;
return $\mathcal{T}$;

Gambar 2. 4 *Pseudocode* Algoritma Dasar BIT*

Algoritma BIT* diawali dengan langkah inisialisasi, dimana variabel-variabel seperti titik awal, titik tujuan, himpunan simpul (V), himpunan tepi (E), dan himpunan sampel ($X_{samples}$) diinisialisasi. Selanjutnya, variabel Q_E dan Q_V digunakan untuk menyimpan simpul dan tepian yang akan di proses pada setiap iterasi pencarian. Variabel r digunakan untuk menyimpan jarak minimum antara titik awal dan titik tujuan.

Setelah inisialisasi, algoritma memasukkan titik awal ke dalam himpunan simpul (V) dan titik tujuan ke dalam himpunan sampel ($X_{samples}$). Algoritma BIT* kemudian memulai iterasi pencarian. Pencarian dilakukan selama himpunan tepi (E) dan himpunan simpul (V) tidak kosong. Pada setiap iterasi, algoritma memilih tepian dengan biaya terendah dari himpunan tepi (E) untuk diproses. Selanjutnya, algoritma memeriksa apakah tepian tersebut dapat meningkatkan solusi saat ini. Jika tepian tidak dapat meningkatkan solusi, algoritma akan menghapus semua tepian dan simpul yang tersisa, lalu memulai *batch* sampel yang baru. Namun, jika tepian dapat meningkatkan solusi, maka algoritma akan memeriksa apakah tepian tersebut merupakan tepian tujuan. Jika tepian tersebut merupakan tepian tujuan, maka algoritma menghentikan pencarian dan mengembalikan solusi saat ini. Jika tepian tersebut bukan tepian tujuan, maka algoritma memperluas pohon pencarian dengan menambahkan simpul baru ke dalam himpunan simpul (V) dan tepian baru ke dalam himpunan tepi (E). Kemudian, algoritma akan memeriksa apakah simpul baru dapat meningkatkan solusi saat ini dan memperbaiki solusi di masa depan. Jika simpul baru dapat meningkatkan solusi, maka ditambahkan ke dalam himpunan sampel ($X_{samples}$). Algoritma juga akan memeriksa ukuran $X_{samples}$, jika telah mencapai batas tertentu maka algoritma akan memangkas $X_{samples}$ untuk mengurangi kompleksitas graf. Setelah itu, algoritma memperbarui nilai r berdasarkan jarak minimum antara titik awal dan titik tujuan. Algoritma kemudian memperbarui himpunan tepi (E) dan himpunan simpul (V) berdasarkan simpul dan tepian yang baru ditambahkan. Proses ini terus diulang hingga algoritma menemukan

solusi atau tidak dapat memperluas pohon pencarian lagi. Bagian *pseudocode* algoritma BIT* untuk memperluas simpul (*vertex*) dan menghapus sampel dapat dilihat pada **Gambar 2.5** dan **Gambar 2.6** berikut ini.

Algorithm 2: ExpandVertex($\mathbf{v} \in \mathcal{Q}_V \subseteq V$)

$\mathcal{Q}_V \xleftarrow{-} \{\mathbf{v}\}$;
$X_{\text{near}} \leftarrow \left\{\mathbf{x} \in X_{\text{samples}} \mid \|\mathbf{x}-\mathbf{v}\|_2 \leq r\right\}$;
$\mathcal{Q}_E \xleftarrow{+} \left\{(\mathbf{v},\mathbf{x}) \in V \times X_{\text{near}} \mid \widehat{g}(\mathbf{v}) + \widehat{c}(\mathbf{v},\mathbf{x}) + \widehat{h}(\mathbf{x}) < g_T(\mathbf{x}_{\text{goal}})\right\}$;
if $\mathbf{v} \notin V_{\text{old}}$ **then**
  $V_{\text{near}} \leftarrow \left\{\mathbf{w} \in V \mid \|\mathbf{w}-\mathbf{v}\|_2 \leq r\right\}$;
  $\mathcal{Q}_E \xleftarrow{+} \left\{(\mathbf{v},\mathbf{w}) \in V \times V_{\text{near}} \mid (\mathbf{v},\mathbf{w}) \notin E, \widehat{g}(\mathbf{v}) + \widehat{c}(\mathbf{v},\mathbf{w}) + \widehat{h}(\mathbf{w}) < g_T(\mathbf{x}_{\text{goal}}), g_T(\mathbf{v}) + \widehat{c}(\mathbf{v},\mathbf{w}) < g_T(\mathbf{w})\right\}$;

Gambar 2. 5 *Pseudocode Expand Vertex* Algoritma BIT*

Algorithm 3: Prune($c \in R_{\geq 0}$)

$X_{\text{samples}} \xleftarrow{-} \left\{\mathbf{x} \in X_{\text{samples}} \mid \widehat{f}(\mathbf{x}) \geq c\right\}$;
$V \xleftarrow{-} \left\{\mathbf{v} \in V \mid \widehat{f}(\mathbf{v}) > c\right\}$;
$E \xleftarrow{-} \left\{(\mathbf{v},\mathbf{w}) \in E \mid \widehat{f}(\mathbf{v}) > c, \text{ or } \widehat{f}(\mathbf{w}) > c\right\}$;
$X_{\text{samples}} \xleftarrow{+} \{\mathbf{v} \in V \mid g_T(\mathbf{v}) \equiv \infty\}$;
$V \xleftarrow{-} \{\mathbf{v} \in V \mid g_T(\mathbf{v}) \equiv \infty\}$;

Gambar 2. 6 *Pseudocode Prune* Algoritma BIT*

Pada *pseudocode* **Gambar 2.5** adalah langkah-langkah dalam algoritma BIT* yang bertanggung jawab untuk memperluas simpul (*vertex*) yang ada dalam antrian simpul (*vertex queue*). Proses ini melibatkan pengambilan simpul dari antrian, pencarian sampel-sampel terdekat, dan penambahan tepian baru ke dalam antrian tepian berdasarkan kriteria tertentu. Langkah-langkahnya termasuk mengambil simpul (V) dari antrian, mencari sampel terdekat, mengevaluasi kandidat tepian, dan menambahkan tepian yang memenuhi syarat

kedalam antrian tepian [22]. *Pseudocode* algoritma 2 pada **Gambar 2.5** merupakan salah satu langkah kunci dalam algoritma BIT* yang memungkinkan algoritma untuk secara dinamis memperluas pohon pencarian dan mengeksplorasi ruang pencarian dengan efisien. Dengan memperluas simpul secara adaptif dan mengevaluasi tepian yang terhubung, algoritma dapat mencapai solusi optimal dengan kompleksitas waktu yang efisien.

Sedangkan *pseudocode* pada **Gambar 2.6** adalah bagian dari algoritma BIT* yang bertanggung jawab untuk menghapus simpul dan sampel yang tidak lagi berkontribusi pada peningkatan solusi atau tidak dapat memberikan solusi yang lebih baik dari biaya yang telah ditentukan. Proses ini melibatkan penghapusan sampel dan simpul berdasarkan biaya estimasi yang telah ditentukan, sehingga memastikan bahwa hanya entitas yang masih relevan dan berpotensi memberikan kontribusi pada peningkatan solusi yang dipertahankan dalam pencarian. Dengan melakukan *pruning* secara teratur, algoritma dapat mempertahankan kinerja pencarian yang optimal dan mengurangi kompleksitas graf yang tidak perlu.

Kedua algoritma ini bekerja sama untuk membantu algoritma BIT* dalam mencapai solusi optimal dengan memperluas ruang pencarian secara adaptif dan mengelola entitas yang terlibat dalam proses pencarian. Namun, algoritma BIT* juga memiliki beberapa kelemahan, yaitu membutuhkan waktu komputasi yang cukup signifikan untuk mencapai solusi optimal, terutama dalam masalah dengan ruang pencarian yang besar atau kompleks. Algoritma ini melibatkan pencarian terurut pada setiap *batch* sampel yang dapat memakan waktu cukup

lama [17]. Algoritma BIT* juga menggunakan estimasi biaya heuristik untuk memandu pencarian. Kualitas estimasi heuristik dapat mempengaruhi kinerja algoritma ini. Jika estimasi heuristik tidak akurat, algoritma dapat menghasilkan solusi yang suboptimal atau membutuhkan waktu yang lebih lama untuk mencapai solusi optimal [32]. Sepengetahuan penulis, belum ada penelitian sebelumnya yang membahas penggabungan antara algoritma BIT* dengan algoritma *Informed* PRM untuk perencanaan jalur yang lebih baik dan lebih cepat dari algoritma perencanaan jalur lainnya.

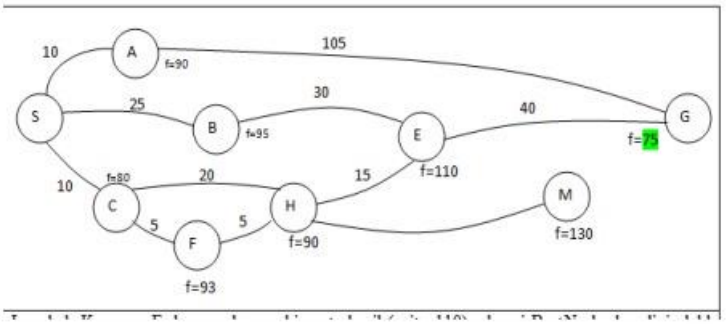
2.4 Solution Cost

Solution cost merupakan nilai biaya yang diperoleh dari jalur yang ditempuh dalam proses perencanaan jalur [16]. *Solution cost* berfungsi sebagai ukuran untuk mengevaluasi kualitas atau efisiensi dari solusi yang dihasilkan oleh algoritma perencanaan jalur. Secara umum, *solution cost* menggambarkan total biaya atau jarak yang ditempuh dari titik awal hingga titik akhir. Salah satu algoritma perencanaan jalur yang menggunakan *solution cost* adalah algoritma A* (A-*star*). Algoritma ini bertujuan untuk menemukan jalur terpendek dengan meminimalkan total *cost* yang terdiri dari:

1. Jarak sebenarnya dari titik awal ke titik saat ini (g(n)).
2. Perkiraan jarak dari titik saat ini ke tujuan akhir (h(n)),

Total *solution cost* dihitung dengan rumus: $f(n) = g(n) + h(n)$

Dimana f(n) adalah total *cost*, g(n) mewakili jarak sebenarnya dari titik awal hingga titik saat ini, dan h(n) adalah perkiraan jarak dari titik saat ini ke titik akhir. Pada **Gambar 2.7**, titik awal adalah A dan titik tujuan adalah G.



Gambar 2. 7 *Solution Cost* dalam Algoritma A*

Algoritma A* akan mencari jalur terpendek dengan meminimalkan total *solution cost* f(n). Setiap *node* memiliki nilai g(n) dan h(n) yang digunakan untuk menghitung total *solution cost*.

2.5 Time Cost

Time cost atau waktu dalam algoritma perencanaan jalur mengacu pada durasi yang diperlukan untuk memncapai tujuan dari titik awal melalui rute tertentu. *Time cost* dalam algoritma perencanaan jalur sering kali menjadi matriks utama yang digunakan untuk menentukan jalur optimal, terutama dalam aplikasi dimana kecepatan atau ketepatan waktu sangat penting, seperti pada navigasi kendaraan, robotika, atau sistem transportasi [16].

Time cost merujuk pada biaya yang dihitung berdasarkan waktu yang diperlukan untuk berpindah dari satu titik ke titik lainnya sepanjang jalur. *Time*

cost bisa dipengaruhi oleh berbagai faktor. Faktor-faktor yang mempengaruhi *time cost* meliputi:

1. Kecepatan: waktu perjalanan di sepanjang jalur sangat dipengaruhi oleh kecepatan. Jalur yang lebih panjang mungkin lebih cepat jika kecepatan di jalur tersebut lebih tinggi.
2. Kondisi jalan atau medan: Jalan yang sulit atau medan yang berat dapat memperlambat perjalanan dan menambah waktu tempuh.

Algoritma Djikstra adalah salah satu algoritma yang paling sering digunakan untuk menemukan jalur terpendek antara dua titik dalam sebuah grafik. Dalam algoritma ini, parameter waktu digunakan untuk menghitung jarak terpendek dari titik awal ke titik tujuan. Algoritma ini beroperasi dengan memperbarui secara bertahap jarak terpendek ke setiap titik dalam grafik.

2.6 Iterasi

Iterasi merupakan salah satu parameter penting dalam algoritma perencanaan jalur, dimana algoritma secara berulang memperbarui informasi mengenai jalur atau solusi hingga memenuhi kriteria tertentu, seperti menemukan jalur optimal atau mencapai batas maksimum iterasi yang ditetapkan. Iterasi mengacu pada langkah-langkah berulang yang dilakukan oleh algoritma untuk menemukan jalur dari titik awal ke titik tujuan [16].

Proses iterasi dalam algoritma perencanaan jalur melibatkan pengulangan. Algoritma perencanaan jalur mengevaluasi dan memperbarui informasi tentang jalur yang sedang dibentuk. Setiap iterasi dapat mencakup pemeriksaan *node*

atau titik baru., pembaruan jarak atau waktu tempuh, serta pemilihan jalur yang lebih optimal.

Dalam setiap iterasi, algoritma akan mempertimbangkan *node* atau jalur baru, memperbarui nilai *solution cost* (seperti jarak atau waktu), dan jika ditemukan jalur yang lebih baik, algoritma akan memilihnya. Proses iterasi ini berlanjut sampai jalur optimal ditemukan atau algoritma berhenti ketika mencapai jumlah iterasi maksimum. Jumlah iterasi yang dilakukan dapat mempengaruhi kecepatan dan efisiensi algoritma. Semakin banyak iterasi, semakin lama waktu yang dibutuhkan untuk menyelesaikan pencarian jalur, namun hal ini juga dapat meningkatkan kemungkinan menemukan solusi yang optimal.

2.7 Penelitian Terdahulu

Untuk menunjang topik penelitian mengenai pengembangan algoritma perencanaan jalur dengan hibridisasi algoritma *Informed* PRM dan algoritma BIT*, perlu adanya penelitian terdahulu yang relevan untuk dijadikan sebagai referensi atau acuan penelitian dalam mengkaji setiap penelitian terdahulu yang fmembahas terkait dengan permasalahan perencanaan jalur ditunjukkan oleh **Tabel 2.1**.

Tabel 2. 1 Penelitian Terdahulu

No	Peneliti	Metode	Deskripsi
1	S. M. LaValle (1998)	Algoritma RRT	Algoritma RRT yang diusulkan oleh S. M. Lavalle dapat memberikan solusi dengan waktu komputasi yang cepat, namun memiliki kelemahan yaitu

No	Peneliti	Metode	Deskripsi
			memberikan solusi yang sub optimal.
2	S. Karaman dan E. Frazzoli (2011)	Algoritma RRT*	Algoritma RRT* memiliki sifat asimtotik optimal dan dapat memperbaiki kualitas jalur yang dihasilkan dibanding algoritma RRT, namun terdapat kelemahan pada algoritma RRT* yaitu waktu komputasi yang didapatkan lebih lambat.
3	J. D. Gammell, dkk. (2015)	Algoritma BIT*	Algoritma BIT* memiliki sifat asimtotik optimal dengan solusi jalur yang lebih konsisten dan waktu komputasi yang lebih cepat.
4	A. Viseras, dkk. (2016)	Algoritma RRT dan ACO	Penggabungan dari algoritma RRT dan ACO ini mampu menciptakan jalur yang optimal dengan efisiensi yang baik dalam berbagai ruang konfigurasi, namun masih memerlukan waktu yang lama.
5	J. D. Gammell, dkk. (2018)	Algoritma *Informed* RRT*	Algoritma *Informed* RRT* menghasilkan solusi jalur yang lebih optimal dengan waktu komputasi yang lebih cepat dibandingkan dengan algoritma *Infored* RRT*. Proses pengambilan sampel pada algoritma ini dilakukan dalam wilayah ellipsoid yang mencakup titik awal dan titik akhir.
6	M. Aria (2020)	Algoritma RRT* dan PRM	Penelitian ini menciptakan algoritma yang memiliki kemampuan konvergensi yang cepat, namun algoritma tersebut tidak dapat mencapai solusi jalur yang bersifat asimtotik optimal
7	M. Aria (2021)	Algoritma RRT dan ACS	Penggabungan algoritma RRT dan ACS berhasil menghasilkan jalur optimal untuk perencanaan pergerakan robot dalam lingkungan yang kompleks. Namun, algoritma ini memiliki beberapa kelemahan yang terkait

No	Peneliti	Metode	Deskripsi
			dengan solusi sub optimal, waktu komputasi yang lama, dan keterbatasan dalam penggunaan pada lingkungan kontinu.
8	M. Aria (2021)	Algoritma *Informed* RRT* dan PRM	Penelitian ini menunjukkan bahwa penggabungan algoritma *Informed* RRT* dan PRM yang menghasilkan algoritma *Informed* PRM, berhasil menghasilkan jalur yang hampir optimal untuk semua kasus yang diberikan. Algoritma ini juga berhasil mengungguli algoritma RRT* dan algoritma *Informed* RRT* dalam hal waktu komputasi dan jalur yang dihasilkan. Kekurangan dari penelitian tersebut yaitu mungkin *Informed* PRM tidak jauh lebih baik dibanding dari *Informed* RRT* pada beberapa lingkungan pengujian tertentu, penelitian ini juga hanya mengusulkan contoh ruang ellipsoid antara konfigurasi awal dan tujuan.