# BAB II

# TINJAUAN PUSTAKA

## 2.1 Pyton

*Python* adalah bahasa pemrograman berorientasi objek tingkat tinggi yang dibuat oleh Guido Van Rossum [9]. filosofi desainnya menekankan kemampuan membaca kode, dan sintaknya memungkinkan pemrogram untuk mengekspresikan konsep dalam baris kode yang lebih sedikit daripada yang mungkin dilakukan dalam bahasa seperti C [10]. Konstruksi bahasanya memungkinkan pengguna untuk menulis program yang jelas, baik dalam skala kecil maupun besar [11].

Fitur terpenting dalam *pyton* adalah mendukungnya beberapa paradigma pemrograman, termasuk orientasi objek, dan prosedural imperatif [12]. *Python* mendukung sistem tipe dinamis dan manajemen memori otomatis serta memiliki perpustakaan yang besar. Pernyataan *python* tidak perlu diakhiri dengan karakter khusus. Penerjemah *python* tahu ketika anda selesai menuliskan program dengan adanya baris baru. Jika pernyataan mencakup lebih dari satu baris, tindakan paling aman adalah menggunakan garis miring terbalik di akhir baris untuk memberi tahu *python* bahwa program akan dilanjutkan di baris berikutnya [13].

## 2.2 Path Planning

*Path planning* adalah proses menemukan jalur dari titik A ke titik B, biasanya menggunakan beberapa algoritma [14]. Proses ini sangat sederhana jika tidak ada hambatan atau jalur percabangan, namun kompleksitasnya meningkat dengan cepat ketika hambatan ditambahkan [15]. Jika suatu algoritma akan memeriksa semua jalur yang memungkinkan atau memeriksa setiap lokasi yang mungkin dalam ruang lingkup grafik, eksekusinya akan terlalu lambat [16]. Oleh karena itu, dibutuhkan algoritma yang dapat memeriksa *node* sesedikit mungkin, dan juga dapat menemukan jalur tercepat ke tujuan [17].

*Planning* terdiri dari beberapa urutan tindakan yang mengubah beberapa keadaan awal menjadi beberapa keadaan tujuan yang diinginkan [18]. Sebuah jalur dapat dikatakan optimal jika jumlah biaya transisinya minimal disemua jalur yang mungkin, dimulai dari posisi awal (status awal) ke posisi tujuan (status tujuan) [19]. *Path planning* digunakan untuk memecahkan masalah di berbagai bidang, mulai dari perencanaan rute yang sederhana hingga pemilihan urutan tindakan tepat yang diperlukan untuk mencapai tujuan tertentu [20]. *Path planning* dapat digunakan di lingkungan yang sepenuhnya diketahui atau sebagian diketahui, serta lingkungan yang sama sekali tidak diketahui [21].

## 2.2 Heuristik

Heuristik adalah sebuah algoritma yang mencoba untuk menemukan contoh tertentu dari $X$ yang memaksimalkan $f$ dengan berulang memanggil fungsi heuristik [22]. Heuristik adalah metode yang melakukan satu atau lebih modifikasi pada solusi atau contoh tertentu, untuk mendapatkan solusi berbeda yang lebih unggul, atau yang mengarah ke solusi unggul [23]. Algoritma pencarian heuristik telah berhasil digunakan untuk menghitung solusi optimal untuk masalah besar [24].

Fungsi heuristik mengontrol perilaku pencarian A* dengan memperkirakan biaya minimum dari setiap *node* $n$ ke tujuan [25]. Desain heuristik membutuhkan kehati-hatian serta pertimbangan waktu yang tepat. Efektivitas pencarian A* dapat menurun saat nilai heuristik terlalu tinggi maupun rendah [26]. Kualitas fungsi heuristik mempengaruhi baik akurasi maupun kecepatan dalam pencarian.

## 2.3 Algoritma A*

A* adalah algoritma pencarian terbaik pertama yang dianggap sebagai versi lanjutan dari algoritma *Dijkstra's* [27]. Diperkenalkan pada tahun 1968 oleh Peter E. Hart, Nils J. Nilson dan Bertram Raphael [28]. Algoritma ini mencoba untuk mengurangi jumlah total keadaan yang akan dijelajahi dengan memasukkan perkiraan biaya heuristik untuk mencapai tujuan dari

keadaan tertentu. A* memiliki dua karakteristik yang membentuk fungsi baru yaitu sebagai berikut :

$$f(n) = g(n) + h(n) \quad (1)$$

Untuk setiap *node* $n$, total biaya $f(n)$ diberikan sebagai jumlah dari biaya $g(n)$ dan $h(n)$. $g$ adalah biaya yang ditempuh dari titik awal *node* menuju *node* $n$. $h$ adalah faktor heuristik yang memperkirakan biaya perpindahan dari *node* $n$ menuju *node* tujuan [29].

A* diyakini didasarkan pada algoritma yang disebutkan diatas karena A* seperti algoritma *Dijkstra's* yang menemukan jalur terpendek tanpa gagal dan seperti *Greedy Best-First Search* yang menggunakan fungsi heuristik untuk memperkirakan jarak ke tujuan. Pseudocode A* diperlihatkan pada **Gambar 2.1**.

```
function AStar
initialize start_node and calculate xy_index from start_node
initialize goal_node and calculate xy_index from goal_node
initialize unvisited_list and visited_list
calculate grid_index start_node and put into unvisited_list
while unvisited_list is not empty
        c_id = unvisited_list.cost + heuristic_value(goal_node,unvisited_list)
        current = the node in unvisited_list having the lowest c_id value
        if current = goal_node, stop the search
        remove current from unvisited_list
        add current to visited_list
        generate eight successor and calculate c_id
        for each successor
                node = find paths from eight possible directions
                n_id = calculate grid_index from node
                if node not safe, do nothing
                if n_id in visited_list, do nothing
                if n_id not in unvisited_list:
                        unvisited_list=node
                if cost from unvisited list greater than cost from node
                        unvisited_list=node
calculate final path
```

**Gambar 2.1** Pseudocode A*.

Algoritma *Dijkstra's* bekerja sangat baik untuk menemukan jalur terpendek dari satu titik akhir ke titik akhir lainnya, tetapi juga menghabiskan waktu dan sumber daya untuk menjelajahi arah yang tidak terlalu meyakinkan. Sebaliknya, *Greedy Best-First Search* mengeksplorasi arah yang memiliki harapan terbaik, tetapi gagal untuk secara konsisten menemukan jalur terpendek ke tujuan. Menggabungkan dua algoritma ini, A* menggunakan jarak dari awal dan perkiraan jarak ke tujuan untuk menghilangkan keterbatasan algoritma konvensional ini.

A* digunakan pada banyak bidang aplikasi, Akshay dkk telah melakukan penelitian mengenai efisiensi waktu pada algoritma A* untuk aplikasi robot. Dalam penelitiannya, sebuah algoritma harus dirancang untuk memastikan jalur bebas hambatan bagi robot dalam bidang industri. Modifikasi terhadap algoritma A* disajikan dalam penelitian ini. Hasil yang didapat yakni dengan algoritma A* menunjukkan pengurangan waktu pemrosesan maksimum sebesar 95% [30].

Shrawan Kr. Sharma dan B.L.Pal membahas perbandingan algoritma *Dijkstra's* dan A* untuk kebutuhan pencarian jalur terpendek pada lalulintas dalam kota. Pencarian jalan terpendek sangat penting dalam beberapa kasus khusus seperti pemadam kebakaran, ambulans yang membawa pasien, penangkapan pencuri oleh pihak kepolisian, dll. Penelitian ini mengatakan bahwa algoritma A* berkinerja lebih baik daripada algoritma *Dijkstra's* di semua kasus (dengan *obstacle* dan tanpa *obstacle*) [31].

## 2.4 Weighted A*

Untuk masalah pencarian yang sulit, A* mungkin membutuhkan terlalu lama waktu untuk menemukan solusi yang optimal, dan solusi perkiraan yang relatif lebih cepat dapat lebih berguna. Banyak peneliti telah mengeksplorasi efek pembobotan nilai $g(n)$ dan $h(n)$ dalam evaluasi *node* secara berbeda, untuk memungkinkan A* menemukan solusi optimal dengan

upaya komputasi yang lebih sedikit. Dalam pendekatan ini, fungsi evaluasi *node* didefinisikan sebagai :

$$f'(n) = g(n) + w.h(n) \quad (2)$$

Dimana parameter *weight* $w \geq 1$ ditetapkan oleh pengguna. Heuristik berbobot mempercepat solusi pencarian karena membuat *node* lebih dekat ke tujuan. Pencarian heuristik berbobot paling efektif untuk masalah pencarian dengan solusi yang mendekati optimal.

Abhinav Bhatia dkk membahas mengenai *Anytime Weighted* A*. Dalam penelitiannya, menemukan bobot terbaik merupakan tantangan karena tidak hanya bergantung pada karakteristik domain yang ada, tetapi juga waktu komputasi yang tersedia. Penelitian ini mengusulkan metode *Randomized Weighted* A*, yang secara acak menyesuaikan bobotnya pada saat *runtime*. Hasilnya adalah algoritma sederhana yang mudah diimplementasikan dan berkinerja baik [32].

Eric A. Hansen dan Rong Zhou telah melakukan penelitian mengenai cara mengubah algoritma pencarian heuristik A* menjadi algoritma *Anytime* A*. Pendekatan yang diadopsi yaitu menggunakan pencarian heuristik berbobot untuk menemukan solusi perkiraan yang cepat, dan kemudian melanjutkan pencarian berbobot untuk menemukan solusi yang lebih baik [33].