Nama : Muhammad Husain Kaasyiful Ghitha

NIM : 1103220047

Kelas TK-45-G09

Robotika dan Sistem Cerdas

Analisis Tugas Robotika Week 14

Pada tugas ini, dilakukan praktik penggunaan bahasa Rust, yang dicontohkan oleh Will Velida di dalam playlist yang berjudul 'Learn Rust With Me!'. Berikut adalah bagian-bagian dari playlist tersebut:

- Menginstall Rust dan (Menjalankan Kode) Hello World!
- Tipe Data dan 'Collection'
- Pernyataan 'If-Else'
- Larik dan Vektor
- Perulangan menggunakan 'For', 'While', dan 'Loop'
- Peta Hash

Selain membuat dan menjalankan kode Rust mengikuti playlist tersebut, terdapat juga pembuatan program-program Rust yang berfokus pada implementasinya pada robot menggunakan kecerdasan buatan generatif. Program yang dibuat antara lain:

- Perencanaan Jalur Sederhana
- Gerakan Robot dengan Input Pengguna
- Simulasi Robot Menghindari Rintangan
- Penjadwalan Robot dengan Prioritas
- Robotik dengan Sistem Event-Driven
- Robot dengan Model Probabilistik

Berikut adalah pembahasan mengenai kode Rust dan program-program yang dibuat.

1. Praktik Tutorial dengan playlist Play Rust With Me! oleh Will Velida.
   Pada kode yang tertera di playlist ini, penjelasan kode yang dimuat di masing-masing video adalah sebagai berikut:

a. Learn Rust with Me! Part 1 - Installing Rust and Hello World!.

Rust adalah bahasa pemrograman sumber terbuka yang berfokus pada pengembangan aplikasi tingkat rendah, dengan fitur bahasa tingkat tinggi secara aman dan efisien, membedakannya dari bahasa seperti C dan C++. Bahasa Rust bersifat aman secara tipe dan memori, dan terbebas dari data race. Bahasa ini juga memiliki ekosistem yang ramah bagi pengembang.

Pada praktik ini, editor kode yang digunakan adalah Visual Studio Code. Untuk memrogram Rust menggunakan Visual Studio Code (di Windows), maka diperlukan C++ Build Tools. Setelah alat tersebut telah terinstal, maka installer Rust (rustup_init) dapat diunduh dan dijalankan untuk menginstal Rust, dan Cargo, build tool untuk Rust. Dengan alat-alat tersebut, sebuah program 'Hello World!' sederhana dapat diprogram dan dikompilasi menggunakan rustc. Untuk membuat sebuah program yang lebih matang, perintah 'cargo new' dapat digunakan dan secara default menuliskan kode 'Hello World!' di dalamnya. Untuk menjalankannya, 'cargo run' dapat dijalankan.

b. Learn Rust with Me! Part 2 Data types and collections.

Pada semua program Rust, terdapat sebuah fungsi utama 'main()' yang dijalankan pertama kali dijalankan di sebuah file kode Rust. Pada Bahasa Rust, menulis argumen (variabel) di dalam output dapat dilakukan dengan menulis '{}' di dalam string output dan menulis parameter secara berurutan setelah string tersebut.

Pada Bahasa Rust, deklarasi sebuah variabel dinyatakan dengan keyword 'let'. Rust akan menentukan tipe datanya secara implisit, namun variabel secara default bersifat immutable (nilai varabel tersebut tidak dapat ditimpa dengan nilai baru). Untuk dapat menimpa nilai suatu variabel, diperlukan modifier 'mut'. Terdapat juga metode 'shadowing' untuk menimpa nilai data dengan melakukan operasi dengan variabel itu sendiri. Selain itu, variabel bersifat statik (tipe data tidak dapat berubah). Tedapat empat tipe data dasar di Rust: integer, floating-point number, boolean, dan karakter. Semua operasi yang biasanya ditemukan di bahasa pemrograman lain (seperti penjumlahan dan operasi logika) dapat dilakukan dengan Rust. Sama seperti bahasa C dan C++, sebuah string merupakan perujukan dari karakter dan menyesuaikan dengan panjang karakter tersebut.

Terdapat dua kelompok data di Rust: ‘tuple’ dan ‘struct’. Sebuah tuple adalah kumpulan data immutable yang dapat diisi oleh data berbagai tipe. Deklarasi tuple dapat dilakukan dengan menggunakan tanda kurung (()) dan elemennya diakses menggunakan indeks dari tuple tersebut, dimulai dari nol. Sebuah struct adalah tipe yang disusun oleh data dengan tipe tertentu yang dapat dinamai (sebagai ‘field’). Terdapat dua cara dalam mendeklarasikan struct, yaitu cara klasik dengan mendeklarasikannya seperti sebuah fungsi dan memberi nama field dan tipenya. Selin itu, sebuah strcut juga dapat dideklarasikan seperti sebuah tuple (dengan membuat tuple berisikan tipe field) dan diakses juga layaknya sebuah struct

c. Learn Rust with Me! Part 3 If Else Statements.

Menulis pernyataan percabangan di Rust pada dasarnya sama dengan bahasa C dan C++. Namun, sebuah percabangan dapat menjadi nilai dari suatu variabel (bertipe boolean).

d. Learn Rust with Me! Part 4 Arrays and Vectors.

Implementasi larik dan tipe data vektor di bahasa Rust dapat dikatakan hampir serupa dengan implementasi yang terdapat pada bahasa C++. Selain mendeklarasikan semua variabelnya secara eksplisit, sebuah larik juga dapat dideklarasikan dengan nilai awal dan banyak elemen pada larik tersebut. Sebuah larik (dan vektor) dapat dicetak secara keseluruhan dengan ‘{:?}’. Deklarasi vektor pada Rust sedikit berbeda dari C++, yaitu dengan cara menulis ‘vec!’ sebelum tubuh vektor yang akan dideklarasikan. Anggota dari sebuah vektor dapat ditambah dengan menggunakan metode push() dan metode pop() digunakan untuk membuang anggota pada indeks terakhir dari suatu vektor. Berbeda dari akses pada elemen tidak valid pada sebuah larik, akses tidak valid pada sebuah vektor dapat berjalan namun akan memasuki keadaan panik dan proses terhenti dengan error.

e. Learn Rust with Me! Part 5 For Loops, While Loops and Loop Loops!

Secara umum, perulangan di bahasa Rust menggunakan keyword ‘loop’, Penghentian perulangan dapat dilakukan dengan keyword ‘break’. Selain loop. Perulangan dapat dilakukan dengan keyword ‘while’, yang mengulang perintah sesuai dengan kondisi yang dipasangkan, dan keyword ‘for’ (seperti pada ‘for x in list.iter()’), yang mengulang perintah untuk setiap elemen pada data yang memiliki

elemen terindeks (yang dapat diiterasi). Selain mengulang pada elemen, ‘for’ juga dapat digunakan untuk mengulang dengan rentang bilangan dengan pernyataan rentang seperti ‘1...10’ (seperti (i = 0; i < 10; i++) pada ‘for’ dalam bahasa C)

f. Learn Rust with Me! Part 6 Hash Maps.

Hash Maps di bahasa Rust adalah kumpulan objek dari pasangan ‘key’ dan ‘value’, seperti map pada C++ dan dictionary pada Python. Untuk menggunakan hash map, pada file kode diperlukan memuat library untuk hash map dengan kode ‘use std::collections::HashMap’. Variabel data bertipe hash map dapat dideklarasikan dengan fungsi ‘HashMap::new()’ dan data dapat dimasukkan dengan ‘HashMap::insert()’. Ketika mengakses data dengan ‘HashMap::get()’, data yang dikelaurkan adalah data sekaligus tipe datanya. Data pada Hash Map dapat dihapus dengan metode remove().

2. Pemrograman Robotika Sederhana di Rust dengan Bantuan Kecerdasan Buatan Generatif.

Pada Program-program robotika ini, penjelasan kode-kode yang dibuat oleh AI adalah sebagai berikut:

a. Perencanaan Jalur Sederhana.

Untuk merencanakan jalur robot dari titik awal (0, 0) ke tujuan (4, 4) dalam sebuah matriks 2D, kode yang dibuat mengimplementasikan algoritma pencarian A*. Matriks ini berisi rintangan yang ditandai dengan angka 1 dan jalan bebas yang ditandai dengan angka 0. Algoritma A* digunakan karena efisiensinya dalam menemukan jalur terpendek dengan menggunakan heuristik jarak Manhattan untuk memperkirakan biaya dari node saat ini ke tujuan. Kode ini menggunakan beberapa struktur data penting seperti Position untuk merepresentasikan koordinat x dan y, serta Node untuk menyimpan informasi tentang posisi, biaya total (f_score), dan biaya dari titik awal ke node saat ini (g_score). BinaryHeap digunakan sebagai priority queue untuk menyimpan node yang akan dieksplorasi, sementara HashMap digunakan untuk melacak jalur terbaik dan biaya dari titik awal ke setiap node.

Fungsi find_path adalah inti dari algoritma A*. Fungsi ini memulai dengan menambahkan node awal ke dalam open_set dan kemudian terus mengeksplorasi node dengan biaya terendah hingga mencapai tujuan atau tidak ada lagi node yang bisa dieksplorasi. Setiap kali node dieksplorasi, tetangganya dihitung dan diperiksa

apakah mereka menawarkan jalur yang lebih murah. Jika ya, tetangga tersebut ditambahkan ke open_set dengan biaya yang diperbarui.

Fungsi main menginisialisasi grid dan memanggil find_path untuk mencari jalur dari titik awal ke tujuan. Jika jalur ditemukan, jalur tersebut dicetak dan grid ditampilkan dengan jalur yang ditandai dengan tanda bintang (*). Jika tidak ada jalur yang ditemukan, pesan "No path found!" akan ditampilkan. Kode ini memberikan solusi yang efisien dan visualisasi yang jelas untuk masalah perencanaan jalur dalam matriks 2D dengan rintangan. Pada implementasinya, kode ini berhasil mengimplementasikan algoritma A* untuk merencanakan jalur robot dalam matriks 2D dengan rintangan. Struktur data yang digunakan memastikan efisiensi dalam pencarian jalur, sementara fungsi utama memastikan bahwa jalur terpendek ditemukan atau menginformasikan jika tidak ada jalur yang tersedia. Output yang dihasilkan memberikan visualisasi yang jelas tentang jalur yang ditemukan, membuatnya mudah untuk diverifikasi dan dipahami.

b. Gerakan Robot dengan Input Pengguna

Kode ini adalah program Rust yang mengimplementasikan sebuah robot yang dapat digerakkan berdasarkan input pengguna. Program ini menggunakan struktur `Position` untuk menyimpan koordinat x dan y dari posisi robot, dan struktur Robot yang memiliki satu field position dari tipe `Position`. Fungsi new() pada implementasi Robot digunakan untuk membuat robot baru pada posisi awal (0,0). Fungsi move_to() pada implementasi Robot menerima string direction sebagai parameter dan mengubah posisi robot berdasarkan arah yang diberikan. Fungsi ini menggunakan `match` untuk memeriksa nilai direction dan menyesuaikan koordinat x atau y sesuai dengan arah yang diberikan (up, down, right, left). Jika perintah tidak valid, program akan mencetak pesan "Perintah tidak valid!".

Fungsi display_position() digunakan untuk mencetak posisi robot saat ini dalam format `(x, y)`. Fungsi ini dipanggil setiap kali posisi robot berubah, serta sekali pada awal program untuk menampilkan posisi awal robot. Pada fungsi main(), program memasuki loop yang meminta input dari pengguna untuk menggerakkan robot. Input dibaca menggunakan io::stdin().read_line(), dan program akan terus meminta input hingga pengguna memasukkan perintah "quit". Setiap kali perintah valid dimasukkan, posisi robot diperbarui dan ditampilkan. Jika perintah "quit"

dimasukkan, program akan mencetak pesan "Program selesai!" dan keluar dari loop.

Ketika program dijalankan, output pertama yang akan terlihat adalah posisi awal robot: ‘Posisi robot: (0, 0)’ Setelah itu, program akan meminta input dari pengguna dengan prompt: ‘Masukkan perintah (up/down/right/left/quit)’. Jika pengguna memasukkan perintah yang valid seperti "up", "down", "right", atau "left", program akan memperbarui posisi robot dan mencetak posisi baru. Misalnya, jika pengguna memasukkan "up", outputnya akan menjadi: ‘Posisi robot: (0, 1)’ Jika pengguna memasukkan perintah yang tidak valid, program akan mencetak pesan: ‘Perintah tidak valid!’. Program akan terus meminta input hingga pengguna memasukkan "quit", yang akan menghasilkan output: ‘Program selesai!’

Kode ini menunjukkan penggunaan struktur dan implementasi fungsi dalam Rust untuk mengelola dan memanipulasi data. Dengan menggunakan loop dan input dari pengguna, program ini memungkinkan interaksi dinamis di mana pengguna dapat menggerakkan robot dan melihat perubahan posisi secara real-time. Program ini juga menunjukkan cara menangani input yang tidak valid dengan memberikan umpan balik yang sesuai kepada pengguna.

c. Simulasi Robot Menghindari Rintangan

Kode yang dibuat mensimulasikan pergerakan robot dalam peta 2D dengan menghindari rintangan menggunakan algoritma Breadth-First Search (BFS). Struktur Point digunakan untuk merepresentasikan koordinat dalam peta, sementara struktur Robot menyimpan posisi awal dan tujuan robot. Fungsi is_valid_move memeriksa apakah gerakan ke titik tertentu valid, yaitu berada dalam batas peta dan tidak mengenai rintangan. Fungsi find_path menggunakan BFS untuk mencari jalur terpendek dari posisi awal ke tujuan, dengan mempertimbangkan rintangan yang ada.

Algoritma BFS diimplementasikan dengan menggunakan antrian VecDeque untuk menyimpan titik-titik yang akan diperiksa. Matriks visited digunakan untuk melacak titik-titik yang sudah dikunjungi, sedangkan matriks prev menyimpan titik sebelumnya untuk setiap titik yang dikunjungi, memungkinkan rekonstruksi jalur setelah tujuan tercapai. Jika tujuan tercapai, jalur direkonstruksi dengan melacak kembali dari tujuan ke posisi awal menggunakan matriks prev, kemudian jalur tersebut dibalik agar dimulai dari posisi awal.

Peta 10x10 dibuat dengan menggunakan vektor dua dimensi map, di mana nilai false menunjukkan titik kosong dan true menunjukkan rintangan. Beberapa rintangan ditambahkan secara manual ke dalam peta. Robot dibuat dengan posisi awal di (0,0) dan tujuan di (9,9). Fungsi find_path kemudian dipanggil untuk mencari jalur dari posisi awal ke tujuan, dan hasilnya ditampilkan ke layar.

Jika jalur ditemukan, program akan mencetak setiap langkah yang diambil robot dari posisi awal ke tujuan, termasuk koordinat setiap langkah. Jika tidak ada jalur yang ditemukan, program akan mencetak pesan bahwa tidak ada jalur yang ditemukan. Output ini memberikan gambaran jelas tentang bagaimana robot bergerak dalam peta, menghindari rintangan, dan mencapai tujuan. Program ini menunjukkan penggunaan algoritma BFS yang efektif untuk masalah pencarian jalur dalam peta 2D dengan rintangan.

d. Penjadwalan Robot dengan Prioritas

Kode yang diberikan adalah implementasi dari sistem penjadwalan tugas untuk robot menggunakan antrian prioritas. Struktur data Task merepresentasikan tugas dengan prioritas, jenis tugas, dan koordinat target. Jenis tugas didefinisikan dalam enum TaskType yang mencakup empat jenis: Move, Pickup, Deliver, dan Charge. Implementasi trait Ord dan PartialOrd pada Task memungkinkan antrian prioritas untuk mengurutkan tugas berdasarkan prioritasnya, dengan prioritas yang lebih rendah memiliki urutan lebih tinggi (diutamakan).

Struktur Robot menyimpan posisi robot, antrian tugas menggunakan BinaryHeap dengan pembungkus Reverse untuk mengatur urutan prioritas, dan tugas yang sedang dikerjakan. Metode add_task menambahkan tugas baru ke antrian, sementara process_next_task mengambil tugas dengan prioritas tertinggi dari antrian dan memprosesnya sesuai dengan jenis tugasnya. Metode move_to, pickup_at, deliver_to, dan charge_at mengimplementasikan logika untuk menggerakkan robot ke posisi target, mengambil objek, mengantarkan objek, dan mengisi daya baterai, masing-masing.

Dalam fungsi main, sebuah objek Robot diinisialisasi dan beberapa tugas dengan prioritas berbeda ditambahkan ke antrian. Loop utama memproses tugas-tugas dalam antrian satu per satu, dengan jeda satu detik antara setiap tugas untuk mensimulasikan waktu pemrosesan. Setelah semua tugas diproses, program

menampilkan pesan "All tasks completed!" untuk menunjukkan bahwa semua tugas telah selesai.

Output dari kode ini akan menunjukkan urutan tugas yang diproses berdasarkan prioritasnya. Tugas dengan prioritas tertinggi (nilai prioritas terendah) akan diproses terlebih dahulu. Dalam contoh ini, tugas Charge dengan prioritas 1 akan diproses pertama, diikuti oleh tugas Pickup dengan prioritas 2, dan terakhir tugas Move dengan prioritas 3. Setiap tugas akan menampilkan pesan yang sesuai dengan jenis tugasnya dan koordinat targetnya, memberikan gambaran yang jelas tentang urutan dan hasil pemrosesan tugas oleh robot.

e. Robotika dengan Sistem Event-Driven

Kode yang dibuat mengimplementasikan sistem robotik berbasis event-driven menggunakan bahasa pemrograman Rust. Struktur utama dalam kode ini adalah RobotState, yang menyimpan status robot seperti posisi, arah, tujuan, dan rintangan yang terdeteksi. Enum RobotEvent mendefinisikan jenis-jenis event yang dapat diterima oleh robot, yaitu NewObstacle, TargetChanged, dan SensorUpdate. Implementasi RobotState mencakup metode untuk menangani event, menyesuaikan jalur robot, memperbarui posisi, dan mencari rintangan terdekat.

Metode handle_event dalam RobotState memproses event yang diterima oleh robot. Jika event NewObstacle terdeteksi, posisi rintangan baru ditambahkan ke daftar rintangan dan jalur robot disesuaikan. Jika event TargetChanged terjadi, tujuan robot diperbarui dan jalur disesuaikan. Event SensorUpdate memperbarui posisi robot berdasarkan arah hadapnya. Metode adjust_path menyesuaikan jalur robot untuk menghindari rintangan terdekat, sementara update_position memperbarui posisi robot berdasarkan arah hadapnya.

Fungsi main menginisialisasi robot dan antrean event menggunakan VecDeque. Dalam loop utama, kode ini mensimulasikan waktu dengan thread::sleep dan menambahkan event secara acak ke dalam antrean. Event NewObstacle dan TargetChanged ditambahkan dengan probabilitas tertentu, sementara event SensorUpdate ditambahkan secara reguler. Semua event dalam antrean diproses oleh robot, dan simulasi berhenti jika robot mencapai tujuan yang ditentukan.

Output dari kode ini adalah serangkaian pesan yang mencerminkan status robot dan event yang terjadi. Misalnya, ketika rintangan baru terdeteksi, pesan "New obstacle detected at ..." akan dicetak. Ketika tujuan robot berubah, pesan "Target

changed to ..." akan muncul. Setiap kali posisi robot diperbarui, pesan "Position updated to: ..." akan dicetak. Jika robot mencapai tujuan, pesan "Target reached!" akan muncul dan simulasi berhenti. Output ini memberikan gambaran tentang bagaimana robot berinteraksi dengan lingkungannya dan menyesuaikan jalurnya berdasarkan event yang diterima.

f. Robot dengan Model Probabilistik

Analisis kode ini menunjukkan implementasi dasar dari sistem navigasi robot menggunakan model probabilistik. Kode ini mendefinisikan struktur Position untuk menyimpan koordinat (x, y) dan tingkat ketidakpastian posisi robot. Struktur SensorData digunakan untuk menyimpan data sensor seperti jarak, sudut, dan noise. Fungsi distance_to dalam Position menghitung jarak Euclidean antara posisi robot saat ini dan target, sementara fungsi update memperbarui posisi robot dengan mempertimbangkan ketidakpastian.

Fungsi get_best_path bertanggung jawab untuk menentukan arah gerakan terbaik menuju target dengan mempertimbangkan noise sensor. Fungsi ini menggunakan generator angka acak untuk menambahkan noise pada perhitungan arah gerakan, kemudian menormalisasi vektor arah untuk mendapatkan unit vector. Ini memastikan bahwa robot bergerak dalam arah yang benar meskipun ada ketidakpastian dalam data sensor.

Dalam fungsi main, posisi awal robot dan target diinisialisasi, serta data sensor dengan noise tertentu. Simulasi pergerakan robot dilakukan dalam loop yang berjalan hingga jumlah langkah maksimum tercapai atau robot mencapai target. Pada setiap langkah, jarak ke target dihitung, dan jika jarak kurang dari 0.5, simulasi berhenti dan mencetak pesan bahwa target telah tercapai. Jika tidak, arah gerakan terbaik dihitung dan posisi robot diperbarui dengan mempertimbangkan ketidakpastian yang meningkat setiap langkah.

Output dari kode ini akan menunjukkan posisi robot pada setiap langkah dan jarak ke target. Jika robot mencapai target dalam jumlah langkah yang ditentukan, pesan akan dicetak menunjukkan langkah di mana target tercapai. Jika tidak, output akan menunjukkan posisi robot dan jarak ke target pada setiap langkah hingga jumlah langkah maksimum tercapai. Analisis ini menunjukkan bahwa kode ini berhasil mensimulasikan pergerakan robot dengan mempertimbangkan

ketidakpastian dalam data sensor, dan memberikan gambaran yang jelas tentang bagaimana robot dapat menavigasi menuju target dalam kondisi ketidakpastian.