TECHNICAL REPORT
ROBOTIKA OPTION 1
Hacking “Mastering ROS2” Book

Disusun oleh :

Nama	**: Thasya Mulia**
Nim	**: 1103201208**
Kelas	**: TK-44-04**
Dosen	**: Dr. Risman Adnan**

PROGRAM STUDI TEKNIK KOMPUTER
FAKULTAS TEKNIK ELEKTRO
UNIVERSITAS TELKOM
BANDUNG
2022/2023

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Robotika adalah ilmu yang berkaitan dengan pengenalan dan manipulating physical world melalui perangkat yang dikendalikan oleh komputer. Sistem robotika berada di physical world, mampu memperoleh informasi tentang lingkungan mereka melalui sensor, dan melakukan manipulasi dengan menggunakan kekuatan fisik. Meskipun sebagian besar bidang robotika masih dalam tahap awal, konsep perangkat yang mampu melakukan manipulasi secara "cerdas" memiliki potensi besar untuk mengubah masyarakat.

Sistem Operasi Robot (ROS) adalah perangkat lunak perantara robotika yang digunakan secara luas di seluruh dunia untuk membantu pengembang dalam memprogram aplikasi robotik. Saat ini, ROS telah diadopsi oleh perusahaan robotika, pusat penelitian, dan universitas untuk memprogram robot-robot canggih.

1.2 Tujuan

- Memahami ROS (Robot Operating System) sebagai framework untuk membangun perangkat lunak untuk robotika.
- Meringkas dan memberikan gambaran menyeluruh tentang isi dari buku "Mastering ROS for Robotics Programming - Edisi Ketiga" dari bab 1 hingga bab 8, sehingga dapat memahami materi yang ada.
- Memenuhi kewajiban pelaksanaan ujian akhir mata kuliah Robotika.

BAB 11
DASAR TEORI

1. Webots

Webots adalah simulator robotika yang digunakan untuk memodelkan dan mensimulasikan perilaku robot di lingkungan virtual. Webots mendukung berbagai jenis robot, termasuk robot beroda, robot berjalan, robot manipulator, dan banyak lagi. Dalam Webots, kita dapat merancang struktur fisik robot, mengatur parameter kinematik dan dinamik, serta mengimplementasikan algoritma kontrol yang diperlukan. Selain itu, Webots menyediakan berbagai sensor dan aktuator virtual yang dapat diintegrasikan dengan robotik.

Fungsi Webots:

- Simulasi Robotika: Dengan Webots, pengembang dapat membuat model virtual dari robot dan lingkungan sekitarnya, serta menguji perilaku dan performa robot dalam simulasi. Ini memungkinkan pengembang untuk melakukan eksperimen, mengoptimalkan algoritma kontrol, dan memvalidasi desain robot sebelum mengimplementasikannya di dunia nyata. Webots juga menyediakan berbagai sensor dan aktuator virtual yang dapat diintegrasikan dengan model robot, sehingga memungkinkan simulasi yang lebih realistis dan mendekati kondisi dunia nyata.
- Pengembangan dan Pemrograman Robot: Webots memungkinkan untuk mengembangkan kode perangkat lunak robot dan menguji fungsionalitasnya dalam lingkungan simulasi. Dalam Webots, pengembang dapat merancang struktur fisik robot, mengatur parameter kinematik dan dinamik, serta mengimplementasikan algoritma kontrol yang diperlukan. Hal ini memungkinkan pengembang untuk memahami dan memvalidasi perilaku robot sebelum melakukan implementasi di dunia nyata.

2. ROS (Robot Operating System)

ROS adalah kerangka kerja perangkat lunak open-source yang banyak digunakan dalam pengembangan robotika. ROS menyediakan lingkungan yang terstruktur

dan modular untuk mengintegrasikan komponen perangkat keras dan perangkat lunak dalam sistem robotika. Salah satu keunggulan utama ROS adalah kemampuannya dalam mengelola komunikasi antara berbagai node yang berjalan secara independen. ROS menggunakan model publikasi-langganan (publish-subscribe) yang memungkinkan node untuk berkomunikasi melalui topik (topic), layanan (service), dan aksi (action).

Fungsi ROS (Robot Operating System):

- Komunikasi dan Integrasi: ROS menyediakan infrastruktur komunikasi yang kuat untuk menghubungkan berbagai komponen perangkat keras dan perangkat lunak dalam sistem robotika. Melalui model publikasi-langganan (publish-subscribe), ROS memfasilitasi pertukaran data dan perintah antara sensor, aktuator, dan algoritma kontrol dalam sistem robotika yang kompleks.
- Pengelolaan Perangkat Lunak: ROS menyediakan kerangka kerja perangkat lunak yang terstruktur untuk mengorganisir dan mengelola kode robotika. Dengan ROS, pengembang dapat merancang arsitektur perangkat lunak yang modular dan berorientasi layanan. Ini memungkinkan reusabilitas kode yang tinggi dan mempermudah pengembangan kolaboratif dalam tim. ROS juga menyediakan perpustakaan perangkat lunak yang kaya dan beragam, termasuk algoritma navigasi, pemrosesan citra, manipulasi objek, dan lainnya, yang dapat digunakan untuk memperluas fungsionalitas robot.

BAB III
ANALISIS CODE

- **Analisis file code "e-puck_avoid_obstacles.c" yang ada di github**

Code diatas bertujuan untuk menghindari rintangan menggunakan metode Braitenberg. Analisis tentang kode tersebut:

1. Inisialisasi Perangkat:

- Kode tersebut menggunakan Webots API untuk menginisialisasi perangkat yang digunakan oleh robot, seperti sensor jarak, sensor tanah, LED, dan motor.
- Perangkat seperti sensor jarak dan sensor tanah diaktifkan dengan memanggil fungsi `wb_distance_sensor_enable()`.
- Motor kiri dan motor kanan diatur dengan menggunakan fungsi `wb_motor_set_position()` dan `wb_motor_set_velocity()`.

2. Sensor dan Aktuator:

- Nilai-nilai sensor jarak dan sensor tanah diperoleh melalui pemanggilan fungsi `wb_distance_sensor_get_value()` dan disimpan dalam array `distance_sensors_values` dan `ground_sensors_values` masing-masing.
- Fungsi `cliff_detected()` digunakan untuk memeriksa apakah ada rintangan yang terdeteksi oleh sensor tanah.
- Nilai-nilai LED diatur dengan menggunakan fungsi `wb_led_set()`.
- Kecepatan motor kiri dan motor kanan diatur melalui array `speeds`, yang dihitung berdasarkan nilai sensor jarak dengan memperhitungkan bobot dan offset tertentu. Nilai kecepatan tersebut kemudian disesuaikan dengan batasan maksimum dan minimum menggunakan kondisi if-else.

3. Loop Utama:

- Loop utama berjalan terus-menerus menggunakan `while (true)` untuk menjalankan kontrol robot secara terus-menerus.

- Setiap iterasi loop, nilai sensor dan aktuator diperbarui menggunakan fungsi-fungsi seperti `reset_actuator_values()`, `get_sensor_input()`, `blink_leds()`, `run_braitenberg()`, dan `set_actuators()`.
- Fungsi `step()` digunakan untuk menggerakkan simulasi robot melalui pemanggilan `wb_robot_step()`.

4. Tindakan Robot:

- Jika ada rintangan yang terdeteksi oleh sensor tanah (`cliff_detected()`), robot akan bergerak mundur selama 0.2 detik dan kemudian berbelok ke kiri selama 0.2 detik.
- Jika tidak ada rintangan, robot akan mengikuti metode Braitenberg yang telah diimplementasikan dalam fungsi `run_braitenberg()`.

- **Analisis file code "robot_motion.cpp" yang ada di github**

Code diatas untuk menggerakkan robot menggunakan library Webots dalam bahasa C++. Analisis tentang kode tersebut:

1. Inisialisasi dan Pengaturan Motor:

- Kode tersebut menggunakan library Webots untuk menginisialisasi robot dan mendapatkan penanganan motor kiri dan motor kanan.
- Target posisi motor diatur ke INFINITY untuk mengontrol kecepatan (speed control).

2. Loop Utama:

- Loop utama `while (true)` digunakan untuk menjalankan kontrol robot secara terus-menerus.
- Dalam setiap iterasi loop, kecepatan motor kiri dan kanan diatur menggunakan fungsi `setVelocity()` dari objek Motor.
- Robot kemudian melangkah (step) menggunakan fungsi `step()` dari objek Robot dengan argumen `TIME_STEP`.
- Variabel `t` digunakan untuk menghitung waktu (dalam milidetik) sejak awal program dimulai.

3. Pergerakan Robot:

- Awalnya, kecepatan motor kiri dan kanan diatur ke `MAX_SPEED * 0.1` untuk menggerakkan robot ke depan dengan kecepatan 10% dari kecepatan maksimum.
- Setelah waktu `t` melebihi 2000 milidetik, arah gerakan motor kanan dibalik dengan mengubah nilai `r_direction` menjadi -1.0.
- Setelah waktu `t` melebihi 4000 milidetik, arah gerakan motor kanan dikembalikan ke 1.0 dan `t` diatur kembali ke 0.0.

4. Pembebasan Sumber Daya:

- Pada akhir program, objek robot dihapus dengan menggunakan operator `delete`.

Kode ini mengimplementasikan gerakan robot dengan mengatur kecepatan motor kiri dan kanan. Robot akan bergerak maju selama beberapa detik, kemudian motor kanan akan dibalik arah gerakannya selama beberapa detik, dan siklus ini akan terus berlanjut.

- **Analisis file code "e_puck_manager.cpp" yang ada di github**

Code ini untuk mengendalikan robot e-puck menggunakan Webots melalui ROS. Analisis tentang kode tersebut:

1. Library dan Pesan ROS:

- Kode ini menggunakan beberapa library ROS, seperti `ros/ros.h` dan pesan-pesan seperti `webots_ros/Int32Stamped.h`, `webots_ros/set_float.h`, `webots_ros/set_int.h`, `webots_ros/robot_get_device_list.h`, `std_msgs/String.h`, dan `geometry_msgs/Twist.h`.
- Pesan `geometry_msgs/Twist` digunakan untuk menerima perintah kecepatan robot.

2. Variabel Global:

- Terdapat beberapa variabel global yang digunakan dalam kode ini, seperti `modelList` untuk menyimpan daftar nama model robot, `cnt` untuk menghitung jumlah model yang ditemukan, `left_vel` dan `right_vel`

untuk menyimpan kecepatan motor kiri dan kanan, serta `wheel_radius` dan `axes_length` untuk menghitung kecepatan motor berdasarkan perintah kecepatan linear dan angular.

3. Fungsi Callback:

- Terdapat dua fungsi callback yang digunakan dalam kode ini.
- `cmdVelCallback` adalah fungsi callback yang dipanggil ketika ada perintah kecepatan yang diterima melalui topik `cmd_vel`. Fungsi ini menghitung kecepatan motor kiri dan kanan berdasarkan perintah kecepatan linear dan angular yang diterima.
- `modelNameCallback` adalah fungsi callback yang dipanggil ketika nama model robot diterima melalui topik `model_name`. Fungsi ini menyimpan nama model robot ke dalam variabel `modelName` dan menampilkan informasi tentang model tersebut.

4. Inisialisasi ROS:

- Kode ini menginisialisasi ROS dengan memanggil `ros::init`.
- Node handle (`n`) dibuat untuk mengatur komunikasi ROS.
- Subscriber dibuat untuk menerima pesan dari topik `model_name` dan `cmd_vel`.

5. Mengatur Kecepatan Motor:

- Kode ini menggunakan layanan ROS untuk mengatur kecepatan motor kiri dan kanan.
- Layanan `set_position` digunakan untuk mengatur target posisi motor ke INFINITY, sehingga motor diatur dalam mode kontrol kecepatan.
- Layanan `set_velocity` digunakan untuk mengatur kecepatan motor kiri dan kanan menjadi 0.0.

6. Loop Utama:

- Kode ini memiliki loop utama yang berjalan selama ROS masih aktif (`ros::ok()`).
- Di dalam loop, kecepatan motor kiri dan kanan diatur menggunakan layanan `set_velocity` berdasarkan nilai `left_vel` dan `right_vel` yang diupdate melalui fungsi callback `cmdVelCallback`.

- Loop dijalankan dengan frekuensi 10 Hz menggunakan objek `ros::Rate`.

7. Pembebasan Sumber Daya:

- Pada akhir program, sumber daya ROS yang digunakan dihapus menggunakan fungsi `ros::shutdown()`.

Kode ini mengimplementasikan pengendalian robot e-puck dengan menggunakan perintah kecepatan yang diterima melalui topik `cmd_vel`. Kecepatan motor kiri dan kanan dihitung berdasarkan perintah.

- **Analisis file code "package.xml" yang ada di github**

Code diatas bertujuan untuk untuk mendefinisikan metadata paket ROS, seperti nama paket, versi, deskripsi, dependensi, dan lain-lain. Informasi ini digunakan oleh sistem manajemen paket ROS untuk mengelola paket dan dependensinya Analisis tentang kode tersebut:

1. Nama dan Versi:

- Paket ini diberi nama `webots_demo_pkg` menggunakan elemen `<name>`.
- Versi paket ditentukan dengan elemen `<version>`. Dalam contoh ini, versi paket diatur sebagai `0.0.0`.

2. Deskripsi:

- Deskripsi paket diberikan dalam elemen `<description>`. Dalam contoh ini, deskripsi paket adalah "The webots_demo_pkg package".

3. Pemelihara:

- Pemelihara paket ditentukan dengan elemen `<maintainer>`. Dalam contoh ini, pemelihara paket adalah `jcacace` dengan alamat email `jcacace@todo.todo`.

4. Lisensi:

- Lisensi paket ditentukan dalam elemen `<license>`. Dalam contoh ini, lisensi paket ditandai sebagai "TODO", yang berarti informasi lisensi belum ditentukan.

5. URL:

- URL terkait paket dapat ditambahkan dengan menggunakan elemen `<url>`. Dalam contoh ini, tidak ada URL yang diberikan.

6. Penulis:

- Informasi tentang penulis paket dapat ditambahkan dengan menggunakan elemen `<author>`. Dalam contoh ini, tidak ada informasi penulis yang diberikan.

7. Dependensi:

- Dependensi paket ditentukan dengan elemen-elemen `<build_depend>`, `<build_export_depend>`, `<exec_depend>`, dan `<test_depend>`.
- Dalam contoh ini, paket memiliki dependensi terhadap paket `geometry_msgs`, `roscpp`, dan `webots_ros` untuk proses kompilasi dan eksekusi.
- Elemen `<buildtool_depend>` menunjukkan dependensi terhadap alat kompilasi, dan dalam contoh ini, paket membutuhkan `catkin` sebagai alat kompilasi.

8. Eksport:

- Elemen `<export>` dapat digunakan untuk menyertakan informasi tambahan yang diminta oleh alat-alat lain.

- **Analisis file code "e_puck_manager.launch" yang ada di github**

Code ini untuk meluncurkan node ROS dengan nama e_puck_manager.
Analisis tentang kode tersebut:

1. `<launch>`:

- Ini adalah elemen root dari file XML dan menandakan bahwa ini adalah file launch ROS.

2. `<arg>`:

- Elemen `<arg>` digunakan untuk mendefinisikan argumen yang dapat digunakan saat meluncurkan file launch.
- Di sini, ada satu argumen bernama `no-gui` yang memiliki nilai default `false`. Argumen ini digunakan untuk mengontrol apakah mode GUI Webots diaktifkan atau tidak.

3. `<include>`:

- Elemen `<include>` digunakan untuk memasukkan file launch lain ke dalam file launch saat ini.
- Di sini, file launch yang dimasukkan adalah `webots.launch` dari paket `webots_ros` menggunakan atribut `file`.
- Elemen `<arg>` di dalam `<include>` digunakan untuk memberikan nilai pada argumen yang diperlukan oleh file launch yang dimasukkan.
- Dalam contoh ini, beberapa argumen diberikan, yaitu `mode` diatur sebagai "realtime", `no-gui` diatur sesuai dengan nilai argumen `no-gui` dari file launch saat ini, dan `world` diatur untuk menunjuk ke lokasi file dunia Webots yang terkait dengan paket `webots_demo_pkg`.

4. `<node>`:

- Elemen `<node>` digunakan untuk meluncurkan node ROS.
- Di sini, node yang diluncurkan memiliki nama "e_puck_manager" dan berasal dari paket `webots_demo_pkg`.
- Atribut `pkg` menunjukkan paket yang mengandung file eksekusi node.
- Atribut `type` menunjukkan nama file eksekusi node.
- Atribut `output` diatur sebagai "screen" untuk mengarahkan output dari node ke terminal.

File `e_puck_manager.launch` ini memulai simulasi Webots menggunakan file dunia `e_puck_ros.wbt` dari paket `webots_demo_pkg`, dan kemudian meluncurkan node `e_puck_manager` dari paket yang sama.

BAB IV
HASIL CODE

Berdasarkan semua codingan yang ada, robot E-Puck dapat melakukan hal berikut:

1. Mengontrol Kecepatan Roda:
    - Robot E-Puck dapat mengontrol kecepatan roda kiri dan kanan dengan menggunakan kode `e_puck_manager.cpp` yang menggunakan topik `/cmd_vel` untuk menerima perintah kecepatan dari node ROS.
    - Dalam contoh kode tersebut, kecepatan roda kiri dan kanan dihitung berdasarkan perintah kecepatan linear dan angular yang diterima.
2. Menggerakkan Robot:
    - Dengan mengontrol kecepatan roda kiri dan kanan, robot E-Puck dapat bergerak maju, mundur, berbelok, dan berputar di tempat.
3. Kontrol Waktu Gerakan:
    - Dalam kode `robot_motion.cpp`, waktu gerakan robot E-Puck dapat dikontrol menggunakan variabel `t`.
    - Dalam contoh tersebut, setelah waktu `t` mencapai 2000, arah gerakan roda kanan akan berubah, dan setelah waktu `t` mencapai 4000, arah gerakan roda kanan akan kembali ke arah semula.
    - Dengan mengubah waktu dan aturan perubahan arah, Anda dapat mengontrol pola gerakan robot E-Puck.
4. Integrasi dengan ROS:
    - Dalam kode `e_puck_manager.cpp`, robot E-Puck diintegrasikan dengan ROS menggunakan node ROS.
    - Node `e_puck_manager` dapat menerima perintah kecepatan melalui topik `/cmd_vel` dan mengontrol kecepatan roda robot berdasarkan perintah tersebut.
    - Ini memungkinkan pengendalian robot E-Puck melalui ROS dan integrasinya dengan lingkungan ROS lainnya.

Dengan menggunakan kode yang disediakan, kita dapat mengontrol gerakan dan perilaku robot E-Puck melalui ROS, memprogramnya untuk melakukan tugas tertentu, dan memanfaatkannya dalam skenario yang sesuai dengan kebutuhan.

BAB V
KESIMPULAN

Dalam technical report ini, kita telah melakukan analisis dan implementasi kontrol gerakan pada robot E-Puck menggunakan platform Webots dan integrasinya dengan ROS. Melalui codingan yang disediakan, kita berhasil mengontrol kecepatan roda kiri dan kanan, sehingga robot E-Puck dapat bergerak maju, mundur, berbelok, dan berputar di tempat. Kita juga mengintegrasikan robot E-Puck dengan ROS menggunakan node `e_puck_manager`, yang memungkinkan pengendalian robot melalui perintah kecepatan yang diterima melalui topik `/cmd_vel`. Selain itu, kita memberikan kontrol waktu gerakan dengan mengubah arah gerakan roda kanan setelah periode waktu tertentu. Dengan demikian, melalui implementasi ini, kita berhasil memperlihatkan kemampuan robot E-Puck untuk melakukan gerakan terkontrol dan berinteraksi dengan lingkungan melalui integrasi dengan ROS.

REFERENSI

Joseph, L., & Cacace, J. (2021). Mastering ROS for Robotic Programming (3rd ed.). Packt Publishing