TECHNICAL REPORT

MASTERING ROS2 WITH WEBOTS

Nama :

Zulian Wahid 1103201049

Table of Content

Introduction

Bidang robotika telah mengalami kemajuan yang signifikan dalam beberapa tahun terakhir, merevolusi berbagai industri dan domain. Karena robot menjadi semakin terintegrasi ke dalam kehidupan kita sehari-hari, sangat penting untuk mengembangkan teknik pemrograman yang kuat dan efisien untuk pengoperasiannya. Salah satu kerangka kerja terkemuka untuk pemrograman robot adalah Robot Operating System (ROS), yang menyediakan seperangkat alat dan pustaka yang komprehensif untuk membangun dan mengendalikan robot. Dalam laporan teknis ini, kami mengeksplorasi konsep dan teknik yang diuraikan dalam buku "Menguasai ROS untuk Pemrograman Robotik." Laporan ini berfungsi sebagai dokumentasi perjalanan kami melalui buku tersebut, dengan fokus pada Bab 1 hingga 8. Melalui contoh-contoh praktis dan latihan langsung, kami mempelajari aspek-aspek utama ROS, seperti komunikasi, persepsi, kontrol, dan simulasi, dengan penekanan khusus pada penggunaan simulator robot Webots.

Tujuan utama dari laporan teknis ini adalah untuk mendapatkan pemahaman yang mendalam tentang ROS dan aplikasinya dalam pemrograman robotik. Dengan mengikuti bab-bab "Menguasai ROS untuk Pemrograman Robotik," kami bertujuan untuk mengembangkan kemahiran dalam konsep dan teknik ROS sambil mengasah keterampilan praktis kami melalui implementasi dan eksperimen. Laporan ini memberikan gambaran ringkas tentang topik-topik yang dibahas dalam setiap bab, bersama dengan refleksi dan wawasan yang kami peroleh selama proses pembelajaran. Selain itu, kami mengeksplorasi integrasi ROS dengan Webots, simulator robot yang populer, untuk mensimulasikan dan menguji sistem robotik kami di lingkungan virtual. Dengan menggabungkan teori dan praktik, laporan ini bertujuan untuk berkontribusi pada basis pengetahuan yang lebih luas tentang ROS dan implementasi praktisnya, menyediakan sumber daya yang berharga bagi para peneliti, insinyur, dan penggemar robotika.

ROS Basic

ROS, atau Robot Operating System, adalah kerangka kerja sumber terbuka yang dirancang untuk membangun dan mengendalikan robot. ROS berfungsi sebagai middleware yang menyediakan seperangkat pustaka perangkat lunak dan alat untuk memfasilitasi pengembangan perangkat lunak robot. ROS menawarkan infrastruktur komunikasi untuk pertukaran data tanpa batas antara berbagai

komponen sistem robot, mengikuti pendekatan modular untuk desain perangkat lunak, dan memiliki ekosistem paket yang luas untuk tugas-tugas robot yang umum. ROS juga menyediakan alat visualisasi dan debugging, terintegrasi dengan simulator untuk menguji algoritme, dan memiliki komunitas yang mendukung. Secara keseluruhan, ROS memainkan peran penting dalam menstandarkan dan memodularisasi pengembangan perangkat lunak robot, memungkinkan implementasi yang efisien dan fleksibel untuk sistem robotik yang kompleks.

Berikut adalah beberapa aspek dan manfaat utama dari penggunaan ROS:

Komunikasi: ROS menyediakan infrastruktur komunikasi yang fleksibel yang memungkinkan pertukaran data tanpa batas antara berbagai komponen sistem robot. ROS menggunakan model pengiriman pesan publish-subscribe, di mana node dapat mempublikasikan dan berlangganan topik, sehingga memudahkan berbagi data sensor, perintah kontrol, dan informasi lainnya.

Modularitas: ROS mempromosikan pendekatan modular untuk pengembangan perangkat lunak robot. Hal ini memungkinkan pengembang untuk menguraikan sistem yang kompleks menjadi komponen yang lebih kecil dan dapat digunakan kembali yang disebut node. Node ini dapat ditulis dalam berbagai bahasa pemrograman dan dapat berkomunikasi satu sama lain melalui infrastruktur komunikasi ROS.

Ekosistem Paket: ROS memiliki ekosistem yang luas dari paket dan pustaka yang ada yang menyediakan fungsionalitas siap pakai untuk tugas-tugas robotik umum. Paket-paket ini mencakup berbagai bidang, termasuk persepsi, pelokalan, pemetaan, perencanaan jalur, manipulasi, dan banyak lagi. Memanfaatkan paket-paket yang ada ini dapat secara signifikan mempercepat pengembangan dan mengurangi upaya yang diperlukan untuk mengimplementasikan fungsi robotik yang umum.

Visualisasi dan Debugging: ROS menyediakan alat visualisasi dan debugging yang kuat yang membantu dalam pengembangan dan debugging sistem robotik. Alat-alat seperti RViz memungkinkan visualisasi data sensor, model robot, dan algoritme perencanaan, membantu pengembang mendapatkan wawasan tentang perilaku robot mereka.

Simulasi: ROS terintegrasi dengan baik dengan simulator robot seperti Gazebo dan Webots, yang memungkinkan pengembang untuk menguji dan memvalidasi algoritme mereka dalam lingkungan simulasi sebelum menerapkannya pada robot nyata. Simulasi memungkinkan pembuatan prototipe yang cepat, verifikasi algoritme, dan pengujian dalam berbagai kondisi tanpa memerlukan perangkat keras fisik.

Komunitas dan Dukungan: ROS memiliki komunitas pengembang dan peneliti yang dinamis dan aktif. Sifat ROS yang digerakkan oleh komunitas ini mendorong kolaborasi, berbagi pengetahuan, dan ketersediaan sumber daya seperti tutorial, kode sampel, dan forum, sehingga memudahkan pendatang baru untuk memulai dan mencari bantuan saat dibutuhkan.

Webots Basic

Webots adalah simulator robot yang banyak digunakan yang menyediakan lingkungan virtual untuk mendesain, memprogram, dan menguji sistem robotik. Dikembangkan oleh Cyberbotics, menawarkan platform simulasi 3D realistis yang memungkinkan pengguna untuk memodelkan dan mensimulasikan berbagai jenis robot, sensor, dan lingkungan. Webots mendukung berbagai bahasa pemrograman dan menyediakan antarmuka yang mudah digunakan untuk merancang perilaku robot, melakukan simulasi berbasis fisika, dan memvisualisasikan interaksi robot dengan dunia simulasi. Dengan serangkaian fitur yang komprehensif dan perpustakaan model robot yang luas, Webots berfungsi sebagai alat yang berharga untuk pembuatan prototipe yang cepat, pengembangan algoritme, dan evaluasi kinerja robot, yang pada akhirnya memfasilitasi pengembangan dan validasi sistem robotik sebelum digunakan dalam skenario dunia nyata.

Analyze the code

1. package.xml

Kode ini adalah file XML bernama package.xml. Dalam konteks ROS (Sistem Operasi Robot), file package.xml adalah bagian penting dari paket ROS. File ini berisi informasi metadata dan ketergantungan tentang paket.

Konten file package.xml mencakup berbagai tag yang memberikan detail penting tentang paket. Berikut ini adalah rincian dari berbagai bagian:

<name>: Menentukan nama paket, yang dalam hal ini adalah webots_demo_pkg.

<version>: Menentukan versi paket. Versi disetel ke 0.0.0 dalam contoh ini.

<description>: Memberikan deskripsi singkat tentang paket, yaitu "Paket webots_demo_pkg" dalam kasus ini.

<maintainer>: Menunjukkan pengelola paket beserta alamat surelnya. Pengelola untuk paket ini adalah jcacace dengan alamat email jcacace@todo.todo.

<license>: Menentukan lisensi yang digunakan untuk mendistribusikan paket. Dalam kasus ini, lisensi ditandai sebagai "TODO," yang berarti perlu diisi dengan informasi lisensi yang sesuai.

Ketergantungan: Bagian ini berisi daftar ketergantungan paket. Bagian ini menentukan paket ROS lain yang menjadi sandaran paket ini selama fase pembuatan dan eksekusi. Ketergantungan yang terdaftar termasuk geometry_msgs, roscpp, dan webots_ros, yang mengindikasikan bahwa paket ini membutuhkan paket-paket ini untuk diinstal.

Secara keseluruhan, file package.xml berfungsi sebagai manifes untuk paket ROS, menyediakan informasi penting tentang paket dan dependensinya.

2. CMakeList.txt

Kode yang disediakan adalah file CMakeLists.txt, yang digunakan dalam paket ROS (Robot Operating System) untuk mengonfigurasi proses pembuatan. Mari kita analisis bagian-bagian yang berbeda dan tujuannya:

CMake versi minimum dan deklarasi proyek:

cmake_minimum_required (VERSI 3.0.2): Menentukan versi minimum CMake yang dibutuhkan.

project(webots_demo_pkg): Menetapkan nama proyek sebagai "webots_demo_pkg".

Menemukan dan mendeklarasikan ketergantungan catkin:

find_package(catkin KOMPONEN YANG DIPERLUKAN ...): Mencari dan menyertakan dependensi catkin yang diperlukan. Dalam kasus ini, komponen yang dibutuhkan adalah geometry_msgs, roscpp, dan webots_ros.

Pesan, layanan, dan tindakan ROS:

Kode ini menyertakan bagian komentar yang menjelaskan cara mendeklarasikan dan menghasilkan pesan, layanan, dan aksi ROS dalam paket. Bagian-bagian ini dikomentari untuk cuplikan kode yang disediakan.

Parameter konfigurasi ulang dinamis ROS:

Kode ini menyertakan bagian komentar yang menjelaskan cara mendeklarasikan dan menghasilkan parameter konfigurasi ulang dinamis di dalam paket. Bagian-bagian ini dikomentari untuk cuplikan kode yang disediakan.

Konfigurasi khusus Catkin:

catkin_package(...): Menentukan konfigurasi khusus paket, seperti menyertakan direktori, pustaka, dan dependensi.

Membangun konfigurasi:

include_directories(...) : Menentukan direktori tambahan yang akan disertakan untuk file header.

add_executable(e_puck_manager src/e_puck_manager.cpp): Mendeklarasikan executable bernama "e_puck_manager" dan menentukan file sumber "src/e_puck_manager.cpp".

target_link_libraries(e_puck_manager ${catkin_LIBRARIES}): Menautkan eksekusi dengan pustaka yang diperlukan (dalam kasus ini, catkin_LIBRARIES).

File CMakeLists.txt ini mengatur konfigurasi build untuk paket "webots_demo_pkg". File ini mendeklarasikan dependensi yang diperlukan, mengonfigurasi paket, dan menentukan eksekusi bernama "e_puck_manager" yang bergantung pada file sumber yang disediakan "src/e_puck_manager.cpp".

3. e_puck_manager.cpp

Kode yang disediakan adalah file "e_puck_manager.cpp", yang tampaknya merupakan implementasi node ROS untuk mengendalikan robot keping elektronik dalam simulasi Webots. Mari kita analisis kode dan fungsinya:

File header:

#include "ros/ros.h": Menyertakan file header ROS yang diperlukan untuk fungsi ROS.

<webots_ros/Int32Stamped.h>: Menyertakan file header pesan Webots ROS yang diperlukan untuk menangani nilai bilangan bulat.

<webots_ros/set_float.h> dan <webots_ros/set_int.h>: Menyertakan file header layanan Webots ROS yang diperlukan untuk mengatur nilai float dan integer.

<webots_ros/robot_get_device_list.h>: Menyertakan file header layanan Webots ROS yang diperlukan untuk mengambil daftar perangkat.

<std_msgs/String.h>: Termasuk file header pesan ROS yang diperlukan untuk menangani pesan string.

<geometry_msgs/Twist.h>: Termasuk file header pesan ROS yang diperlukan untuk menangani pesan Twist, yang biasanya digunakan untuk mengendalikan gerakan robot.

Variabel Global:

static char modelList[10][100]: Mendefinisikan larik statis untuk menyimpan nama-nama model robot.

static int cnt = 0: Menginisialisasi variabel penghitung untuk melacak jumlah model robot.

Fungsi Panggilan Balik:

cmdVelCallback: Fungsi ini adalah panggilan balik untuk topik "cmd_vel". Fungsi ini menerima pesan Twist dan menghitung kecepatan roda kiri dan kanan berdasarkan kecepatan linier dan sudut.

modelNameCallback: Fungsi ini adalah pemanggilan balik untuk topik "model_name". Fungsi ini menerima nama model robot dan menyimpannya dalam array modelList.

Fungsi Utama:

ros::init: Menginisialisasi node ROS.

ros::NodeHandle n: Membuat NodeHandle ROS untuk berkomunikasi dengan sistem ROS.

ros::Subscriber nameSub: Berlangganan ke topik "model_name" untuk menerima nama model robot.

while (cnt == 0): Menunggu sampai setidaknya satu nama model diterima.

modelName = modelList[1]: Mengatur variabel modelName dengan nama model robot pertama.

ros::Subscriber cmdVelSub: Berlangganan ke topik "cmd_vel" untuk menerima perintah kecepatan.

Menyiapkan kontrol kecepatan untuk motor:

Kode ini membuat instance dari layanan webots_ros::set_float dan menetapkan nilainya ke INFINITY, yang menetapkan mode kontrol motor ke kontrol kecepatan.

Klien layanan dibuat untuk motor roda kiri dan kanan untuk mengatur posisi dan nilai kontrol kecepatannya.

ros::Rate r(10): Mengatur laju putaran ke 10 Hz (menjalankan putaran 10 kali per detik).

Loop kontrol utama:

Di dalam loop, kecepatan roda kiri dan kanan diatur berdasarkan nilai pesan Twist yang diterima.

Panggilan servis dilakukan untuk memperbarui kecepatan roda kiri dan kanan.

r.sleep(): Menunda eksekusi loop untuk mencapai kecepatan loop yang diinginkan.

ros::spinOnce(): Mengizinkan ROS untuk memproses setiap pemanggilan yang tertunda.

Secara keseluruhan, kode ini mengatur node ROS yang berlangganan topik "cmd_vel" untuk menerima perintah kecepatan dan mengontrol kecepatan roda kiri dan kanan robot keping es dalam simulasi Webots. Kode ini juga berlangganan topik "model_name" untuk menerima nama model robot.

4. e-puck_avoid_obstacles.c

Kode tersebut adalah program C berjudul e-puck_avoid_obstacles.c. Kode tersebut adalah pengontrol default untuk robot e-puck, yang bertujuan untuk menghindari rintangan dengan menggunakan algoritme Braitenberg.

Kode dimulai dengan beberapa inklusi header, diikuti dengan definisi berbagai tag perangkat dan nilai serta nama yang sesuai. Perangkat tersebut termasuk sensor jarak, lampu LED, dan motor. Kode ini juga mendefinisikan konstanta dan variabel yang digunakan dalam algoritma Braitenberg.

Program ini terdiri dari beberapa fungsi, termasuk get_time_step, step, passive_wait, init_devices, reset_actuator_values, get_sensor_input, cliff_detected, set_actuators, blink_leds, run_braitenberg, go_backward, dan turn_left. Fungsi-fungsi ini menangani berbagai aspek perilaku robot, seperti menginisialisasi perangkat, membaca input sensor, mengendalikan aktuator, dan menjalankan algoritma Braitenberg untuk menghindari rintangan.

Pada fungsi utama, program menginisialisasi robot e-puck, menginisialisasi perangkat, dan memasuki loop tak terbatas. Di dalam loop, program ini terus membaca input sensor, mengedipkan LED, memeriksa deteksi tebing, dan melakukan penghindaran rintangan atau bergerak sesuai dengan algoritme Braitenberg. Terakhir, kode ini menetapkan nilai aktuator dan langkah simulasi.

Secara keseluruhan, kode ini menyediakan implementasi dasar pengontrol penghindaran rintangan untuk robot keping elektronik menggunakan perilaku gaya Braitenberg.

5. robot_motion.cpp

Kode tersebut adalah program C++ berjudul robot_motion.cpp. Ini adalah pengontrol sederhana untuk robot di lingkungan simulasi Webots. Pengontrol mengatur motor robot untuk berputar ke arah yang berbeda untuk jangka waktu tertentu.

Kode ini menyertakan file header yang diperlukan dari perpustakaan Webots untuk berinteraksi dengan komponen robot seperti motor dan sensor jarak. Kode ini mendefinisikan konstanta MAX_SPEED yang mewakili kecepatan maksimum motor dan konstanta lain TIME_STEP yang menentukan durasi setiap langkah simulasi dalam milidetik.

Pada fungsi utama, program ini membuat sebuah instance dari kelas Robot untuk menginisialisasi robot dan membuat koneksi dengan simulasi Webots. Kemudian mengambil objek motor untuk roda kiri dan kanan robot menggunakan namanya.

Di dalam perulangan perulangan tak terbatas, program mengatur kecepatan motor kiri dan kanan ke sebagian kecil dari kecepatan maksimum (MAX_SPEED). Arah rotasi untuk setiap motor (l_direction dan r_direction) dikontrol dengan mengalikan kecepatan dengan 1.0 atau -1.0, yang menghasilkan gerakan maju atau mundur.

Fungsi robot->step(TIME_STEP) dipanggil untuk memajukan simulasi sebanyak satu langkah. Variabel t ditambah dengan TIME_STEP untuk melacak waktu yang telah berlalu. Setelah periode tertentu (t > 2000), arah motor kanan dibalik (r_direction *= -1.0), menyebabkan robot mengubah putarannya. Jika t melebihi 4000, arah motor kanan disetel ulang ke 1.0, dan t disetel kembali ke 0.0, memulai rotasi lagi.

Program melanjutkan gerakan bolak-balik ini tanpa batas waktu hingga dihentikan secara manual. Akhirnya, objek Robot dihapus, dan program diakhiri dengan mengembalikan nilai 0.

Secara keseluruhan, kode ini menunjukkan contoh dasar untuk mengontrol gerakan robot dalam lingkungan simulasi Webots menggunakan C++.

6. e_puck_manager.launch

Kode ini adalah file peluncuran untuk kerangka kerja ROS (Robot Operating System). File ini berjudul e_puck_manager.launch dan digunakan untuk meluncurkan simulasi Webots dengan file dunia tertentu dan memulai simpul ROS yang disebut e_puck_manager.

Berikut ini adalah rincian file peluncuran:

File dimulai dengan deklarasi XML <?xml version="1.0"?>.

Tag <launch> menunjukkan awal file peluncuran.

Tag <arg> digunakan untuk mendefinisikan argumen bernama "no-gui" dengan nilai default "false". Argumen ini digunakan untuk menentukan apakah akan memulai Webots dengan atau tanpa antarmuka pengguna grafis (GUI).

Tag <include> digunakan untuk menyertakan dan meluncurkan file peluncuran lain, dalam hal ini, file webots.launch dari paket webots_ros. Hal ini dilakukan untuk memulai lingkungan simulasi Webots.

Di dalam tag <include>, terdapat beberapa tag <arg> yang menentukan argumen tambahan untuk file peluncuran yang disertakan. Argumen mode diatur ke "realtime" untuk menjalankan simulasi dalam waktu nyata, dan argumen no-gui diatur ke nilai argumen "no-gui" yang ditentukan sebelumnya dalam file peluncuran ini. Argumen world menentukan jalur ke file world Webots yang akan dimuat untuk simulasi.

Setelah menyertakan file peluncuran Webots, tag <node> digunakan untuk mendefinisikan node ROS yang disebut "e_puck_manager". Node ini merupakan bagian dari paket "webots_demo_pkg" dan tipenya adalah "e_puck_manager". Atribut keluaran diatur ke "screen" untuk mengarahkan keluaran node ke layar.

File peluncuran diakhiri dengan tag penutup </launch>.

Secara keseluruhan, file peluncuran ini digunakan untuk meluncurkan lingkungan simulasi Webots dengan file dunia tertentu dan memulai node ROS yang disebut "e_puck_manager". Ini menyediakan integrasi antara Webots dan ROS, yang memungkinkan komunikasi dan kontrol antara robot simulasi dan ekosistem ROS.

Result

Sebagai hasilnya, robot e-puck dapat melakukan tugas-tugas berikut:

Menghindari Rintangan: Kode e-puck_avoid_obstacles.c mengimplementasikan perilaku penghindaran rintangan menggunakan algoritme Braitenberg. Robot menggunakan sensor jarak untuk mendeteksi rintangan di sekelilingnya dan menyesuaikan kecepatan motor untuk menghindarinya. Hal ini memungkinkan robot e-puck untuk bernavigasi di lingkungan dengan rintangan sambil menghindari tabrakan.

Gerakan Maju dan Mundur: Kode robot_motion.cpp menunjukkan perilaku gerakan sederhana di mana robot e-puck bergerak maju dengan kecepatan konstan. Setelah durasi tertentu, arah gerakan dibalik untuk waktu yang singkat, menyebabkan robot bergerak mundur. Pola ini berulang, menghasilkan gerakan bolak-balik robot e-puck.

Integrasi ROS: File peluncuran e_puck_manager.launch mengintegrasikan robot keping elektronik dengan kerangka kerja ROS (Sistem Operasi Robot). Ini meluncurkan simulator Webots dengan file dunia yang ditentukan dan memulai node e_puck_manager dari paket webots_demo_pkg. Integrasi ini memungkinkan komunikasi dan interaksi antara robot e-puck dan node ROS lainnya, memfasilitasi pengembangan perilaku dan aplikasi robot yang lebih kompleks.

Singkatnya, robot e-puck dapat secara mandiri menghindari rintangan, bergerak dalam pola gerakan maju-mundur, dan berinteraksi dengan node ROS lainnya untuk meningkatkan fungsionalitas.

Conclusion

Kesimpulannya, robot e-puck yang dilengkapi dengan sensor jarak dan motor, menunjukkan kemampuan yang mengesankan dalam menghindari rintangan dan kontrol gerak. Melalui penerapan

algoritma Braitenberg, robot secara cerdas menavigasi lingkungannya dengan menyesuaikan kecepatan motor berdasarkan input sensor, secara efektif menghindari tabrakan dengan rintangan. Selain itu, robot ini menampilkan pola gerakan maju-mundur, bergantian antara gerakan maju dan mundur, memberikan keserbagunaan dalam pergerakannya. Selain itu, integrasi robot e-puck dengan kerangka kerja ROS membuka kemungkinan untuk komunikasi dan interaksi tanpa batas dengan node ROS lainnya, memungkinkan pengembangan perilaku dan aplikasi yang lebih canggih. Robot e-puck berfungsi sebagai platform yang berharga untuk mengeksplorasi strategi navigasi otonom dan menawarkan potensi untuk kemajuan lebih lanjut dalam penelitian dan pengembangan robotika.

Reference

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