Міністерство освіти та науки України
Львівський національний університет імені Івана Франка
Факультет електроніки та ком´ютерних технологій
Кафедра радіофізики та комп’ютерних технологій

КУРСОВА РОБОТА

на тему: «Огляд можливостей ROS»

Виконав:
студент групи ФеІм-11
Сало Роман Андрійович
Науковий керівник:
доц. Кушнір О. О.

Львів – 2023

АНОТАЦІЯ

У цій курсовій роботі розглядаються різні засоби, які використовуються у робототехніці: мови програмування, фреймворки, бібліотеки, середовища розробки та симуляції. Оскільки тема роботи зв’язана з ROS, далі акцент робиться саме на ній. Розглядаються та детально описуються основні можливості ROS, його часто використовувані пакети, функції. В кінці приводиться простий приклад коду, який використовує можливості ROS.

ANNOTATION

This coursework examines the various instruments used in robotics: programming languages, frameworks, libraries, development and simulation environments. Since the topic is related to ROS, the emphasis is placed on it. The main features of ROS, its frequently used packages, and functions are discussed and described in detail. At the end, there is given a simple code example that uses the capabilities of ROS.

ЗМІСТ

Вступ

1. Загальний огляд засобів у робототехніці
   1.1. Мови програмування
   1.2. Фреймворки та бібліотеки
   1.3. Засоби для симуляцій
2. Огляд можливостей ROS
   2.1. Головні концепції ROS
   2.2. Структура файлової системи ROS
   2.3. Структура обчислювального графа
   2.4. Структура пакетів ROS
   2.5. Огляд основних пакетів та бібліотек ROS
      2.5.1. Засоби комунікації
      2.5.2. Обробка зображень, 3D-даних
      2.5.3. Робота з координатами, їх перетворення
      2.5.4. Навігація, планування руху
3. Приклад коду програми на ROS

Висновки

Література

Додатки

ВСТУП

Робота в галузі робототехніки стає все більш важливою в сучасному світі. Розробники роботів та автоматизованих систем шукають надійні та ефективні інструменти, які допоможуть їм впроваджувати складні функціональні можливості та забезпечувати взаємодію між різними компонентами системи. У цьому контексті Robot Operating System (ROS) визнаний як одна з провідних платформ для розробки роботів та розподілених систем.

ROS — це відкрита та гнучка система для розробки роботів, яка забезпечує широкий набір інструментів і бібліотек для побудови розподілених систем у робототехніці. Вона була розроблена з орієнтацією на спрощення процесу створення роботів та сприяння колаборативній розробці. ROS надає набір інструментів для керування ресурсами, комунікації між модулями, візуалізації даних та багато іншого. Завдяки модульній архітектурі та багатому співтовариству розробників, ROS дозволяє ефективно створювати складні роботизовані системи, розподілені по різних комп'ютерах та платформах.

1. ЗАГАЛЬНИЙ ОГЛЯД ЗАСОБІВ У РОБОТОТЕХНІЦІ

Засоби робототехніки охоплюють багато різних інструментів: мов програмування, середовищ розробки, моделювання та симуляцій.

1.1. Мови програмування

Опишемо мови програмування, які найчастіше використовуються у цій галузі:

- C/C++: одна з найпоширеніших мов програмування у робототехніці. Вона відома своєю продуктивністю, близьким до машинного рівня доступом до апаратного забезпечення та потужними бібліотеками.
- Python: теж одна з найпопулярніших мов у цій галузі, завдяки своїй простоті, легкості використання та широкому спектру наявних бібліотек. Зазвичай використовується для розробки високорівневих функцій, обробки даних та швидкого прототипування.
- Java: також має застосування у робототехніці, завдяки своїй переносимості – один і той самий код можна легко переносити між пристроями, які підтримують JVM.
- C#: використовується для програмування на рівні портів та сокетів, та для впровадження нейронних мереж.
- MATLAB: часто використовується для обчислень, моделювання та симуляцій.
- HDL: більш низькорівневі засоби, які дозволяють програмувати на рівні електронних та логічних схем.

1.2. Фреймворки та бібліотеки

У робототехніці пропонується великий вибір готових систем, як цілих фреймворків для різноманітних задач, так і більш вузькоспеціалізованих бібліотек певних алгоритмів. Опишемо деякі з них:

- ROS (Robot Operating System) – операційна система для програмування роботів, яка надає великий спектр інструментів та пакетів для різних цілей. Підтримує мови C++ та Python.
- YARP (Yet Another Robot Platform) – набір бібліотек для забезпечення взаємодії між сенсорами, приводами та процесорами.
- MRPT (Mobile Robot Programming Toolkit) – надає набір бібліотек для розробки мобільних роботів, який включає в себе засоби комп’ютерного зору, кінематики, SLAM-алгоритми та багато іншого.
- OROCOS (Open Robot Control Software) – набір засобів для контролю роботів, який включає в себе засоби для кінематики, фільтрації, машин станів. Може інтегруватися з ROS та YARP.
- OpenCV – бібліотека засобів та алгоритмів комп'ютерного зору та обробки зображень. Надає можливості для аналізу вмісту зображень, розпізнавання образів, відстежування руху об'єктів, перетворення зображень та багато іншого.
- PCL (Point Cloud Library) – набір інструментів для обробки зображень та хмар точок.
- MoveIt – набір засобів для керування роботом, включаючи алгоритми планування руху, кінематики, перевірки колізій та інше.
- DART (Dynamic Animation and Robotics Toolkit) – бібліотека, яка надає структури даних та алгоритми для кінематичних і динамічних додатків у робототехніці та комп'ютерній анімації.

1.3. Засоби для симуляцій

Симуляції дозволяють відтворити поведінку робота, його взаємодію з різними предметами та навколишнім середовищем. Симулятори пропонують багато різних функцій та можливостей, наприклад фізичні рушії з підтримкою фізичних сил, твердих та м’яких тіл, рідин; рушії для рендерингу сцен, різні компоненти для побудови моделей роботів (двигуни, приводи, маніпулятори), симуляцію сенсорів, середовища програмування.

Серед досить популярних симуляторів можна виділити Gazebo, Webots, CoppeliaSim, SOFA, а також декілька інших. Кожен має свої переваги і недоліки, і підтримує різний перелік засобів та інструментів. Нижче наведена порівняльна таблиця симуляторів за підтримуваними інструментами.

Simulator	RGBD + LiDAR	Force Sensor	Linear + Cable Acutator	Multi-Body Import	Soft-Body Contacts	DEM Simulation	Fluid Mechanics	Headless Mode	ROS Support	HITL	Teleoperation	Realistic Rendering	Inverse Kinematics
Airsim	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓, unreal	✗
CARLA	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓, unreal	✗
CoppeliaSim	✓	✓	Linear only	✓	✗	✓	✗	✓	✓	✓	✓	✗	✓
Gazebo	✓	✓	Linear only	✓	✗	Through Fluidix	Through Fluidix	✓	✓	✓	✓	✗	✓
MuJoCo	✓	✓	✓	✓	✓	✓	Limited	✓	✗	HAPTIX only	HAPTIX only	✗	✗
PyBullet	✓	✓	Linear only	✓	✓	✓	✗	✓	✗	✗	✓	✗	✓
SOFA	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓, Unity	✗
UWSim	RGBD only	✓	✗	✓	✗	✗	✗	✓	✓	✓	✓	✓, custom	✗
Chrono	✓	✓	✓	✗	✓	✓	✓	✓	✗	✗	✓	✓, offline	✓
Webots	✓	✓	linear	✓	✗	✗	Limited	✓	✓	✗	✓	✗	✗

Simulator	GPS	Tracks	Wheels	Legs	Mecanum / Omni Wheels	Heightmap Import	OpenDrive / OpenStreetMap	Pathplanning	ROS Support	RGBD	LiDAR	Realistic Rendering
Gazebo	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
CoppeliaSim	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
Raisim	✗	✗	✓	✓	✗	✓	✗	✗	✗	✓	✓	✓, Unity
Webots	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
PyBullet	✗	✗	✓	✓	✗	✓	✗	✓	✗	✓	✓	✗
CARLA	✓	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓, Unreal
Project Chrono	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓, POV-Ray

Simulator	Pathplanning	Inverse Dynamics	Inverse Kinematics	Suction	Deformable Objects	Force/Torque Sensor	Realistic Rendering
SimGrasp	✓	✓	✓	✗	✗	✓	✗
Gazebo	✓	✓	✓	✗	✗	✓	✗
CoppeliaSim	✓	✗	✓	✓	✗	✓	✗
Pybullet	✓	✓	✓	✗	✓	✓	✗
MuJoCo	✗	✓	✗	✗	✓	✓	✗
NVidia Isaac	✓	✗	✓	✓	✓	✓	✓

Рис. 1.1: порівняльні таблиці робототехнічних симуляторів

2. ОГЛЯД МОЖЛИВОСТЕЙ ROS

Robot Operating System (ROS) – це широко використовуваний у робототехніці фреймворк. У ROS є стандартні можливості операційної системи, як-от апаратна абстракція, керування пристроями на низькому рівні, реалізовано часто використовувану функціональність, передавання повідомлень між процесами та керування бібліотеками.

Архітектура ROS заснована на графі з централізованою топологією. Обробка відбувається у вузлах, які можуть приймати або відправляти дані з датчиків, систем контролю стану і планування, приводів і т. д. Бібліотека орієнтована на Unix-подібні системи.

Пакет *-ros-pkg є загальним репозиторієм для розробки високорівневих бібліотек. Багато з можливостей часто асоційовані з ROS, такі як бібліотеки навігації та візуалізатор Rviz, зберігаються в цьому репозиторії. Ці бібліотеки надають потужний набір інструментів (різні візуалізатори, симулятори, засоби налагодження) для спрощення роботи.

ROS поширюється на умовах ліцензії BSD і є ПЗ з відкритим вихідним кодом. ROS безкоштовна для дослідницьких і комерційних цілей. ROS сприяє повторному використанню коду, тож розробники робототехніки та вчені можуть не створювати потрібну функціональність кожен раз з нуля. Можна отримати код із репозиторію, змінити його і поділитися поліпшеним кодом. Є можливість написати драйвер власного датчика для ROS.

ROS підтримує паралельні обчислення, має хорошу інтеграцію з популярними C++ бібліотеками, такими як як OpenCV, Qt, Point Cloud Library та ін. Вона може працювати на одноплатних комп'ютерах та мікроконтролерах.

2.1. Головні концепції ROS

В архітектурі ROS можна виділити три концептуальні рівні:

1. Рівень файлової системи (Filesystem level);

2. Рівень обчислювального графа (Computation Graph level);

3. Рівень спільноти (Community level).

Перший рівень – це рівень файлової системи. На цьому рівні розташована внутрішня структура ROS – структура папок, файли, необхідні для роботи.

Другий рівень – це рівень обчислювального графа, на якому відбувається взаємодія між процесами та системами. На цьому рівні знаходяться концепції та модулі, які слугують для створення систем, обробки всіх процесів, комунікації з більш ніж одним комп'ютером і т. д.

Третій рівень – це рівень спільноти. Цей рівень містить інструменти та концепції для обміну знаннями, алгоритми та код від будь-якого розробника. Цей рівень дуже важливий, оскільки ROS швидко зростає за потужної підтримки з боку спільноти.

2.2. Структура файлової системи ROS

Докладніше розглянемо перший рівень архітектури ROS. Як і у випадку звичайної операційної системи, програми в ROS розділені на папки, у яких містяться деякі файли, що описують її функціональність:

1. Пакети (Packages): формують атомарний рівень ROS. Пакет має мінімальну структуру і вміст, щоб створити програму в ROS. Він може мати виконувані процеси (вузли або node), файли конфігурації тощо;

2. Декларації (Manifests): міститься інформація про пакети, ліцензійна інформація, залежності, прапорці компіляції та інше. Керування деклараціями здійснюється через файл manifests.xml;

3. Стеки (Stacks): коли ви збираєте разом кілька пакетів для отримання деякої функціональності, то отримаєте стек. У ROS, існує багато таких стеків для різних цілей, наприклад, стек навігації;

4. Декларації стеків (Stack manifests): надають дані про стек, включно з його ліцензійною інформацією та його залежністю від інших стеків;

5. Типи повідомлень (Message types, msg): повідомлення є інформацією, яку процес відправляє іншим процесам. В ROS є безліч стандартних типів повідомлень. Опис повідомлення зберігається в my_package/msg/MyMessageType.msg;

6. Типи сервісів (Service types, src): описи сервісів зберігаються в my_package/srv/MyServiceType.srv. Вони визначають у ROS структуру запитів і відповідей для сервісів.

7. Репозиторії: місце, де зберігаються пакети та стеки, які можна отримати, використовуючи систему контролю версій.

2.3. Структура обчислювального графа

Обчислювальний граф – це peer-to-peer мережа процесів ROS, які спільно обробляють дані. Основними одиницями обчислювального графа ROS є вузли, майстер, сервер параметрів, повідомлення, сервіси, теми та пакети, кожен з яких надає дані до графа у різний спосіб. Ці концепції реалізовані в репозиторії ros_comm.

Розглянемо детальніше кожен елемент:

- Вузли: це процеси, які виконують обчислення. ROS розроблена таким чином, щоб бути модульною. Система управління роботом зазвичай складається з багатьох вузлів. Наприклад, один вузол керує лазерним далекоміром, інший керує колісними двигунами, третій – виконує локалізацію, інший – планування шляху, ще один – забезпечує графічне представлення системи і так далі. Вузол ROS пишуться з використанням клієнтської бібліотеки ROS, такої як roscpp або rospy.
- Майстер: забезпечує реєстрацію імен та пошук по обчислювальному графу. Без майстра вузли не змогли б знайти один одного, обмінятися повідомленнями або викликати сервіси.
- Сервер параметрів: дозволяє зберігати дані за ключами у централізованому місці. В останніх версіях цей функціонал виконує майстер.
- Повідомлення: вузли спілкуються один з одним за допомогою повідомлень. Повідомлення – це просто структура даних, що складається з набору полів. Підтримуються стандартні примітивні типи (int, float, bool тощо), а також масиви примітивних типів. Повідомлення можуть містити довільно вкладені структури та масиви.
- Теми: повідомлення проходять через транспортну систему з семантикою публікації/підписки. Вузол надсилає повідомлення, публікуючи його у певній темі. Тема – це ім'я, яке використовується для ідентифікації вмісту повідомлення. Вузол, який зацікавлений у

певному типі даних, підписується на відповідну тему. Для однієї теми може існувати декілька публікуючих і підписників, а один вузол може публікувати і/або підписуватися на декілька тем. Загалом, публікуючі та підписники не знають про існування один одного. Ідея полягає в тому, щоб відокремити виробництво інформації від її споживання. Логічно, можна уявити тему як строго типізовану шину повідомлень. Кожна шина має назву, і будь-хто може підключитися до неї, щоб надсилати або отримувати повідомлення, якщо вони мають правильний тип.

- Сервіси: модель публікації/підписки є дуже гнучкою комунікаційною парадигмою, але її односторонній транспорт "багато-до-багатьох" не підходить для взаємодії «запит/відповідь», яка часто потрібна в розподіленій системі. Запит/відповідь здійснюється через сервіси, які визначаються парою структур повідомлень: одна для запиту і одна для відповіді. Вузол, що надає послугу, пропонує послугу під певним ім'ям, а клієнт використовує послугу, надсилаючи повідомлення із запитом та очікуючи на відповідь. Клієнтські бібліотеки ROS зазвичай представляють цю взаємодію програмісту так, ніби це віддалений виклик процедури.
- Пакети (Bags): це формат для збереження та відтворення даних ROS-повідомлень. Пакети є важливим механізмом для зберігання даних, таких як дані з датчиків, які може бути важко зібрати, але які необхідні для розробки та тестування алгоритмів.

Розглянемо детальніше деякі моменти. Майстер ROS діє як служба імен у обчислювальному графі. Він зберігає реєстраційну інформацію про теми та сервіси для вузлів ROS. Вузли зв'язуються з майстром, щоб повідомити свою реєстраційну інформацію. Спілкуючись з майстром, ці вузли можуть отримувати інформацію про інші зареєстровані вузли та встановлювати з'єднання, якщо це необхідно. Майстер також робить зворотні виклики до цих

вузлів, коли ця реєстраційна інформація змінюється, що дозволяє вузлам динамічно створювати з'єднання при запуску нових вузлів.

Вузли з'єднуються з іншими вузлами напряму; майстер лише надає інформацію для пошуку, подібно до DNS-сервера. Вузли, які підписуються на тему, запитують з'єднання у вузлів, які публікують цю тему, і встановлюють це з'єднання за узгодженим протоколом з'єднання. Найпоширеніший протокол, що використовується в ROS, називається TCPROS, який використовує стандартні сокети TCP/IP.

Ця архітектура дозволяє здійснювати відокремлену роботу, де імена є основним засобом, за допомогою якого можна будувати більші та складніші системи. Кожна клієнтська бібліотека ROS підтримує зміну імен з командного рядка, що означає, що скомпільовану програму можна переконфігурувати під час виконання для роботи в іншій топології обчислювального графа.

2.4. Структура пакетів ROS

Пакет може містити вузли ROS, незалежну від ROS бібліотеку, набір даних, конфігураційні файли, стороннє програмне забезпечення або будь-що інше, що логічно становить корисний модуль. Мета пакетів – надати цю корисну функціональність у зручному для використання вигляді, щоб програмне забезпечення можна було легко використовувати повторно. Загалом, пакети ROS дотримуються принципу «золотої середини»: достатньо функціональності, щоб бути корисними, але не надто багато, щоб він був важким і його було важко використовувати з іншим програмним забезпеченням.

Зазвичай пакети притримуються такої структури:

- include/package_name: включає заголовки бібліотек, які потрібно експортувати в CMakeLists.txt;
- msg/: папка для зберігання типів повідомлень;
- src/: вихідний код програми;
- srv/: папка для зберігання типів сервісів;
- scripts/: папка для виконуваних Bash- чи Python-скриптів;
- CMakeLists.txt: файл, необхідний для збірки пакетів. Такий файл описує, як зібрати код і куди його встановити;
- package.xml: маніфест пакета, який описує його властивості – ім’я пакета, версію, авторів, залежності.

2.5. Огляд основних пакетів та бібліотек ROS

Розглянемо детальніше основні та найбільш використовувані пакети.

2.5.1. Засоби комунікації

Пакет ros_comm включає в себе основні бібліотеки та компоненти ROS. Нижче представлені деякі з них:

- roscpp / rospy – клієнтські бібліотеки ROS (для C++ та Python відповідно), які дозволяють взаємодіяти з екосистемою ROS: темами, сервісами, параметрами;
- rosgraph – пакет, який дозволяє отримати інформацію про обчислювальний граф;
- rostopic, rosnode, rosservice, rosparam, rosmaster, rosmsg – надають засоби для роботи з компонентами ROS: темами, вузлами, сервісами і т.д.;
- message_filters – фільтри, які дозволяють передавати повідомлення при певних умовах або через певний час.

Пакет common_msgs містить типи найчастіше використовуваних повідомлень, наприклад:

- actionlib_msgs – повідомлення для різних дій;
- diagnostic_msgs – повідомлення для діагностики та моніторингу систем у ROS, які використовуються у стеку діагностики;
- geometry_msgs, shape_msgs – типи, які представляють геометричні примітиви та фігури (точка, полігон, вектор та ін.);
- nav_msgs – повідомлення, які використовуються в навігації;
- sensor_msgs – використовуються різними сенсорами, наприклад лазерами чи камерами.

Стек actionlib надає стандартизований інтерфейс для взаємодії з різноманітними завданнями. Прикладами цього є переміщення бази в цільове місце, виконання лазерного сканування і повернення отриманої хмари точок, виявлення ручки дверей тощо.

У будь-якій великій ROS-системі бувають випадки, коли хтось хоче відправити запит на вузол для виконання якогось завдання, а також отримати відповідь на запит. Це можна зробити за допомогою сервісів ROS. Однак у деяких випадках, якщо виконання сервісу займає багато часу, користувачеві може знадобитися можливість скасувати запит під час виконання або періодично отримувати зворотній зв'язок про хід виконання запиту. Пакет actionlib надає інструменти для створення серверів, які виконують довготривалі завдання. Він також надає клієнтський інтерфейс для надсилання запитів на сервер.

ActionClient і ActionServer взаємодіють за допомогою «ROS Action Protocol», який побудований на основі повідомлень ROS. Клієнт і сервер надають користувачам простий API для запиту цілей (на стороні клієнта) або для виконання цілей (на стороні сервера) за допомогою викликів функцій і зворотних викликів.

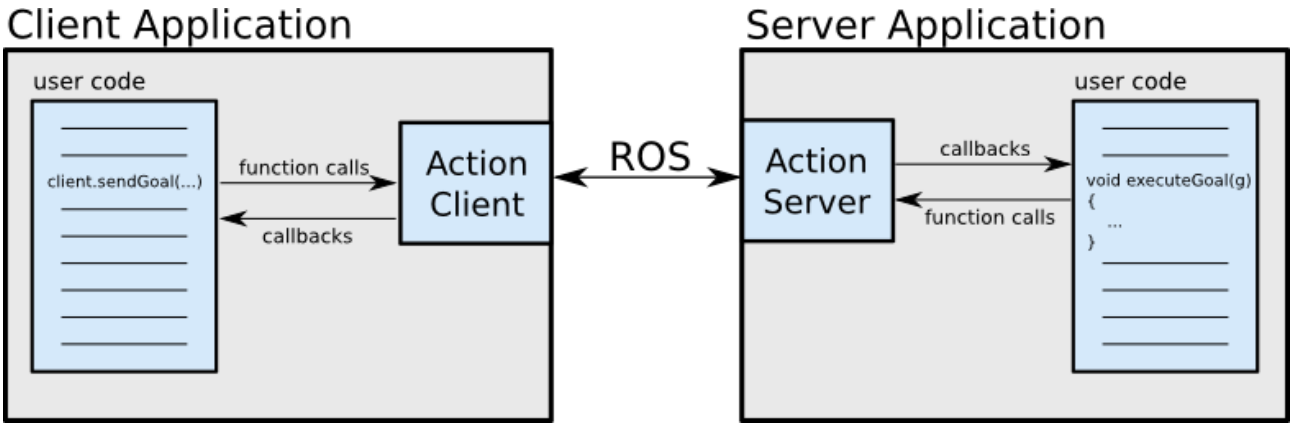

Рис. 2.1: ілюстрація взаємодії клієнта і сервера у actionlib

Для того, щоб клієнт і сервер могли взаємодіяти, потрібно визначити кілька повідомлень, за допомогою яких вони спілкуватимуться. Це робиться за

допомогою специфікації дій. Вона визначає повідомлення «Мета», «Зворотній зв'язок» і «Результат», за допомогою яких клієнт і сервер взаємодіють:

- Мета – її клієнт може надіслати серверу дій ActionServer. Наприклад, якщо ми хочемо перемістити робота, метою буде повідомлення PoseStamped, яке містить інформацію про те, куди робот повинен переміститися у світі. Для керування лазерним сканером мета буде містити параметри сканування (мінімальний кут, максимальний кут, швидкість і т.д.);
- Зворотній зв'язок – надає розробникам сервера спосіб повідомити ActionClient про поступовий прогрес у досягненні цілі. Для переміщення основи це може бути поточна позиція робота на шляху. Для керування лазерним сканером це може бути час, що залишився до завершення сканування;
- Результат – надсилається від ActionServer до ActionClient після завершення цілі. Він відрізняється від зворотного зв'язку, оскільки надсилається лише один раз. Це корисно, коли метою дії є надання певної інформації. Результатом переміщення може буде кінцева позиція робота. Для керування лазерним сканером результат може містити хмару точок, згенеровану на основі сканування.

2.5.2. Обробка зображень, 3D-даних

Зазвичай роботи оснащені різними сенсорами, які зчитують інформацію про навколишнє середовище у вигляді 2D-зображень чи набору точок у 3D. У ROS є пакети для обробки такої інформації.

Одним з основних засобів обробки зображень є стек image_common, який включає в себе:

- image_transport: надає підтримку для транспортування зображень у стиснених форматах. Має використовуватися усіма, хто публікує або підписується на зображення;

- camera_calibration_parsers: містить процедури для читання і запису параметрів калібрування камери. Переважно використовується драйверами камер;
- camera_info_manager: інтерфейс для збереження, відновлення та налаштування інформації про калібрування камери. Використовується переважно драйверами камер;
- polled_camera: інтерфейс для запиту зображень від драйвера камери.

Стек image_pipeline призначений для обробки необроблених зображень з камери у вхідні дані для алгоритмів. Компоненти включають:

- Калібрування: камери мають бути відкалібровані, щоб пов'язати зображення, які вони створюють, з тривимірним світом. Пакет camera_calibration надає інструменти для калібрування монокулярних і стереокамер у системі ROS.
- Монокулярна обробка: Потік необроблених зображень можна пропустити через вузол image_proc для усунення спотворень, спричинених камерою. Вузол також виконує інтерполяцію кольорів для кольорових камер Байєра.

- Стереообробка: вузол stereo_image_proc виконує обов'язки image_proc для пари камер, відкаліброваних для стереозображення. Він також використовує стереообробку для створення диспропорційних зображень і хмар точок.

- Обробка глибини: depth_image_proc надає вузли для обробки глибинних зображень, наприклад, для створення хмар точок.

- Візуалізація: пакет image_view надає можливість для перегляду зображень. До нього також входить інструмент stereo_view для перегляду стереопар та диспропорційних зображень.

Пакет vision_opencv надає інтерфейс до популярної бібліотеки OpenCV у ROS. OpenCV надає велику кількість методів для обробки зображень.

Для роботи з 3D-даними, наприклад хмарами точок, слугують laser_pipeline та perception_pcl.

Метапакет laser_pipeline включає в себе засоби для обробки даних, отриманих з лазера та перетворення цих даних. Це перетворення складається з трьох етапів:

- Фільтрація: вимірювання, що надходять безпосередньо з лазерних сканерів, часто бувають зашумленими або містять невірні виміри. Зазвичай потрібно забезпечити певне згладжування, видалити помилкові виміри. Це виконується за допомогою різноманітних фільтрів у пакеті laser_filters;
- Проекція: після фільтрації варто перетворити отриманий набір даних у набір 3D-точок. Функції для ефективного перетворення з урахуванням таких ефектів, як нахил лазера, визначені у пакеті laser_geometry;
- Збірка: лазерний сканер перемщується у тривимірному просторі, щоб згенерувати хмару точок. Пакет laser_assembler ефективно агрегує сканування у часі і робить такі хмари доступними для інших вузлів за допомогою виклику сервісу ROS.

Метапакет perception_pcl надає функціонал для обробки хмар точок, їх перетворення у інші типи (повідомлення та ін.)

2.5.3. Робота з координатами, їх перетворення

Одним з найважливіших пакетів для цього є tf. Це пакет, який дозволяє користувачеві відстежувати декілька систем координат у часі. Пакет підтримує зв'язок між системами координат у деревовидній структурі, буферизованій у часі, і дозволяє користувачеві трансформувати точки, вектори тощо між будь-якими двома системами координат у будь-який бажаний момент часу.

Роботизована система зазвичай має багато 3D-координатних систем, які змінюються з часом, наприклад: світова система координат, базова система координат, системи координат маніпулятора, голови і т.д. Засоби пакету tf дозволяють відстежувати всі ці системи координат у часі.

tf може працювати у розподіленій системі. Це означає, що вся інформація про системи координат робота доступна всім компонентам ROS на будь-якому комп'ютері в системі.

Ще один важливий компонент – tf_conversions. Він містить набір функцій для перетворення типів даних tf (точка, вектор, позиція тощо) у семантично ідентичні типи даних, що використовуються іншими бібліотеками.

Пакет robot_state_publisher дозволяє публікувати стан робота в tf. Після публікації стан стає доступним для всіх компонентів системи, які також використовують tf. Пакет приймає кути з’єднань робота як вхідні дані і публікує 3D-позиції ланок робота, використовуючи кінематичну деревоподібну модель робота.

2.5.4. Навігація, планування руху

Стек navigation дозволяє керувати рухом та навігацією робота. Засоби цього стеку беруть на вхід покази сенсорів та цільову позицію, і на вихід

посилають команди, які відповідають за швидкість і напрям руху, щоб досягти цілі. Умовою для використання цього стеку є те, що робот повинен мати дерево систем координат з tf та публікувати дані з датчиків, використовуючи правильні типи повідомлень ROS. Крім того, навігаційний стек має бути налаштований відповідно до форми і динаміки робота, щоб продуктивно працювати.

Пакет move_arm стеку actionlib забезпечує реалізацію дії, яка, отримавши мету, намагатиметься досягти її за допомогою руки робота. Вузол move_arm об'єднує планувальник, монітор траєкторії та контролер траєкторії для досягнення мети. Пакет підтримує будь-який планувальник, який надає інтерфейс GetMotionPlan, і будь-який контролер, який надає послуги для встановлення, запиту і скасування траєкторій. Вузол спроектований як модульний, тому використовує сервіси ROS для планування, контролю та зворотної кінематики і виконує мінімум обчислень всередині самого вузла.

Також подібний функціонал надають пакети ompl, chomp_motion_planner, sbpl.

3. ПРИКЛАД КОДУ ПРОГРАМИ НА ROS

Розглянемо простий приклад програми – створення двох вузлів, один з яких розсилає повідомлення, а інший приймає їх.

Спочатку створимо вузол розсилки повідомлень. Для цього скачаємо файл talker.py, який буде містити код вузла-відправника, і дозволимо виконання цього файлу, ввівши наступні команди:

- wget https://raw.github.com/ros/ros_tutorials/kinetic-devel/rospy_tutorials/001_talker_listener/talker.py
- chmod +x talker.py

Після отримання файлу можемо переглянути його вміст та розглянути по рядках.

```
1   #!/usr/bin/env python3
2   # license removed for brevity
3   import rospy
4   from std_msgs.msg import String
5
6   def talker():
7       pub = rospy.Publisher('chatter', String, queue_size=10)
8       rospy.init_node('talker')
9       rate = rospy.Rate(10) # 10hz
10      while not rospy.is_shutdown():
11          hello_str = "hello world %s" % rospy.get_time()
12          rospy.loginfo(hello_str)
13          pub.publish(hello_str)
14          rate.sleep()
15
16  if __name__ == '__main__':
17      try:
18          talker()
19      except rospy.ROSInterruptException:
20          pass
```

Рис. 3.1: код файлу talker.py

Спочатку імпортуємо саму бібліотеку ROS (rospy) та тип повідомлень, які вузол буде надсилати. Далі, задаємо інтерфейс вузла. У рядку 7 вказуємо, що вузол публікує повідомлення типу String в тему chatter. У рядку 8 створюємо вузол, задавши йому ім’я, щоб він міг комунікувати із системою. Ім’я є важливою частиною, оскільки вузли ідентифікуються саме за ним. Важливо, щоб ім’я було унікальним. Додавання параметра anonymous=True вирішує цю проблему, додаючи кілька випадкових цифр до імені вузла. У рядку 9 задаємо частоту виконання циклу. В даному випадку 10 разів за секунду.

Далі, починаючи з рядка 10, запускається цикл, в якому все буде виконуватись, поки програма запущена. У цьому прикладі виводимо текстовий рядок на екран та у логи, публікуємо його як повідомлення. Метод sleep() призупиняє виконання на деякий час, щоб відповідати заданій частоті виконання.

Після блоку об’явлення вузла йде власне запуск цієї функції, в якій описали вузол. Рядок «except rospy.ROSInterruptException» у цій конструкції перехоплює виключення, яке викидається при натисканні Ctrl+C або при виключенні вузла. Це гарантує те, що код не продовжить виконуватись після методу sleep().

Приступимо до створення вузла, який приймає повідомлення. Отримаємо відповідний файл з кодом, ввівши команди:

- wget https://raw.github.com/ros/ros_tutorials/kinetic-devel/rospy_tutorials/001_talker_listener/listener.py
- chmod +x listener.py

```python
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the sam
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is
    rospy.spin()

if __name__ == '__main__':
    listener()
```

Рис. 3.2: код файлу listener.py

Спочатку оголосимо функцію callback(data), яка визначить, як вузол буде реагувати на отримане повідомлення та що з ним робити. Далі оголошуємо саме створення вузла: задаємо ім’я і підписуємо його, вказавши тему, тип повідомлень та callback-функцію, оголошену раніше. Рядок rospy.spin() не дає завершитись виконанню файлу, поки вузол не буде вимкнено.

Додамо ці два файли у CMakeLists.txt, запустимо catkin_make і запустимо наші два вузли. Нижче можемо побачити вивід у консолі: як і очікувалось, публікуючий вузол виводить своє повідомлення, приймаючий вузол виводить те, що він прочитав опубліковане повідомлення.

```
[INFO] [1664767089.808396]: hello world 1664767089.8082786
[INFO] [1664767089.908136]: hello world 1664767089.9079938
[INFO] [1664767090.008121]: hello world 1664767090.008
[INFO] [1664767090.108167]: hello world 1664767090.1079464
[INFO] [1664767090.208043]: hello world 1664767090.2079341
[INFO] [1664767090.308047]: hello world 1664767090.3079493
[INFO] [1664767090.408382]: hello world 1664767090.4080873
[INFO] [1664767090.508288]: hello world 1664767090.5081513
[INFO] [1664767090.608095]: hello world 1664767090.607984
[INFO] [1664767090.708295]: hello world 1664767090.708047
[INFO] [1664767090.808419]: hello world 1664767090.8080955
[INFO] [1664767090.908192]: hello world 1664767090.908053
[INFO] [1664767091.008039]: hello world 1664767091.0079508
[INFO] [1664767091.108439]: hello world 1664767091.1081138
[INFO] [1664767091.208467]: hello world 1664767091.2081246
[INFO] [1664767091.308234]: hello world 1664767091.3080456
[INFO] [1664767091.408532]: hello world 1664767091.4081488
[INFO] [1664767091.508188]: hello world 1664767091.5080357
[INFO] [1664767091.608459]: hello world 1664767091.6081212
[INFO] [1664767091.708657]: hello world 1664767091.7082067
[INFO] [1664767091.808531]: hello world 1664767091.8081598
[INFO] [1664767091.908404]: hello world 1664767091.9081273
[INFO] [1664767092.008213]: hello world 1664767092.0080926
[INFO] [1664767092.108135]: hello world 1664767092.1079993
[INFO] [1664767092.208466]: hello world 1664767092.2081287
[INFO] [1664767092.308290]: hello world 1664767092.3080893
```

```
[INFO] [1664767090.609500]: /listener_105107_1664767063616I heard hello world 1664767090.607984
[INFO] [1664767090.712314]: /listener_105107_1664767063616I heard hello world 1664767090.708047
[INFO] [1664767090.813191]: /listener_105107_1664767063616I heard hello world 1664767090.8080955
[INFO] [1664767090.909311]: /listener_105107_1664767063616I heard hello world 1664767090.908053
[INFO] [1664767091.009114]: /listener_105107_1664767063616I heard hello world 1664767091.0079508
[INFO] [1664767091.112881]: /listener_105107_1664767063616I heard hello world 1664767091.1081138
[INFO] [1664767091.212788]: /listener_105107_1664767063616I heard hello world 1664767091.2081246
[INFO] [1664767091.311032]: /listener_105107_1664767063616I heard hello world 1664767091.3080456
[INFO] [1664767091.413059]: /listener_105107_1664767063616I heard hello world 1664767091.4081488
[INFO] [1664767091.509889]: /listener_105107_1664767063616I heard hello world 1664767091.5080357
[INFO] [1664767091.613042]: /listener_105107_1664767063616I heard hello world 1664767091.6081212
[INFO] [1664767091.713342]: /listener_105107_1664767063616I heard hello world 1664767091.7082067
[INFO] [1664767091.812869]: /listener_105107_1664767063616I heard hello world 1664767091.8081598
[INFO] [1664767091.912937]: /listener_105107_1664767063616I heard hello world 1664767091.9081273
[INFO] [1664767092.009676]: /listener_105107_1664767063616I heard hello world 1664767092.0080926
[INFO] [1664767092.109804]: /listener_105107_1664767063616I heard hello world 1664767092.1079993
[INFO] [1664767092.213088]: /listener_105107_1664767063616I heard hello world 1664767092.2081287
[INFO] [1664767092.310750]: /listener_105107_1664767063616I heard hello world 1664767092.3080893
[INFO] [1664767092.410489]: /listener_105107_1664767063616I heard hello world 1664767092.4082022
[INFO] [1664767092.510535]: /listener_105107_1664767063616I heard hello world 1664767092.5080082
[INFO] [1664767092.612989]: /listener_105107_1664767063616I heard hello world 1664767092.6081326
[INFO] [1664767092.713122]: /listener_105107_1664767063616I heard hello world 1664767092.7081435
[INFO] [1664767092.812704]: /listener_105107_1664767063616I heard hello world 1664767092.8080943
```

Рис. 3.3: консольний вивід роботи вузлів talker та listener

ВИСНОВКИ

У цій курсовій роботі був проведений як поверхневий огляд інструментів у робототехніці в загальному, так і більш глибокий огляд конкретного інструменту (ROS).

Короткий опис різних засобів (фреймворків, мов програмування і т.п.) був приведений, щоб дати загальне представлення, що з себе представляє галузь робототехніки в плані наявних інструментів. Другий розділ з описом ROS мав на меті ознайомити читача з основними компонентами цієї системи, як вони виглядають та для чого використовуються. Наочно показано використання цих засобів у третьому розділі.

ЛІТЕРАТУРА

1. Top 5 Free and Open Source Robotics Frameworks. URL: https://medevel.com/top-5-free-and-open-source-robotics-frameworks/

2. Top 10 Python Libraries For Robotics. URL: https://analyticsindiamag.com/top-10-python-libraries-for-robotics/

3. What is the best simulation tool for robotics? URL: https://robohub.org/what-is-the-best-simulation-tool-for-robotics/

4. Баськов В. С. Кооперативний пошук шляху у середовищі ROS / Баськов В. С., Данилов В. Я.: Київ, 2021: https://ela.kpi.ua/bitstream/123456789/45279/1/Baskov_bakalavr.pdf

5. ROS.org. ROS / Concepts. URL: http://wiki.ros.org/ROS/Concepts

6. ROS.org. Packages. URL: http://wiki.ros.org/Packages

7. ROS.org. APIs. URL: http://wiki.ros.org/APIs

8. ROS.org. ROS / Tutorials / WritingPublisherSubscriber(python). URL: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

ДОДАТКИ

Додаток 1

Код файлу talker.py з оголошенням вузла, який публікує повідомлення

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Додаток 2

Код файлу listener.py з оголошенням вузла, який отримує повідомлення

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```