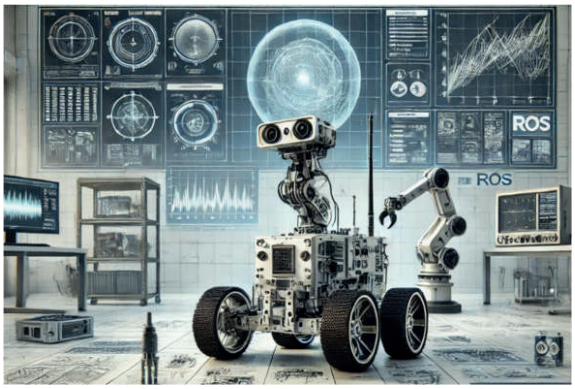
ЗМІСТ

ПЕРЕДСЛОВО



Світ сучасної робототехніки, це поєднання алгоритмів, сенсорних систем і програмного забезпечення, що дозволяє машинам приймати рішення та взаємодіяти із зовнішнім середовищем [3].

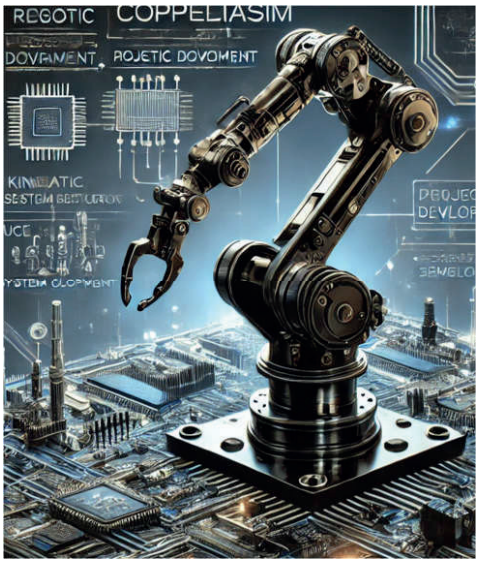
В практичному контексті програмування роботів виходить за межі звичайного написання коду, а перетворюється в мистецтво створення автономних систем, які можуть орієнтуватися у просторі, ухилятися від перешкод, взаємодіяти з іншими пристроями та приймати оптимальні рішення на основі отриманих даних. А ROS (Robot Operating System) є одним із найпотужніших інструментів для створення складної програмної архітектури, що дозволяють реалізовувати як базові алгоритми керування, так і високорівневі моделі машинного навчання та штучного інтелекту. Опанування ROS є ключовим етапом для будь-якого розробника, що прагне працювати в галузі мобільної робототехніки, промислових систем або безпілотних платформ.

Цей навчальний посібник присвячений саме практичному програмуванню роботів, охоплюючи не лише основи роботи з ROS, а й методики інтеграції сенсорів, налаштування симуляційних середовищ і реалізацію автономних алгоритмів керування. Особливу увагу приділено побудові гнучкої програмної архітектури, яка дозволяє швидко розгортати нові робототехнічні рішення та адаптувати їх під різні сценарії використання. Матеріал містить приклади коду, реальні кейси використання, а також рекомендації щодо тестування систем у симуляційних середовищах.

Опанування ROS, це не просто знайомство з операційною системою для роботів, а розуміння принципів модульного програмування, ефективної комунікації між вузлами та інтеграції апаратних компонентів. Саме тому книга охоплює розробку та налагодження програмних модулів для роботів, використання сенсорних систем, алгоритмів навігації та управління рухом**.**

Найважливішим у процесі навчання є не просто засвоєння технологій, а розвиток інженерного мислення. Основна увага приділяється практичним аспектам, що дозволяють отримати реальні навички**,** необхідні для роботи з автономними платформами. А практичні завдання створені так, щоб ви могли експериментувати, знаходити оптимальні рішення та створювати власні робототехнічні системи та світи.

Програмування роботів у сучасному світі неможливе без інтеграції штучного інтелекту, навчання з підкріпленням, адаптивного керування та інших інновацій сучасності. Саме тому в серії книг "Практичне програмування роботів" ми розглянемо автоматизоване навчання роботів, оптимізацію руху та взаємодію між кількома автономними платформами. Додатково вивчатимуться можливості військового застосування ROS, зокрема створимо й проведемо опис розширення Military Edition, що дозволить моделювати складні бойові сценарії, управління автономними дронами та аналіз стратегій у симуляціях бойових середовищ.



Цей матеріал створений для тих, хто хоче не лише ознайомитися з теоретичними основами ROS, а й навчитися практично застосовувати ці знання у реальних проєктах. Виконуючи завдання та експерименти, ви зможете не лише зрозуміти, як працюють автономні системи, але й розпочати власний шлях у сфері робототехнічного програмування, інтегруючи свої рішення не тільки у симуляційні середовища, а також в справжні апаратні рішення.

Робототехніка, це не просто майбутнє, а вже наше сьогодення.

Ця книга є запрошенням у світ автономних роботизованих систем, у якому знання, набуті сьогодні, відкриють двері до інноваційних розробок завтрашнього дня. Вперед до практичного програмування роботів!

Бажаємо Натхнення. Разом до Перемоги!
Автор і Колектив кафедри.

РОЗДІЛ 1
ЩО ТАКЕ ROS?

У розділі детально розглянуто основи роботи з ROS, зокрема, здійснення запуску першого вузла, процес взаємодії вузлів через топіки, управління черепашкою Turtlesim та візуалізацію зв'язків між компонентами. Завдяки цим прикладам, закладається основа для глибшого розуміння принципів розподіленої робототехнічної системи. Новачкам надана змога вперше зіткнутися з фундаментальними концепціями ROS, такими як вузли, топіки, повідомлення та їхній взаємозв'язок через ROS Master.
Особлива увага надана практичному вивченню CLI-команд й інструменту rqt_graph, які показали, як архітектура ROS дозволяє вузлам незалежно працювати та одночасно гармонійно інтегруватися в систему. Показано як інформація генерується вузлом turtlesim_node і передається через топіки у вигляді повідомлень та перетворюється на реальні дії, такі як керування рухом черепашки.
Цей розділ є символічним стартом у подорожі до освоєння ROS. Щоб надати розуміння основам комунікації між вузлами, механізмам інтеграції та модульність всієї системи в ROS. Досвід опанування цього розділу в подальшому закладає основу для створення більш складних систем, де автономність вузлів та їхня взаємодія через стандартизовані механізми є ключовими елементами.

Питання:

1. Ознайомлення з ROS, архітектурою та можливостями.
2. Встановлення ROS.
3. Запуск першого вузла та базові команди

Завдання:

1. Встановити ROS на власному комп'ютері.
2. Виконати налаштування catkin_ws.
3. Створити простий вузол, який друкує повідомлення в термінал кожні 5 секунд.

Результат опанування:

Готовність до роботи з вузлами та базовими командами.

1.1. Вступ до ROS

ROS (Robot Operating System), це програмний фреймворк, який надає структуру та інструменти для розробки робототехнічних систем. Незважаючи на назву, ROS не є операційною системою у традиційному розумінні. Це набір бібліотек і служб, які працюють поверх існуючих операційних систем (переважно Linux) та забезпечують середовище для створення модульних, взаємодіючих і багатофункціональних робототехнічних рішень.

До основних переваг ROS відносять, його *модульність, структуровану взаємодію* і *багатофункціональність*. Розглянемо кожну окремо взяту перевагу.

Модульність. ROS використовує концепцію модульної архітектури, що дозволяє розробникам розділяти функціональність робототехнічної системи на окремі компоненти, названі **вузлами (Nodes)**. Кожен вузол виконує конкретне завдання, наприклад, обробляє дані сенсорів, керує двигунами чи здійснює

навігацію. Це забезпечує легкість налаштування, тестування та повторного використання компонентів у різних проєктах.

Взаємодія. ROS надає можливість вузлам взаємодіяти через стандартизовані механізми, такі як *топіки (Topics)*, *сервіси (Services)* та *дії (Actions)*.

Топіки *Topics*	це асинхронні канали для обміну даними, які дозволяють одному вузлу публікувати інформацію, а іншим вузлам підписуватися на ці дані. Наприклад, сенсор може публікувати поточну позицію робота в топік, на який підписується вузол навігації.
Сервіси *Services*	забезпечують синхронну взаємодію вузлів за принципом "запит-відповідь". Наприклад, вузол може відправити запит іншому вузлу, для виконання обчислень або отримання певних даних.
Дії *Actions*	дозволяють організовувати асинхронні задачі, які потребують довготривалого виконання. Наприклад, рух робота до заданої точки.

Багатофункціональність. ROS пропонує широкий набір готових бібліотек і пакетів для роботи з сенсорами, камерами, маніпуляторами, алгоритмами навігації та візуалізації. Це дозволяє розробникам зосередитися на розв'язанні специфічних задач без необхідності створювати базову інфраструктуру з нуля.

Визначення і основні поняття ROS

ROS – це інструмент, що став стандартом у світі робототехніки [8]. Його модульна архітектура, універсальні механізми взаємодії та потужний набір бібліотек дозволяють розробникам ефективно створювати, тестувати та впроваджувати складні робототехнічні системи. Завдяки ROS інженери отримують можливість працювати на рівні високорівневих задач, а не заглиблюватися в технічні деталі інтеграції базових компонентів.

Розглянемо визначення і основні поняття, що існують в ROS.

ROS Master (roscore), це центральний компонент, який забезпечує взаємодію між вузлами. Він працює як координатор, відстежуючи, які вузли запущені і які топіки та сервіси вони використовують, а також сприяє встановленню зв'язків між ними. Компонент ROS Master потрібен для роботи всієї системи й запускається за допомогою команди (Listing 1.1):

Listing 1.1: command line

```
roscore
```

Вузли (Nodes), це незалежні програми, які виконують конкретні завдання в системі. Наприклад, один вузол може обробляти дані від LIDAR, а інший обробляє дані від камери. Завдяки їх автономності вузли можуть бути запущені на різних комп'ютерах і навіть у різних операційних системах, якщо вони взаємодіють через ROS Master.

Топіки (Topics), призначені для передачі потокових даних між вузлами. Наприклад, вузол публікує інформацію про швидкість робота в топік `/cmd_vel`, а

вузол контролера підписується на цей топік і керує рухом. Використання топіків дозволяють масштабувати систему та легко додавати нові компоненти.

Сервіси (Services), застосовуються для задач, де потрібна миттєва відповідь. Наприклад, сервіс може бути використаний для команди "перемістити маніпулятор у певну точку". У цьому випадку вузол-клієнт надсилає запит, а вузол-сервер відповідає, коли завдання завершено.

Дії (Actions), надають можливість виконувати довготривалі задачі з відстеженням прогресу. Наприклад, під час навігації робот може періодично повідомляти про свій статус "виконується", "досягнуто цілі" або "завдання скасовано". Це дозволяє зберігати контроль над складними процесами.

В якості прикладу застосування ROS, можна назвати багато реальних проектів таких як роботи Atlas від Boston Dynamics, або антропоморфний робот від Amazon Robotics. Яскравим прикладом існуючого DIY проєкту є *TurtleBot,* його системна платформа й набір сенсорів.

TurtleBot, це популярна електрона DIY платформа для навчання та досліджень у галузі робототехніки, що базується на ROS [10, 16]. Ця платформа оснащена різноманітними сенсорами, які дозволяють роботу сприймати навколишнє середовище та взаємодіяти з ним.

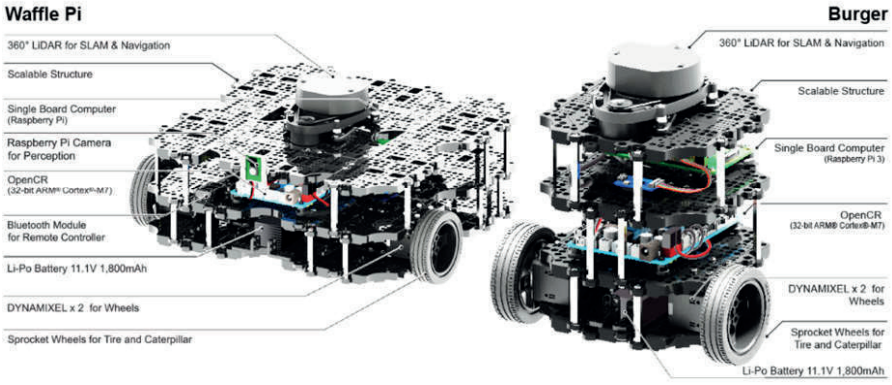


Рис 1.1. Вигляд й склад платформи TurtleBot, модель Waffle PI та Burger

Розглянемо більш детально основні компоненти TurtleBot та його сенсори. Платформа TurtleBot має декілька моделей (рис. 1.1), Waffle PI і Burger й містить:

1. Основну платформу, що складається з *механічної конструкції*, *двигунів й коліс*. Де ***механічна конструкція*** містить місця для кріплення додаткових сенсорів і обладнання, тим самим забезпечуючи універсальність платформи і мобільність робота. А ***двигуни і колеса*** відповідають за рух робота в просторі.

2. Набір сенсорів, який містить *лідар (LiDAR), камеру RGB-D, інфрачервоні та ультразвукові датчики,* а також *енкодери* що встановлені на колесах. ***Лідар,*** дозволяє виконувати SLAM картографування (будувати карти оточуючого середовища) та здійснювати навігацію, завдяки скануванню оточуючого середовища з кутом огляду 360°. ***Камера RGB-D*** забезпечує отримання

кольорових зображень та глибини бачення, що корисно для розпізнавання об'єктів і побудови тривимірних моделей. ***Інфрачервоні*** та ***ультразвукові датчики,*** використовуються при виявленні перешкод для забезпечення безпечного руху, а також можуть проводити вимірювання відстані до об'єкту або перешкоди. ***Енкодери, що встановлені на колесах***, дозволяють вимірювати швидкість руху і пройдену роботом відстань, що є важливими параметрами для одометрії.

3. Обчислювальний модуль (контролер прийняття рішення), зазвичай є одноплатним комп'ютером (SBC - single-board computer), наприклад Raspberry Pi, NVidia Jetson або Intel NUC. Він виконує обробку даних від сенсорів, реалізує алгоритми навігації та контролює робота.

4. Джерело живлення, зазвичай це ***акумулятор***, що забезпечує енергією всі електронні компоненти робота, дозволяючи йому працювати автономно протягом певного часу.

Офіційні ресурси містять детальні описи всіх компонентів, їх функцій та взаємодії між собою, тому вивчення їх буде корисним для розуміння структури системи та можливостей TurtleBot. Для більш детального ознайомлення з апаратною архітектурою та сенсорами TurtleBot, рекомендується самостійно ознайомитися з офіційною документацією:

Корисні ресурси, документація та посилання.

TurtleBot3 Overview: Robotis e-Manual [10] можна ознайомитися за посиланням: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

А **TurtleBot3 Additional Sensors:** Robotis e-Manual розташовано за посиланням: https://emanual.robotis.com/docs/en/platform/turtlebot3/additional_sensors/

1.2. Системні вимоги для роботи з ROS

Відповідність системним вимогам є критичною для роботи з ROS, оскільки навіть мінімальні затримки або нестача ресурсів можуть негативно вплинути на продуктивність робототехнічних систем. Забезпечення необхідної конфігурації дозволить розробникам повною мірою скористатися всіма перевагами ROS.

Для ефективної роботи з *Robot Operating System* необхідно забезпечити відповідність апаратного та програмного забезпечення певним вимогам. Ці вимоги спрямовані на підтримку стабільності системи, забезпечення обробки даних у реальному часі та використання інструментів, які входять до складу ROS.

1.2.1. Загальні вимоги до операційної системи

Існує багато різновидів ROS, наприклад *ROS Noetic* або *ROS 2 Humble*. Всі вони підтримуються виключно системами з операційними системами на базі Linux. Для встановлення і роботи з ROS існують рекомендації від розробника [8]:

Рекомендовані версії ОС, для ROS Noetic - Ubuntu 20.04, а для ROS 2 Humble - Ubuntu 22.04.

Для користувачів операційних систем MS Windows й macOS рекомендовано використання *Docker-контейнерів* або віртуальних машин Canonical Ubuntu.
Важливо зазначити щодо вибору ОС є певні особливості, такі як:

- Ubuntu забезпечує довготривалу підтримку (LTS), що є ключовим фактором для стабільної роботи ROS.
- Користувачам із попередніми версіями ROS (Kinetic, Melodic) слід враховувати, що підтримка цих версій припинена.

1.2.2. Вимоги до апаратної складової ЕОМ

A. Враховуючи багато ядерність процесора, що дозволяє ефективно виконувати одночасно кілька вузлів ROS, особливо для обробки даних сенсорів у реальному часі, рекомендованими вимогами до *центрального процесора (CPU) є:*

- *мінімум* двоядерний процесор із тактовою частотою 1.8 ГГц;
- *рекомендовано*, для нормальної роботи – чотириядерний процесор (Intel i5 або AMD Ryzen 5) із частотою 2.5 ГГц або вище.

B. Симуляція складних середовищ та запуск декількох вузлів може споживати значний обсяг пам'яті. Тому вимоги до *оперативної пам'яті (RAM)* становлять:

- *мінімум* 4 ГБ для базових проєктів;
- *рекомендовано* ОЗУ з об'ємом 8–16 ГБ для роботи з симулятором Gazebo та обробки великої кількості даних.

C. Симулятор Gazebo та інші інструменти візуалізації, наприклад Rviz, значною мірою використовують ресурси GPU для рендерингу. А програмно-апаратна архітектура паралельних обчислень CUDA забезпечує оптимальну обробку візуальних даних. Вимогами до *GPU графічної карти* є:

- *мінімум* графічна карта з підтримкою OpenGL 3.3 (інтегрована графіка, наприклад Intel HD):
- *рекомендовано* використання дискретної графіки Nvidia з підтримкою CUDA для роботи з паралельними обчисленнями.

D. Для повноцінної роботи ROS, вимагає встановлення значної кількості залежностей, бібліотек і моделей, а тому об'єм необхідного *дискового простору* становить:

- *мінімум* 20 ГБ вільного простору.
- розробником *рекомендовано* 50 ГБ для установки додаткових пакетів, таких як моделі роботів, бібліотеки та файли симуляцій.

E. Вимоги до *інформаційної мережі та мережевого обладнання* не є критичними, але *рекомендовано*:

- наявність *стабільного інтернет з'єднання,* для завантаження пакетів оновлень з репозиторію apt-get і можливості підключення до онлайн-документації та отримання доступу до спільноти ROS;
- для здійснення роботи з розподіленими системами може знадобитися локальна мережа LAN із низькою PING затримкою.

1.2.3. Додаткові рекомендації до апаратної складової ПЕОМ

Додатковими рекомендаціями до апаратної складової ПЕОМ для роботи з симулятором Gazebo становлять: