Форма №24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Інформаційна безпека та електронні комунікації

(найменування факультету)

Кафедра інформаційних технологій електронних засобів

(найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістра

(ступінь вищої освіти)

на тему: ДОСЛІДЖЕННЯ МЕРЕЖЕВОЇ ВЗАЄМОДІЇ ТА РОЗРОБКА ПРОГРАМИ З'ЄДНАННЯ МОБІЛЬНОГО РОБОТУ З ВІДДАЛЕНОЮ СИСТЕМОЮ ROS2 З ВИКОРИСТАННЯМ WEBSOCKET ТА ІНТЕРНЕТ-БРАУЗЕРА

Виконав студент 2 курсу БК-512м групи

Спеціальності 172 «Телекомунікації та радіотехніка»

Освітня програма (спеціалізація) «Інтелектуальні технології мікросистемної радіоелектронної техніки»

МНИХ К.Р.

(ПРІЗВИЩЕ та ініціали)

Керівник ФАРАФОНОВ О. Ю.

(ПРІЗВИЩЕ та ініціали)

Рецензент КОРОЛЬКОВ Р. Ю.

(ПРІЗВИЩЕ та ініціали)

2024

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет ФІБЕК
Кафедра інформаційні технології електронних засобів
Ступінь вищої освіти магістр
Спеціальність 172 «Телекомунікація та радіотехніка»
(код і найменування)
Освітня програма (спеціалізація) інтелектуальні технології мікросистемної радіоелектронної техніки
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТЕЗ, к.т.н, доцент
Олександр МАЛИЙ
«_____» _____2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Мниха Костянтина Романовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження мережевої взаємодії та розробка програми з'єднання мобільного робота з віддаленою системою ROS2 з використанням websocket та інтернет-браузера
керівник проєкту (роботи) к.т.н., доцент ФАРАФОНОВ Олексій Юрійович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від «21» листопада 2023 року № 448
2. Строк подання студентом проєкту (роботи) 23 грудня 2023 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Теоретичні відомості та постановка завдання. 2. Розробка програми з'єднання мобільного роботу з віддаленою системою ros2 з використанням websocket та інтернет-браузера.. 3. Опис програми для з'єднання мобільного робота із віддаленою системою ros2.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1–3 Основна частина	ФАРАФОНОВ О.Ю., доцент		
Нормоконтроль	ПОСПЕЄВА І.Є., старший викладач		

7. Дата видачі завдання «4» вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області	1 тиждень	Завдання, ТЗ
2	Постановка завдання роботи	2 тиждень	Розділ 1
3	Розробка програми	7-8 тиждень	Розділ 2
	Опис програми для з'єднання мобільного робота із віддаленою системою ros2	9-11 тижні	Розділ 3
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	12-13 тиждень	Додатки
7	Захист роботи	14 тиждень	
8			

Студент(ка) _____ Костянтин МНИХ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Олексій ФАРАФОНОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 78с., 12 рис., 2 дод., 22 джерела.

ДОСЛІДЖЕННЯ МЕРЕЖЕВОЇ ВЗАЄМОДІЇ ТА РОЗРОБКА ПРОГРАМИ З'ЄДНАННЯ МОБІЛЬНОГО РОБОТУ З ВІДДАЛЕНОЮ СИСТЕМОЮ ROS2 З ВИКОРИСТАННЯМ WEBSOCKET ТА ІНТЕРНЕТ-БРАУЗЕРА

Об’єкт дослідження – Взаємодія між мобільним роботом і віддаленою системою ROS2 через мережеві протоколи.

Предмет роботи – розробка методології та програмного забезпечення для з'єднання мобільного робота з системою ROS2, використовуючи WebSocket та інтернет-браузер.

Мета роботи – аналіз мережевих протоколів та інструментів для інтеграції мобільного робота з ROS2, розробка та імплементація програми для ефективної взаємодії через WebSocket.

Матеріали, методи та технічні засоби: використання сучасних мережевих технологій, програмування на мові Python, використання бібліотек для WebSocket і розробка користувацького інтерфейсу через інтернет-браузер.

Результатом роботи стане розроблена програма для з'єднання мобільного робота з ROS2, яка забезпечує надійну та ефективну мережеву взаємодію. Це підвищить гнучкість та ефективність використання мобільних роботів у різних умовах.

Наукова новизна роботи полягає у розробці спеціалізованої програми, яка дозволяє мобільному роботу зв'язуватися з ROS2 через WebSocket, що є новим напрямком у робототехніці. Паралельно з цим розроблений веб-інтерфейс для віддаленого управління роботом, що надає можливість користувачам обирати,

чи бажають вони самостійно управляти роботом, чи передавати управління ROS2. Це спрощує розуміння процесів управління роботом та надає користувачам інтуїтивну можливість взаємодії з ним.

Галузь використання: мережева взаємодія та керування мобільними роботами в різних сферах, включаючи дослідницькі та промислові застосування.

ЗМІСТ

Вступ..... 9

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ТА ПОСТАНОВКА ЗАВДАННЯ ……11

1.1 NodeMcu..... 11

1.1.1 Моделі nodeMcu..... 12

1.1.2 Застосування nodeMcu 15

1.2 Веб-сторінки 17

1.2.1 Історія виникнення веб-сторінки..... 18

1.2.2 Типи веб-сторінок..... 20

1.2.3 HTTP протокол 22

1.2.4 Приклад роботи веб-сторінки 23

1.3 Системи ROS 26

1.3.1 Історія виникнення системи ROS 28

1.3.2 Архітектура системи ROS..... 32

1.3.3 Інструменти системи ROS..... 34

1.4 Постановка завдань магістерської роботи 35

2 РОЗРОБКА ПРОГРАМИ З'ЄДНАННЯ МОБІЛЬНОГО РОБОТУ З ВІДДАЛЕНОЮ СИСТЕМОЮ ROS2 З ВИКОРИСТАННЯМ WEBSOCKET ТА ІНТЕРНЕТ-БРАУЗЕРА..... 37

2.1 Опис взаємодії у системі ROS2: publisher and subscription 37

2.1.1 Створення Publisher в ROS2..... 37

2.1.2 Створення Subscriber в ROS2..... 42

2.2 Опис стандартної взаємодії клієнт – сервер 45

2.2.1 Опис моделі "клієнт – сервер" 45

2.2.2 Мережа "клієнт-сервер" 47

2.2.2 Різниця між клієнтом та сервером 48

2.2.2 Опис взаємодії за допомогою WebSocket 50

2.2.2 Історія виникнення WebSocket 51

3 ОПИС ПРОГРАМИ ДЛЯ З'ЄДНАННЯ МОБІЛЬНОГО РОБОТА ІЗ ВІДДАЛЕНОЮ СИСТЕМОЮ ROS2 53

3.1 Опис коду для nodeMcu 53

3.1.1 Підключення бібліотек для nodeMcu 53

3.1.2 Налаштування мережі WIFI 54

3.1.3 Ініціалізація серверів 55

3.1.4 Конфігурація апаратного забезпечення 56

3.1.5 Веб-інтерфейс для управління 57

3.1.6 Обробка запитів на сервері 58

3.1.7 Налаштування та запуск сервера 60

3.1.8 Головний цикл виконання 61

3.2 Опис коду для вузла ROS2 62

3.2.1 Ініціалізація ROS2 Node 62

3.2.2 Взаємодія через WebSocket 62

3.2.2 Функція send_websocket_request 63

3.2.2 Головна функція та обробка вводу 64

3.3 Опис механізму взаємодії nodeMcu та вузла ROS2 65

3.3.1 Тестування коду на nodeMcu 65

3.3.2 Тестування коду вузла ROS2 67

Висновки 71

Перелік джерел посилання..... 72

Додаток А..... 74

Додаток Б..... 77

ВСТУП

У епоху стрімкого технологічного прогресу, особливу увагу привертають три взаємопов'язані сфери: електроніка, робототехніка та системи дистанційного управління. Їх розвиток не лише формує основу сучасної технічної цивілізації, а й задає тон для майбутніх інновацій, які будуть визначати наше повсякденне життя, економіку та наукові дослідження.

Електроніка, як фундамент усіх сучасних технологій, проходить через період небувалих інновацій. Від мікросхем і напівпровідників до складних інтегральних схем, які лежать в основі усіх сучасних електронних пристроїв – телефонів, комп'ютерів, автомобілів та багатьох інших. Розвиток мікроелектроніки, зокрема, призвів до революційних змін у виробничих технологіях, комунікаціях та обчислювальній техніці. Нові методи проектування та виробництва дозволяють створювати все більш компактні, ефективні та потужні електронні компоненти. Це, у свою чергу, відкриває шлях для більш складних та розумних пристроїв та систем.

Робототехніка, яка тісно пов'язана з розвитком електроніки, також переживає свою золоту епоху. Роботи вже давно вийшли за межі виробничих ліній та знайшли своє застосування у найрізноманітніших сферах. Від промислових роботів, здатних виконувати монотонну та небезпечну роботу на заводах, до медичних роботів, які допомагають у проведенні складних операцій, та навіть домашніх роботів, які полегшують повсякденні завдання. Особливий інтерес представляють розвиваючі сфери, такі як м'яка робототехніка, автономні транспортні засоби та дрон-технології. Ці інновації не лише змінюють спосіб, яким ми виконуємо роботу, але й корінним чином змінюють наше сприйняття можливостей робототехніки.

Системи дистанційного управління, у свою чергу, є невід'ємною частиною сучасної робототехніки та електроніки. Вони дозволяють людині управляти складними машинами та системами на відстані, що відкриває нові горизонти у

сферах, де особиста присутність людини або неможлива, або небезпечна. Системи дистанційного управління знаходять своє застосування в таких сферах, як аерокосмічна промисловість, глибоководні дослідження, військові технології та багато інших. Завдяки розвитку бездротових технологій та удосконаленням у галузі штучного інтелекту та машинного навчання, системи дистанційного управління стають все більш ефективними, надійними та зручними у використанні.

Інтеграція цих трьох напрямків – електроніки, робототехніки та систем дистанційного управління – відкриває нові можливості для створення складних, інтелектуальних та автономних систем. Від автоматизації виробничих процесів та підвищення безпеки праці до розвитку розумних міст та досліджень далекого космосу, ці технології мають величезний вплив на багато аспектів нашого життя. Вони допомагають вирішувати складні проблеми, підвищувати якість життя та відкривають двері у світ, де межі між наукою та фантастикою стають все більш розмитими.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 NodeMcu

NodeMcu - це платформа, заснована на ESP8266, призначена для створення різноманітних IoT-пристроїв. Цей модуль може відправляти та отримувати дані в локальну мережу або Інтернет через Wi-Fi. Через свою доступність, він часто використовується для розробки систем "розумний дім" та віддалено керованих роботів на які використовують модулі Arduino. Ми зосередимо увагу на характеристиках плати, відмінностях між версіями та розподілі контактів останньої версії Esp8266 NodeMcu. Також коротко оглянемо мову програмування Lua, яка використовується для написання програм для NodeMcu.

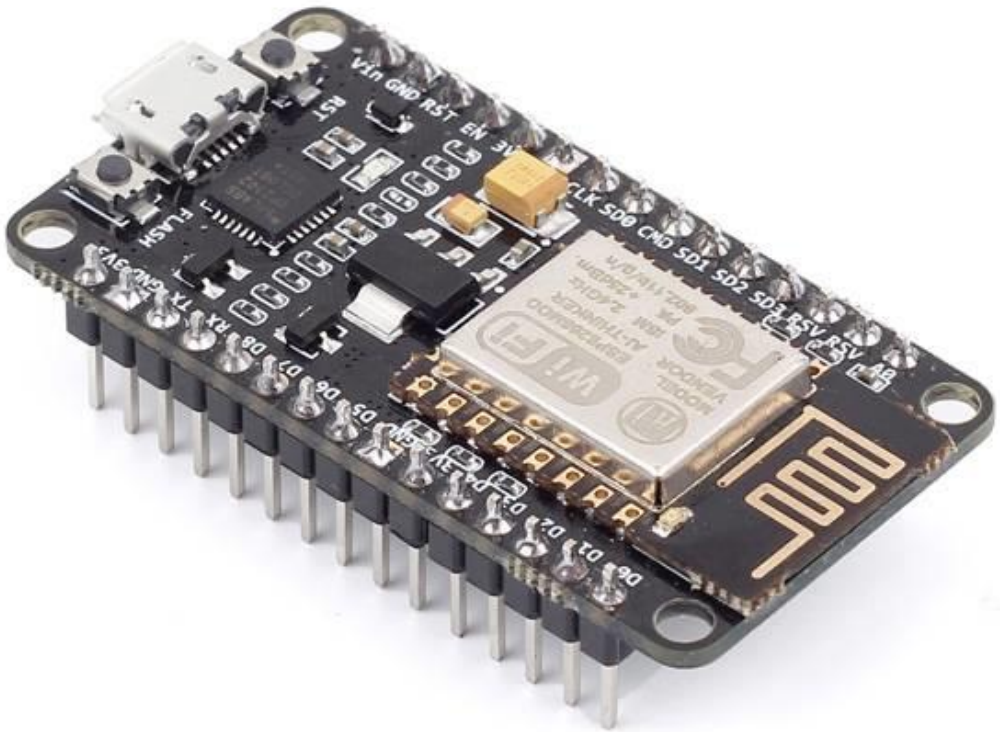

Рисунок 1 – Плата NodeMcu

Технічні характеристики модуля:

– підтримує Wi-Fi протокол 802.11 b/g/n;

– підтримувані режими Wi-Fi – точка доступу, клієнт;

– вхідна напруга 3,7 В – 20 В;

– робоча напруга 3 В-3,6 В;

– максимальний струм 220 мА;

– вбудований стек TCP/IP;

– діапазон робочих температур від -40С до 125С;

– 80 МГц, 32-бітний процесор;

– час пробудження та відправлення пакетів 22 мс;

– вбудовані TR-перемикач і PLL;

– наявність підсилювачів потужності, регуляторів, систем управління живленням.

1.1.1 Моделі nodeMcu

Існують декілька поколінь плат nodeMcu: V1 (версія 0.9), V2 (версія 1.0) та V3 (версія 1.0). На ринку, зокрема в інтернет-магазинах, ці плати часто позначаються як V1, V2, V3. Іноді трапляється змішування моделей, наприклад, V3 може виглядати зовні схожою на V2. Важливо відзначити, що всі плати NodeMcu працюють на засадах відкритого вихідного коду, тому їх можуть виробляти різні компанії. Станом на сьогодні, ключовими виробниками плат NodeMcu є Amica, DOIT та LoLin/Wemos.

Моделі плат NodeMcu V1 та V2 легко розрізнити за розміром. Друге покоління включає в себе покращену версію чіпа ESP-12 та 4 Мб флеш-пам'яті. Перше покоління, яке застаріло, має яскраво-жовтий колір і не зручно у використанні, адже займає 10 виходів на макетній платі. У вдосконаленій моделі другого покоління цей недолік усунуто – плата стала вужчою і її виходи краще підходять до контактів. Плати V3 зовні схожі на V2, але вони мають більш надійний USB-вихід. Модель V3, яку виробляє компанія LoLin, відрізняється

використанням одного з двох резервних виходів для додаткової землі, а другого - для USB живлення. Також ця модель має більші розміри, ніж попередні версії.

Переваги NodeMcu v3:

– наявність UART-USB інтерфейсу з роз'ємом micro USB дозволяє легко підключати плату до комп'ютера;

– обладнана 4 Мбайтами флеш-пам'яті;

– можливість оновлення прошивки через USB.

– здатність створювати скрипти на LUA та зберігати їх у файловій системі.

Недоліки модуля NodeMcu:

Основним недоліком є обмеженість у виконанні лише LUA скриптів, які зберігаються в оперативній пам'яті. Оскільки об'єм оперативної пам'яті складає всього 20 Кбайт, розробка великих скриптів може бути складною. Перш за все, алгоритм необхідно розділити на лінійні блоки. Ці блоки потрібно записати в окремі файли системи. Усі ці модулі виконуються за допомогою оператора dofile.

Під час написання скриптів важливо дотримуватися правила: при обміні даними між модулями використовувати глобальні змінні, а при обчисленнях всередині модулів - локальні. Також необхідно в кінці кожного написаного скрипта викликати функцію collectgarbage (збірник сміття).

Модуль V3 оснащений 11 універсальними входами-виходами. Окрім цього, деякі з цих виводів мають спеціальні функції:

– D1-D10 – виводи з широтно-імпульсною модуляцією;

– D1, D2 – виводи для інтерфейсу I2C/TWI;

– 3D5–D8 – виводи для інтерфейсу SPI;

– D9, D10 – UART;

– A0 – вхід з АЦ

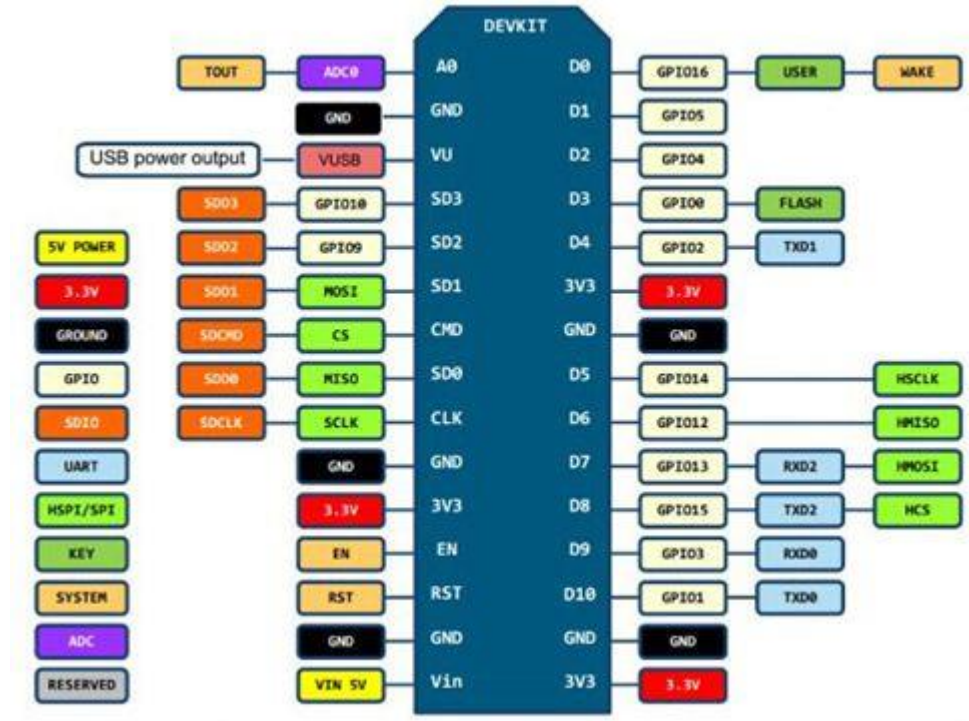

Рисунок 2 – Конфігурація виводів NodeMcu v3

Стандартна прошивка Node MCU, яка завантажена на платформу, включає в себе інтерпретатор мови Lua. Завдяки командам Lua можливо виконувати такі дії:

– з'єднання з Wi-Fi мережею;

– робота в режимі Wi-Fi точки доступу;

– перехід у режим глибокого сну для зменшення споживання енергії;

– вмикання або вимикання світлодіода на виході GPIO16;

– операції з файлами у флеш-пам'яті;

– пошук відкритих Wi-Fi мереж та підключення до них;

– виведення MAC-адреси;

Для програмування NodeMCU можна використовувати Arduino IDE, або комплект розробки SDK – ESPlorer, який має ряд переваг:

– сумісність з різними платформами;

– підтримка відкриття декількох файлів одночасно;

– підсвічування коду Lua;

– можливість зручної відправки файлів;

– підтримка декількох типів прошивок одночасно.

Для забезпечення правильної та стабільної роботи важливо оновити прошивку до останньої версії. Існує кілька способів оновлення: через хмарний сервіс, Docker Image або компіляцію в Linux. Кожен із цих методів має свої переваги та недоліки. Найбільш простим і зрозумілим є перший спосіб.

1.1.2 Застосування nodeMcu

ESP8266 може працювати як точка доступу, так і в режимі клієнта. У нормальному режимі роботи в локальній мережі ESP8266 налаштовується як кінцева станція. Для цього пристрою потрібно задати SSID Wi-Fi мережі та пароль доступу в закритих мережах. Для початкового налаштування цих параметрів буде зручним режим точки доступу. У цьому режимі пристрій видно при стандартному пошуку мереж на планшетах та комп'ютерах. Потрібно підключитися до пристрою, відкрити HTML-сторінку налаштування та задати мережеві параметри, після чого пристрій стандартно підключиться до локальної мережі як кінцева станція.

Для виключно локального використання можна завжди залишати пристрій у режимі точки доступу.

Після підключення до Wi-Fi мережі, пристрій отримує IP-параметр локальної мережі. Ці параметри можна налаштувати вручну разом з параметрами Wi-Fi або використовувати сервіси автоматичного налаштування IP-параметрів, такі як DHCP.

Доступ до серверу пристрою в локальній мережі зазвичай відбувається за його IP-адресою, мережевим ім'ям або через сервіс, якщо підтримується автоматичний пошук сервісів.

Часто необхідний доступ до пристрою з Інтернету. Наприклад, користувач зі свого мобільного телефону перевіряє стан свого "розумного будинку". У цьому випадку пристрій працює як сервер, до якого звертається зовнішній клієнт.

Пристрій на базі ESP8266 зазвичай розташований у локальній мережі офісу або будинку. Маршрутизатор, підключений до локальної мережі та мережі інтернет-провайдера, забезпечує вихід в Інтернет. Провайдер присвоює маршрутизатору статичну або динамічну IP-адресу, а маршрутизатор здійснює трансляцію адрес локальної мережі. За замовчуванням правила цієї трансляції забезпечують видимість інтернет-адрес з локальної мережі, але не дозволяють звернутися до локальних адрес з боку Інтернету. Існує кілька способів обійти це обмеження.

Більшість сучасних маршрутизаторів дозволяє встановлювати додаткові правила трансляції мережевих адрес між локальною та глобальною мережами через технології Virtual server або DMZ. Обидві технології дозволяють звернутися до сервера в локальній мережі з глобальної мережі, знаючи лише IP-адресу, присвоєний маршрутизатору провайдером. У випадку статичної IP-адреси маршрутизатора, це може бути задовільним рішенням для обмеженого кола користувачів системи. Однак такий підхід має недоліки: потрібно вручну налаштовувати маршрутизатор і знати IP-адресу маршрутизатора, який може регулярно змінюватися. Проблему невідомої IP-адреси можна вирішити за допомогою DDNS.

Для доступу до сервера пристрою, кінцевому користувачеві потрібно знати IP-адресу пристрою. Однак отримати статичну IP-адресу у провайдера Інтернета не завжди можливо. Для вирішення цієї проблеми були створені спеціальні інтернет-сервіси з динамічним DNS. Ці сервіси працюють як спеціальні сервери з фіксованими назвами в Інтернеті. Розробник створює на такому сервісі свій акаунт з унікальним ім'ям. Параметри цього акаунта вносяться в пристрій. Пристрій як клієнт періодично звертається до серверу сервісу, повідомляючи йому назву свого акаунта та свою поточну IP-адресу. Кінцевий користувач у Інтернеті звертається до цього ж сервісу і отримує від нього поточні IP-параметри пристрою. У такому випадку пристрій у мережі видно з доменним ім'ям третього рівня, наприклад esp8266.ddns.org.

Основна проблема DDNS-сервісів - це гарантії існування конкретного сервісу. Зазвичай гарантовано тільки комерційний сервіс, коли за його використання стягується плата.

Для спрощення доступності пристрою в Інтернеті і полегшення інсталяції пристрою для користувача було розроблено ряд рішень. Механізм цих рішень базується на наявності в Інтернеті спеціального сервера, до якого може підключитися як IoT-пристрій, так і планшет/комп'ютер користувача. Пристрій працює в режимі клієнта, не потрібно жодних спеціальних налаштувань маршрутизатора або особливих навичок від інсталятора чи користувача пристрою. Обмін даними з пристроєм відбувається через посередництво цього спеціального сервісу, параметри якого має закласти розробник. Розповсюдження використання таких сервісів ускладнюється необхідністю тривало підтримувати свій сервіс в Інтернеті або користуватися чужими сервісами з невідомими перспективами тривалого існування безкоштовних можливостей або регулярної оплати комерційних варіантів.

1.2 Веб-сторінки

Веб-сторінка - це документ, який зазвичай написаний на HTML (мові розмітки гіпертексту), який переглядається в інтернет-браузері. До веб-сторінки можна отримати доступ, ввівши URL-адресу (уніфікований локатор ресурсів) у адресний рядок браузера. Веб-сторінка може містити текст, графіку та гіперпосилання на інші веб-сторінки та файли.

Веб-сторінка надає інформацію глядачам, включаючи зображення чи відео, щоб допомогти ілюструвати важливі теми. Веб-сторінка також може використовуватися як спосіб продажу продуктів чи послуг глядачам.

Наприклад, ви переходите на веб-сторінку, коли клацаєте на посилання, надане пошуковою системою. Інтернет складається з мільйонів веб-сторінок, і щодня додається все більше.

Перегляд веб-сторінки вимагає браузера, як-от Internet Explorer, Edge, Safari, Firefox чи Chrome. Перебуваючи у браузері, ви можете відкрити веб-сторінку, ввівши URL у адресний рядок. Якщо ви не знаєте URL сайту, який хочете відвідати, ви можете використовувати пошукову систему, щоб його знайти.

1.2.1 Історія виникнення веб-сторінки

Історія Інтернету почалася зусиллями вчених та інженерів у створенні та взаємопідключенні комп'ютерних мереж. Набір правил Internet Protocol Suite, що використовуються для комунікації між мережами та пристроями в Інтернеті, виник з досліджень та розробок у США та включав міжнародне співробітництво, зокрема з дослідниками з Великобританії та Франції.

Інформатика, яка була новою дисципліною наприкінці 1950-х, почала розглядати можливість спільного використання часу між користувачами комп'ютерів, а згодом - можливість реалізації цього через широкомасштабні мережі. Дж. С. Р. Ліклайдер розробив ідею універсальної мережі в Офісі технік обробки інформації (IPTO) Департаменту оборони США Агентства перспективних дослідницьких проектів (ARPA). Незалежно від цього, Пол Баран у RAND Corporation запропонував розподілену мережу, засновану на даних у блоках повідомлень на початку 1960-х, а Дональд Девіс у 1965 році в Національній фізичній лабораторії (NPL) задумав комутацію пакетів, запропонувавши національну комерційну дата-мережу у Великобританії.

ARPA у 1969 році уклала контракти на розробку проекту ARPANET під керівництвом Роберта Тейлора та управлінням Лоуренса Робертса. ARPANET ухвалив технологію комутації пакетів, запропоновану Девісом та Бараном. Мережу процесорів повідомлень інтерфейсів побудувала команда в Bolt, Beranek і Newman, з дизайном та специфікацією, які очолював Боб Кан. Протокол хост-до-хоста було визначено групою аспірантів UCLA під керівництвом Стіва Крокера, разом з Джоном Постелем та Вінтом Серфом. ARPANET швидко розширився по всій території США з підключеннями до Великобританії та Норвегії.

У 1970-х роках з'явилося кілька ранніх пакетно-комутованих мереж, які досліджували та надавали дата-мережі. Луї Пуазен та Хуберт Циммерман піонерськи працювали над спрощеним підходом від кінця до кінця до міжмережевих з'єднань у IRIA. Пітер Кірстейн в Університетському коледжі Лондона в 1973 році втілив міжмережеві з'єднання у практику. Боб Меткалф розробив теорію за

Ethernet та PARC Universal Packet. Проекти ARPA, Міжнародна робоча група з мереж та комерційні ініціативи призвели до розвитку різних ідей для міжмережевих з'єднань, у яких кілька окремих мереж можуть бути об'єднані у мережу мереж. Вінт Серф, тепер у Стенфордському університеті, та Боб Кан, тепер в DARPA, опублікували дослідження у 1974 році, які перетворилися на протоколи Transmission Control Protocol (TCP) та Internet Protocol (IP) - два протоколи набору протоколів Інтернету. Дизайн включав концепції з французького проекту CYCLADES під керівництвом Луї Пуазена. Розвиток мереж комутації пакетів підкріплювався математичними роботами 1970-х років Леонарда Клейнрока з UCLA.

У кінці 1970-х з'явилися національні та міжнародні громадські дата-мережі на основі протоколу X.25, розробленого Ремі Депре та іншими. У США Національний науковий фонд (NSF) фінансував національні суперкомп'ютерні центри у кількох університетах США та забезпечив міжпідключення в 1986 році з проектом NSFNET, створивши доступ до цих суперкомп'ютерних сайтів для дослідницьких та академічних організацій у США. Міжнародні підключення до NSFNET, поява архітектури, такої як Система доменних імен, та прийняття TCP/IP на існуючих мережах у США та у всьому світі позначили початки Інтернету. Комерційні інтернет-провайдери (ISP) з'явилися у 1989 році у США та Австралії. Обмежені приватні підключення до частин Інтернету офіційно комерційними суб'єктами з'явилися у декількох американських містах наприкінці 1989 та 1990 років. Оптичний каркас NSFNET був демонтований у 1995 році, знявши останні обмеження на використання Інтернету для перенесення комерційного трафіку, оскільки трафік перейшов на оптичні мережі, керовані Sprint, MCI та AT&T у США.

Дослідження в CERN у Швейцарії британським комп'ютерним вченим Тімом Бернерс-Лі у 1989–90 роках призвели до створення Всесвітньої павутини, поєднуючи гіпертекстові документи в інформаційну систему, доступну з будь-якого вузла в мережі. Драматичне розширення потужності Інтернету, забезпечене появою в середині 1990-х років мультиплексування за допомогою поділу

хвиль (WDM) та розгортання волоконно-оптичних кабелів, мало революційний вплив на культуру, комерцію та технології. Це зробило можливим майже миттєве спілкування за допомогою електронної пошти, миттєвих повідомлень, голосових дзвінків через Інтернет (VoIP), відеочатів та Всесвітньої павутини з її дискусійними форумами, блогами, соціальними мережами та онлайн-магазинами. Зростаючі обсяги даних передаються з все більшою та більшою швидкістю через волоконно-оптичні мережі зі швидкістю 1 Гбіт/с, 10 Гбіт/с та 800 Гбіт/с до 2019 року. Перевага Інтернету в глобальному ландшафті зв'язку була швидкою в історичному масштабі: в 1993 році він передавав лише 1% інформації, що тече через двосторонні телекомунікаційні мережі, 51% до 2000 року та більше 97% телекомунікаційної інформації до 2007 року. Інтернет продовжує рости, керований все більшими обсягами онлайн-інформації, комерції, розваг та соціальних мережевих послуг. Однак майбутнє глобальної мережі може бути сформоване регіональними відмінностями.

1.2.2 Типи веб-сторінок

Статична веб-сторінка, або стаціонарна сторінка, - це вебсторінка, яка доставляється веб-браузеру точно так, як зберігається, на відміну від динамічних вебсторінок, які генеруються веб-додатком.

Отже, статична веб-сторінка відображає однакову інформацію для всіх користувачів, з будь-якого контексту,але за дпомогою JavaScript веб-сторінка може ввести динамічну функціональність, яка може зробити статичну веб-сторінку динамічною.

Статичні веб-сторінки часто є HTML-документами, збереженими як файли в файловій системі та наданими через веб-сервер за допомогою HTTP (однак URL-адреси, що закінчуються на ".html", не завжди є статичними). Проте, широке трактування терміна може включати веб-сторінки, збережені в базі даних, і навіть може включати сторінки, сформатовані за допомогою шаблону та надані

через сервер додатків, за умови, що надана сторінка є незмінною і представлена по суті так, як зберігається.

Зміст статичних веб-сторінок залишається незмінним незалежно від кількості переглядів. Такі веб-сторінки підходять для вмісту, який рідко потребує оновлення, хоча сучасні системи веб-шаблонів змінюють це. Підтримка великої кількості статичних сторінок як файлів може бути непрактичною без автоматизованих інструментів, таких як генератори статичних сайтів. Будь-яка персоналізація або інтерактивність має виконуватися на стороні клієнта, що є обмежуючим.

Переваги:

– забезпечують покращену безпеку порівняно з динамічними веб-сайтами (на динамічні веб-сайти здійснюють атаки);

– покращена продуктивність для кінцевих користувачів порівняно з динамічними веб-сайтами;

– менше або жодних залежностей від систем, таких як бази даних або інші сервери додатків;

– заощадження витрат за рахунок використання хмарного зберігання, а не розміщеного середовища;

– налаштування безпеки легко встановити, що робить їх більш безпечними.

Недоліки:

– динамічну функціональність потрібно виконувати на стороні клієнта.

Динамічна вебсторінка - це веб-ресурс, який включає контент, що регулярно оновлюється за допомогою серверних або клієнтських скриптів. Цей тип сторінки часто використовується на інформаційних платформах, таких як новинні портали або блоги, де контент додається або оновлюється часто. Динамічні сторінки також використовуються у електронній комерції для представлення змінного асортименту товарів.

Динамічне відтворення вмісту вебсторінки традиційно здійснювалося на сервері за допомогою таких мов програмування, як PHP, Java або .NET. Завдяки розвитку можливостей сучасних веб-браузерів, частина обробки контенту

переноситься на сторону клієнта. Це знижує навантаження на сервер і забезпечує швидшу взаємодію з користувачем.

Завдяки застосуванню сучасних JavaScript-фреймворків, таких як React, Angular, Vue, створення динамічних веб-додатків стало значно простішим. Веб-додатки споживають дані у форматі JSON, після чого фреймворк здійснює збірку HTML на стороні клієнта для відображення контенту.

З точки зору управління досвідом користувача, динамічні вебсторінки дозволяють адаптувати контент до потреб кожного відвідувача. Це суттєво підвищує рівень персоналізації та інтерактивності в порівнянні зі статичними вебсторінками.

Розробка динамічних вебсторінок вимагає комплексного підходу та різнопланових навичок, включаючи глибокі знання JavaScript і фреймворків. Крім того, необхідність створення великої кількості контенту та його ефективного управління може стати суттєвим викликом.

Спостерігається поступовий перехід від статичних вебсторінок до повнофункціональних веб-додатків. Авторам контенту все частіше доводиться працювати з складними динамічними контекстами, що вимагає від них адаптації до нових умов управління контентом.

Динамічні вебсторінки є ключовим елементом сучасного веб-дизайну, забезпечуючи високий рівень персоналізації та інтерактивності. Однак, успішна інтеграція цих технологій вимагає комплексного підходу, який враховує як технічні аспекти розробки, так і потреби управління контентом.

1.2.3 HTTP протокол

HTTP (протокол передачі гіпертексту) є основним протоколом для отримання ресурсів, таких як документи HTML. Це основа будь-якого обміну даними в Інтернеті, і він є протоколом клієнт-сервер, що означає, що запити ініціюються отримувачем, зазвичай веб-браузером. Повний документ відтворюється з різних

піддокументів, які отримуються, наприклад, текст, опис макету, зображення, відео, сценарії тощо.

Коли користувач вводить URL-адресу у веб-браузері та натискає Enter, відбувається наступне:

– створення запиту: браузер формує HTTP-запит та відправляє його на сервер, де знаходиться цільова веб-сторінка;

– передача запиту: запит передається через мережу до веб-сервера, який обробляє запит і відправляє відповідь;

– отримання відповіді: веб-браузер отримує відповідь від сервера, яка зазвичай містить статус (наприклад, успішно, помилка) та запитуваний контент;

– рендеринг сторінки: браузер інтерпретує отримані дані, часто HTML-документ, і відображає їх у вигляді веб-сторінки. Якщо сторінка містить посилання на інші ресурси (наприклад, зображення, CSS, JavaScript), браузер додатково запитує ці ресурси, використовуючи додаткові HTTP-запити.

1.2.4 Приклад роботи веб-сторінки

Саме таким прикладом є HTML сторінка на якій реалізовано дистанційне керування роботом.

Алгорит роботи коду:

– створення WiFi-мережі для робота: на початку коду встановлюється назва та пароль для WiFi-мережі. Це дозволяє роботу створити власну бездротову мережу, до якої можна підключитися для управління ним;

– веб-сторінка для управління роботом: код містить HTML-структуру, яка формує веб-сторінку. На цій сторінці розміщені кнопки та інші елементи управління, через які можна керувати роботом. Веб-сторінка відображається в браузері, коли користувач підключається до WiFi-мережі робота;

– елементи керування на веб-сторінці: на веб-сторінці є кнопки для різних команд, таких як рух вперед, вліво, вправо, назад, включення та вимкнення

світла. Ці кнопки використовують JavaScript для відправки команд роботу через веб-сокети;

– використання веб-сокетів для передачі команд: коли користувач натискає на кнопку на веб-сторінці, через веб-сокет відправляється команда роботу. Веб-сокети забезпечують двосторонній зв'язок між веб-сторінкою та роботом;

– обробка команд роботом: коли робот отримує команду через веб-сокет, він виконує відповідні дії, наприклад, рухається вперед або повертає вліво;

– вимірювання відстані сенсором VL53L0X: Робот оснащений сенсором відстані, який дозволяє вимірювати відстань до перешкод. Ця інформація може використовуватися для автоматичного уникнення перешкод або для надання інформації користувачу.

Загалом, цей код створює систему, за допомогою якої можна дистанційно через веб-інтерфейс керувати роботом, надаючи користувачу гнучке та інтуїтивне управління різними функціями робота.

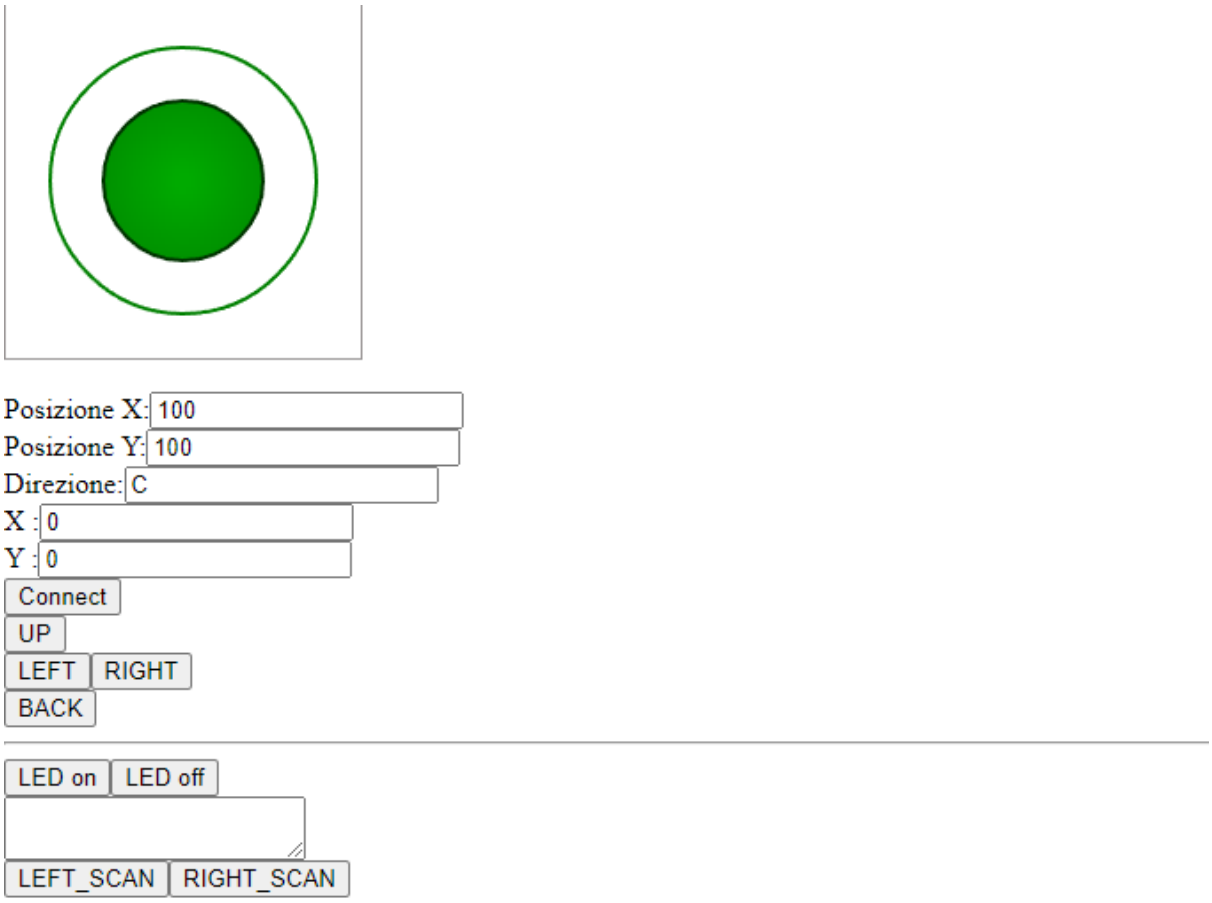

Рисунок 3 – HTML сторінка із кнопками керування та джойстиком

HTTP — це протокол взаємодії клієнта і сервера, де запити ініціюються стороною, що бере участь у комунікації (клієнт), або проксі-сервером замість нього. Зазвичай цю роль виконує веб-браузер, але це може бути будь-який клієнт, наприклад, робот, який обходить мережу для оновлення і поповнення індексації веб-сторінок у пошукових системах.

Кожен запит відправляється на сервер, який обробляє його та надає відповідь. Між запитами та відповідями зазвичай працюють численні проміжні агенти, відомі як проксі, які виконують різні операції та діють як шлюзи або кеші.

Клієнт-серверна взаємодія

Між браузером і сервером зазвичай працюють різні проміжні пристрої, які відіграють певну роль у обробці запитів: роутери, модеми тощо. Завдяки багаторівневій структурі мережі, ці проміжні клієнти "приховані" на мережевому та транспортному рівнях. В цій системі рівнів HTTP знаходиться на найвищому, так званому "прикладному", рівні. Знання про різні рівні мережі є важливими для розуміння її роботи та діагностики можливих проблем.

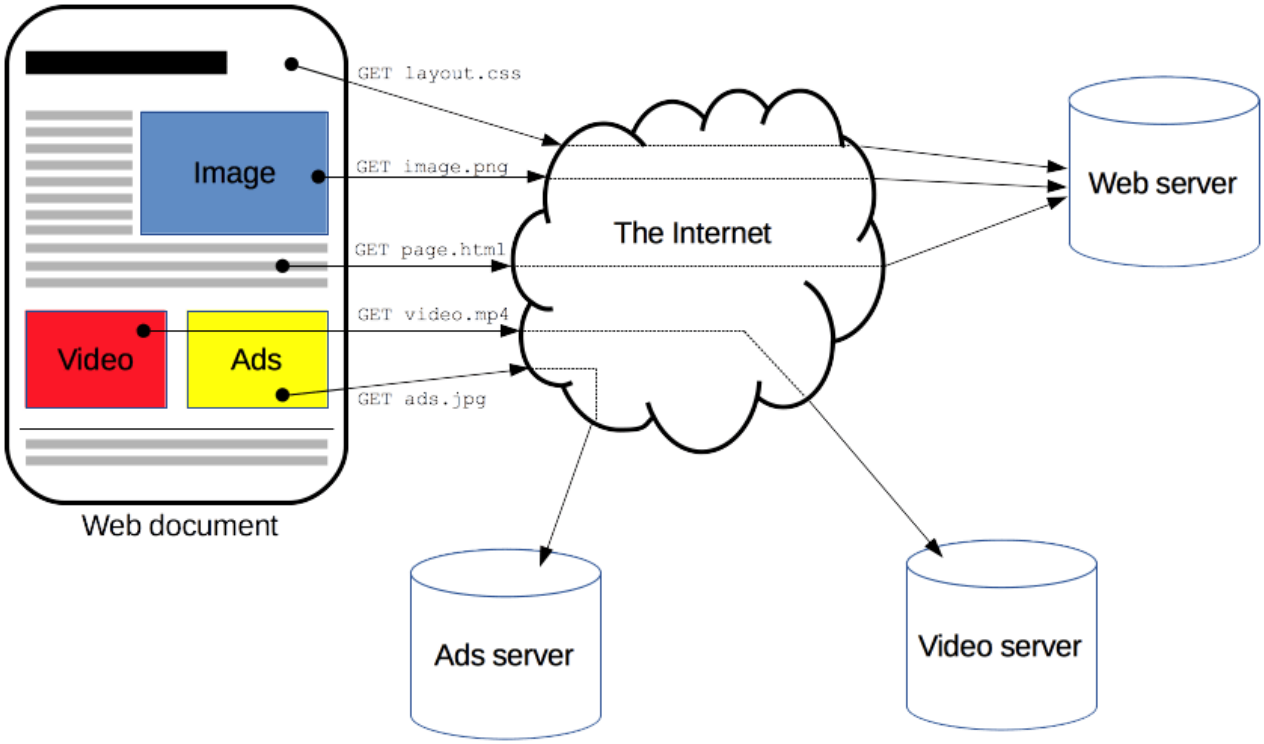

Рисунок 4 – Взаємодія клієнта і сервера

Між веб-браузером та сервером знаходиться велика кількість мережевих вузлів, що передають HTTP-повідомлення. Ці операції на рівні застосунків називаються проксі та можуть виконувати різні функції: кешування, фільтрація, балансування навантаження, аутентифікація, протоколювання.

HTTP є простим та розширюваним, не зберігає стан, але може використовувати сесії з кукіс для збереження стану. Управління з'єднанням відбувається на транспортному рівні, і хоча HTTP не вимагає з'єднання, він зазвичай використовує TCP.

HTTP/1.0 відкривало TCP-з'єднання для кожного обміну даними, але HTTP/1.1 ввів стабільні з'єднання, а HTTP/2 додав мультиплексування повідомлень через одне з'єднання.

HTTP дозволяє керувати кешем, ослабленням обмежень джерела, аутентифікацією, проксі та тунелюванням, сесіями, що підвищує контроль і функціональність мережі.

HTTP - це легкий у використанні та розширюваний протокол. Структура клієнт-сервер разом із здатністю легко додавати заголовки дозволяє HTTP розвиватися разом із розширюваними можливостями мережі.

Хоча HTTP/2 додає деяку складність, інтегруючи HTTP-повідомлення в рамки для покращення продуктивності, базова структура повідомлень залишається такою ж, як у HTTP/1.0. Сесійний потік залишається простим, що дозволяє здійснювати дослідження та відлагодження з простотою.

1.3 Системи ROS

Система ROS, або ROS, представляє собою набір програмного забезпечення середнього рівня з відкритим вихідним кодом для робототехніки. Хоча ROS не є операційною системою (ОС), а скоріше набором програмних фреймворків для розробки програмного забезпечення для роботів, вона надає послуги, призначені для гетерогенного комп'ютерного кластера, такі як абстракція апаратного

забезпечення, контроль низького рівня пристроїв, реалізація загальновживаних функцій, передача повідомлень між процесами та управління пакетами. Процеси, засновані на ROS, представлені у вигляді графічної архітектури, де обробка відбувається у вузлах, які можуть отримувати, публікувати та мультиплексувати дані сенсорів, керування, стан, планування, актуатори та інші повідомлення. Незважаючи на важливість реактивності та низької затримки у керуванні роботами, ROS не є системою реального часу (СРЧ). Однак можливо інтегрувати ROS з кодом реального часу. Брак підтримки систем реального часу був вирішений у створенні ROS 2, значному оновленні API ROS, яке використовуватиме сучасні бібліотеки та технології для основних функцій ROS та додасть підтримку коду реального часу та апаратного забезпечення вбудованих систем.

Програмне забезпечення в екосистемі ROS можна поділити на три групи:

– інструменти, незалежні від мови та платформи, які використовуються для створення та розповсюдження програмного забезпечення на базі ROS;

– реалізації клієнтських бібліотек ROS, такі як roscpp, rospy та roslisp;

– пакети, що містять код, пов'язаний з додатками, які використовують одну або кілька клієнтських бібліотек ROS.

Як інструменти, незалежні від мови, так і основні клієнтські бібліотеки (C++, Python та Lisp) розповсюджуються на умовах ліцензії BSD, тому є відкритим програмним забезпеченням і безкоштовними для комерційного та наукового використання. Більшість інших пакетів ліцензовані за різними ліцензіями відкритого коду. Ці інші пакети реалізують часто використовувані функції та додатки, такі як драйвери апаратного забезпечення, моделі роботів, типи даних, планування, сприйняття, одночасну локалізацію та картографування (SLAM), інструменти симуляції та інші алгоритми.

Основні клієнтські бібліотеки ROS орієнтовані на системи, подібні до Unix, переважно через їхню залежність від великих наборів залежностей відкритого програмного забезпечення. Для цих клієнтських бібліотек Ubuntu Linux

вказана як "Підтримується", тоді як інші варіанти, такі як Fedora Linux, macOS та Microsoft Windows, позначені як "експериментальні" і підтримуються спільнотою. Однак рідна Java клієнтська бібліотека ROS, rosjava, не має цих обмежень і дозволила створювати програмне забезпечення на базі ROS для Android OS. rosjava також дозволила інтегрувати ROS в офіційно підтримуваний набір інструментів MATLAB, який можна використовувати на Linux, macOS та Microsoft Windows. Була також розроблена клієнтська бібліотека JavaScript, roslibjs, яка дозволяє інтегрувати програмне забезпечення в систему ROS через будь-який веб-браузер, що відповідає стандартам.

1.3.1 Історія виникнення системи ROS

Перші елементи того, що згодом перетворилося на ROS, з'явилися у Стенфордському університеті ще до 2007 року. Ерік Бергер та Кінен Вайробек, аспіранти, які працювали в лабораторії робототехніки Кеннета Солсбері, очолювали Програму персональної робототехніки. Працюючи над роботами для виконання завдань у людському середовищі, вони помітили, що багато їхніх колег зіткнулися з проблемою різноманітності в робототехніці: відмінний розробник програмного забезпечення міг не мати необхідних знань з апаратного забезпечення, а хтось, хто розробляв передові методи планування маршрутів, міг не знати, як працювати з комп'ютерним зором. Щоб вирішити цю проблему, вони вирішили створити базову систему, яка б стала відправною точкою для інших у науковому світі.

На перших етапах створення цієї єдиної системи вони побудували апаратний прототип PR1 і почали працювати над програмним забезпеченням для нього, запозичуючи кращі практики з інших ранніх відкритих програмних платформ для роботів, особливо з системи Switchyard, над якою працював інший аспірант Стенфорда, Морган Квіглі, у підтримку проекту STAIR (Стенфордського штучного інтелекту і робота) від Лабораторії штучного інтелекту Стенфорда. На

ранній стадії розвитку PR1 було виділено фінансування у розмірі 50 000 доларів США від Джоанни Хоффман та Алена Россманна. У пошуках фінансування для подальшої розробки Ерік Бергер та Кінен Вайробек зустріли Скотта Хассана, засновника технологічного інкубатора Willow Garage. Хассан поділився їхнім баченням "Лінукса для робототехніки" і запросив їх працювати в Willow Garage. Willow Garage було засновано у січні 2007 року, а перший код ROS було опубліковано на SourceForge 7 листопада 2007 року.

Willow Garage почав розробку робота PR2 як продовження PR1, а також ROS як програмне забезпечення для його управління. Групи з більш ніж двадцяти установ зробили свій вклад у ROS, як у основне програмне забезпечення, так і у зростаючу кількість пакетів, що працювали з ROS, формуючи більшу програмну екосистему. Той факт, що люди поза Willow Garage вносили свій вклад у ROS (особливо з проекту STAIR Стенфорда), означав, що ROS була багатороботною платформою з самого початку. Хоча в Willow Garage спочатку були інші проекти, вони були відкинуті на користь Програми персональної робототехніки: зосередженість на виробництві PR2 як дослідницької платформи для наукових кіл і ROS як відкритого програмного стеку для робототехніки, який би ліг в основу як наукових досліджень, так і технологічних стартапів, подібно до того, як LAMP стек зробив для веб-стартапів.

У грудні 2008 року Willow Garage досяг першої з трьох внутрішніх віх: безперервне навігації для PR2 протягом двох днів на відстані в пі кілометрів. Незабаром після цього була випущена рання версія ROS (0.4 Mango Tango), за якою послідувала перша документація RVIZ та перша стаття про ROS. На початку літа була досягнута друга внутрішня віха: PR2 відкривав двері та сам підключався до зарядного пристрою. Ранні навчальні матеріали з ROS були опубліковані у грудні, готуючись до випуску ROS 1.0 у січні 2010 року. Це була третя віха: створення великої кількості документації та навчальних матеріалів для величезних можливостей, які інженери Willow Garage розробили за попередні 3 роки.

Після цього Willow Garage досяг однієї зі своїх найбільш тривалих цілей: безкоштовної передачі 10 роботів PR2 академічним установам. Це давно було метою засновників, оскільки вони вважали, що PR2 може стимулювати дослідження в області робототехніки по всьому світу. Вони врешті-решт вручили одинадцять PR2 різним установам, включаючи Університет Фрайбурга (Німеччина), Robert Bosch GmbH, Технологічний інститут Джорджії, KU Leuven (Бельгія), Массачусетський технологічний інститут (MIT), Стенфордський університет, Технічний університет Мюнхена (Німеччина), Каліфорнійський університет, Берклі, Університет Пенсільванії, Університет Південної Каліфорнії (USC) та Університет Токіо (Японія). Це, разом з високоефективною програмою стажування Willow Garage (проведеною з 2008 по 2010 рік Мелоні Вайз), допомогло поширити інформацію про ROS у світі робототехніки. Перший офіційний випуск дистрибуції ROS: ROS Box Turtle, був випущений 2 березня 2010 року, позначивши перший раз, коли ROS офіційно поширювався з набором версіонованих пакетів для загального використання. Ці розвитки призвели до появи першого дрона на ROS, першого автономного автомобіля на ROS, та адаптації ROS для Lego Mindstorms. З програмою PR2 Beta, що була вже на повному ходу, робот PR2 був офіційно випущений для комерційного продажу 9 вересня 2010 року.

2011 рік став значущим для ROS з запуском ROS Answers, форуму питань і відповідей для користувачів ROS, 15 лютого; введенням вельми успішного набору робота TurtleBot 18 квітня; і загальною кількістю репозиторіїв ROS, що перевищила 100 5 травня. Willow Garage розпочав 2012 рік, створивши Фонд відкритої робототехніки (OSRF) у квітні. OSRF одразу ж отримав програмний контракт від Агентства перспективних дослідницьких проектів в оборонній сфері (DARPA). Пізніше того року відбулася перша ROSCon у Сент-Полі, Міннесота; була опублікована перша книга про ROS, "ROS на прикладах"; і Baxter, перший комерційний робот, що працює на ROS, був оголошений Rethink Robotics. Незабаром після відзначення п'ятої річниці у листопаді, ROS почав працювати на кожному континенті 3 грудня 2012 року.

У лютому 2013 року Організація з відкритого джерельного робототехнічного програмування (OSRF) перейняла відповідальність за основну розробку Робототехнічної операційної системи (ROS), що збіглося з оголошенням у серпні про поглинання Willow Garage її засновниками, компанією Suitable Technologies. На той час ROS вже мала сім важливих версій, включаючи ROS Groovy, та знайшла користувачів по всьому світу. Ця стадія розвитку ROS завершилася, коли Clearpath Robotics взяла на себе обов'язки підтримки PR2 на початку 2014 року.

З тих пір, як OSRF стала головним розробником ROS, щороку виходила нова версія, а інтерес до ROS продовжував рости. Щорічно з 2012 року проводяться конференції ROSCon, які проходять паралельно з двома провідними робототехнічними конференціями, ICRA або IROS. Зустрічі розробників ROS організовуються в різних країнах, видаються книги про ROS, і запускаються численні освітні програми. 1 вересня 2014 року NASA оголосила про першого робота в космосі, який працює на ROS: Robotnaut 2 на Міжнародній космічній станції. У 2017 році OSRF змінила свою назву на Відкрите робототехнічне програмування. Технологічні гіганти, такі як Amazon і Microsoft, почали цікавитися ROS, зокрема Microsoft адаптував ROS для Windows у вересні 2018 року, а Amazon Web Services випустили RoboMaker у листопаді 2018 року.

Можливо, найважливішою подією періоду OSRF/Open Robotics стало запропонування ROS 2, значного оновлення API ROS, призначеного для підтримки реального часу, ширшого спектра обчислювальних середовищ і сучасних технологій. ROS 2 було анонсовано на ROSCon 2014, перші коміти в репозиторій ros2 були зроблені у лютому 2015 року, за якими послідували альфа-версії у серпні 2015 року. Перша дистрибуція ROS 2, Ardent Apalone, була випущена 8 грудня 2017 року, відкриваючи нову епоху розвитку ROS нового покоління.

1.3.2 Архітектура системи ROS

ROS була розроблена як відкрите програмне забезпечення, з наміром, щоб користувачі могли обирати конфігурацію інструментів та бібліотек, які взаємодіють з ядром ROS, таким чином, щоб користувачі могли адаптувати свої програмні стеки під свої роботи та область застосування. Таким чином, в ROS дуже мало чого є ядром, за винятком загальної структури, в якій програми повинні існувати та комунікувати. В одному сенсі, ROS є підкладкою для вузлів та передачі повідомлень. Однак, насправді, ROS це не тільки ця підкладка, але й багатий і зрілий набір інструментів, широкий спектр можливостей для роботів, незалежних від платформи, що надаються пакетами, та більша екосистема доповнень до ROS.

Рисунок 5 – Креативна візуалізація роботи системи ROS

Процеси ROS представлені як вузли в структурі графа, з'єднані ребрами, які називаються темами. Вузли ROS можуть передавати повідомлення один одному через теми, робити виклики сервісів до інших вузлів, надавати сервіс для інших вузлів, або отримувати чи зберігати спільні дані з комунальної бази даних, що називається сервером параметрів. Процес, який називається ROS Master, робить все це можливим, реєструючи вузли для себе, налаштовуючи комунікацію між вузлами для тем і контролюючи оновлення сервера параметрів. Повідомлення та виклики сервісів не проходять через майстра, натомість майстер налаштовує однорангову комунікацію між усіма процесами вузлів після того, як вони зареєстрували себе у майстра. Ця децентралізована архітектура добре підходить для роботів, які часто складаються з підмножини мережевого комп'ютерного

обладнання, і можуть спілкуватися з зовнішніми комп'ютерами для важких обчислень або команд.

Вузол представляє один процес, що виконується в графі ROS. Кожен вузол має назву, яку він реєструє у ROS master, перш ніж може вживати будь-які інші дії. Може існувати кілька вузлів з різними назвами під різними просторами імен, або вузол може бути визначений як анонімний, у цьому випадку він випадковим чином генерує додатковий ідентифікатор для додавання до свого існуючого імені. Вузли є центром програмування ROS, оскільки більшість клієнтського коду ROS представлена у вигляді вузла ROS, який вживає дії на основі інформації, отриманої від інших вузлів, відправляє інформацію іншим вузлам або відправляє та отримує запити на дії від інших вузлів.

Теми є іменованими шинами, через які вузли відправляють та отримують повідомлення. Назви тем також повинні бути унікальними у межах їх простору імен. Щоб відправляти повідомлення на тему, вузол повинен публікувати на цій темі, а для отримання повідомлень - підписуватися. Модель публікації/підписки є анонімною: жоден вузол не знає, які вузли відправляють або отримують інформаціютемі, що він відправляє/отримує на цій темі. Типи повідомлень, що передаються по темі, можуть значно відрізнятися і можуть бути визначені користувачем. Зміст цих повідомлень може бути даними датчиків, командами управління двигунами, інформацією про стан, командами актуаторів або будь-чим іншим.

Вузол також може оголошувати сервіси. Сервіс представляє дію, яку вузол може виконати, яка матиме один результат. Таким чином, сервіси часто використовуються для дій, які мають визначений початок і кінець, таких як захоплення одного кадру зображення, а не обробка команд швидкості до колісного двигуна або даних одометра від кодера колеса. Вузли оголошують сервіси та викликають сервіси один в одного.

Сервер параметрів є базою даних, якою діляться між собою вузли, що дозволяє спільний доступ до статичної або напівстатичної інформації. Дані, які не змінюються часто і, як такі, будуть рідко доступні, такі як відстань між двома фіксованими точками в середовищі або вага робота, є гарними кандидатами для зберігання на сервері параметрів.

1.3.3 Інструменти системи ROS

Основну функціональність ROS доповнює різноманіття інструментів, які дозволяють розробникам візуалізувати та записувати дані, легко орієнтуватися у структурах пакетів ROS, а також створювати сценарії для автоматизації складних процесів налаштування та конфігурації. Додавання цих інструментів значно збільшує можливості систем на базі ROS, спрощуючи рішення багатьох поширених проблем розробки в робототехніці. Ці інструменти поставляються у вигляді пакетів, як і будь-які інші алгоритми, але вони забезпечують не реалізацію драйверів обладнання чи алгоритмів для різних робототехнічних завдань, а надають інструменти, які не залежать від завдання та робота і входять до ядра більшості сучасних інсталяцій ROS.

rviz (інструмент візуалізації роботів) - це тривимірний візуалізатор, який використовується для візуалізації роботів, середовищ, у яких вони працюють, та даних сенсорів. Це дуже налаштовуваний інструмент з багатьма різними типами візуалізацій та плагінів. Unified Robot Description Format (URDF) - це формат XML-файлу для опису моделі робота.

rosbag - це інструмент командного рядка, який використовується для запису та відтворення даних повідомлень ROS. rosbag використовує формат файлу, названий "bags", який реєструє повідомлення ROS, прослуховуючи теми та записуючи повідомлення, як тільки вони надходять. Відтворення повідомлень з "bags" майже таке ж, як і мати оригінальні вузли, які генерували дані в обчислювальному графі ROS, що робить "bags" корисним інструментом для запису даних,

які будуть використовуватися пізніше у розробці. Хоча rosbag є інструментом лише командного рядка, rqt_bag надає графічний інтерфейс для rosbag.

catkin - це система збірки ROS, яка замінила rosbuild з часів ROS Groovy. catkin базується на CMake і також є кросплатформенним, відкритим і незалежним від мови.

Пакет rosbash надає набір інструментів, які покращують функціональність оболонки bash. До цих інструментів входять rosls, roscd та roscp, які відтворюють функції ls, cd та cp відповідно. Версії ROS цих інструментів дозволяють користувачам використовувати імена пакетів ROS замість шляху до файлу, де розташований пакет. Пакет також додає автодоповнення для більшості утиліт ROS та включає rosed, який редагує заданий файл за допомогою обраного текстового редактора, а також rosrun, який запускає виконувані файли в пакетах ROS. rosbash підтримує ті ж функціональності для zsh та tcsh, хоча і в меншій мірі.

roslaunch - це інструмент, який використовується для запуску кількох вузлів ROS як локально, так і віддалено, а також для встановлення параметрів на сервері параметрів ROS. Конфігураційні файли roslaunch, які пишуться за допомогою XML, можуть легко автоматизувати складний процес запуску та налаштування в одну команду. Сценарії roslaunch можуть включати інші сценарії roslaunch, запускати вузли на певних машинах і навіть перезапускати процеси, які помирають під час виконання.

1.4 Постановка завдань магістерської роботи

У даній магістерській роботі буде розглянуто тему: "Дослідження мережевої взаємодії та розробка програми з'єднання мобільного роботу з віддаленою системою ROS2 з використанням websocket та інтернет-браузера".

Основна мета роботи – розробити програму для взаємодії робота з віддаленою системою ROS2.

У рамках цієї роботи буде розглянуто такі завдання:

– ознайомлення із операційною системою Linux Ubuntu;

– дослідження роботи HTTP запитів та веб-сокетів;

– ознайомлення із роботою плати nodeMcu, використання документації для написання функціоналу програми;

– ознайомлення з системою ROS2 та її інструментами для роботи.

Результатом цієї роботи буде розроблена програма для взаємодії робота із системою ROS2

2 РОЗРОБКА ПРОГРАМИ З'ЄДНАННЯ МОБІЛЬНОГО РОБОТУ З ВІДДАЛЕНОЮ СИСТЕМОЮ ROS2 З ВИКОРИСТАННЯМ WEBSOCKET ТА ІНТЕРНЕТ-БРАУЗЕРА.

2.1 Опис взаємодії у системі ROS2: publisher and subscription

2.1.1 Створення Publisher в ROS2

Щоб створити публічну тему в ROS2 за допомогою Python, необхідно розробити вузол-видавець. Ось приклад базового Python-скрипта для створення такого видавця в ROS2:

Спочатку створіть каталог для зберігання ваших пакетів ROS2. Зазвичай його називають "робочим простором". Його можна створити в будь-якому місці. Наприклад:

```
mkdir -p ~/my_ros2_ws/src
cd ~/my_ros2_ws/src
```

Це створює каталог під назвою my_ros2_ws з підкаталогом src. У каталозі src ви будете створювати пакунки ROS2.

Щоб створити новий пакет, ви можете скористатися командою ros2 pkg create. Наприклад:

```
ros2 pkg create --build-type ament_python my_package
```

Ця команда створює новий пакет під назвою my_package із типом збірки Python.

У каталозі my_package є підкаталог my_package. Тут ви розміщуєте свої вузли Python. Наприклад, ви можете створити файл my_nodepub.py у цьому каталозі.

```
my_package/
├── my_package/
│   ├── __init__.py
```

```
│   └── my_nodepub.py
├── package.xml
└── setup.py
```

Створення вузла Python у ROS2 передбачає написання сценарію Python, який використовує бібліотеку rclpy для взаємодії з екосистемою ROS2.

Ось простий приклад вузла Python. Цей вузол кожну секунду публікуватиме рядкове повідомлення «Hello, ROS2» на тему під назвою «chatter»:

```
create file my_nodepub.py
import rclpy
from rclpy.node import Node
from std_msgs.msg import String


class MyNode(Node):

    def __init__(self):
        super().__init__('my_node')
        self.publisher_ = self.create_publisher(String, 'chatter', 10)
        timer_period = 1.0  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello, ROS2 %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1
def main(args=None):
```

```
    rclpy.init(args=args)

    node = MyNode()

    rclpy.spin(node)

    # Destroy the node explicitly
    node.destroy_node()
    rclpy.shutdown()

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

Ви можете створити цей вузол, створивши файл my_nodepub.py у каталозі my_package/my_package, а потім скопіювавши цей код у my_nodepub.py.

Створивши my_nodepub.py, не забудьте зробити його виконуваним. Ви можете зробити це за допомогою такої команди:

```
chmod +x my_nodepub.py
```

Якщо ваші вузли Python мають залежності від інших пакетів ROS2, вам потрібно додати ці залежності до файлів package.xml і setup.py.

Крім того, вам потрібно додати правила встановлення для ваших вузлів Python у файлі setup.py. Це повідомляє ament (система збірки, яку використовує ROS2), як встановити ваші вузли.

Файл setup.py у пакеті ROS2 Python використовується для визначення способу збирання та встановлення пакета. Зокрема, він використовується для визначення модулів Python і сценаріїв (виконуваних файлів), які надаються пакетом.

Ось приклад файлу setup.py для пакета ROS2 Python**:**

```
from setuptools import setup

package_name = 'my_package'
```

```
setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='Your Name',
    maintainer_email='your.email@example.com',
    description='Description of my_package',
    license='Apache License 2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'my_nodepub = my_package.my_nodepub:main',
        ],
    },
)
```

У цьому прикладі аргумент entry_points використовується для визначення виконуваних файлів, наданих пакетом. Кожен виконуваний файл визначається рядком форми

'name = module:callable'.

У цьому випадку 'my_nodepub = my_package.my_nodepub:main' означає, що основна функція в модулі my_nodepub у пакеті my_package має бути встановлена як виконуваний файл із назвою my_nodepub.

Якщо у вас є кілька сценаріїв Python, які ви хочете встановити як виконувані файли, ви можете додати їх до списку console_scripts:

```
'console_scripts': [
  'my_nodepub = my_package.my_nodepub:main',
  'my_other_node = my_package.my_other_node:main',
],
```

Не забудьте замінити my_package, my_node і my_other_node фактичними назвами вашого пакета та вузлів.

Після оновлення файлу setup.py ви зможете створити свій пакет за допомогою збірки colcon, а вказані сценарії Python слід інсталювати як виконувані файли.

Після того, як ви налаштували свій пакет, ви можете створити його за допомогою colcon. Спочатку поверніться до кореневої папки вашої робочої області:

```
cd ~/my_ros2_ws
```

Потім створіть свій робочий простір за допомогою:

```
colcon build
```

Після створення робочого простору вам потрібно отримати його джерело, щоб ROS2 міг знайти ваші вузли:

```
source install/setup.bash
```

Ви створили та зібрали пакет ROS2 Python. Тепер ви можете запускати свої вузли за допомогою ros2 run. Наприклад:

```
ros2 run my_package my_nodepub
```

Ця команда запускає вузол my_nodepub з пакету my_package.

Зверніть увагу, що це загальні кроки для створення та збірки пакета ROS2 Python. Залежно від ваших конкретних потреб вам може знадобитися зробити більше (наприклад, визначити власні повідомлення чи служби або створити файли запуску).

2.1.2 Створення Subscriber в ROS2

Додайте свої вузли Python:

У каталозі my_package є підкаталог my_package. Тут ви розміщуєте свої вузли Python. Наприклад, ви можете створити файл my_nodesub.py у цьому каталозі.

```
my_package/
├── my_package/
│   ├── __init__.py
│   └── my_nodepub.py
│   └── my_nodesub.py
├── package.xml
└── setup.py
```

Створення вузла Python у ROS2 передбачає написання сценарію Python, який використовує бібліотеку rclpy для взаємодії з екосистемою ROS2.

Ось простий приклад вузла Python. Цей вузол підпишеться рядкове повідомлення «Hello, ROS2» у темі «chatter» кожну секунду:

створити файл my_nodesub.py

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class MySubscriber(Node):

  def __init__(self):
      super().__init__('my_subscriber')
      self.subscription = self.create_subscription(
          String,
          'chatter',
          self.listener_callback,10)
```

```
        self.subscription  # prevent unused variable warning

    def listener_callback(self, msg):
        self.get_logger().info('I heard: "%s"' % msg.data)
def main(args=None):
    rclpy.init(args=args)

    my_subscriber = MySubscriber()

    rclpy.spin(my_subscriber)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    my_subscriber.destroy_node()
    rclpy.shutdown()
if __name__ == '__main__':
    main()
```

Цей сценарій створить підписника, який прослуховує тему «тема» та викликає listener_callback кожного разу, коли отримує повідомлення на цю тему.

Зверніть увагу, що назву теми та тип даних можна змінити відповідно до ваших потреб. У цьому прикладі використовується тип повідомлення std_msgs/String, але ROS2 надає широкий вибір стандартних типів повідомлень, і ви також можете визначити власні.

Файл setup.py у пакеті ROS2 Python використовується для визначення способу збирання та встановлення пакета. Зокрема, він використовується для визначення модулів Python і сценаріїв (виконуваних файлів), які надаються пакетом.

Ось приклад файлу setup.py для пакета ROS2 Python:

```
from setuptools import setup

package_name = 'my_package'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='Your Name',
    maintainer_email='your.email@example.com',
    description='Description of my_package',
    license='Apache License 2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'my_nodepub = my_package.my_nodepub:main',
            'my_nodesub = my_package.my_nodesub:main',
        ],
    },
)
```

Щоб запустити свій сценарій, вам потрібно переконатися, що він виконуваний, і створити пакет ROS2:

```
chmod +x my_nodesub.py
colcon build
source install/setup.bash
ros2 run my_package my_nodesub
```

2.2 Опис стандартної взаємодії клієнт – сервер

Термін "клієнт-сервер" визначає відносини між співпрацюючими програмами в додатку, що складається з клієнтів, які ініціюють запити на послуги, та серверів, які надають цю функцію чи службу.

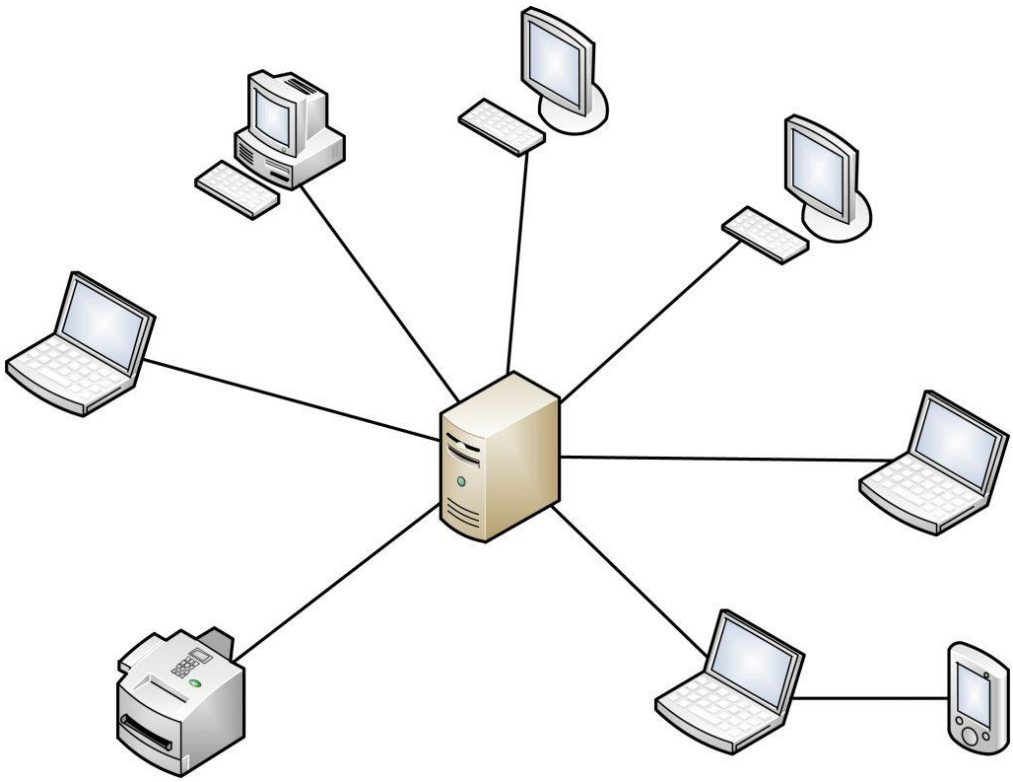

Рисунок 6 - Зображення з Automation Mission

2.2.1 Опис моделі "клієнт – сервер"

Модель "клієнт-сервер" або архітектура "клієнт-сервер" - це розподілена програмна структура, яка розділяє завдання між серверами та клієнтами, які

можуть розміщуватися в одній системі або спілкуватися через комп'ютерну мережу чи Інтернет. Клієнт покладається на відправлення запиту до іншої програми для доступу до сервісу, наданого сервером. Сервер запускає одну або кілька програм, які діляться ресурсами з клієнтами та розподіляють роботу між ними.

Клієнт-сервер комунікують за допомогою моделі запит-відповідь і мають дотримуватися спільного комунікаційного протоколу, який формально визначає правила, мову та шаблони діалогу, що використовуються. Комунікація клієнт-сервер зазвичай дотримується набору протоколів TCP/IP.

Протокол TCP підтримує з'єднання до завершення обміну повідомленнями між клієнтом та сервером. Протокол TCP визначає найкращий спосіб розподілу даних додатків на пакети, які можуть доставляти мережі, передає пакети до мережі та отримує пакети від мережі, а також управляє контролем потоку та повторною передачею втрачених або спотворених пакетів. IP - це протокол безперервного з'єднання, в якому кожен пакет, що подорожує через Інтернет, є незалежною одиницею даних, не пов'язаною з іншими одиницями даних.

Запити клієнтів організовуються та пріоритетизуються в системі планування, яка допомагає серверам справлятися у випадку отримання запитів від багатьох різних клієнтів за короткий проміжок часу. Підхід клієнт-сервер дозволяє будь-якому універсальному комп'ютеру розширити свої можливості, використовуючи спільні ресурси інших хостів. Популярні додатки клієнт-сервер включають електронну пошту, Всесвітню павутину та мережевий друк.

Існують чотири основні категорії обчислень "клієнт-сервер":

– однорівнева архітектура: складається з простої програми, що працює на одному комп'ютері без необхідності доступу до мережі. Запити користувача не управляють мережевими протоколами, тому код простий, а мережа звільнена від додаткового трафіку;

– дворівнева архітектура: складається з клієнта, сервера та протоколу, який з'єднує два рівні. Код графічного інтерфейсу користувача клієнта-сервера знаходиться на хості клієнта, а логіка домену - на хості сервера. Графічний інтерфейс

користувача клієнта-сервера написаний на високорівневих мовах, таких як C++ та Java;

– трирівнева архітектура: складається з презентаційного рівня, який є шаром інтерфейсу користувача, шару додатків, який виконує детальну обробку, та шару даних, який складається з сервера бази даних, який зберігає інформацію;

– багаторівнева архітектура: розділяє додаток на логічні шари, які розділяють обов'язки та управляють залежностями, та фізичні шари, які працюють на окремих машинах, покращують масштабованість та додають затримку з-за додаткового мережевого спілкування. Багаторівнева архітектура може бути закритошаровою, в якій шар може спілкуватися тільки з наступним шаром внизу, або відкритошаровою, в якій шар може спілкуватися з будь-якими шарами нижче.

Популярним прикладом трьохрівневої архітектури є Microsoft MySQL Server, який складається з трьох основних компонентів: протокольного шару, реляційного двигуна та двигуна зберігання. Будь-які клієнтські машини, які безпосередньо підключаються до SQL Server, повинні мати встановлений клієнт SQL Server. Процес виконання клієнт-сервера Microsoft допомагає управляти більшістю графічних наборів інструкцій на операційній системі Windows.

2.2.2 Мережа "клієнт-сервер"

Мережа "клієнт-сервер" - це середовище, через яке клієнти отримують доступ до ресурсів та послуг з центрального комп'ютера через локальну мережу (LAN) або широкомасштабну мережу (WAN), таку як Інтернет. Унікальний сервер, званий демоном, може використовуватися для того, щоб чекати запитів клієнтів, після чого ініціюється мережеве з'єднання до виконання запиту клієнта.

Трафік мережі класифікується як клієнт-сервер (північ-південь) або сервер-сервер (схід-захід). Популярні мережеві послуги включають електронну пошту, обмін файлами, друк та Всесвітню павутину. Однією з основних переваг мережі клієнт-сервер є централізоване управління додатками та даними.

Існує безліч переваг моделі архітектури "клієнт-сервер":

– один сервер, який вміщує всі необхідні дані в одному місці, полегшує захист даних та управління авторизацією та аутентифікацією користувачів;

– ресурси, такі як мережеві сегменти, сервери та комп'ютери, можна додавати до мережі без істотних перерв;

– дані можна ефективно отримувати без необхідності, щоб клієнти та сервер були розташовані поруч;

– усі вузли в системі клієнт-сервер незалежні, запитуючи дані тільки від сервера, що полегшує оновлення, заміну та переміщення вузлів.

2.2.2 Різниця між клієнтом та сервером

Клієнти, також відомі як запитувачі послуг, є частинами комп'ютерного обладнання або програмного забезпечення сервера, які запитують ресурси та послуги, що надаються сервером. Обчислення клієнта класифікуються як товсті, тонкі або гібридні:

– товстий клієнт: клієнт, який надає багатий функціонал, виконує більшість обробки даних самостійно і дуже мало покладається на сервер;

– тонкий клієнт: тонкий клієнт-сервер - це легкий комп'ютер, який значною мірою покладається на ресурси хост-комп'ютера - сервер додатків виконує більшість необхідної обробки даних;

– гібридний клієнт: маючи комбінацію характеристик тонкого та товстого клієнтів, гібридний клієнт покладається на сервер для зберігання постійних даних, але здатний до локальної обробки.

Сервер - це пристрій або комп'ютерна програма, що надає функціональність для інших пристроїв або програм. Будь-який комп'ютерний процес, який може бути використаний або викликаний клієнтом для спільного використання ресурсів та розподілу роботи, є сервером. Деякі загальні приклади серверів включають:

– сервер додатків: розміщує веб-додатки, якими користувачі в мережі можуть користуватися без необхідності мати власну копію;

– обчислювальний сервер: ділиться величезною кількістю комп'ютерних ресурсів з мережевими комп'ютерами, які потребують більше потужності процесора та оперативної пам'яті, ніж зазвичай доступно для персонального комп'ютера;

– сервер баз даних: підтримує та ділиться базами даних для будь-якої комп'ютерної програми, яка споживає добре організовані дані, такі як обліковий програмний забезпечення та електронні таблиці;

– веб-сервер: розміщує веб-сторінки та сприяє існуванню Всесвітньої павутини.

Візуалізація на стороні сервера означає здатність програми перетворювати файли HTML на сервері на повністю відтворену сторінку для клієнта. Веб-браузер робить запит на отримання інформації від сервера, який відповідає, як правило, за мілісекунди, повністю відтворюючи HTML-код. Пошукові системи можуть індексувати та сканувати вміст перед його доставкою, що робить рендеринг на стороні сервера дуже корисним для SEO.

Під час візуалізації клієнт-сервер замість отримання всього вмісту з документа HTML вміст відображається в браузері за допомогою бібліотеки JavaScript на стороні клієнта. Браузер не робить новий запит до сервера, коли завантажується нова сторінка. Це може негативно вплинути на рейтинг у пошуковій системі, оскільки вміст не відображається, доки сторінка не завантажиться у веб-переглядач, однак візуалізація веб-сайту, як правило, відбувається швидше на стороні клієнта.

Однорангова (P2P) мережа - це децентралізована модель комунікацій, у якій всі вузли в мережі мають еквівалентну можливість та можуть функціонувати як клієнт і сервер. Вузли в однорангових обчисленнях колективно використовують свої ресурси та спілкуються один з одним безпосередньо за потреби.

Алгоритм в протоколі однорангових комунікацій збалансовує навантаження, роблячи інші вузли доступними для компенсації будь-якого часу простою ресурсів та перенаправлення запитів, коли змінюється ємність навантаження та

доступність вузлів. Однією з основних переваг однорангових мереж є можливість розширення мережі для управління великою кількістю клієнтів.

У клієнт-серверних обчисленнях, централізованій моделі комунікацій, сервер є центральним вузлом, який спілкується з іншими клієнтськими вузлами. Одна з основних переваг відносин клієнт-сервер над відносинами однорангової мережі - це здатність управляти даними та додатками на одному централізованому сервері.

2.2.2 Опис взаємодії за допомогою WebSocket

WebSocket — це протокол комп’ютерного зв’язку , який забезпечує одночасний двосторонній зв’язок через одне з’єднання протоколу керування передачею (TCP). Протокол WebSocket був стандартизований IETF як RFC 6455 у 2011 році. Поточна специфікація, яка дозволяє веб-додаткам використовувати цей протокол, відома як WebSockets .

WebSocket відрізняється від протоколу передачі гіпертексту (HTTP), який використовується для обслуговування більшості веб-сторінок. Обидва протоколи розташовані на рівні 7 у моделі OSI та залежать від TCP на рівні 4. Незважаючи на те, що вони різні, RFC 6455 заявляє, що WebSocket "розроблений для роботи через HTTP-порти 443 і 80, а також для підтримки HTTP-проксі та посередників", що робить його сумісним з HTTP. Для досягнення сумісності WebSocket використовує заголовок HTTP Upgrade для зміни протоколу HTTP на протокол WebSocket.

Протокол WebSocket забезпечує повнодуплексну взаємодію між веб-браузером (або іншим клієнтським додатком) і веб-сервером із меншими накладними витратами, ніж напівдуплексні альтернативи, такі як опитування HTTP , полегшуючи передачу даних у реальному часі від сервера та до нього. Це стало можливим завдяки забезпеченню стандартизованого способу для сервера надсилати вміст клієнту без попереднього запиту клієнта, а також дозволу передачі повідомлень вперед і назад, зберігаючи з’єднання відкритим. Таким чином між

клієнтом і сервером може відбуватися двостороння передача данних. Зв’язок зазвичай здійснюється через TCP- порт номер 443 (або 80 у випадку незахищених з’єднань), що є корисним для середовищ, які блокують не веб-з’єднання з Інтернетом за допомогою брандмауера . Крім того, WebSocket дозволяє надсилати потоки повідомлень поверх TCP. Тільки TCP має справу з потоками байтів без внутрішньої концепції повідомлення. Подібний двосторонній зв’язок між браузером і сервером було досягнуто нестандартизованими способами за допомогою тимчасових технологій, таких як Comet або Adobe Flash Player .

Більшість браузерів підтримують протокол, зокрема Google Chrome , Firefox , Microsoft Edge , Internet Explorer , Safari та Opera .

Специфікація протоколу WebSocket визначає ws(WebSocket) і wss(WebSocket Secure) як дві нові схеми ідентифікатора ресурсу (URI) , які використовуються для незашифрованих і зашифрованих з’єднань відповідно. Крім назви схеми та фрагмента, решта компонентів URI визначено для використання загального синтаксису URI.

2.2.2 Історія виникнення WebSocket

WebSocket вперше згадувався як TCPConnection у специфікації HTML5 як заповнювач для API сокетів на основі TCP. У червні 2008 року під керівництвом Майкла Картера відбулася низка дискусій , результатом яких стала перша версія протоколу, відомого як WebSocket. До WebSocket повнодуплексний зв’язок через порт 80 був доступний за допомогою каналів Comet ; однак реалізація Comet є нетривіальною, і через зв’язок TCP і накладні витрати заголовка HTTP вона неефективна для невеликих повідомлень. Протокол WebSocket спрямований на вирішення цих проблем без шкоди для безпеки з'єднання. Назва WebSocket» була придумана Яном Хіксоном і Майклом Картером незабаром після цього під час співпраці в чаті і згодом розроблена для включення до специфікації HTML5 Яном Хіксоном. У грудні 2009 року Google Chrome 4 став першим браузером, який повністю підтримував стандарт із увімкненим WebSocket за

замовчуванням. Після того, як протокол було надіслано та ввімкнено за замовчуванням у кількох браузерах, у грудні 2011 року Ian Fette завершив роботу над RFC 6455 .

RFC 7692 представив розширення стиснення для WebSocket за допомогою алгоритму DEFLATE на основі кожного повідомлення.

3 ОПИС ПРОГРАМИ ДЛЯ З'ЄДНАННЯ МОБІЛЬНОГО РОБОТА ІЗ ВІДДАЛЕНОЮ СИСТЕМОЮ ROS2

3.1 Опис коду для nodeMcu

Цей проект демонструє використання плати ESP8266 для створення веб-сервера з метою приймання текстових повідомлень від системи ROS2. У цьому проекті, ESP8266 налаштовано для підключення до місцевої Wi-Fi мережі та створення веб-сервера, який дозволяє користувачам віддалено взаємодіяти з мікроконтролером через веб-браузер. Цей інтерфейс включає варіанти відправки стандартного повідомлення, налаштованого повідомлення та параметрів аналогового піну.

3.1.1 Підключення бібліотек для nodeMcu

У цьому проекті використовуються три основні бібліотеки, кожна з яких відіграє ключову роль у функціонуванні системи:

– ESP8266WiFi: ця бібліотека забезпечує підтримку Wi-Fi функціональності для плати ESP8266. Вона використовується для налаштування Wi-Fi з'єднання, включаючи підключення до мережі за допомогою SSID та пароля. Ця бібліотека є фундаментом для будь-якого проекту, який вимагає бездротового з'єднання, і є важливою для інтеграції ESP8266 в мережеві застосування;

– WebSocketsServer: WebSocket є протоколом комунікації, який забезпечує двосторонній канал зв'язку між веб-клієнтом і сервером по єдиному TCP-з'єднанню. Бібліотека WebSocketsServer використовується для створення WebSocket сервера на ESP8266. Це дозволяє реалізувати ефективний обмін даними в реальному часі між веб-сторінкою та ESP8266, що є критично важливим для завдань, які вимагають швидкої відповіді, наприклад, управління світлодіодом;

– ESP8266WebServer: Ця бібліотека використовується для створення HTTP сервера на платі ESP8266. Вона дозволяє обробляти HTTP запити та відправляти HTTP відповіді. У цьому проекті, ESP8266WebServer використовується для створення веб-інтерфейсу, який користувачі можуть використовувати для взаємодії з мікроконтролером через веб-браузер. Це включає надсилання команд для зміни текстового повідомлення, що відправляється на вузол ROS2.

Комбінація цих бібліотек створює потужну платформу для розробки IoT проектів, надаючи можливість не тільки для підключення до Wi-Fi мережі, але й для створення взаємодії в реальному часі та обробки веб-запитів, що є ключовими компонентами сучасних IoT рішень.

3.1.2 Налаштування мережі WIFI

Код проекту включає важливий етап налаштування та підключення до мережі WiFi, що є фундаментальним для функціонування IoT пристрою. У цьому контексті, SSID (Service Set Identifier) та пароль використовуються для ідентифікації та доступу до конкретної Wi-Fi мережі.

SSID (Service Set Identifier): SSID – це назва Wi-Fi мережі, до якої підключається ESP8266. У коді ця назва мережі визначена як змінна ssid. Ця інформація необхідна для того, щоб мікроконтролер міг ідентифікувати і вибрати правильну мережу Wi-Fi серед інших доступних.

Пароль Wi-Fi: Для забезпечення безпечного підключення, Wi-Fi мережі часто використовують паролі. У коді пароль мережі вказується у змінній password. Цей пароль необхідний для авторизації та отримання доступу до мережі.

Процес підключення до Wi-Fi ініціюється в функції setup(), де викликається метод WiFi.begin(ssid, password). Цей метод бере два аргументи: SSID та пароль, і використовує їх для запуску процесу підключення до вказаної Wi-Fi

мережі. Під час підключення, ESP8266 перевіряє доступні Wi-Fi мережі, знаходить мережу з відповідним SSID та намагається авторизуватися, використовуючи наданий пароль.

Після ініціації підключення, код виконує цикл, який чекає на успішне підключення до мережі, перевіряючи статус підключення за допомогою WiFi.status(). Цей цикл також відповідає за виведення повідомлень у серійний порт для інформування про процес підключення. Коли з'єднання встановлено, виводиться IP-адреса, присвоєна мікроконтролеру в мережі, що дозволяє користувачам легко знайти пристрій у мережі та взаємодіяти з ним.

Таким чином, налаштування Wi-Fi є критично важливим для забезпечення зв'язку між ESP8266 та мережею, що є основою для всіх подальших мережевих операцій і взаємодії з користувачем.

3.1.3 Ініціалізація серверів

В цьому проекті, використовуються два типи серверів - WebSocket та HTTP. Їх ініціалізація та налаштування є ключовими для взаємодії між ESP8266 та користувачем через веб-інтерфейс.

Ініціалізація WebSocket сервера здійснюється за допомогою створення екземпляра `WebSocketsServer` з портом 81, як показано в рядку `WebSocketsServer webSocket = WebSocketsServer(81);`. WebSocket сервер дозволяє здійснювати двосторонній обмін даними у реальному часі. У цьому проекті, він використовується для прийому текстових повідомлень від користувача через веб-інтерфейс та відправки відповідей назад. Використання порту 81 є стандартною практикою для WebSocket серверів, оскільки він зазвичай не використовується іншими сервісами.

HTTP сервер ініціалізується через клас `ESP8266WebServer` з портом 80, що є стандартним портом для HTTP з'єднань. Це відображено у рядку `ESP8266WebServer server(80);`. HTTP сервер відповідає за обслуговування веб-сторінок та обробку HTTP запитів від користувачів. Наприклад, коли користувач відвідує основну сторінку веб-інтерфейсу, запит обробляється HTTP сервером, який надсилає HTML контент для відображення у веб-браузері користувача.

Обидва сервери мають свої унікальні ролі у проекті. WebSocket сервер є ідеальним для ситуацій, коли необхідний швидкий обмін даними, таких як зміна стану світлодіода в реальному часі на основі входів користувача. Водночас, HTTP сервер використовується для більш традиційних веб-запитів та надання статичного вмісту, такого як веб-сторінки.

Це розділення завдань між двома серверами дозволяє більш ефективно розподіляти ресурси та оптимізувати взаємодію користувача з системою, забезпечуючи плавність та відгук на дії користувача без затримок.

3.1.4 Конфігурація апаратного забезпечення

Ефективна робота проекту базується не тільки на програмному коді, а й на правильній конфігурації апаратного забезпечення. Основним апаратним компонентом у цьому проекті є світлодіод (LED), який підключається до певного піну на платі ESP8266.

Конфігурація піну для світлодіода: У коді використовується константа ledPin, яка визначена як D2. Це означає, що світлодіод підключений до піну D2 на платі ESP8266. Пін D2 встановлюється як вихідний (output) в функції setup(), використовуючи команду pinMode(ledPin, OUTPUT);. Це важливо, оскільки вихідні піни використовуються для відправлення сигналів з мікроконтролера до інших пристроїв.

Контроль світлодіода: У функції handleWebSocketMessage, світлодіод контролюється за допомогою команд digitalWrite(ledPin, HIGH) та digitalWrite(ledPin, LOW). Ці команди включають та вимикають світлодіод відповідно, шляхом подачі або припинення подачі напруги на пін D2. Це дозволяє візуалізувати реакцію системи на дії користувача у веб-інтерфейсі.

Код також включає використання функції delay(500), що забезпечує часову затримку в 500 мілісекунд. Ця затримка використовується після кожного включення та вимикання світлодіода, створюючи ефект миготіння.

Крім використання цифрового піну D2 для управління світлодіодом, у проекті також задіяний аналоговий пін A0, до якого підключений змінний резистор. Цей пін використовується для зчитування аналогових сигналів, які змінюються в залежності від опору резистора. В розділі коду, де обробляється вибір `messageType == "analog"`, зчитується аналогове значення з піну A0, що відображає опір змінного резистора. Це значення потім використовується для генерації текстового повідомлення про поточний опір, яке надсилається через WebSocket. Ця функція дозволяє користувачу моніторити зміни опору в реальному часі через веб-інтерфейс.

Взаємодія між програмним та апаратним забезпеченням є критично важливою для успіху проекту. Правильна конфігурація пінів, контроль світлодіода та використання аналогових входів дозволяють створити багатофункціональну систему, яка може відповідати на веб-запити й взаємодіяти з фізичним світом.

3.1.5 Веб-інтерфейс для управління

Центральним елементом взаємодії з користувачем у цьому проекті є веб-інтерфейс, створений за допомогою HTML та JavaScript коду, вбудованого в функцію `handleRoot`. Цей інтерфейс дозволяє користувачам віддалено керувати змістом тексту, який буде відправлено на вузол ROS2.

Веб-сторінка створена за допомогою HTML, який задає структуру та вміст сторінки. В коді змінна `html` визначає HTML-контент, що включає основні елементи, такі як заголовок, форму для вибору типу повідомлення та відповідні пункти вибору (радіокнопки). Кожна радіокнопка відповідає певному режиму роботи - стандартне повідомлення, налаштоване повідомлення, та параметри аналогового піну.

JavaScript використовується для додання динамічності веб-інтерфейсу. У коді, JavaScript функція `sendMessageType(value)` відправляє HTTP GET запит на сервер при зміні вибраного режиму повідомлення. Це забезпечує взаємодію між користувачем та сервером без необхідності перезавантаження сторінки. Коли користувач вибирає різні режими відправки тексту, ця функція активується, відправляючи запит на сервер з новим значенням, що дозволяє змінити текст повідомлення.

Коли сервер отримує запит від JavaScript, він обробляється відповідною функцією на стороні сервера. Наприклад, якщо користувач вибирає опцію для відправлення стандартного повідомлення, сервер отримує цей запит та змінює стан рядка на інший.

Цей веб-інтерфейс є ключовим для інтерактивності проекту, оскільки він забезпечує зручний та інтуїтивно зрозумілий спосіб для користувачів керувати змістом тексту. Використання стандартних веб-технологій, таких як HTML та JavaScript, робить інтерфейс доступним через будь-який сучасний веб-браузер, забезпечуючи широку сумісність та легкість використання.

3.1.6 Обробка запитів на сервері

Обробка запитів на сервері відіграє ключову роль, оскільки це дозволяє забезпечити інтерактивність та зв'язок між користувачем і сервером. Це досягається за допомогою двох основних функцій: `handleMessageTypeChange` та `handleWebSocketMessage`.

Функція handleMessageTypeChange обробляє HTTP запити, які надходять від користувача через веб-інтерфейс. Коли користувач вибирає різний тип повідомлення (стандартне, налаштоване, аналогове), функція JavaScript `sendMessageType(value)` відправляє HTTP GET запит на сервер з параметром `type`, що відображає вибір користувача. На сервері, `handleMessageTypeChange` обробляє цей запит, встановлюючи глобальну змінну `messageType` відповідно до отриманого типу повідомлення. Сервер після цього відправляє підтвердження назад до веб-інтерфейсу.

Функція handleWebSocketMessage – це функція для обробки повідомлень, що надходять через WebSocket. Коли на сервер надходить текстове повідомлення від користувача, ця функція активує миготіння світлодіода, яке служить як індикатор отримання нового повідомлення. Після цього, в залежності від поточного значення `messageType`, сервер генерує відповідне текстове повідомлення та відправляє його назад до користувача через WebSocket. Це означає, що світлодіод не використовується для керування, а лише як індикатор отримання повідомлення. Наприклад, при виборі "стандартного" повідомлення сервер відправляє "Standard Message", для "налаштованого" - "Custom Message when RadioButton is Selected". У разі вибору аналогового режиму, функція читає значення з аналогового піну A0 і відправляє це значення як частину повідомлення.

Ці функції забезпечують ефективну взаємодію між сервером та користувачем, реагуючи на дії користувача в реальному часі. Вони відіграють важливу роль у забезпеченні інтерактивності проекту, дозволяючи користувачам віддалено змінювати тип повідомлень і отримувати відповідь від системи через веб-інтерфейс.

3.1.7 Налаштування та запуск сервера

Функція `setup` у коді має критичне значення, оскільки вона відповідає за ініціалізацію та налаштування всіх необхідних компонентів системи перед її запуском. Ця функція виконує кілька ключових завдань:

Першочергово, функція `setup` ініціалізує Wi-Fi з'єднання, використовуючи SSID та пароль, задані в змінних `ssid` та `password`. Це робиться за допомогою виклику `WiFi.begin(ssid, password)`. Після цього, функція входить у цикл, який очікує підключення до Wi-Fi мережі, відображаючи статус підключення через серійний порт.

Функція `setup` також встановлює пін, призначений для світлодіода, як вихідний, використовуючи `pinMode(ledPin, OUTPUT)`. Це гарантує, що мікроконтролер зможе керувати світлодіодом для відображення стану системи.

Важливою частиною функції `setup` є ініціалізація та запуск WebSocket сервера. Це робиться шляхом виклику `webSocket.begin()`, який запускає WebSocket сервер на порту, вказаному в конструкторі `WebSocketsServer`. Функція також визначає `handleWebSocketMessage` як обробник подій для сервера, що дозволяє ефективно реагувати на повідомлення, отримані через WebSocket.

Подібно до WebSocket сервера, HTTP сервер ініціалізується та запускається в функції `setup`. Використовуючи `server.on("/", handleRoot)` та `server.on("/setMessageType", handleMessageTypeChange)`, код встановлює обробники для основного шляху та шляху налаштування типу повідомлення відповідно. Нарешті, `server.begin()` запускає HTTP сервер, дозволяючи обробляти веб-запити.

В цілому, функція `setup` виконує всі необхідні налаштування, що забезпечують стабільну роботу системи. Вона ініціалізує з'єднання з Wi-Fi, налаштовує піни для взаємодії з апаратними компонентами, а також запускає сервери,

необхідні для взаємодії з користувачем через веб-інтерфейс. Ця функція є фундаментом, на якому будується вся подальша діяльність системи.

3.1.8 Головний цикл виконання

Функція `loop` є серцем програми на платі ESP8266, вона виконується безперервно після виконання функції `setup`. Основна роль цієї функції полягає у підтримці активності WebSocket та обробці HTTP запитів. Виклик `webSocket.loop()` у функції `loop` забезпечує підтримку активного з'єднання WebSocket. Це означає, що сервер регулярно перевіряє наявність нових повідомлень від клієнтів і обробляє їх за потреби. Це критично важливо для забезпечення відповіді сервера на дії користувача в реальному часі.

Обробка HTTP запитів: `server.handleClient()` викликається у кожному циклі `loop`, що дозволяє HTTP серверу обробляти запити, які надходять від користувачів. Це включає обробку запитів на зміну типу повідомлення та відправлення HTML сторінки користувачеві.

Виконання цих функцій у бескінечному циклі гарантує, що сервер завжди готовий реагувати на дії користувача, забезпечуючи надійну та стабільну роботу системи.

Цей проект на платі ESP8266 демонструє розробку простої, але функціональної системи для дистанційного управління та взаємодії через веб-інтерфейс. Використовуючи комбінацію WebSocket та HTTP серверів, проект дозволяє користувачам віддалено змінювати тип повідомлення, яке відображається на сервері, та отримувати індикацію отримання повідомлення через миготіння світлодіода. Можливі сценарії використання цього проекту включають створення основи для розширених IoT-застосувань, таких як домашня автоматизація, системи моніторингу, або навіть освітні цілі, що демонструють взаємодію між мікроконтролером та веб-технологіями. Цей проект слугує чудовим прикладом того, як з використанням відносно простих компонентів та коду можна створити

інтерактивні та зв'язані системи, відкриваючи шлях для більш складних та інноваційних IoT рішень.

3.2 Опис коду для вузла ROS2

3.2.1 Ініціалізація ROS2 Node

Код розпочинається з ініціалізації вузла ROS 2, який є основою для взаємодії з системою ROS. Вузол названий `ros2_node` і створюється як екземпляр класу `MyNode`, що успадковує від базового класу `Node`, наданого ROS 2.

У класі `MyNode` ініціалізуються публікатори та підписники для двох тем: `ros2_request_topic` та `ros2_response_topic`. Публікатори (`publisher_request` та `publisher_response`) використовуються для відправлення повідомлень до цих тем, тоді як підписник (`subscriber_request`) реєструється для одержання повідомлень з `ros2_request_topic`.

Для обробки вхідних повідомлень з теми `ros2_request_topic`, в класі `MyNode` визначена функція зворотного виклику `callback_request`. Ця функція автоматично викликається, коли на вузол надходить нове повідомлення у відповідній темі. Вона використовується для логування отриманих повідомлень та може бути розширена для виконання додаткової обробки даних.

Ця ініціалізація вузла ROS 2 та його компонентів є критично важливою для функціонування системи, оскільки вона забезпечує засоби для комунікації та обміну даними у мережі ROS.

3.2.2 Взаємодія через WebSocket

В центрі цього коду лежить взаємодія з NodeMCU через WebSocket, що відкриває шлях для двостороннього зв'язку між ROS 2 вузлом і фізичним пристроєм.

Процес взаємодії розпочинається з підключення до WebSocket сервера, що розгорнуто на NodeMCU. URI сервера (у цьому випадку 'ws://192.168.2.103:81') використовується для встановлення з'єднання. Це з'єднання ініціалізується та управляється за допомогою бібліотеки `websockets` у Python, яка надає асинхронний API для роботи з WebSocket.

Після встановлення з'єднання, код дозволяє відправляти текстові повідомлення на NodeMCU. Це включає як початкове тестове повідомлення "Hello from ROS 2!", так і повідомлення, введені користувачем через консоль. Використання асинхронної функції `send_websocket_request` дозволяє відправку повідомлень та очікування відповідей без блокування основного потоку виконання.

Коли відповідь надходить з NodeMCU, вона приймається та обробляється. Отримані відповіді виводяться в консоль для інформування користувача. Крім того, відповіді також публікуються у відповідній ROS 2 темі (`ros2_response_topic`), що дозволяє інтеграцію з іншими компонентами ROS 2 системи.

Цей механізм взаємодії через WebSocket є ключовим елементом системи, оскільки він забезпечує гнучкий та ефективний спосіб комунікації між ROS 2 вузлом і віддаленими пристроями, такими як NodeMCU.

3.2.2 Функція send_websocket_request

Функція `send_websocket_request` відіграє центральну роль у взаємодії між ROS 2 вузлом та NodeMCU через WebSocket. Ця функція є асинхронною, що дозволяє ефективно виконувати мережеві запити без блокування основного потоку програми.

Функція приймає три параметри: вузол ROS 2 (`node`), URI WebSocket сервера (`uri`), та повідомлення, яке потрібно відправити (`message`). Вона встановлює з'єднання з WebSocket сервером за допомогою `websockets.connect(uri)`

та відправляє задане повідомлення на сервер. Це відправлення реалізується асинхронно, використовуючи `await websocket.send(message)`.

Після відправлення повідомлення, функція очікує на отримання відповіді від сервера. Відповідь приймається за допомогою `await websocket.recv()`. Отримана відповідь виводиться у консоль, що дозволяє користувачу бачити зворотній зв'язок від NodeMCU.

Окрім відправлення та отримання даних через WebSocket, функція також здійснює публікацію цих даних у відповідні теми ROS 2. Відправлене повідомлення публікується у темі `ros2_request_topic`, а отримана відповідь – у темі `ros2_response_topic`. Це реалізується шляхом створення нових екземплярів типу `String`, їх наповнення відповідними даними та використання методу `publish` відповідних публікаторів вузла.

Ця функція є ключовою для інтеграції ROS2 системи з зовнішніми пристроями через WebSocket, надаючи засоби для двостороннього зв'язку та обміну даними між різними компонентами системи.

3.2.2 Головна функція та обробка вводу

Головна функція `main` відіграє вирішальну роль у цьому коді, оскільки вона ініціалізує вузол ROS 2 та управляє всіма основними процесами програми.

Початок функції `main` присвячений ініціалізації ROS2. За допомогою виклику `rclpy.init()`, ініціалізується ROS 2 клієнтська бібліотека, яка дозволяє програмі взаємодіяти з ROS 2 екосистемою. Далі створюється екземпляр `MyNode`, який активує вузол ROS 2, встановлюючи зв'язок з публікаторами та підписниками.

В головній функції визначається URI WebSocket сервера, до якого буде встановлено з'єднання. Після цього, програма переходить у цикл, де користувач може ввести текстове повідомлення. Введене повідомлення відправляється на

NodeMCU за допомогою функції `send_websocket_request`, яка також відповідає за отримання відповіді та публікацію в ROS 2 теми.

Після завершення роботи з програмою (коли користувач не вводить повідомлення та натискає Enter), функція `main` виконує очищення ресурсів. Це включає знищення створеного вузла (`node.destroy_node()`) та виклик `rclpy.shutdown()` для коректного завершення роботи з ROS 2. Ці кроки гарантують, що всі ресурси, використані вузлом, будуть належним чином звільнені.

Головна функція `main` відіграє важливу роль у координації всіх важливих процесів програми, забезпечуючи інтерактивний зв'язок між користувачем, ROS2 та зовнішніми пристроями через WebSocket.

3.3 Опис механізму взаємодії nodeMcu та вузла ROS2

3.3.1 Тестування коду на nodeMcu

Робота починається з завантаження скетчу в плату nodeMcu через Arduino IDE.

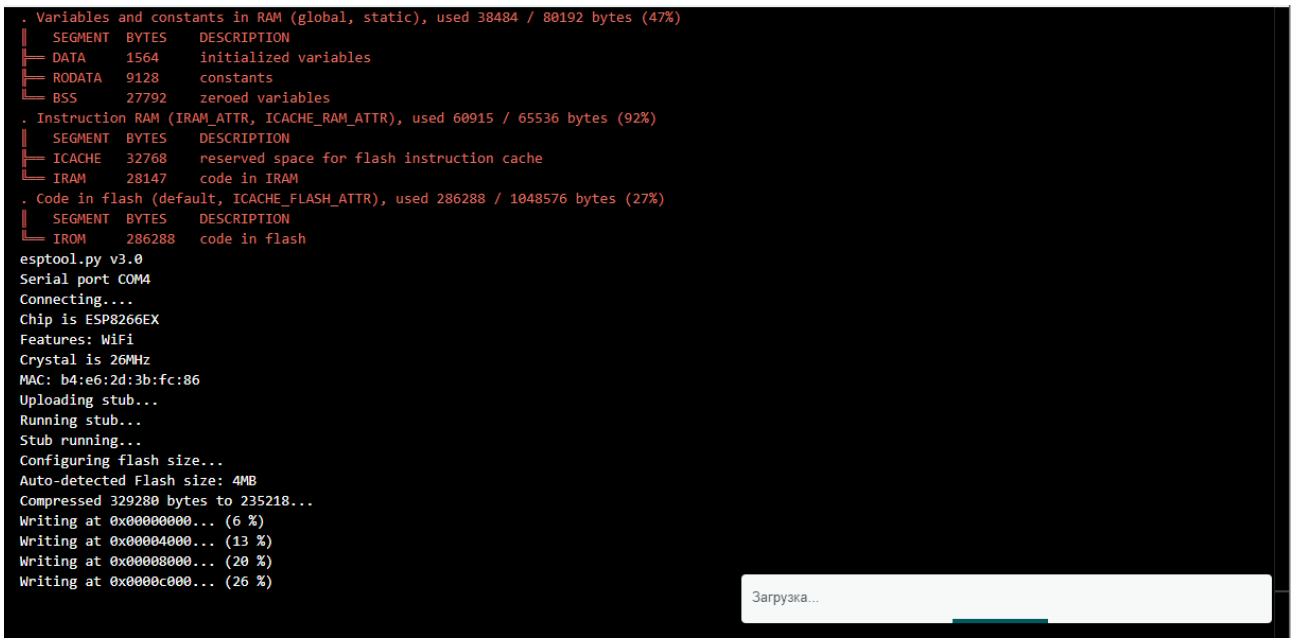

Рисунок 7 – Завантаження скетчу за допомогою Arduino IDE

Після цього в моніторі порта з’являється інформація стосовно підключення до мережі WIFI та IP адреса веб-серверу nodeMcu.

```
Connecting to WiFi...
Connecting to WiFi...
Connecting to WiFi...
Connecting to WiFi...
Connected to WiFi
IP Address: 192.168.1.101
```

Рисунок 8 – Процес під’єднання до мережі WIFI

Так як плата nodeMcu запущена, при вводі IP адреси в пошуковий рядок браузера, нам стає доступна HTML сторінка із вибором кастомного повідомлення для відправки на вузол ROS2. Але головною умовою є те, що важливо бути підключеним до однієї і тієї ж WIFI , або локальної мережі.

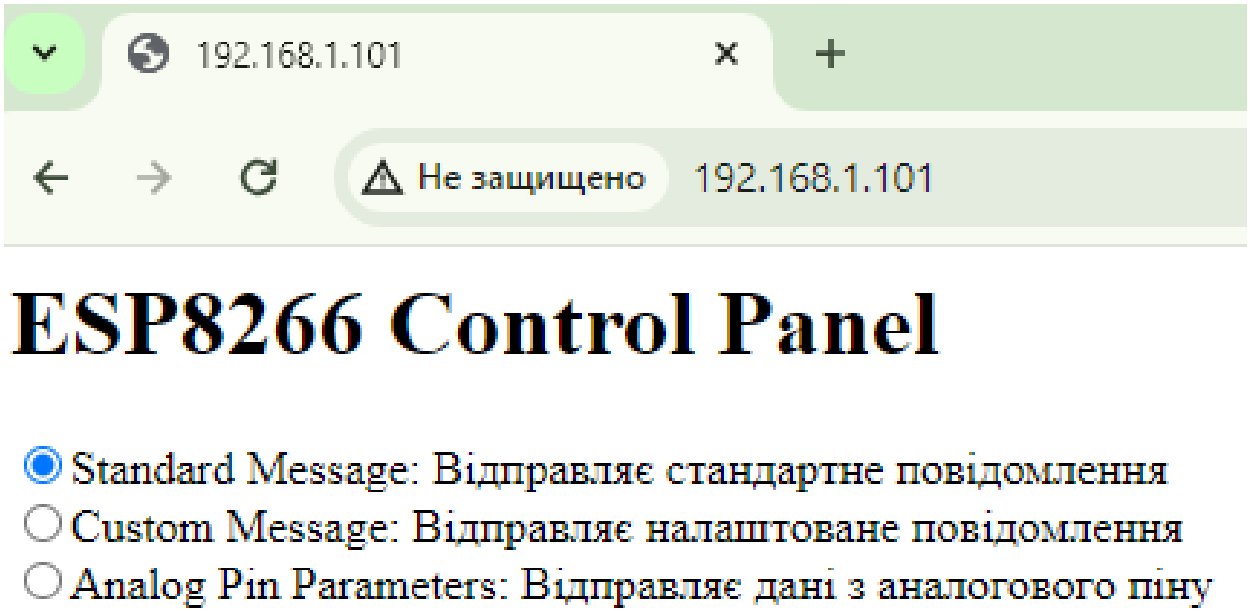

Рисунок 9 – HTML сторінка із веб-інтерфейсом

Після завершення роботи з програмою (коли користувач не вводить повідомлення та натискає Enter), функція `main` виконує очищення ресурсів. Це включає знищення створеного вузла (`node.destroy_node()`) та виклик `rclpy.shutdown()` для коректного завершення роботи з ROS 2. Ці кроки гарантують, що всі ресурси, використані вузлом, будуть належним чином звільнені.

Головна функція `main` відіграє важливу роль у координації всіх важливих процесів програми, забезпечуючи інтерактивний зв'язок між користувачем, ROS2 та зовнішніми пристроями через WebSocket.

3.3.2 Тестування коду вузла ROS2

Слідуючим етапом буде запуск вузла системи ROS2. Для цього нам знадобиться операційна система Linux Ubuntu та термінал. Для запуска вузла потрібно ввести команду:

ros2 run my_robot_controller test_node

Дуже важливо в коді на Python записати IP сервера nodeMcu. Після запуску вузла термінал сповістить про успіше встановлення з’єднання та готовність до роботи.

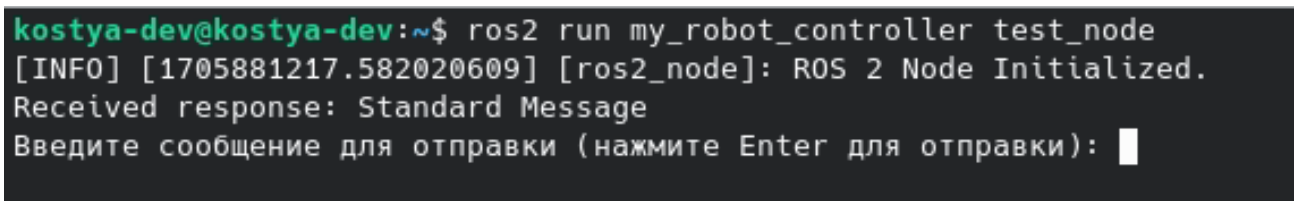

Рисунок 10 – Успішний запуск вузла ROS2

На цьому етапі вузол ROS2 та nodeMcu готові до взаємодії. При запущеному вузлі ROS2 нам доступні два топіки: з запитом який ми відправляємо та відповідь від nodeMcu.

```
kostya-dev@kostya-dev:~$ ros2 run my_robot_controller test_node
[INFO] [1705881833.092196011] [ros2_node]: ROS 2 Node Initialized.
Received response: Standard Message
Введите сообщение для отправки (нажмите Enter для отправки): this is my message
Received response: Standard Message
```

Рисунок 11 – Передача тексту із системи ROS2

Тепер за допомогою веб-інтерфейсу який створює nodeMcu ми можемо змінювати текст відповіді плати на запит.

```
kostya-dev@kostya-dev:~$ ros2 topic echo /ros2_request_topic
data: this is my message
---
data: My second message
---
```

~ : ros2

```
kostya-dev@kostya-dev:~$ ros2 topic echo /ros2_response_topic
data: Standard Message
---
data: Custom Message when RadioButton is Selected
---
```

Рисунок 12 – Приклад кастомної відповіді від сервера nodeMcu

Швидкість публікації повідомлень у паблішері ROS2:

– у контексті даного проекту, ROS2 не має вбудованих обмежень на частоту публікації повідомлень. Ефективність публікації залежить від швидкості отримання даних через WebSocket, що передбачає адаптивну частоту оновлення залежно від вхідного потоку даних;

– продуктивність ROS2 у плані обробки та передачі повідомлень піддається впливу зовнішніх факторів, таких як обчислювальні ресурси системи, складність даних і мережева інфраструктура. Це підкреслює необхідність оптимізації як програмного забезпечення, так і апаратної частини для досягнення максимальної ефективності.

Швидкість Роботи З'єднання WebSocket:

– з'єднання WebSocket забезпечує високу продуктивність завдяки своїй повнодуплексній природі. Однак, практична швидкість передачі даних обмежується мережевими умовами та якістю Wi-Fi з'єднання.

– використання ESP8266 (NodeMCU) вносить певні обмеження через його відносно низьку обчислювальну потужність та обмежені мережеві можливості порівняно з більш потужними системами.

Поведінка при розриві інтернет-з'єднання:

– при втраті інтернет-з'єднання спостерігається припинення оновлення даних на веб-сторінці, завантаженій з NodeMCU. У очікуванні даних від сервера, сторінка може тимчасово "зависання";

– з втратою мережі відбувається автоматичне роз'єднання WebSocket. Це вимагає впровадження стратегій обробки винятків та відновлення з'єднання у програмному забезпеченні ROS2 для запобігання системних збоїв;

– теоретично, через WebSocket можна передавати значну кількість повідомлень у секунду. Однак фактичні показники обмежуються затримками у

мережі та швидкістю обробки як на стороні nodeMCU, так і в ROS2, що зазвичай обмежується кількома повідомленнями у секунду.

У рамках дипломного проєкту, інтеграція ROS2 з WebSocket через платформу nodeMCU демонструє потенціал для ефективного двостороннього зв'язку та передачі даних. Однак, досягнення оптимальної продуктивності вимагає врахування апаратних обмежень, мережевих умов та надійного управління винятками, особливо у сценаріях втрати з'єднання. Ці фактори мають ключове значення для забезпечення стабільності та надійності системи у реальних умовах експлуатації.

ВИСНОВКИ

Дослідження мережевої взаємодії та розробка програми для з'єднання мобільного роботу з віддаленою системою ROS2, використовуючи WebSocket та інтернет-браузер, є важливим кроком у розвитку сучасних методів керування робототехнікою. Використання WebSocket у даному контексті є особливо ефективним рішенням, оскільки ця технологія забезпечує двосторонній, неперервний обмін даними між клієнтом і сервером, що є критично важливим для забезпечення оперативного управління роботами.

Перевагами використання WebSocket є, насамперед, низька затримка в обміні повідомленнями, що дозволяє роботам оперативно реагувати на команди та зміни у навколишньому середовищі. Також, WebSocket сприяє ефективній роботі в реальному часі, що є необхідним для робототехніки, де швидка відповідь на події може бути критичною. Іншою важливою перевагою є простота інтеграції з сучасними веб-технологіями, що дозволяє розробникам легко створювати та вдосконалювати інтерфейси для керування роботами.

Дослідження цієї теми є значущим не лише з технічної точки зору, але й з практичної. У сучасному світі, де робототехніка знаходить все більше застосувань у різних сферах життя, ефективне та надійне управління роботами стає ключовим фактором. Інтеграція веб-технологій із системами управління роботами відкриває нові можливості для розвитку робототехніки, зокрема, з погляду віддаленого управління, моніторингу та взаємодії з роботами через Інтернет.

Таким чином, використання WebSocket для обміну даними у системі управління мобільним роботом з ROS2 є перспективним рішенням, що відкриває широкі можливості для покращення якості та ефективності керування робототехнічними системами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Radartutorial – Радіолокатори підповерхневого зондування [Електронний ресурс]. – Режим доступу : https://www.radartutorial.eu/02.basics/rp16.uk.html

1. Instructables – Get Started With NodeMCU (ESP8266) [Electronic resource]. – Access mode : https://www.instructables.com/Get-Started-With-NodeMCU/

2. Wikipedia – NodeMCU [Electronic resource]. – Access mode:https://en.wikipedia.org/wiki/NodeMCU#:~text=for%20%22Enhanced%22.-,History,of%20nodemcu%2Dfirmware%20to%20GitHub.

3. Academic-accelerator – NodeMCU (ESP8266) [Electronic resource]. – Access mode : https://academic-accelerator.com/encyclopedia/nodemcu

4. Wikipedia – Web page [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Web_page

5. Techopedia.– Web Page [Electronic resource]. – Access mode : https://www.techopedia.com/definition/4774/web-page-page

6. Computerhope.– Web Page [Electronic resource]. – Access mode : https://www.computerhope.com/jargon/w/webpage.htm

7. Allaboutcookies.– What Is a Web Page? [Electronic resource]. – Access mode : https://allaboutcookies.org/what-is-a-web-page

8. Wikipedia – Static Web page [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Static_web_page

9. Blog.hubspot – Static vs. Dynamic Websites: Here's the Difference [Electronic resource]. – Access mode: https://blog.hubspot.com/website/static-vs-dynamic-website

10. w3schools – Why create a static website? [Electronic resource]. – Access mode: https://www.w3schools.com/howto/howto_website_static.asp

11. Wikipedia – HTTP [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/HTTP

12. Developer.mozilla – HTTP request methods [Electronic resource]. – Access mode: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

13. Learn.javascript – XMLHttpRequst [Electronic resource]. – Режим доступу : https://learn.javascript.ru/xmlhttprequest

14. Wikipedia – Robot Operating System [Electronic resource]. – Access mode: https://uk.wikipedia.org/wiki/Robot_Operating_System

15. Ros – Humble [Electronic resource]. – Access mode: https://docs.ros.org/en/humble/index.html

16. Theconstructsim – A History of ROS (Robot Operating System) [Electronic resource]. – Access mode: https://www.theconstructsim.com/history-ros/

17. Medium – Ros 2 Publisher and Subscriber Python [Electronic resource]. – Access mode: https://medium.com/@poommin2543/ros-2-publisher-and-subscriber-python-af299fc834a6

18. Theconstructsim – Combine Publisher, Subscriber & Service in ROS2 Single Node (Robot Operating System) [Electronic resource]. – Access mode: https://www.theconstructsim.com/combine-publisher-subscriber-service-in-ros2-single-node/

19. ROS2 workshop – Understanding ROS 2 nodes with a simple Publisher - Subscriber pair [Electronic resource]. – Access mode: https://ros2-industrial-workshop.readthedocs.io/en/latest/_source/basics/ROS2-Simple-Publisher-Subscriber.html

20. Developer.mozilla – WebSocket [Electronic resource]. – Access mode: https://developer.mozilla.org/ru/docs/Web/API/WebSocket

21. Learn.javascript – Websockets [Electronic resource]. – Access mode: https://learn.javascript.ru/websockets

22. . Wikipedia – WebSocket [Electronic resource]. – Access mode: https://ru.wikipedia.org/wiki/WebSocket

ДОДАТОК А

Текст програми nodeMcu

```
#include <ESP8266WiFi.h>
#include <WebSocketsServer.h>
#include <ESP8266WebServer.h>
const char *ssid = "netis_2.4G_552940";
const char *password = "33555den";
WebSocketsServer webSocket = WebSocketsServer(81);
ESP8266WebServer server(80);
const int ledPin = D2;
unsigned long lastLEDChange = 0;
String messageType = "standard";
void handleRoot() {
  String html = "<html>"
                "<head><meta charset='UTF-8'></head>"
                "<body>"
                "<h1>ESP8266 Control Panel</h1>"
                "<form>"
                "<input type='radio' id='standardMessage' name='messageType' value='standard' checked onchange='sendMessageType(this.value)'><label for='standardMessage'>Standard Message</label><span>: Відправляє стандартне повідомлення</span><br>"
                "<input type='radio' id='customMessage' name='messageType' value='custom' onchange='sendMessageType(this.value)'><label for='customMessage'>Custom Message</label><span>: Відправляє налаштоване повідомлення</span><br>"
                "<input type='radio' id='analogParameters' name='messageType' value='analog' onchange='sendMessageType(this.value)'><label for='analogParameters'>Analog Pin Parameters</label><span>: Відправляє дані з аналогового піну</span>"
                "</form>"
                "<script>"
                "function sendMessageType(value) {"
                "  var xhr = new XMLHttpRequest();"
```

```
            "  xhr.open('GET', '/setMessageType?type=' + value, true);"
            "  xhr.send();"
            "}"
            "</script>"
            "</body></html>";
  server.send(200, "text/html", html);
}
void handleMessageTypeChange() {
  if (server.hasArg("type")) {
    messageType = server.arg("type");
  }
  server.send(200, "text/plain", "Message Type Updated");
}
void handleWebSocketMessage(uint8_t num, WStype_t type, uint8_t *payload, size_t length) {
  switch (type) {
    case WStype_TEXT:
      Serial.printf("[%u] Received text message: %s\n", num, payload);
      for (int j = 0; j < 2; ++j) {
        digitalWrite(ledPin, HIGH);
        delay(500);
        digitalWrite(ledPin, LOW);
        delay(500);
      }
      lastLEDChange = millis();
      if (messageType == "standard") {
        webSocket.sendTXT(num, "Standard Message");
      } else if (messageType == "custom") {
        webSocket.sendTXT(num, "Custom Message when RadioButton is Selected");
      } else if (messageType == "analog") {
        int analogValue = analogRead(A0);
        String analogMessage = "Analog Value: " + String(analogValue);
```

```
        webSocket.sendTXT(num, analogMessage);
      }
      break;
  }
}
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
  pinMode(ledPin, OUTPUT);
  webSocket.begin();
  webSocket.onEvent(handleWebSocketMessage);
  server.on("/", handleRoot);
  server.on("/setMessageType", handleMessageTypeChange);
  server.begin();
}
void loop() {
  webSocket.loop();
  server.handleClient();
}
```

ДОДАТОК Б

Текст програми для вузла ROS2

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
import websockets
import asyncio
class MyNode(Node):
    def __init__(self):
        super().__init__("ros2_node")
        self.publisher_request = self.create_publisher(String, 'ros2_request_topic', 10)
        self.publisher_response = self.create_publisher(String, 'ros2_response_topic', 10)
        self.subscriber_request = self.create_subscription(String, 'ros2_request_topic', self.callback_request, 10)
        self.get_logger().info("ROS 2 Node Initialized.")
    def callback_request(self, msg):
        # Обработка сообщений, полученных из топика 'ros2_request_topic'
        self.get_logger().info(f"Received request: {msg.data}")


def main():
    rclpy.init()
    node = MyNode()
    # Замените IP-адрес на адрес NodeMCU
    websocket_uri = 'ws://192.168.2.103:81'


    try:
        asyncio.get_event_loop().run_until_complete(send_websocket_request(node, websocket_uri, "Hello from ROS 2!"))
```

```
        while True:
            message = input("Введите сообщение для отправки (нажмите Enter для отправки): ")
            if message:
                asyncio.get_event_loop().run_until_complete(send_websocket_request(node, web-
socket_uri, message))
            else:
                break
    finally:
        node.destroy_node()
        rclpy.shutdown()
async def send_websocket_request(node, uri, message):
    async with websockets.connect(uri) as websocket:
        await websocket.send(message)
        response = await websocket.recv()
        print(f"Received response: {response}")
        # Публикация отправленного сообщения в топик 'ros2_request_topic'
        msg_request = String()
        msg_request.data = message
        node.publisher_request.publish(msg_request)
        # Публикация полученного ответа в топик 'ros2_response_topic'
        msg_response = String()
        msg_response.data = response
        node.publisher_response.publish(msg_response)
if __name__ == '__main__':
    main()
```