Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розробка системи керування автономної роботизованої платформи у виробничому приміщенні на базі Robot Operation System
(тема)

Виконав:
студент 4 курсу, групи АКТАКІТ-20-3
Бобков М. В.
(прізвище, ініціали)

Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Автоматизація та комп'ютерно-інтегровані технології
(повна назва освітньої програми)

Керівник ст. викл. Гурін Д. В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Невлюдов І. Ш.
(підпис) (прізвище, ініціали)

2024 р.

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра КІТАР
Рівень вищої освіти перший (бакалаврський)
Спеціальність 151 Автоматизація та комп`ютерно-інтегровані технології
Тип програми Освітньо-професійна
Освітня програма Автоматизація та комп`ютерно-інтегровані технології
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри КІТАМ _____
(підпис)
«___» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Бобкову Микиті Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка системи керування автономної роботизованої платформи у виробничому приміщенні на базі Robot Operation System

Затверджена наказом по університету від 03.06.2024 р. № 544 СТ
2. Термін подання студентом роботи до екзаменаційної комісії 12.06.2023 р.
3. Вихідні дані до роботи напруга джерел живлення 3,3-5 В для мікроконтролера, джерело живлення для платформи в залежності від бази платформи, мова програмування C/C++, фреймворк ROS, фреймворк ESP-IDF.
4. Перелік питань, що потрібно опрацювати в роботі 4.1 Вступ; 4.2 Аналіз предметної області; 4.3 Розробка апаратної реалізації проекту; 4.4 Розробка програмної реалізації проекту; 4.5 Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій
Демонстраційний матеріал у вигляді презентації в форматі ppt 10 с.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Приміт-ка
1	Складання технологічного завдання	22.04 – 26.04.24	виконано
2	Аналіз літературних джерел та аналогічних рішень	26.04 – 03.05.24	виконано
3	Складання розділу розробки апаратної частини реалізації проекту	10.05 – 20.05.24	виконано
4	Складання розділу розробки програмної частини реалізації проекту та експериментальних дослідів	20.05 – 01.06.24	виконано
5	Оформлювання пояснювальної записки	11.06.2024	виконано
6	Подання роботи на перевірку Інтернет-сервісом Unicheck	11.06 – 12.06.24	
7	Подання роботи на рецензію		
8	Подання роботи на підпис зав. кафедри		
9	Подання кваліфікаційної роботи в ЕК		

Дата видачі завдання 22.04.2023 р.

Студент _____ Бобков М. В.
(підпис)

Керівник роботи _____ ст. викл. Гурін Д.В.
(підпис) (посада, прізвище, ініціали)

Я, як студент ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав(ла) і не одержував(ла) недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

12 червня 2024 р.  Бобков М. В.

РЕФЕРАТ

Пояснювальна записка: 66 с., 8 табл., 41 рис., 2 дод., 21 джерело.

РОБОТОТЕХНІКА, МІКРОКОНТРОЛЕРИ, ROBOT OPERATION SYSTEM, ESP-IDF, КОМУНІКАЦІЯ, ROSSERIAL.

Мета роботи – розробка розширення функціоналу програмного забезпечення rosserial.

Об’єкт розробки – програмне забезпечення rosserial для інтеграції мікроконтролера до системи на основі ROS.

Предмет розробки – програмне забезпечення для розширення функціоналу.

Для досягнення мети було проведено аналіз причин та актуальності використання фреймворку ROS, проаналізовано проблему інтеграції мікроконтролера до платформи на базі ROS, проаналізовано ефективність використання мікроконтролеру ESP32, обрано апаратно-програмну частину проекту, розроблено алгоритм роботи мікроконтролеру, розроблено схему підключення макету та програмний код для реалізації роботи алгоритму, проведено експериментальні дослідження на правильність виконання отриманої реалізації, оформлено роботу відповідно до стандартів та рекомендацій.

ABSTRACT

The explanatory note contains: 66 p., 8 tables, 41 drawing, 2 pp., 21 source.

ROBOTICS, MICROCONTROLLERS, ROBOT OPERATION SYSTEM, ESP-IDF, COMMUNICATION, ROSSERIAL.

The aim of the work is to develop an additional functionality to software rosserial.

The object of development is a software rosserial, which is used for an integration of microcontroller to ROS-based platform.

The subject of development is a software for an additional functionality.

To achieve this goal, the reasons and relevance of the ROS framework were analyzed, as well as the problem of integrating the microcontroller into the ROS-based platform and the effectiveness of using the ESP32 microcontroller. Also, the hardware and software part of the project were selected, neither as were developed: the algorithm of the microcontroller; the layout connection diagram and program code to implement the algorithm, experimental studies on the correctness of the implementation were made. In the end the work were formalized in accordance with the standards and recommendations.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕННЬ

МК – мікроконтролер;

API – Application Program Interface;

ESP-IDF – Espressif IoT development Framework;

ROS – Robot Operation System;

UART – Universal Asynchronious Receiver-Tranciever;

USB – Universal Serial Bus.

ВСТУП

В наш час роботи є невід'ємною частиною життя кожної людини. Вони впливають на наше життя майже у кожній сфері – починаючи з розумного будинку та закінчуючи індустріальними роботами, що виконують найбільш небезпечні для людини дії на підприємствах. Наразі у сфері робототехніки дуже великої популярності набув фреймворк ROS [1,2], що дозволяє швидко розробити роботизовану систему завдяки використанню вже розробленим компонентам, що легко інтегруються в єдину систему.

Існує дуже багато прикладів використання ROS, як для комерційного використання –наприклад, систем моніторингу, систем життєзабезпечення [3], так і для навчальних цілей [4] або ж наукових досліджень – створення гуманоїдних роботів [5] та ін. Для перегляду повного списку проектів на базі фреймворк існує відповідний сайт [6] з публікаціями.

Так як для роботи даного фреймворку необхідна операційна система не-реального часу [7,8], то для інтегрування мікроконтролерів у єдину систему разом із ROS необхідно використовувати спеціальні програми, що служать мостом для обміну інформації між ними. Наприклад, найбільш популярними є програмні засоби rosserial [9] та microros [10].

За допомогою цих програмних засобів вже було розроблено багато проектів. Наприклад, роботизована рука [11], бездротова система телеуправління [12], чотириногий робот-платформа HyperDog [13].

Ця кваліфікаційна робота спрямована на покращення взаємодії між фреймворком ROS та програмного засобу для інтеграції мікроконтролерів до нього, зокрема додавання функціоналу мікроконтролеру ESP32 на базі фреймворку esp-idf до роботи разом з програмним засобом rosserial.

Можливі сфери застосування: будь-яка роботизована система, що бере за основу гібрид роботи платформи на базі ROS, а також мікроконтролер ESP32.

Мета роботи – розробка розширення функціоналу програмного забезпечення rosserial.

Об’єкт розробки – програмне забезпечення rosserial для інтеграції мікроконтролера до системи на основі ROS.

Предмет розробки – програмне забезпечення для розширення функціоналу.

Для виконання мети роботи необхідно:

– провести аналіз причини використання фреймворку ROS;

– провести аналіз проблеми інтеграції мікроконтролера до платформи на основі ROS та можливі програмні засоби для вирішення цієї задачі;

– провести аналіз вибору програмного засобу;

– вибір апаратної частини;

– провести аналіз порівняння використання мікроконтролеру ESP32 на базі фреймворку EPS-IDF та Arduino фреймворку;

– розробити алгоритм роботи мікроконтролеру ESP32 разом із програмним засобом для інтеграції до платформи;

– розробити схему підключення макету;

– розробити програмний код для реалізації алгоритму;

– провести ескпериментальні дослідження на правильність виконання отриманої реалізації алгоритму;

– оформити кваліфікаційну роботу згідно ДСТУ 3008:2015 [14], а також з методичними вказівками з підготовки й оформлення кваліфікаційної роботи здобувачами першого (бакалаврського) рівня вищої освіти спеціальності 151 Автоматизація та комп’ютерно-інтегровані технології освітньої програми «Автоматизація та комп’ютерно-інтегровані технології» [15], використовуючи навчальний посібник з дипломного проекту [16] та методичні вказівки [17].

Результати дослідження надані та опубліковані у [18] .

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз використання фреймворку ROS в сучасних роботах

Robot Operating System (ROS) – це фреймворк для розробки систем робота з відкритим вихідним кодом. Він позиціонує себе як мета-операційна система через факт того, що він надає послуги, які очікуються від операційної системи, зокрема абстрагування від апаратного забезпечення, низькорівневе керування пристроями, реалізацію загальновживаної функціональності, передачу повідомлень між процесами та керування компонентами. Фреймворк також надає інструменти та бібліотеки для отримання, створення, написання та запуску коду на декількох комп'ютерах.

ROS є унікальним фреймворком, який неможливо порівняти з іншими платформами для розробки програмного коду для роботів. Його нестандартність зумовлена його метою – розробити програмний код для роботів, який буде універсальним для кожного з них, через що не потрібно буде створювати один й той самий функціонал по декілька разів. Тому ROS вважається розподіленим фреймворком процесів (нодів), що дозволяє індивідуально розробляти виконувані файли та вільно об'єднувати їх під час виконання. Ці процеси можуть бути згруповані у пакети та стеки, які можна легко поширювати та ділитися ними. ROS також підтримує об'єднану систему репозиторіїв коду, що також дозволяє поширювати спільну роботу. Такий дизайн, від рівня файлової системи до рівня спільноти користувачів, дозволяє приймати незалежні рішення щодо розробки та впровадження, але всі вони можуть бути об'єднані за допомогою інструментів інфраструктури ROS.

1.1.1 Структура фреймворку ROS. Ієрархія файлової системи

Основою ідеї файлової системи в фреймворку є використання пакетів.

Пакет – основна одиниця для організації програмного коду, що легко інтегрується до різних проектів. Вона може містити додаткову логіку роботи у вигляді ноду, необхідну для іншого пакету бібліотеку, базу даних, конфігурацію роботу та інше. Базова структура фреймворку теж розподілена на пакети.

Існують також мета-пакети, які є спецальним типом пакетів, що містять в собі лише посилання на группу інших пакетів. Таким чином за допомогою одного мета-пакету можна одразу завантажити декілька пакетів разом.

Всю інформацію про пакет можна знайти всередні нього у маніфесті (файл package.xml) – назва, версія, опис, залежності та ін.

Всю ієрархію пакетів можна побачити на рис. 1.1.

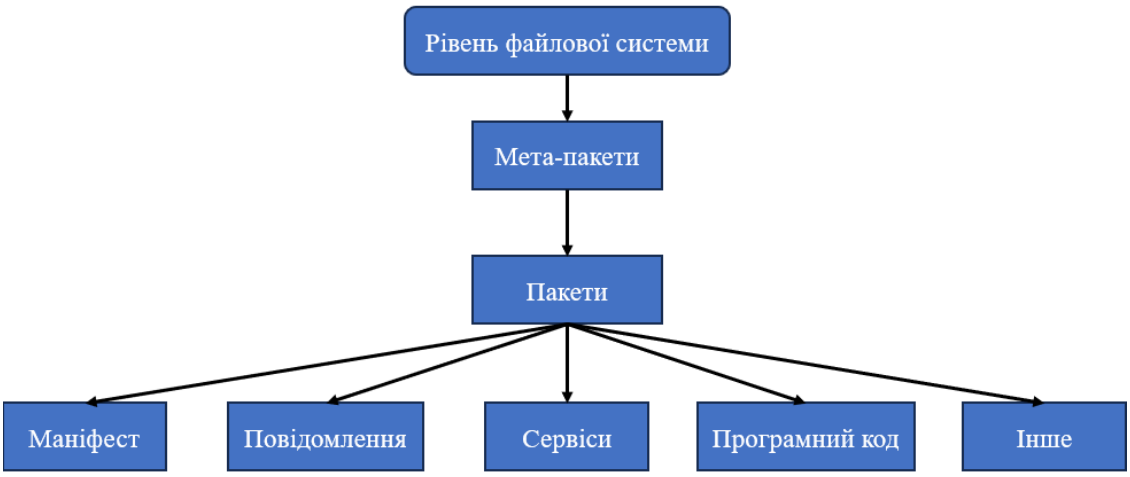


Рисунок 1.1 – Структура файлової системи в ROS

Як показано на рис. 1.1, до структури кожного пакету відносяться:

– маніфест;

– повідомлення;

– сервіси;

– програмний код;

– інше.

1.1.2 Маніфест пакету

До маніфесту пакету відноситься файл формату XML та з відповідною назвою package.xml. Цей файл визначає властивості пакету, такі як його назва, номери версій, автори, супровідники та залежності від інших пакетів.

Правильне використання цього файлу грає велику роль, адже в ньому вказані всі залежності пакету від інших пакетів. Таким чином при першому використанні, система зможе автоматично під'єднати всі вказані залежності, якщо вони не були заздалегідь налаштованні.

Структура файлу маніфеста доволі проста та складається з мінімальної кількості тегів, необхідних для опису пакету. Приклад повноцінного опису пакету наведено на рис. 1.2.

```
<package format="2">
  <name>foo_core</name>
  <version>1.2.4</version>
  <description>
    This package provides foo capability.
  </description>
  <maintainer email="ivana@willowgarage.com">Ivana Bildbotz</maintainer>
  <license>BSD</license>

  <url>http://ros.org/wiki/foo_core</url>
  <author>Ivana Bildbotz</author>

  <buildtool_depend>catkin</buildtool_depend>

  <depend>roscpp</depend>
  <depend>std_msgs</depend>

  <build_depend>message_generation</build_depend>

  <exec_depend>message_runtime</exec_depend>
  <exec_depend>rospy</exec_depend>

  <test_depend>python-mock</test_depend>

  <doc_depend>doxygen</doc_depend>
</package>
```

Рисунок 1.2 – Приклад змісту файлу package.xml

На рисунку 1.2 показано, що необхідні теги можна розділити на 2 групи:

– загальні теги (обов'язкові);

– теги залежностей (необов'язкові).

До загальних тегів відносять таку інформацію, як назва пакету, його версія, опис, автор та ліцензія для випуску.

До тегів залежностей в свою чергу відносяться 6 можливих тегів:

- залежності від інших пакетів для компіляції пакету;
- залежності від використаних бібліотек;
- залежності для виконання коду в цьому пакеті;
- залежності для тестування правильності роботи пакету;
- залежності для інструкцій компіляції проекту;
- залежності для документації проекту.

На практиці найчастіше використовуються залежності від інших пакетів, від використаних бібліотек та для виконання коду.

1.1.3 Повідомлення пакету

До структури пакету також відносяться повідомлення. Вони уявляють собою прості типи даних або структури простих типів даних, які легко та швидко обробляються в фреймворку. Повідомлення служать для передачі інформації в ROS. Зазвичай інформація передається по темам та публікується в ноди. Для цього кожному ноду необхідно заздалегідь знати тип повідомлення з інформацію.

Існують базові типи повідомлень, але кожний користувач може створювати свої власні типи, що заснові на базі стандартних типів. Стандартні типи входять до пакету «std_msgs». Вони детально наведені у табл. 1.1.

Таблиця 1.1 – Стандарті типи повідомлень в фреймворку ROS

Примітивний тип	Серілізований тип	Тип у C++	Тип у Python
1	2	3	4
bool	unsigned 8-bit int	uint8_t	bool
int8	signed 8-bit int	int8_t	int

Продовження таблиці 1.1

1	2	3	4
uint8	unsigned 8-bit int	uint8_t	int
int16	signed 16-bit int	int16_t	int
int32	signed 32-bit int	int32_t	int
uint32	unsigned 32-bit int	uint32_t	int
int64	signed 64-bit int	int64_t	long int
uint64	unsigned 64-bit int	uint64_t	long int
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ascii string	std::string	str bytes
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time
duration	secs/nsecs unsigned 32-bit ints	ros::Duration	rospy.Duration

Користувач може також створювати масиви базових типів даних.

Приклад повідомлення наведено на рис. 1.3.

```
std_msgs/msg/Header header
geometry_msgs/msg/Quaternion orientation
double[9] orientation_covariance
geometry_msgs/msg/Vector3 angular_velocity
double[9] angular_velocity_covariance
geometry_msgs/msg/Vector3 linear_acceleration
double[9] linear_acceleration_covariance
```

Рисунок 1.3 – Повідомлення «sensor_msgs/Imu»

На рисунку 1.3 наведено приклад комплексного повідомлення, яке складається з базових та інших типів повідомлень: «geometry_msgs/Quaternion» та «geometry_msgs/Vector3». Ці повідомлення також в свою чергу створені з інших типів (рис. 1.4).

```
double x=0.0
double y=0.0
double z=0.0
double w=1.0
```

Рисунок 1.4 – Повідомлення «geometry_msgs/Quaternion», що входить до складу «sensor_msgs/Imu»

1.1.4 Сервіси пакету

Сервіси так само, як і теми в нодах, використовуються для обміну інформацією в фреймворку. Сервіси можуть отримувати та передавати інформацію, що отримується у вигляді повідомлень від одного нода до іншого.

Особливістю сервісів є те, що вони не передають автоматично інформацію, а чекають на виклик функції для її публікації. Зазвичай вони використовуються для отримання інформації, що не обов’язково потрібно знати системі в кожний момент часу, а тільки в конкретні проміжки.

1.1.5 Програмна реалізація пакету

При використанні фреймворку весь програмний код для роботу необхідно створювати у вигляді спеціальних нодів.

Нод – процес, який виконує основні обчислення. Зазвичай для кожного завдання або частини робота створюється окремий нод. Наприклад, один нод отримує значення з сенсору, інший контролює рух моторів, інший планує маршрут робота.

Робота платформи неможлива без використання головного ноду – Мастеру (англ. Master). Цей нод створюється першим та адмініструє створення всіх

подальших нодів та їх комунікацію. Він надає послуги іменування та реєстарції інших нодів, відстежує відправників та абонентів різних тематик.

Ноди можуть обмінюватись інформацією між собою за допомогою спеціальних повідомлень. Кожне повідомлення має свій тип – зазвичай це проста структура даних з примітивних типів даних (цілі числа, числа з плаваючою точкою, строки та ін.).

Також для обміну повідомленнями в ROS існує спецальна транспортна система – тематика або тема (з англ. Topic) повідомлення. Кожна тема має своє ім’я та підтримує тільки один тип повідомлень. Ноди можуть відправляти повідомлення до теми –тобто бути відправником, або ж отримувати дані – бути абонентом. Кожен нод може одночасно бути й відправником, й абонентом. Кожна тема може мати декілька відправників та абонентів.

Схема комунікацій між нодами за допомогою повідомлень та тем наведено на рисунку нижче:

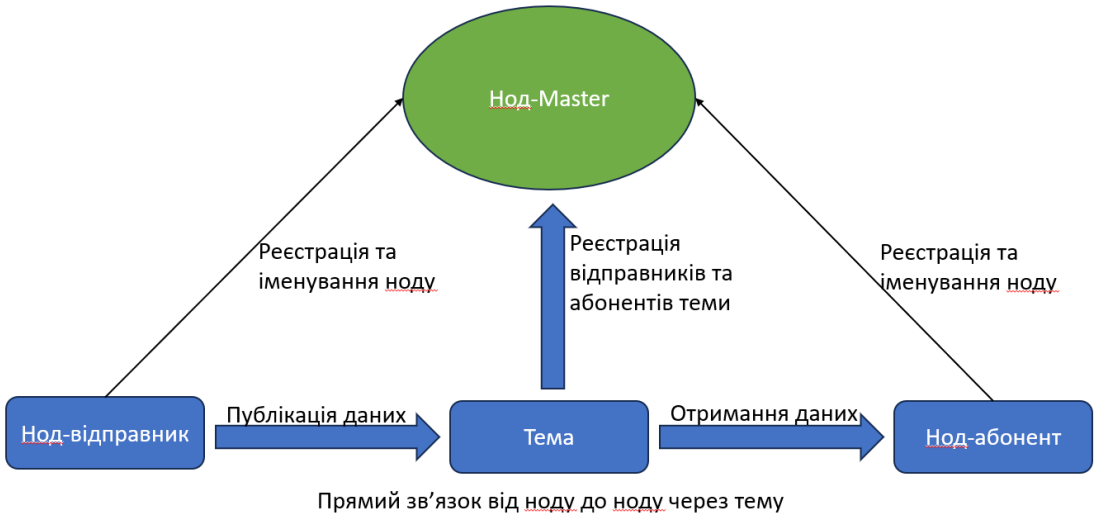


Рисунок 1.5 – Схема зв’язку між нодами та мастером

Фреймворк розподілен на різні дистрибутиви, кожен з яких має свої нововведення. Для кожного дистрибутиви розробники створюють свою версію пакету, через це деякі з пакетів могуть бути недоступними на різних версіях фреймворку.

Наразі існує дві версії фреймворку – ROS 1 та ROS 2. Кожна з версій має свої дистрибутиви (табл. 1.2, 1.3). Розробка ROS 1 на даний момент зупинена і надалі тільки нові дистрибутиви для ROS 2 будуть публікуватись раз на рік, як це раніше було з ROS 1.

Таблиця 1.2 – Сучасні дистрибутиви для ROS 1

Дистрибутив	Дата випуску	Дата кінця підтримки	Час підтримки
Noetic Ninjemys	23.05.2020	30.05.2025	5 років
Melodic Morenia	23.05.2018	30.05.2023	5 років
Lunar Loggerhead	23.05.2017	30.05.2019	2 роки
Kinetic Kame	23.05.2016	30.05.2021	5 років
Jade Turtle	23.05.2015	30.05.2017	2 роки

Таблиця 1.3 – Сучасні дистрибутиви для ROS 2

Дистрибутив	Дата випуску	Дата кінця підтримки	Час підтримки
Iron Irwini	23.05.2023	xx.11.2024	1,5 роки
Humble Hawksbill	23.05.2022	30.05.2027	5 років
Galactic Geochelone	23.05.2021	09.12.2022	1,5 роки
Foxy Fitzroy	05.06.2020	20.06.2023	3 роки

Розробники ROS 2 хочуть зробити його більш привабливим варіантом для індустріальних рішень. Адже за основу ROS 1 виступала ідея використання фреймворку насамперед для наукових досліджень в області робототехніки, через що ця версія не мала змоги підходити під стандарти в індустрії.

Наприклад, ROS не є системою реального часу і розробники ROS 2 показали на своїй конференції ROSCon у 2015р. [19], що вони хочуть зробити нову версію, що буде підходити для створення програмного забезпечення на базі операційної системи реального часу.

В ROS 2 розробники хочуть також усунути й деякі інші з критичних проблем, вирішення яких було неможливим у минулій версії. Тому зараз ROS 2 активно розвивається.

Через це для версії ROS 1 більше не буде нововведеннь та останній дистрибутив офіційно підтримується до квітня 2025 року. Незважаючи на це, більша частина розробників все ще планує залишатись і на далі на використанні оригінальної версії фреймворку через низку причин, таких як занадто низька кількість доступних пакетів для ROS 2, його ненадійність, як занадто молодого проекту та ін. Звісно, з часом ці проблеми будуть усунені, але наразі ROS 2 виглядає занадто експериментальною версією.

Так як в роботі використано тільки ROS 1, то надалі при використанні фрази ROS буде матись на увазі саме ROS 1.

Наразі для роботи ROS необхідно мати операційну систему на базі Unix. Рекомендовано використовувати Ubuntu або Mac OS X. При цьому важно розуміти, що для різних версій оперативної системи є різні версії фреймоворку. Наприклад для ROS Noetic необхідно використовувати Ubuntu 20.04.

1.2 Актуальність використання ROS

Фреймворк був створений в 2007 році й дуже швидко набув популярності у розробників та науковців.

Вже в 2010 році було опубліковано статтю [2] про експоненціальне збільшення популярності ROS та кількості доступних пакетів, створених та доступних для розробників зі всього світу.

Наразі популярніть фреймворку не зменшилась, а тільки зростає. Актуальність видно через декілька факторів.

Офіційний сайт «ROS Metrics» [https://metrics.ros.org/] збирає інформацію про регістрацію нових користувачів на форумах пов'язаних з питаннями використання фреймворку. Отримані дані можна побачити у вигляді графіку (рис. 1.6).

Також у дослідженні [https://www.therobotreport.com/2022-ros-2-metrics-report/] наведено, що кількість користувачів збільшилась на 33% в порівнянні з 2021 роком. Серед них більш, ніж 740 компаній по всьому світу. Фреймворк було завантажено за рік 501333806 разів за рік, а також завантажено пакетів на 173 терабайти даних.

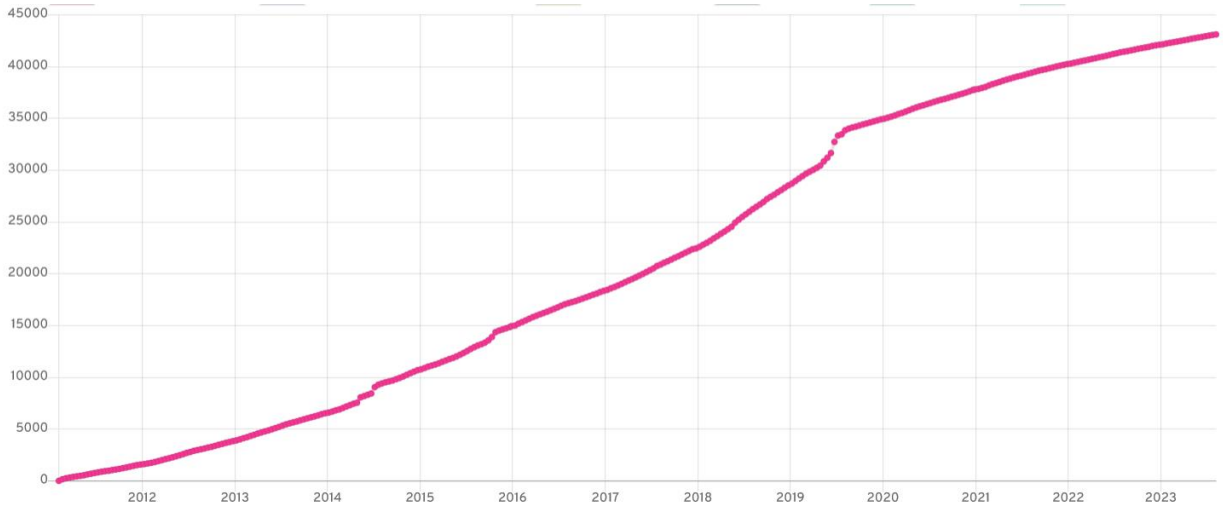


Рисунок 1.6 – Графік користувачів веб-ресурсів для питань пов'язаних з ROS

Розробники фреймворку – Open Software Robotics Foundation (OSRF) також створили механізм для оприлюднення робота або проекту на базі їх фреймворку. Всі опубліковані роботи доступні на відповідній веб-сторінці [6]. На сайті можна знайти відомості по різним типам роботів: наземні, повітряні, наводні, підводні,

маніпулятори, компоненти, тощо; по типу використаного обладнання: тип сенсору, тип головного комп'ютерного модуля, тощо; по характеристикам та ін (рис. 1.7).

Так як фреймворк найбільше використовується дослідниками, більшість проектів пов'язаних з ROS унікальні та намагаються розробити новітні рішення в роботехніці. Завдяки відкритому програмному коду, багато проектів є у відкритому доступі і кожний ентузіаст може модифікувати їх та персоналізувати.

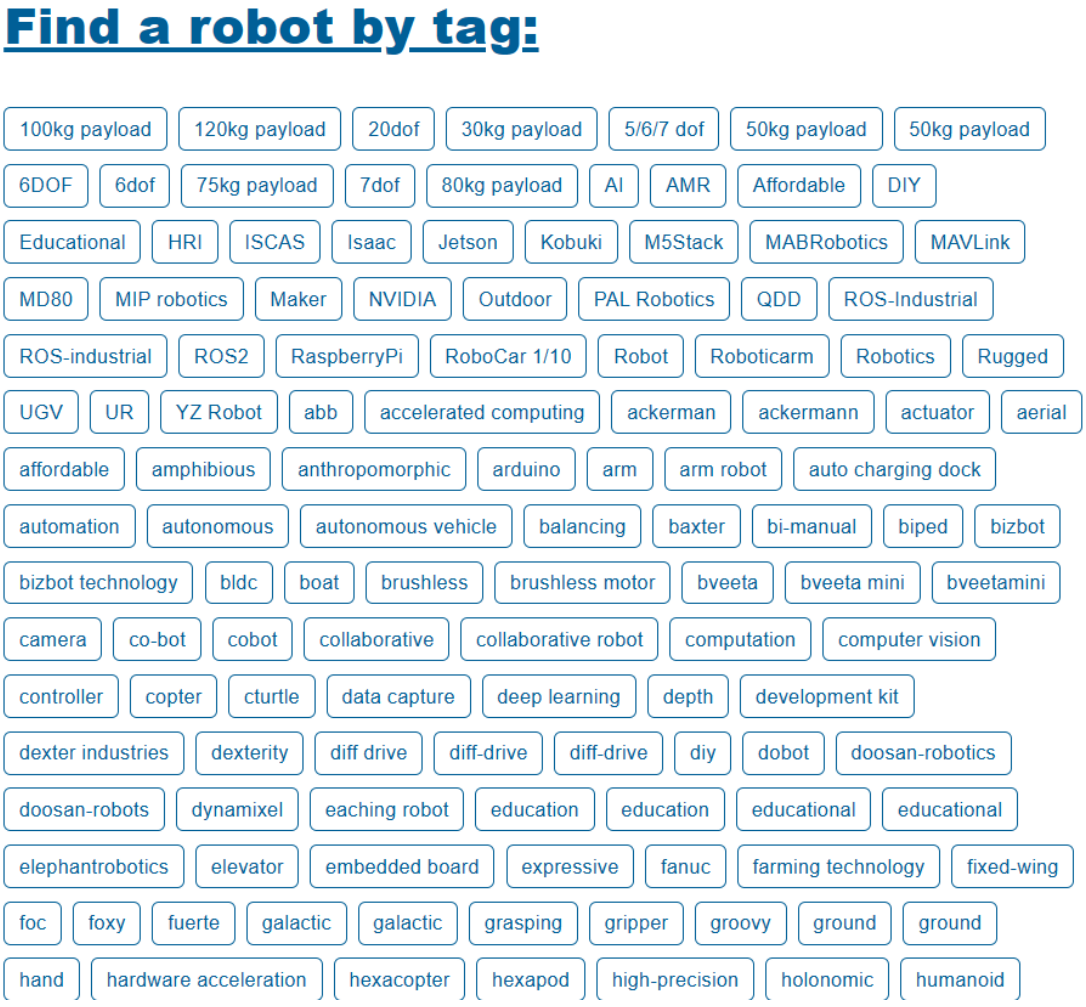


Рисунок 1.7 – Теги для пошуку роботу на сайті OSRF

1.3 Використання ROS разом з мікроконтролерами

При створенні робота велику роль відіграють також мікроконтролери.

Мікроконтролер (МК) – це об’єднання в одну систему мікропроцесору, що обробляє всі вхідні дані та передає оброблені дані на вихідні периферійні пристрої, а також оперативних та постійних запам’ятовуючих пристроїв, інтерфейсів, тактових генераторів та ін.

Необхідно приділити особливу увагу інтерфейсам мікроконтролерів, особливо на серійний (англ. SERIAL). Зазвичай в МК в ролі нього виступає Universal Asynchronous Receiver-Transmitter (UART), він пов’язаний з цифровими висновками 0 (RX) та 1 (TX), а також використовується для зв’язку з комп’ютером через USB.

Також важливо виділити параметри, що впливають на швидкодію мікроконтролера, а саме – тактову частоту процесору, розрядність шини даних та обсяг оперативної пам’яті.

Тактова частота – частота послідовності імпульсів, що задає ритм роботи внутрішніх пристроїв.

Розрядність шини даних – кількість розрядів визначає швидкість та ефективність інформаційного обміну. Зазвичай може бути 8, 16, 32 або 64 розряди.

Обсяг оперативної пам’яті дозволяє зберігати більше проміжної інформації під час роботи програм.

Мікроконтролери ідеально підходять для мобільних роботів, де важливі обмеження по простору та енергоспоживанню. Вони можуть швидко обробляти вхідні дані від сенсорів, керувати моторами для точного маніпулювання та багато іншого. Але використання роботів на базі фреймворку ROS разом із мікроконтролерами має деякі ускладнення.

Як було зазначено в 1.1, фреймворк на даний момент підтримує тільки операційні системи не-реального часу, а також версія доступного дистрибутива прив’язана до використання конкретної версії операційної системи. Через це фреймворк неможливо встановити на мікроконтролер, так як вони зазвичай використовують тільки системи реального часу, такі як FreeRTOS [20].

Але існують рішення цієї проблеми, такі як створення емулятора роботи ROS. Тобто можна використовувати зрозумілий для ROS тип комунікації – формат повідомлень, тематик, тощо.

Наразі найчастіше використовуються 2 програми – rosserial та microros [9,10].

MicroROS – це набір бібліотек, що дозволяє розробляти роботизовані додатки для розгортання на мікроконтролерах. Він має відкритий вихідний код і може бути дуже корисним для всіх робототехніків, які прагнуть інтегрувати низькорівневі мікроконтролери в робототехнічну систему. Особливістю є те, що фреймворк працює тільки з ROS 2 та може бути використаний тільки для двостороннього зв'язку між вузлами ROS 2.

ROSSerial – це протокол для обгортання стандартних серіалізованих повідомлень ROS і мультиплексування декількох тем і сервісів через серійний пристрій, такий як послідовний порт або мережевий сокет. Його особливістю також є те, що він працює тільки з однією версією фреймворку – ROS 1.

Для роботи rosserial необхідно використовувати програмний засіб на головному комп’ютері, де встановлено ROS – серверну сторону, а також необхідну версію на мікроконтролері – клієнтську.

Протокол може бути використаним на будь-якомі мікроконтролері, що має процесор з ANSI C++ компілятором, а також що має серійний порт. Необхідно тільки створити програмний код для зв’язування мікроконтролера разом із абстрактними методами із бібліотеки rosserial.

Наразі вже існують рішення для таких платформ: Arduino, ESP32 (Arduino фреймворк), STM32 (Arduino фреймворк та STM32CubeMX фреймворк), embedded linux, Teensy, Tivac та ін.

Більш детально про протокол розглянуто в науковій публікації [18]. Результатом роботи було наведення позитивних та негативних сторін використання протоколу (табл. 1.4), а також висновок щодо можливого розширення функціоналу протоколу. А саме, що для мікроконтролеру ESP32

існує версія тільки на основі фреймворку Arduino, що суттєво лімітує функціонал даного мікроконтролера.

Таблиця 1.4 – Позитивні та негативні сторони використання ROSSerial

Плюси	Мінуси
Не залежить від операційної системи і легко портується на будь-яку платформу, що підтримує мову програмування C++	Наразі порти існують лише для Arduino, Embedded Linux та Xbee. Наприклад, для використання разом з ESP32 потрібно написати власний порт
Це простий та швидкий спосіб для роботи роботів малих габаритів і невеликої ваги	Він підходить лише для тих програм, в яких одноплатний комп'ютер не може забезпечити достатню продуктивність обчислень через накладні витрати з точки зору часу обчислень і вимог до пам'яті
Користувачі можуть легко отримати функціонуючу та актуальну версію пакунка, оскільки rosserial є пакетом ROS	Кількість публікаторів та підписників обмежена до 25
Низьке споживання пам'яті у порівнянні з іншими методами	Системна частина rosserial базується на мові Python, який має меншу обчислювальну потужність у порівнянні з C++.

2 РОЗРОБКА АПАРАТНОЇ РЕАЛІЗАЦІЇ ПРОЕКТУ

2.1 Вибір мікроконтролера

За висновком підрозділу 1.3 в роботі буде використано мікроконтролер ESP32 від Espressif.

ESP32 – високоінтегрований, суміщений (Wi-Fi + Bluetooth) чіп, виконаний для рішень, що потребують мінімальних показників енергоспоживання. розроблений для електроніки і додатків інтернету речей, виконаний в мініатюрному корпусі, що вимагає для інтеграції близько 10-ти зовнішніх компонентів. Мікроконтролер з’явився на ринку восени 2015 року. Він має гарний функціонал і багатообіцяючі можливості. Поєднання в одному чипі WiFi та Bluetooth, двох процесорних ядер та багатого набору периферії може зробити ESP32 лідером у своєму сегменті. ESP32 обіцяє знову зробити революцію у світі IoT, як свого часу зробив його молодший брат ESP8266. Зовнішній вигляд мікроконтролера наведено на рис. 2.1.

Рисунок 2.1 – Зовнішній вигляд ESP-32-DevKit v1

ESP32 отримав значний приріст у продуктивності порівняно зі своїм попередником ESP8266. Обчислювальна потужність зросла вчетверо. ESP32 має два ядра, кожен з яких працює на частоті 160 МГц, таким чином одне ядро може взяти на себе завдання реального часу по роботі з графікою або керуванням двигунами, а друге може обробляти комунікаційні протоколи і в цілому відповідати за зв'язок.

Існують різні версії МК серії ESP32 від компанії Espressif:

– ESP32 – оригінальна версія;

– ESP32-S(2,3) – урізана версія, що має лише одне ядро та WIFI у версії S2, а також два ядра, але урізані можливості WIFI та BLUETOOTH у версії S3;

– ESP32-C(2,3,6) – версія на базі мікропроцесору RISC-V, має лише одне ядро, виступає як покращена версія ESP8266;

– ESP32-H(2) – версія на базі мікропроцесору RISC-V, аналог оригінальної версії.

Також мікроконтролери компанії Espressif класифікують за форматом:

– мікропроцесор;

– модуль;

– набір для розробників.

Кожний наступний формат включає в себе минулий, а також додаткові засоби (рис. 2.2).

Рисунок 2.2 – Приклад класифікації за форматом

Мікропроцесор являє собою сам чіп. Зазвичай такі замовляються у великому обсязі виробниками пристроїв. Модуль – це вже припаяний чіп, який перевірено на працездатність, зазвичай використовується в професійних проектах. Набір для розробників – спеціальні плати з модулем або мікропроцесором та великим набіром додаткових можливостей для підключення та ін.

В макеті буде використано модуль для розробників під назвою ESP-32-DevKit v1, тому надалі під МК ESP32 на увазі матиметься саме він.

Характеристики та особливості МК наведено у табл. 2.1.

Таблиця 2.1 – Технічні характеристики ESP32

Параметр	Значення
1	2
Мікропроцесор	Xtensa Dual-Core 32-bit LX6
Тактова частота мікропроцесору, МГц	80 - 240
Оперативна пам'ять, кбайт	520
Внутрішня пам'ять програм, кбайт	448
Флеш пам’ять, Мбайт	4
Бездротові комунікації	Wi-Fi: 802.11b/g/n/e/i
	Bluetooth: v4.2 BR/EDR та BLE
Периферійні інтерфейси	12-біт АЦП
	2 8-біт ЦАП
	4 SPI
	2 I2C
	3 UART
	CAN 2.0
	Ethernet MAC з підтримкою DMA
	Широтно-імпульсна модуляція

Продовження таблиці 2.1

1	2
Напруга живлення модуля, В	5
Напруга логічних рівнів, В	3,3
Споживаний струм, мА	Від 30 до 500

На рисунку 2.3 наведено розпіновку мікроконтролера.

Рисунок 2.3 – Схема виходів та входів мікроконтролера

2.2 Вибір фреймворку при розробці програмного забезпечення для ESP32

Як для розробки будь-якого програмного засобу, так й для розробки програмного забезпечення для мікроконтролера необхідно використовувати засоби для компіляції та збірки проекту, такі як Toolchain, Makefiles, CMake, Ninja, тощо.

2.2.1 Фреймворк Espressif IoT Development Framework

Для свого мікроконтролера ESP32 компанія Espressif рекомендує [21] використовувати схему, наведену на рис. 2.4.

Для контролю периферії МК Espressif створила свій фреймворк: Espressif IoT Development Framework (ESP-IDF).

ESP-IDF надає базові апаратні та програмні ресурси, які допомагають розробникам реалізувати свої ідеї з використанням обладнання серії ESP32. Він складається з декількох компонентів, де ці компоненти або містять код, спеціально написаний для мікросхем ESP, або містять бібліотеку сторонніх розробників (тобто сторонні компоненти). У деяких випадках сторонні компоненти містять «специфічну для ESP-IDF» обгортку, щоб забезпечити інтерфейс, який або простіший, або краще інтегрований з іншими функціями ESP-IDF. В інших випадках сторонні компоненти представляють оригінальний Application Program Interface (API) базової бібліотеки безпосередньо.

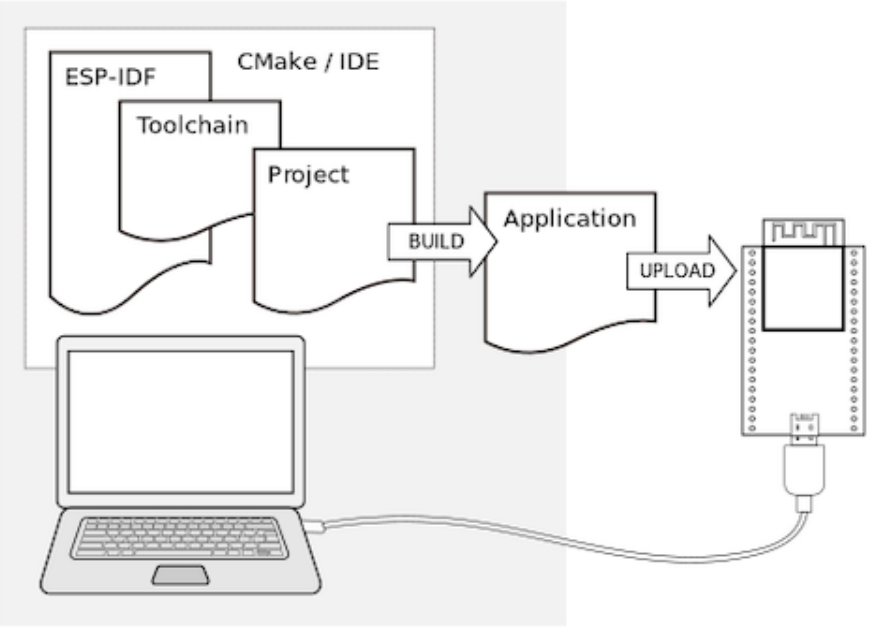


Рисунок 2.4 – Алгоритм створення програми для мікроконтролера

До базових бібліотек в ESP-IDF відносяться бібліотеки для:

– загальних системних функцій;

– bluetooth;

– мережевого зв’язку;

– аналогов-цифрових перетворювачів;

– таймери;

– General Purpose Inputs & Outputs;

– зберігання даних;

– системи повідомлень про стан роботи МК;

– економії споживання енергії.

Кожна з бібліотек складається з набору більш специфікованих бібліотек. Таким чином, в бібліотеці для роботи з мережевим зв’язком існують окремі бібліотеки для роботи з WiFi, Ethernet, HTTP сервером, MQTT сервером та ін.

Окремим випадком такого компоненту для обгортки фреймворку є Arduino Framework. Він дозволяє використовувати API, що застосовується при програмуванні мікроконтролерів серії Arduino (Uno, Nano, MKR, Micro, Mega, тощо).

2.2.2 Фреймворк Arduino Framework

Arduino Framework є більш простим у використанні, має велику базу користувачів та бібліотек, через що є дуже популярним засобом для розробки.

Для його використання необхідно встановити середу розробки для Arduino – Arduino IDE (рис. 2.5). Вона містить текстовий редактор для написання коду, область повідомлень, текстову консоль, панель інструментів з кнопками для загальних функцій та ряд меню. Він підключається до апаратного забезпечення Arduino для завантаження програм і спілкування з ними.

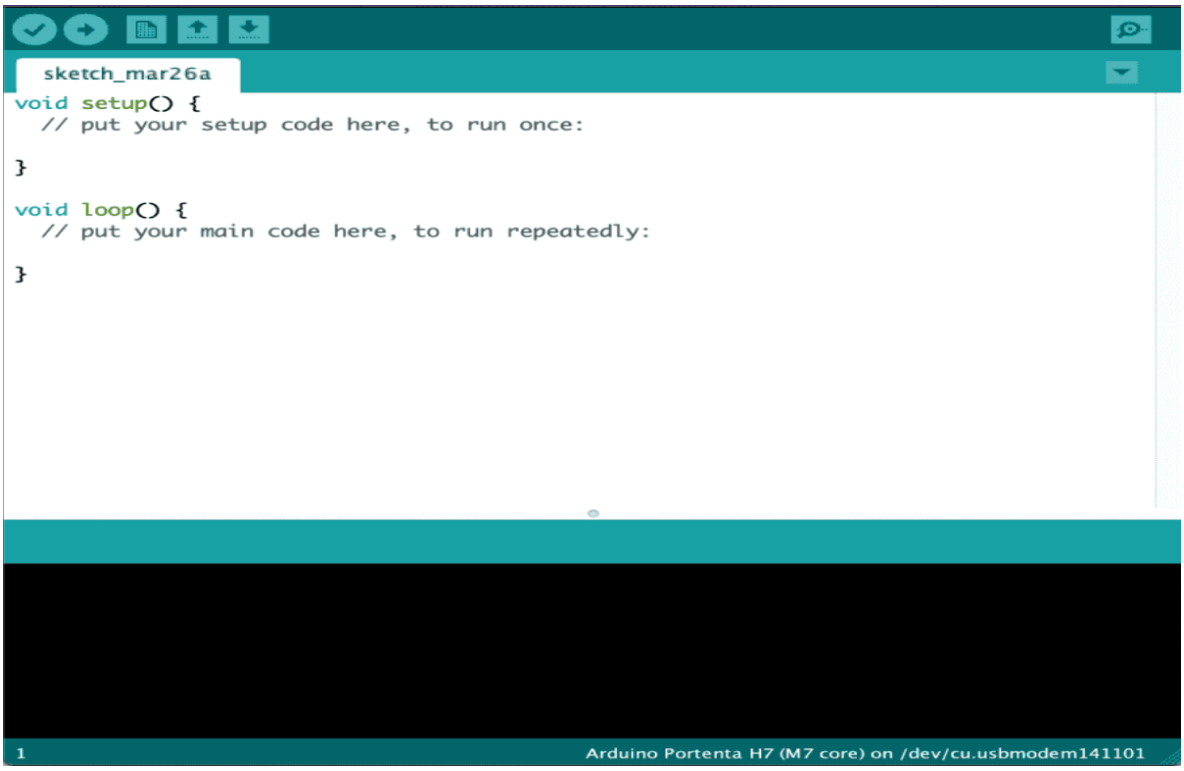


Рисунок 2.5 – Зовнішній вигляд вікна програми Arduino IDE

Для програмування мікроконтролеру серії ESP32 в Arduino IDE необхідно завантажити менеджер для плат та встановити бібліотеку esp32 (рис. 2.6).

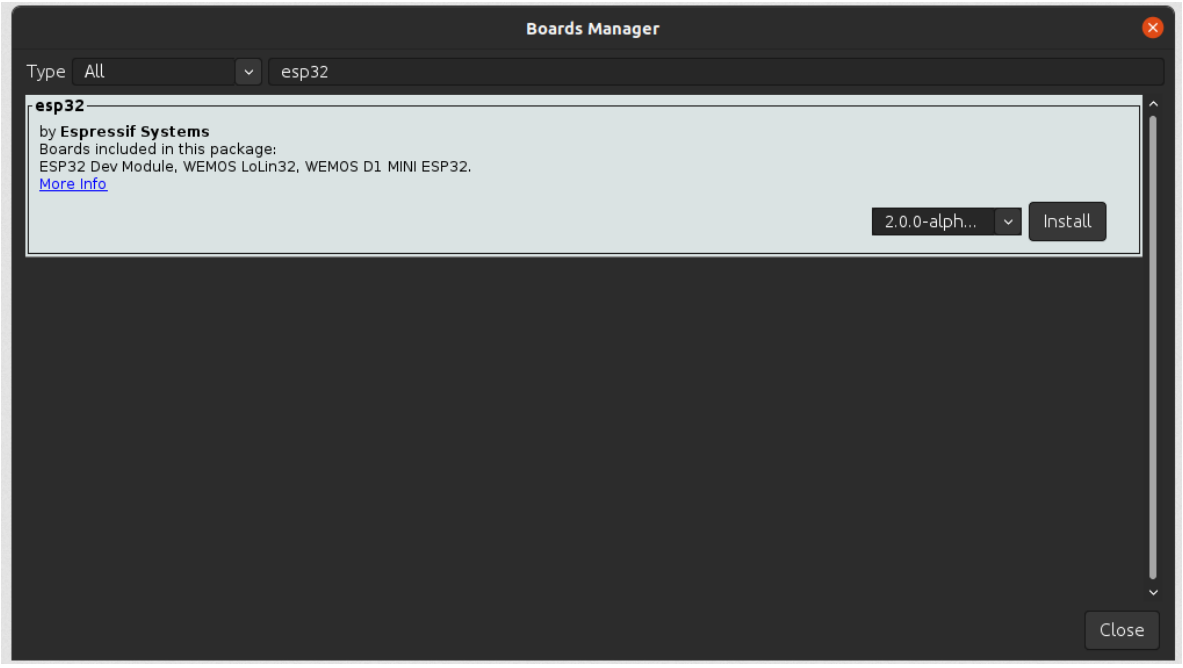


Рисунок 2.6 – Бібліотека в Arduino IDE для підтримки ESP32

2.2.3 Порівняння фреймворків

В Arduino та ESP-IDF є деякі ключові відмінності, наприклад:

– Arduino Framework орієнтований на використання мови C++, а ESP-IDF на використання C (в деяких випадках можливе також й використання C++);

– ESP-IDF використовує CMake для побудови проекту, в той час, як Arduino Framework – має свій унікальний спосіб для побудови проектів (заснований на мові програмування Java);

– ESP-IDF дозволяє дуже детально налаштовувати проект та апаратну частину за допомогою свого API у вигляді конфігураційних файлів та утиліти Kconfig. Ця утиліта грає важливу роль в створені конфігурації та її шаблонів у проекті. За допомогою команди «idf.py menuconfig» розробник може викликати у терміналі графічне вікно з меню для налаштування мікроконтролера (рис. 2.7). Всі зміни в конфігурації будуть збережені в файлі sdkconfig.defaults;

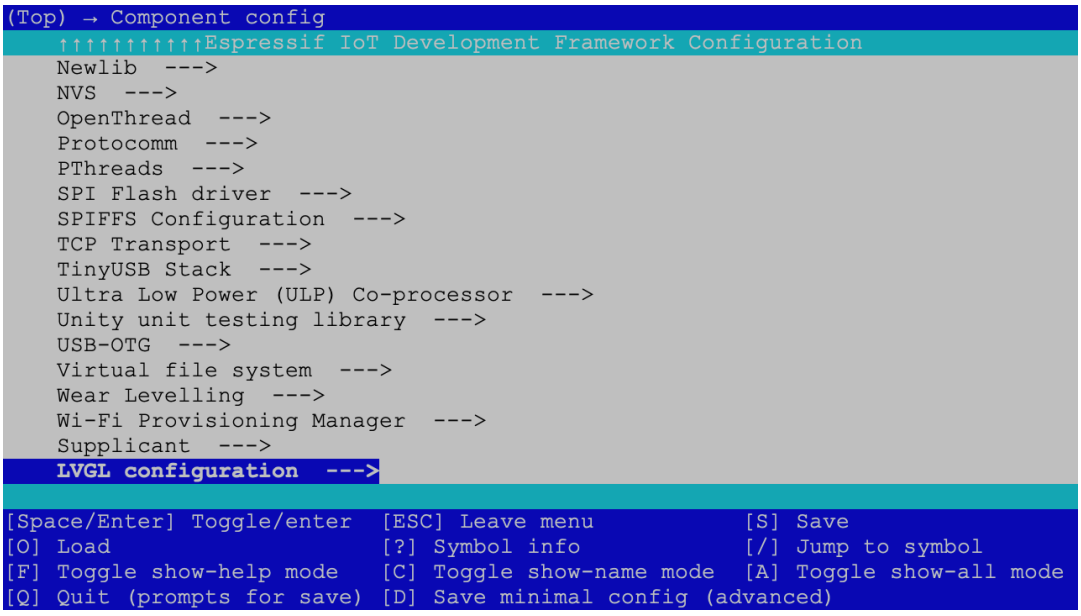


Рисунок 2.7 – Графічна репрезентація утиліти Kconfig

– наявність системи реального часу FreeRTOS у фреймворку ESP-IDF.

FreeRTOS це компактна та ефективна система реального часу, тобто система, що виконує кожну операцію у чітко відведений для цього час без зайвої

затримки головного процесору, яка має відкритий вихідний код. Вона є найбільш популярною операційною системою для вбудованих систем наразі. Використання FreeRTOS дозволяє виконання декількох завдань одночасно, FreeRTOS планує декілька завдань відповідно до їх пріоритету і полегшує взаємодію двох або більше незалежних завдань (рис. 2.8).

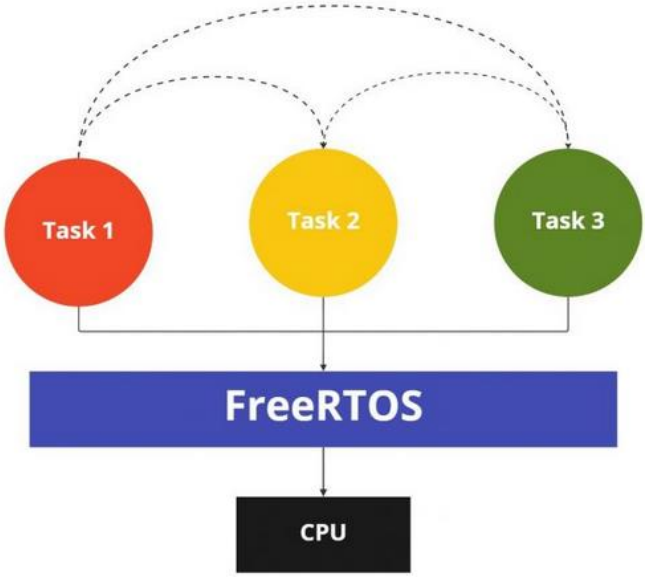


Рисунок 2.8 – Взаємодія завдань за допомогою FreeRTOS

До позитивних сторін використання FreeRTOS відносять:

– простоту використання: мінімальний набір для використання включає в себе лише 3 файли на мові програмування С;

– налічування м’ютексів та семафор: ефективний ресурс для кращого розподілу ресурсів мікроконтролеру;

– можливість управління пам’яттю: FreeRTOS ефективно використовує динамічну пам’ять мікроконтролера;

– налічування завдань та їх черг: можна легко розділяти завдання, а також керувати ними за допомогою черг.

Так як звичайна версія FreeRTOS має підтримку тільки роботи на одному ядрі, то для своїх двоядерних мікроконтролерів компанія Espressif використовує власну модифіковану версію операційної системи, що дозволяє багатозадачність одночасно на обох ядрах МК.

Для конфігурації FreeRTOS можна використовувати утиліту Kconfig.

Для порівняння використання цих фреймворків на рис. 2.9 та 2.10 наведено код, що необхідний для ініціалізації роботи серійного порту на мікроконтролері.

```
Serial.begin(115200);
```

Рисунок 2.9 – Код в Arduino Framework

```
/* Configure parameters of an UART driver,
 * communication pins and install the driver */
 uart_config_t uart_config = {
 .baud_rate = 115200,
 .data_bits = UART_DATA_8_BITS,
 .parity = UART_PARITY_DISABLE,
 .stop_bits = UART_STOP_BITS_1,
 .flow_ctrl = UART_HW_FLOWCTRL_DISABLE
 };
 uart_param_config(EX_UART_NUM, &uart_config);
 //Set UART pins (using UART0 default pins ie no changes.)
 uart_set_pin(EX_UART_NUM, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE,
UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE);
 //Install UART driver, and get the queue.
 uart_driver_install(EX_UART_NUM, BUF_SIZE * 2, BUF_SIZE * 2, 20,
&uart0_queue, 0);
```

Рисунок 2.10 – Код в ESP-IDF

Як можна побачити, ESP-IDF дозволяє налаштувати порт у набагато більш детальному методі. За допомогою Arduino Framework користувачу необхідно тільки викликати метод класу Serial та передати параметром швидкість обміну даними у серійному порті, коли в ESP-IDF необхідно виконати такі налаштування, як:

а) створити змінну конфігурації порту з відповідними параметрами;

1) швидкість обміну в бод;

2) кількість даних у пакеті;

3) перевірка парності (для перевірки цілісності та правильності отриманих даних);

4) кількість бітів для стопу передачі пакету;

5) тип реалізації порту.

б) викликати функцію для збереження конфігурації;

в) викликати функцію для вибору піну для серійного порту;

г) ініціалізувати драйвер порту.

Саме через наявність широкого спектру налаштувань, використання фреймворку ESP-IDF є більш пріоритетним при створенні професіональних та комерційних продуктів. Але з іншої сторони, вони вимагають необхідних професійних навичок у розробників програмного коду та інженерів вбудованих систем.

2.3 Розробка алгоритму роботи платформи на базі ROS та мікроконтролера ESP32 разом із rosserial

Для розробки алгоритму спочатку необхідно детермінувати учасників комунікації. При роботі алгоритму в нас повинно бути дві сторони – сервер та клієнт.

У ролі серверу виступає платформа на базі ROS. Це може бути будь-який головний обчислювальний модуль, що підпадає під умови роботи фреймворку. Зазвичай для цього використовують персональний комп’ютер, ноутбук, або ж більш спеціалізовані для роботів модулі, такі як одноплатні комп’ютери (Raspberry Pi, Nvidia JETSON Nano/Orin, тощо).

Для роботи серверу необхідно встановити та налаштувати серверну частину програмного забезпечення rosserial. Після цього, необхідно розпочати роботу фреймворку та запустити нод під назвою rosserial_python (рис. 2.11). Нод буде автоматично намагатись підключитись до клієнту. Для вибору клієнта використовуються:

– порт для підключення. За замовчуванням /dev/ttyUSB0, тобто перший підключений пристрій до інтерфейсу USB);

– швидкість обміну даними. За замовчуванням 57600 бод, тобто 57600 символів за секунду. Може бути від 9600 до 1000000 бод.

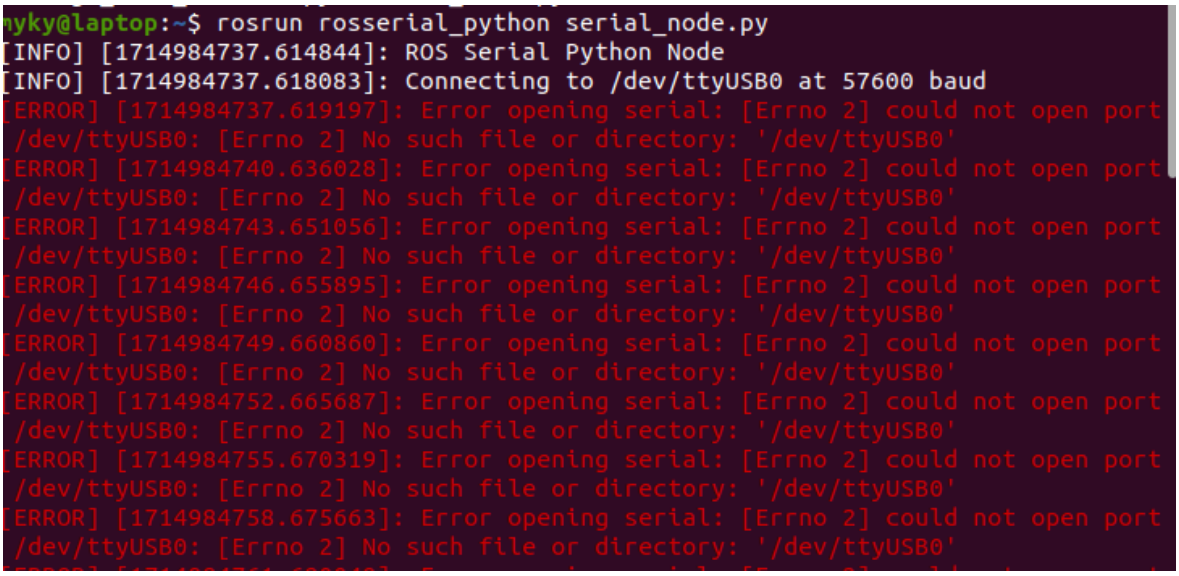


Рисунок 2.11 – Вивід у консоль під час роботи ноду rosserial_python

Як видно на рис. 2.11, так як наразі немає жодного підключеного клієнту, то нод циклічно видає попередження про це. При правильній роботі системи, нод повинен відображати інформацію про клієнта (рис. 2.12) та публікувати дані у відповідній темі.

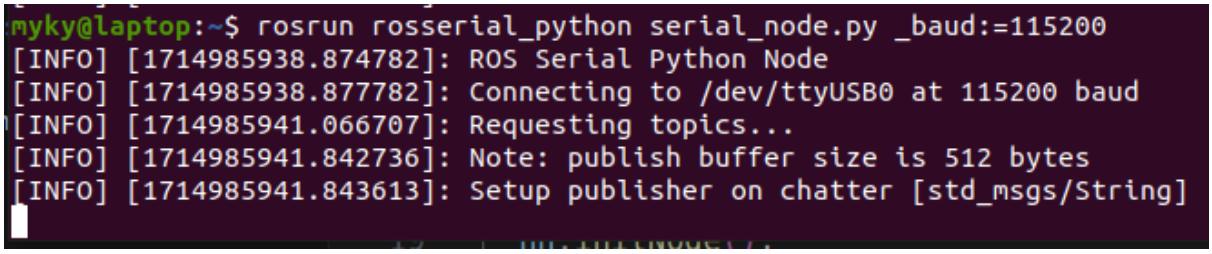


Рисунок 2.12 – Вивід у консоль під час успішної комунікації між пристроями

У ролі клієнту виступає мікроконтролер ESP32 із встановленим програмним забезпеченням на базі фреймворку ESP-IDF. Схему взаємодії між сервером та клієнтом наведено на рис. 2.13.

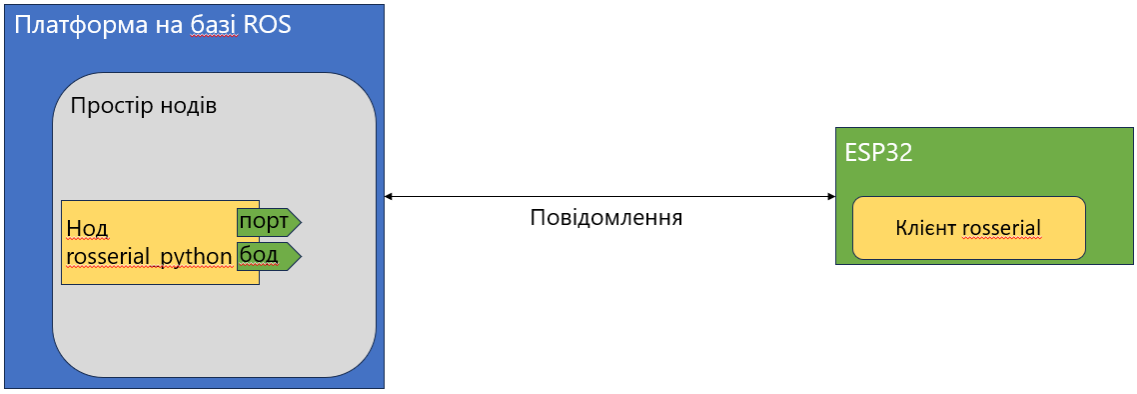
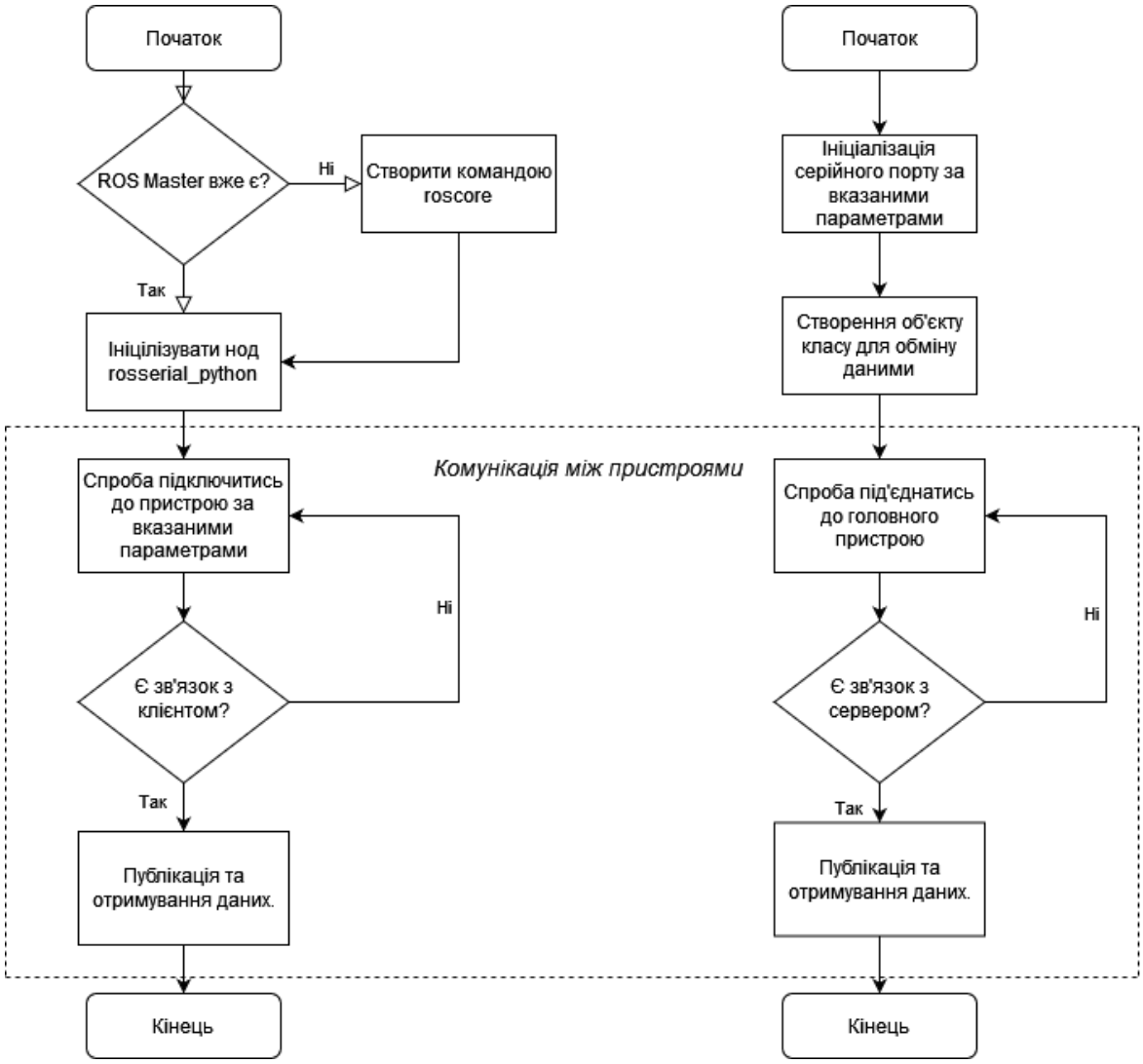


Рисунок 2.13 – Схема взаємодії між сервером та клієнтом

Розроблені алгоритми роботи для серверу та клієнту наведено відповідно на рис. 2.14.

На стороні серверу (рис. 2.14, а) необхідно спочатку створити Master-нод, це необхідно через структуру та принцип роботи фреймворку ROS. Без цього неможливо запускати будь-які інші ноди. Зазвичай це виконується за допомогою команди roscore або roslaunch. Після цього необхідно розпочати роботу ноду серверної частини, тобто rosserial_python, та вказати порт та швидкість обміну. При успішному під'єднанні до МК, сервер почне отримувати та передавати відповідні дані. Обмін інформацією буде циклічно відбуватись, поки жодна із сторін не закінчить його. Для припинення обміну необхідно вимкнути нод серверу.

Також, на рис. 2.14, б видно, що алгоритм роботи на стороні МК доволі схожий на серверну частину. Спочатку за допомогою функцій фреймворку ESP-IDF налаштовується використання серійного порту мікроконтролера з відповідними параметрами. Після цього створюється головний об'єкт класу, що є рівнем з'єднання абстракцій фреймворку ROS разом із реальними методами та функціями для впливу на мікроконтролер.



а) б)

а) серверна частина;

б) клієнтська частина

Рисунок 2.14 – Розроблені алгоритми роботи та їх зв’язок

Створений об’єкт розпочинає комунікацію між пристроями. Таким чином, спочатку перевіряються налаштування зв’язку, такі як, наприклад, відповідність версій та типу протоколу та ін. При успішній перевірці та наявності зв’язку мікроконтролер починає отримувати та передавати дані у зрозумілому для платформи вигляді – за допомогою тем та повідомлень. Обмін інформацією відбувається також циклічно й не буде припинений, якщо це не реалізовано в подальшій логікі роботи.

Для обміну інформацією між об’єктами використовується протокол, який також використано в комунікації самого ROS – TCP та XMLRPC. TCP використовується для обміну інформацією у темах, а XMLRPC для зв’язку з нодом-мастером. Процес ініціалізації обміну даними наступний:

а) створюється нод-публікатор чи нод-абонент;

б) нод інформує нод-мастер про себе;

в) нод-мастер прив’язує нод до необхідної теми;

г) нод-публікатор чи нод-абонент починають безпосередню комунікацію без ноду-мастеру.

Структура повідомлення для обміну даними за допомогою протоколу TCP наведено на рис. 2.15.

Рисунок 2.15 – Структура повідомлення в ROS та ROSSerial

2.4 Схема підключення макету

Підключення макету наведено на рис. 2.16. Схема має вигляд підключення мікроконтролеру за допомогою USB-інтерфейсу мікроконтролеру до USB-інтерфейсу ноутбуку. За допомогою цього мікроконтролер має змогу передавати дані, а також живитись від ноутбуку.

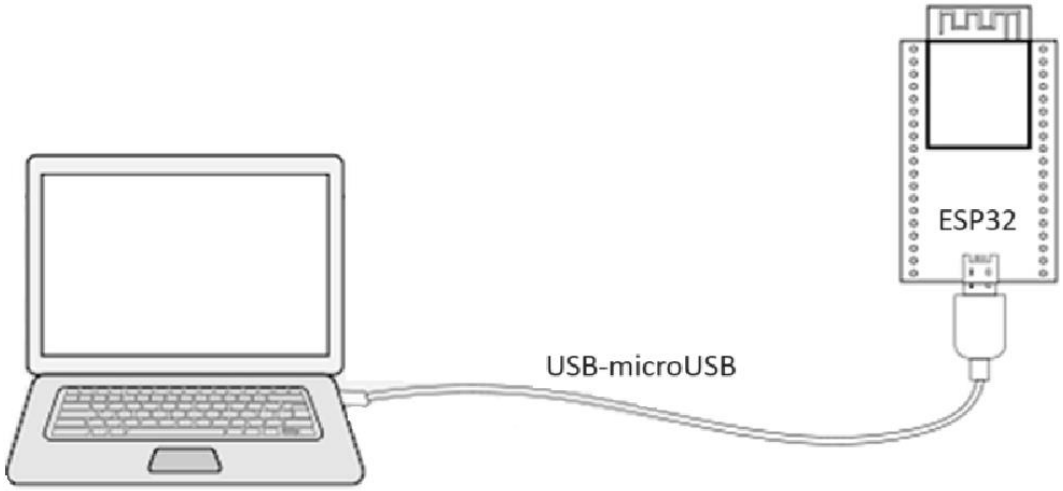


Рисунок 2.16 – Схема підключення МК до ноутбуку

3 РОЗРОБКА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ПРОЕКТУ

3.1 Розробка програмної реалізації проекту

Як наведено у підрозділі 1.3, для роботи програмного засобу rosserial необхідно створити рівень зв’язку між rosserial та функціями нижнього рівню для контролю мікроконтролеру. Для досягнення цієї мети необхідно:

– створити інтерфейс (клас ESP32Hardware, файл ESP32Hardware.h);

– створити ініціалізатор нодів для системи ROS (файл ros.h) та ініціалізувати там створений інтерфейс.

3.1.1 Реалізація інтерфейсу між rosserial та нижнім рівнем управління МК

До інтерфейсу між мікроконтролером та бібліотекою rosserial відносяться:

– функція ініціалізації класу;

– функція зчитування даних з серійного порту;

– функція запису даних до серійного порту;

– змінна для отримання актуального часу системи у мілісекундах.

В функції ініціалізації класу викликається функція для ініціалізації протоколу, в залежності від актуальної конфігурації. На макетній платі для серійної комунікації використано модуль CP2102 для конвертації даних від USB до UART. Також бібліотека rosserial має підтримку обміну даними за допомогою TCP.

Отже під час ініціалізації класу буде викликатись функція для початку роботи із протоколом UART або TCP, в залежності від конфігурації.

Конфігурація здійснена за допомогою макросу, а також, як було зазначено у підрозділі 2.2, за допомогою класичного для ESP-IDF файлу конфігурації Kconfig (рис. 3.1).

Окрім вибору інтерфейсу, так само можна налаштувати окремі параметри кожного з них (рис. 3.2). Наприклад піни, що використовуються для отримання та передачі даних, швидкість обміну даних, IP адреса та порт для бездротової комунікації, тощо.

```
menu "rosserial"

config ROSSERIAL_OVER_WIFI
    bool "rosserial over WiFi using TCP"
    default n
    help
        Use rosserial over WiFi using TCP

config ROSSERVER_AP
    string "WiFi SSID"
    default "myssid"
    depends on ROSSERIAL_OVER_WIFI
    help
        SSID (Access point name) of WiFi to connect

config ROSSERVER_PASS
    string "WiFi Password"
    default "mypass"
    depends on ROSSERIAL_OVER_WIFI
    help
        Password of the WiFi
```

Рисунок 3.1 – Конфігурація за допомогою файлу Kconfig

```
#define ROS_SERVER_IP       CONFIG_ROSSERVER_IP
#define ROS_SERVER_PORT     CONFIG_ROSSERVER_PORT

#define UART_PORT           UART_NUM_0
#define UART_TX_PIN         GPIO_NUM_1
#define UART_RX_PIN         GPIO_NUM_3
```

Рисунок 3.2 – Конфігурація за допомогою макросів

Для зчитування даних в залежності від конфігурації буде використовуватись функція зчитування байтів з порту UART або ж TCP. Отримані дані зберігаються в буфері (рис. 3.3).

Також наявна проста перевірка на правильність отримання даних (рис. 3.3). Якщо буфер з отриманими даними пустий, то функція повертає код помилки.

Для запису даних буде аналогічно використано функцію запису байтів до UART або WIFI інтерфейсу (рис. 3.3).

Для отримання часу буде також використано функцію з бібліотеки ESP-IDF для отримання поточного часу. Так як функція повертає час в секундах, то отримане значення необхідно поділити на 1000 для отримання мілісекунд. (рис. 3.3).

```
class ESP32Hardware
        int read()
        {
            int read_len;
#ifdef CONFIG_ROSSERIAL_OVER_WIFI
            read_len = ros_tcp_read(rx_buf, 1);
#else
            read_len = uart_read_bytes(UART_PORT, (uint8_t *)rx_buf, 1, 0);
#endif
            if (read_len == 1) {
                return rx_buf[0];
            } else {
                return -1;
            }
        }

        // write data to the connection to ROS
        void write(uint8_t* data, int length)
        {
#ifdef CONFIG_ROSSERIAL_OVER_WIFI
            ros_tcp_send(data, length);
#else
            uart_write_bytes(UART_PORT, (char*)data, (size_t)length);
#endif
        }

        // returns milliseconds since start of program
        unsigned long time()
        {
            return esp_timer_get_time() / 1000;
        }
};
#endif
```

Рисунок 3.3 – Частина реалізації інтерфейсу між МК та rosserial

Програмний код наведено у додатку А.

3.1.2 Реалізація ініціалізатору нодів для системи ROS

Для створення цього програмного забезпечення у rosserial є шаблон реалізації у вигляді файлу ros.h (рис. 3.4).

У цьому файлі підключаються бібліотека node_handle.h від оригінального фреймворку ROS, а також створений у попередньому пункті файл ESP32Hardware.h для інтерфейсу.

При успішному підключенні файлів необхідно тільки створити об’єкт нового класу, в нашому випадку – ESP32Hardware. Тепер при виклику класу NodeHandle разом із ним буде запускатись також й клас, який було створено – ESP32Hardware.

```
#include "ros/node_handle.h"
#include "ESP32Hardware.h"

namespace ros
{
  typedef NodeHandle_<ESP32Hardware, 25, 25, 1024, 1024> NodeHandle; // default 25, 25, 512, 512
}

#endif
```

Рисунок 3.4 – Частина реалізації для ініціалізації ноду

Як видно на рис. 3.4, при створенні об’єкту необхідно крім назви класу також ввести декілька додаткових аргументів (рис. 3.4). Кожний з них є параметром, що контролює ліміти використання rosserial на мікроконтролері:

– MAX_PUBLISHERS репрезентує максимальну кількість нодів-відправників;

– MAX_SUBSCRIBERS репрезентує максимальну кількість нодів-абонентів;

– IN_BUFFER_SIZE репрезентує розмір вхідного буферу. Він впливає на максимальний розмір повідомлень, що надсилаються під час комунікації. Значення потрібно вводити в байтах;

– OUT_BUFFER_SIZE репрезентує розмір вихідного буферу. Він впливає на максимальний розмір повідомлень, що отримуються під час комунікації. Значення потрібно вводити в байтах.

Підбір цих параметрів потрібно робити, розраховуючи кількість ресурсів МК. Наприклад, якщо використати занадто багато оперативної пам’яті, то МК буде вести себе хаотично. Якщо надсилається повідомлення, що більше за розміром ніж максимальний вказаний розмір буферу, то замість повідомлення прийде попередження або помилка.

За стандартом в rosserial рекомендується використовувати максимум 25 нодів-відправників та абонентів, так само як й 512 байт для розміру повідомлень.

Створені файли кодів дозволяють використовувати rosserial разом із МК ESP32 на базі фреймворку ESP-IDF так само, як й на інших платформах та мати зв’язок з платформою на базі ROS. А саме реалізується можливість відправляти та отримувати дані, в уніфікованому для платформи вигляді одночасно з можливістю використання функцій фреймворку та його бібліотек.

Програмний код наведено у додатку Б.

3.2 Експериментальні дослідження

Для перевірки отриманої реалізації та тестування правильності виконання алгоритму роботи необхідно провести експериментальні дослідження.

Ціль дослідження – перевірити, що МК ESP32 на базі фреймворку ESP-IDF з використанням програмного засобу rosserial та реалізованої логіки роботи може обмінюватись даними з платформою на базі ROS.

3.2.1 Макет для дослідження

Як описано у підрозділі 2.3, для роботи алгоритму необхідно мати як мінімум один сервер та одного клієнта. У ролі серверу буде виступати персональний комп’ютер з операційною системою Ubuntu 20.04 та з

налаштованою останньою версією фреймворку – ROS Noetic, де буде налаштовано серверну частину програмного засобу rosserial – rosserial_python. Серверна частина буде очікувати підключення клієнта через порт /dev/ttyUSB0 з швидкістю обміну 115200 бод.

У ролі клієнту виступає мікроконтролер ESP32 із встановленим програмним забезпеченням на базі фреймворку ESP-IDF. На МК попередньо завантажено робочу версію реалізації програмного коду, наведеного у підрозділі 3.1.

Схему роботи експериментального макету наведено на рис. 3.5.

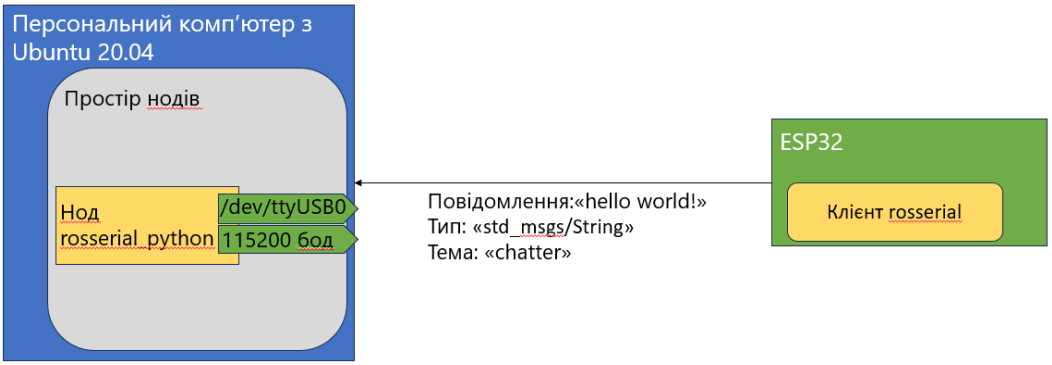


Рисунок 3.5 – Схема взаємодії між сервером та клієнтом

Фізичний вигляд макету наведено на рис. 3.6

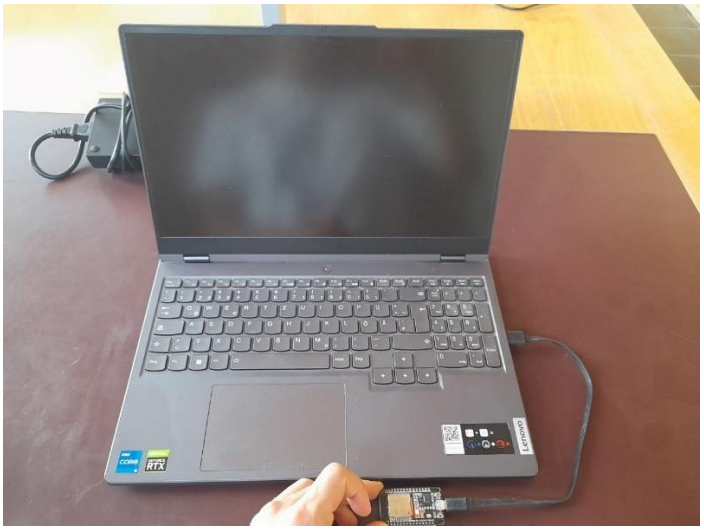


Рисунок 3.6 – Фізичний вигляд макету

3.2.2 План проведення та результат першого експерименту

До ноутбуку, відповідно до схеми підключення макету, за допомогою USB-проводу під’єднано МК.

Для спростування робочого макету, мікроконтролер буде відправляти стандартне повідомлення «hello world!» у тему «chatter», тип повідомлення – «std_msgs/String».

На персональному комп’ютері у вікні консолі повинні відображатись дані, що показуються при успішному з’єднанні (рис. 3.7), а також у темі «chatter» повинно відображувати всі опубліковані дані з мікроконтролеру.

При виконанні експерименту було отримано відповідні дані (рис. 3.7 – 3.8)

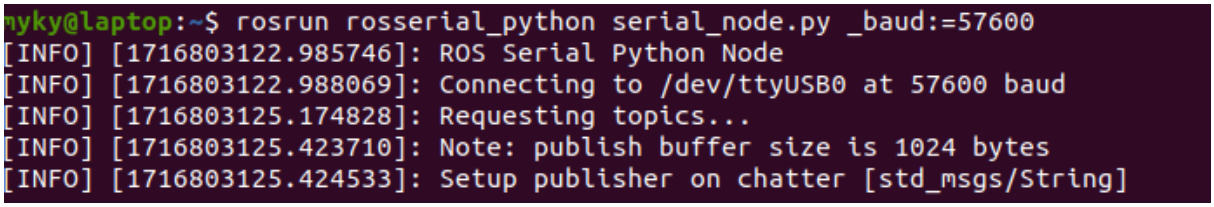


Рисунок 3.7 – Вивід у консоль про успішне підключення та реєстрацію ноду

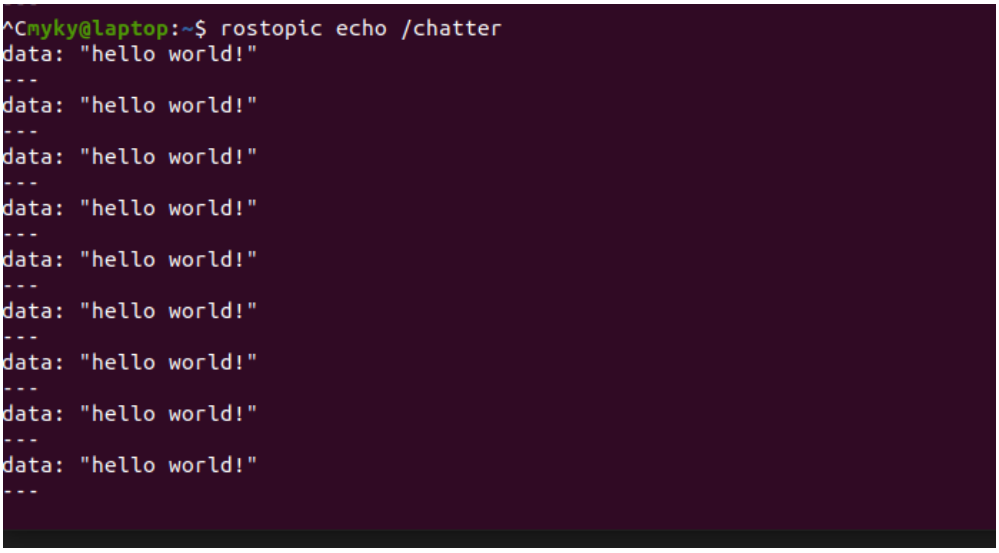


Рисунок 3.8 – Вивід у консоль отриманих даних

3.2.3 План проведення та результат другого експерименту

Для більш наглядного прикладу правильності виконання алгоритму та ефективності роботи макету, проведемо дослід з управлінням двигуном за допомогою команд з платформи на базі ROS.

Для цього модифікуємо макет з першого досліду та додамо до нього серводвигун SG-90 (рис. 3.9).

Серводвигун SG-90 – найпоширеніша модель серводвигунів для використання в робототехніці. Він дуже простий в використанні, через що має велику популярність у початківців.

Рисунок 3.9 – Зовнішній вигляд корпусу серводвигуна

Усередині корпусу знаходиться невеликий модуль керування, який під дією вхідного сигналу подає живлення відповідної полярності електродвигуну. Вхідний сигнал керування містить дані про потрібне положення валу. Для визначення поточного положення валу редуктор з'єднаний із двигуном змінного резистора. Електроніка SG90 обчислює різницю між поточним положенням

редуктора та необхідним. Модуль управління орієнтуючись на опір змінного резистора подає живлення необхідної полярності на двигун для повороту редуктора, що приводить у відповідність положення, що передається сигналом управління.

Характеристика серводвигуна наведена у табл. 3.1.

Таблиця 3.1 – Технічні дані SG90

Параметр	Значення
Швидкість без навантаження, сек / град	0,12 / 60 при живленні 4,8 В
Крутний момент, кг /см	2
Температурний діапазон, °С	0 – 50
Ширина мертвої зони, мкс	4
Робоча напруга живлення, В	3,5 – 6
Споживаний струм при русі, мА	50 – 80
Споживаний струм в утриманні, мА	5 – 10
Кут повороту, град	120
Розміри, см	3,3 х 3 х 1,3
Вага, г	9
Частота керуючого сигналу, Гц	50

Схема підключення експериментального макету наведено на рис. 3.10.

Як видно на рис. 3.10, для роботи серводвигуна необхідно підключити 3 піна – сигнальний та два для позитивиного та негативного живлення. Відповідно до табл. 3.1, для живлення мотору необхідно використовувати напругу більше 3,5 В, в макеті буде використано блок живлення на 5 В. Останній сигнальний пін під’єднується до піна 32 на ESP32. Через цей пін мікроконтролер має змогу комунікувати, тобто управляти двигуном чи отримувати інформацію про стан положення двигуна за допомогою енкодера.

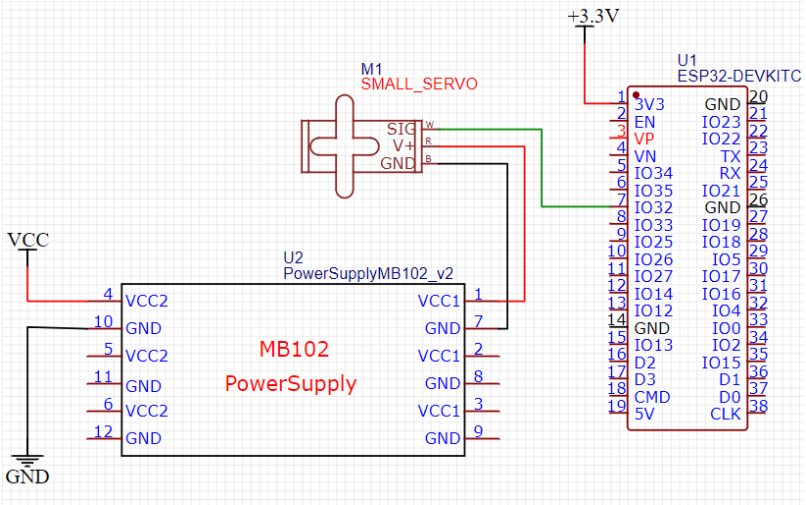


Рисунок 3.10 – Схема підключення макету

Алгоритм роботи макету наведено на рис. 3.11.

Рисунок 3.11 – Алгоритм роботи макету

Фізичний вигляд макету наведено на рис. 3.12.

Як джерело живлення в макеті використано піни мікроконтроллера Arduino Uno – 5 В та заземлення відповідно.

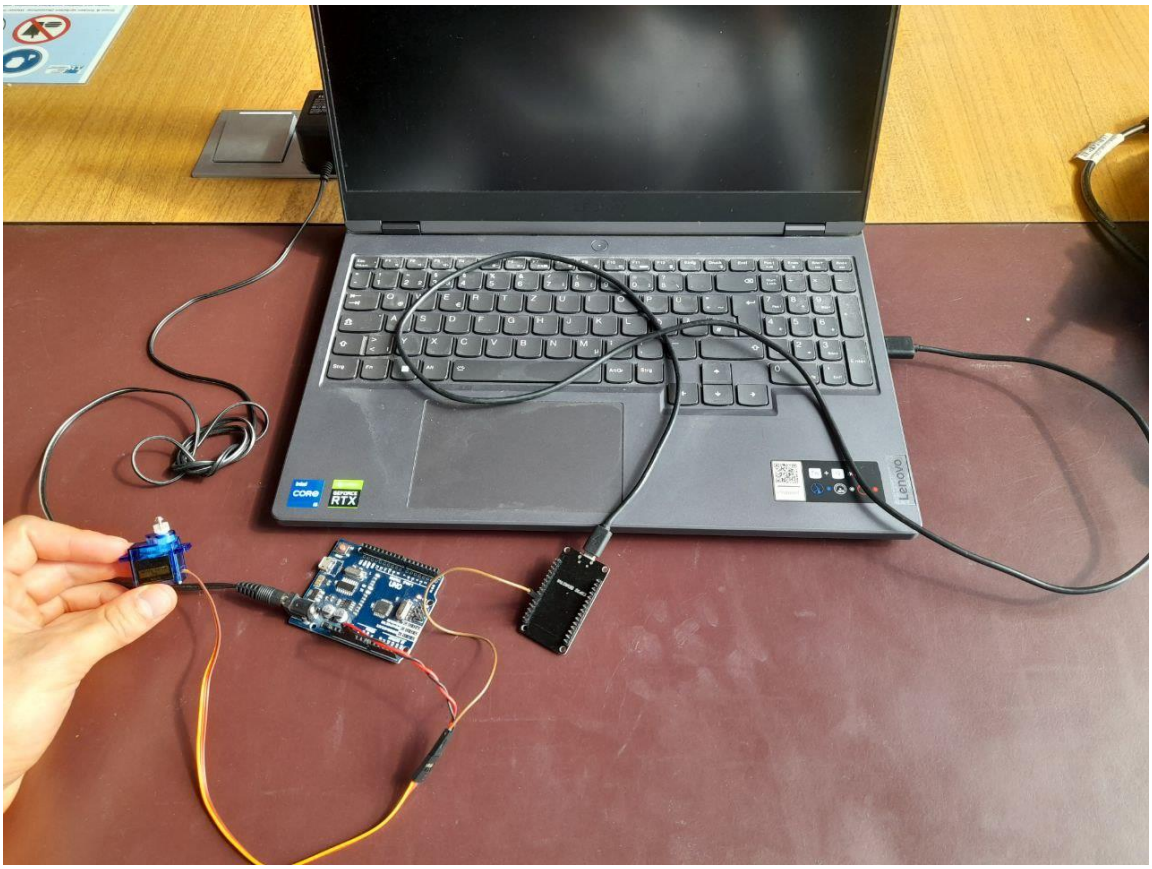


Рисунок 3.12 – Фізичний вигляд експериментального макету

В результаті експерименту було отримано відповідні дані (рис. 3.13 та 3.14)

```
myky@laptop:~$ rostopic list
/diagnostics
/motor_angle
/motor_encoder
/rosout
/rosout_agg
myky@laptop:~$
```

Рисунок 3.13 – Список тем в ROS під час роботи макету

На рисунку 3.13 наведено список тем, що можна спостерігати під час роботи макету. Теми «/motor_angle» та «/motor_encoder» комунікують з мікроконтролером. Перша тема передає необхідний кут повороту для двигуна, а друга – повератє актуальне значення куту повороту. Для успішного виконання досліду обидві теми повинні мати однакове актуальне значення кута повороту двигуна.

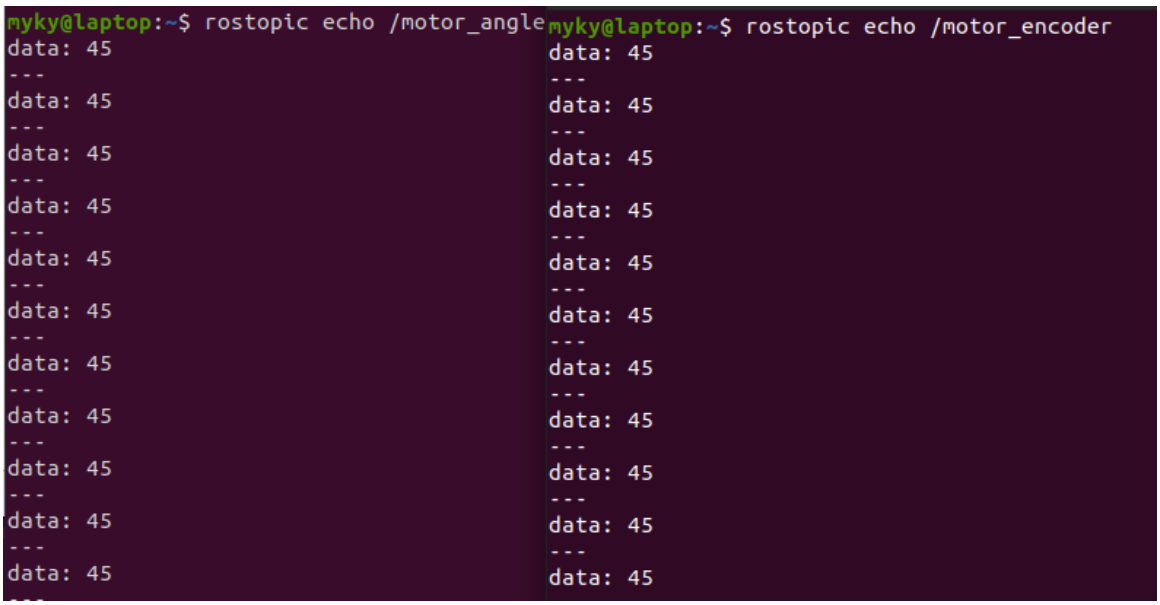


Рисунок 3.14 – Дані в темах «/motor_angle» та «/motor_encoder»

Як видно на рис. 3.14, значення в обох темах однакове, одже алгоритм працює успішно.

Тепер проведемо розрахунки стійкості системи. Для цього спочатку необхідно побудувати математичну модель системи.

Необхідні для обчислення параметри серводвигуна наведено у табл. 3.2.

Таблиця 3.2 – Параметри SG90, необхідні для розрахунків математичної моделі

Параметр	Значення
Опір обмотки, R, Ом	2
Константа крутного моменту, k_t, Н · м/А	0,01
Константа проти-ЕРС, k_b, В · с/рад	0,01
Момент інерції ротора, J, кг · м^2	0,01
Коефіцієнт в'язкого тертя, b, Н · м · с/рад	0,1

Підставимо наведені в табл. 3.2 дані до формули передавальної функції (3.1):

$$W(s) = \frac{\Omega(s)}{V(s)} = \frac{k_t}{s(Js + b)(R + k_b s)}, \tag{3.1}$$

$$W(s) = \frac{0{,}01}{s(0{,}01s + 0{,}1)(2 + 0{,}01s)}.$$

Спростимо вираз:

$$W(s) = \frac{0{,}01}{0{,}02s^2 + 0{,}201s + 0{,}2}.$$

Далі проведемо розрахунки критеріїв стікості системи.

Оскільки маємо систему другого порядку, то необхідно розрахувати алгебраїчні та частотні критерії стійкості другого порядку. А саме, необхідно:

– перевірити виконання необхідної умови стійкості;

– визначити критерій Михайлова.

Для перевірки виконання необхідної умови стійкості потрібно, щоб коефіцієнти характеристичного рівняння системи були одного знаку. Отже, складемо характеристичне рівняння системи:

$$0{,}02\lambda^2 + 0{,}201\lambda + 0{,}2 = 0\,.$$

Оскільки всі коефіцієнти одного знаку та більне нуля, то необхідна умова виконується і система стійка.

Для системи другого порядку виконання необхідної умови є достатнім для визначення стійкості системи.

Розрахуємо теперь частотні критерії.

Для того, щоб система була стійка, достатньо, щоб крива Михайлова при зміні частоти ω від 0 до $+\infty$, починаючись при $\omega = 0$ на дійсній додатній півосі, обходила тільки проти годинникової стрілки послідовно n квадрантів координатної площини, де $n = 2$, ніде не перетворюючись в нуль.

Змінимо характеристичне рівняння для побудови кривої Михайлова:

$$W(j\omega) = 0{,}02(j\omega)^2 + 0{,}201(j\omega) + 0{,}2\,,$$
$$W(j\omega) = X(\omega) + jY(\omega)\,,$$
$$X(\omega) = 0{,}2 - 0{,}02\omega^2, \omega = \pm\sqrt{10}\,,$$
$$Y(\omega) = 0{,}201\omega, \omega = 0\,.$$

Для побудови кривої розрахуємо значення дійсної та уявної частин характеристичного рівняння при зміні частоти від 0 до $+\infty$ (табл. 3.3).

Вид кривої в межах квадранта не впливає (рис. 3.15), тому вона будується приблизно. За видом кривої можна зробити висновок, що задана система стійка.

Таблиця 3.3 – Дані для побудови кривої Михайлова

ω	0	$0 < \omega < \sqrt{10}$	$\sqrt{10}$	$\sqrt{10} < \omega < \infty$	$\to \infty$
$X(\omega)$	0.2	> 0	0	< 0	$\to -\infty$
$Y(\omega)$	0	> 0	0.64	> 0	$\to \infty$

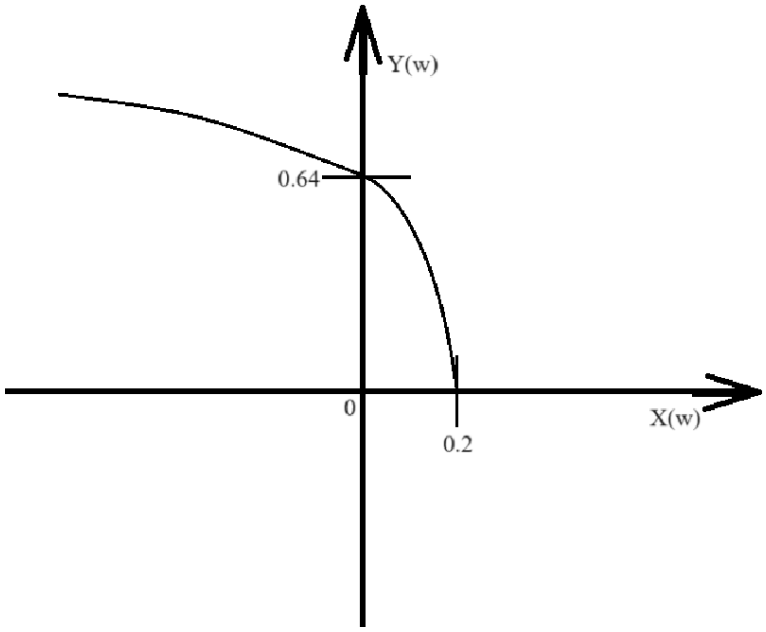


Рисунок 3.15 – Крива Михайлова для досліджуваної системи

3.2.4 План проведення та результат третього експерименту

Для повного досліду необхідно також перевірити те, що мікроконтролер та програмне забезпечення може стабільно використовуватись при максимальному навантаженні протоколу обміну.

Для цього, згідно з пунктом 3.1.2, перевірено працездатність при максимальній кількості нодів-відправників. Так як аргумент MAX_PUBLISHERS = 25, то було створено 25 нодів, кожний з яких відправляє повідомлення типу «std_msgs/String» у тему «chatter». Для спростування, кожний нод відправляє повідомлення з власною назвою. Наприклад перший нод-відправник відправляє повідомлення «publisher 1». Схему роботи експериментального макету більш детально наведено на рис. 3.16.

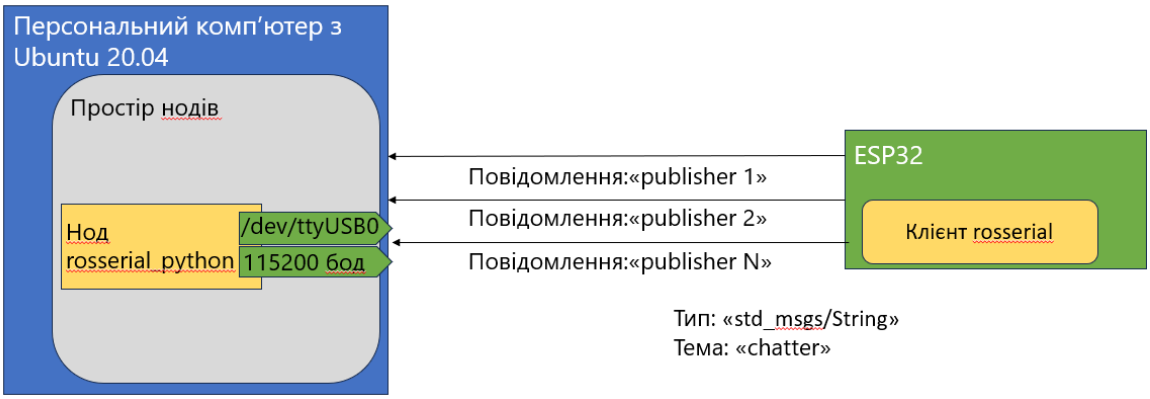


Рисунок 3.16 – Схема макету для максимального навантаження

При виконанні експерименту було отримано відповідні дані (рис. 3.17 – 3.18). Всі ноди-відправники успішно відправляють дані на персональний комп'ютер, що обробляє їх та виводить на екран консолі. Це підтверджує, що програмний засіб успішно виконує поставлену задачу та його можливо використовувати навіть при максимальному навантаженні програмних можливостей.

Для порівняння, мікроконтролер Arduino Uno (ATmega 328P), що має процесор з тактовою частотою в 16 МГц, а також 2 Кбайти оперативної пам'яті спроможній мати максимум 15 нодів-відправників та нодів-отримувачів, а також буфер розміром в 512 байт. Це майже вдвічі зменшує його потенціал використання в порівнянні з ESP32.

Завдяки використанню програмного забезпечення на базі фреймворку ESP-IDF, в розробника залишається можливість використання бібліотек фреймворку для будь-якої подальшої реалізації. Це є найбільшою перевагою використання фреймворку. Для експерименту та наглядного прикладу модифікуємо макет.

```
nyky@laptop:~$ rosrun rosserial_python serial_node.py _baud:=115200
[INFO] [1716989567.705197]: ROS Serial Python Node
[INFO] [1716989567.707419]: Connecting to /dev/ttyUSB0 at 115200 baud
[INFO] [1716989569.884923]: Requesting topics...
[INFO] [1716989569.963004]: Note: publish buffer size is 1024 bytes
[INFO] [1716989569.963923]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989569.971773]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989569.976961]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989569.984742]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989569.990412]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989569.997940]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.003947]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.011715]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.020405]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.024316]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.031286]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.037926]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.047729]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.053566]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.059393]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.067550]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.073278]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.081868]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.089636]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.095409]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.101338]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.109084]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.116868]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.123650]: Setup publisher on chatter [std_msgs/String]
[INFO] [1716989570.132937]: Setup publisher on chatter [std_msgs/String]
```

Рисунок 3.17 – Вивід у консоль про успішне підключення та реєстрацію нодів

```
nyky@laptop:~$ rostopic echo /chatter
data: "publisher 1"
---
data: "publisher 2"
---
data: "publisher 3"
---
data: "publisher 4"
---
data: "publisher 5"
---
data: "publisher 6"
---
data: "publisher 7"
---
data: "publisher 8"
---
data: "publisher 9"
---
data: "publisher 10"
---
data: "publisher 11"
---
data: "publisher 12"
---
data: "publisher 13"
---
data: "publisher 14"
---
data: "publisher 15"
---
data: "publisher 16"
---
data: "publisher 17"
---
data: "publisher 18"
---
data: "publisher 19"
---
data: "publisher 20"
---
data: "publisher 21"
---
data: "publisher 22"
---
data: "publisher 23"
---
data: "publisher 24"
---
data: "publisher 25"
---
```

Рисунок 3.18 – Вивід у консоль отриманих даних

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проведено аналіз використання фреймворку ROS, його структуру, ієрархію його файлової системи та принцип роботи з ним. Також було проаналізовано актуальність його використання в сучасному світі та майбутньому, а також наведено проблему та причини інтеграції мікроконтролеру до платформ на базі ROS. Було проаналізовано існуючі програмні засоби для вирішення проблеми інтеграції та обрано програмний засіб rosserial.

На основі аналізу програмного засобу та доступних мікроконтролерів було обрано мікроконтролер ESP32, наведено його основні технічні дані та класифікацію. Було проаналізовано можливі засоби для програмування мікроконтролера, а саме фреймворки ESP-IDF та Arduino. Для цього було наведено основні відомості про кожний з фреймворків, а також наведено їх порівняння. Після висновку використання фреймворку ESP-IDF було розроблено алгоритм роботи платформи на базі ROS та мікроконтролера, наведено інформацію щодо ролей серверу та клієнту, їх схему взаємодії, приклад повідомлення для комунікації, а також схему підключення.

Було розроблено програмну реалізацію алгоритму комунікації, а саме інтерфейс між нижнім рівнем управління мікроконтролером та rosserial, а також ініціалізатор ноду для фреймворку ROS. Програмний код наведено відповідно у додатках А та Б.

На основі отриманих даних було створено макет для експериментальних досліджень, які було успішно проведено при різних умовах використання. А саме, було проведено три експерименту: найпростішу версію макету; перевірку на працездатність при максимальному навантаженню, а також реальний приклад використання отриманої реалізації для контролю серводвигуна. Для останнього досліду було обрано використовувати серводвигун SG90, його актуальність

використання, основні технічні відомості, схему підключення до макету та алгоритм роботи наведено відповідно у підрозділі другого досліду. Також для мотору було розраховано критерії стійкості. Кожний дослід було виконано успішно, що свідчить із опису отриманих результатів.

Завдяки отриманим результатам реалізовано можливість використання мікроконтролера ESP32 разом із платформою на базі ROS за допомогою програмного засобу rosserial при підтримці офіційного фреймворку для мікроконтролеру – ESP-IDF, що значно розширює потенціал його використання, це є особливо актуально із збільшістю популярності використання ROS.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ROS [Електронний ресурс] /- Режим доступу: www / URL: https://www.ros.org/ (дата звернення 08.05.2024).

2. Steve Cousins. Exponentila Growth of ROS / Steve Cousins // IEEE Robotics & Automation Magazine. – 2011. – P.19-20.

3. Eduardo Pinto. A Health and Usage Monitoring System for ROS-based Service Robots / Eduardo Pinto, Pedro Deusdado, Francisco Marques, Andre Lourenco, Ricardo Mendonca, Pedro Santana, Luis Flores, Jose Barata // Mechatronics and its Applications (ISMA). – 2015.

4. Courses [Електронний ресурс] /- Режим доступу: www / URL: https://wiki.ros.org/Courses (дата звернення 08.05.2024).

5. Nao-robot [Електронний ресурс] /- Режим доступу: www / URL: https://en.wikipedia.org/wiki/Nao_(robot) (дата звернення 08.05.2024).

6. Robots [Електронний ресурс] /- Режим доступу: www / URL: https://robots.ros.org/ (дата звернення 08.05.2023).

7. National Instruments [Електронний ресурс] /- Режим доступу: www / URL: What is a Real-Time Operating System (RTOS)? - NI (дата звернення 08.05.2024).

8. Installation [Електронний ресурс] /- Режим доступу: www / URL: http://wiki.ros.org/Installation (дата звернення 08.05.2024).

9. Rosserial [Електронний ресурс] /- Режим доступу: www / URL: rosserial - ROS Wiki (дата звернення 08.05.2024).

10. Microros [Електронний ресурс] /- Режим доступу: www / URL: micro-ROS | ROS 2 for microcontrollers (дата звернення 08.05.2024).

11. Sergio Hernandez-Mendez. Design and Implemetation of a Robotic Arm using ROS and MoveIt! / Sergio Hernandez-Mendez, Hector Vasquez-Leal, Elvia R.

Palacios-Hernandez // IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC 2017). – 2017.

12. Arthur Gomes. ROS Based Wireless Teleoperation System for Robots / Atrhur Gomes, Medhang Nagavekar, Jeane Marina Dsouza // 2nd International Conference for Innovation in Technology (INOCON). – 2023.

13. Nipun Dhananjaya Weerakkodi. HyperDog: An Open-Source Quadruped Robot Platform Based on ROS2 and micro-ROS / Nipun Dhananjaya Weerakkodi, Iana Zhura, Ildar Babataev // IEEE International Conference on Systems, Man and Cybernetics (SMC). – 2022. – P. 436-441.

14. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. Структура та правила оформлення. Введ. 2015-06-22. К. Держстандарт України, 2017. – 29 с.

15. Харківський національний університет радіоелектроніки [Електронний ресурс] /– Режим доступу: www / URL: https://nure.ua/department/kafedra-komp-yuterno-integrovanih-tehnologiy avtomatizatsiyi-ta-mehatroniki-kitam.

16. Невлюдов І. Ш. Дипломне проектування для студентів усіх форм навчання спеціальностей 151 «Автоматизація та комп'ютерно-інтегровані технології» [Текст]: навч. посібник / І. Ш. Невлюдов, А. О. Андрусевич, О. В. Токарєва, Г. В. Пономарьова. – Київ: НАУ, – 2016. – 320 с.

17. Методичні вказівки до Підготовки атестаційної роботи бакалавра для студентів усіх форм навчання спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» освітньої програми: «Автоматизація та комп'ютерно-інтегровані технології» / упоряд.: І. Ш. Невлюдов, О. В. Токарєва, Г. В. Пономарьова. – Харків: ХНУРЕ, – 2019. – 36 с.

18. Bobkov M. Rosserial`s Role in Microcontroller Integration with ROS Robotics / Bobkov M., Starodubtsev E. // XVI International Scientific Student Conference> Contemporary global Trends: Challenges and Risks for Central Asia. – 2024.

19. Jackie Kay. Real-time control in ROS and ROS 2.0 / Jackie Kay, Adolfo Rodriguez Tsouroukdissian // Robotic Operation System Conference (ROSCon). – 2015.

20. FreeRTOS [Електронний ресурс] /- Режим доступу: www / URL: https://www.freertos.org/index.html (дата звернення 08.05.2024).

21. Get Started [Електронний ресурс] /- Режим доступу: www / URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/ (дата звернення 08.05.2024).