НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

1. ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 Інженерія програмного забезпечення

на тему Програмно-апаратний комплекс управління роботизованою платформою

Виконав: студент 4 курсу, групи ТІ-61

Голець Владислав Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, доцент, к.т.н. Ковальчук А. М. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент доцент кафедри ТЕУ Т Та АЕС Сірий О.А _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення веб-технологій та мобільних пристроїв

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту
Голець Владислав Олександрович
(прізвище, ім’я, по батькові)

1. Тема роботи Програмно-апаратний комплекс управління роботизованою платформою
керівник роботи доцент, доцент, к.т.н. Ковальчук А. М.
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 202__р. № __

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи персональний комп’ютер, мови програмування С++.
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні та апаратні рішення, можливі методи реалізації взаємодії елементів, обґрунтувати обрані програмні застосунки та шляхи розробки програмного забезпечення, розробити необхідне програмне та апаратне забезпечення, розробити інтерфейс користувача, зробити висновки за результатами роботи.
5. Перелік ілюстративного матеріалу
1. Моделювання системи 2. Апаратна архітектура системи. 3.Програмна архітектура системи. 4. Поєднання апаратних засобів системи. 5. Демонстрація роботи програмних та апаратних засобів системи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”___”_____201___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	2.12.2019	
2.	Вивчення та аналіз задачі	2.12.2019-13.04.2020	
3.	Розробка архітектури та загальної структури системи	13.04.2020-27.04.2020	
4.	Розробка структур окремих підсистем	27.04.2020-04.05.2020	
5.	Програмна реалізація системи	20.04.2020-13.05.2020	
6.	Оформлення пояснювальної записки	04.05.2020-05.06.2020	
7.	Захист програмного продукту	17.05.2020	
8.	Передзахист	12.06.2020	
9.	Захист	15.06.2020	

Студент _____ Голець В.О.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Ковальчук А.М.
(підпис) (прізвище та ініціали,)

Аннотация

Целью работы было внедрение программного и аппаратного интегрированного управления робототехнической системой для тестирования механизмов мониторинга и контроля технологических процессов. Был разработан, запрограммировано и установлено программное и аппаратное стенд. После этого серверная часть была реализована. Одним из преимуществ проекта является возможность добавления или удаления отдельных модулей платформы, что делает его функционально гибким и позволяет настроить. Система собирает и обрабатывает технологические данные и работает в режиме реального времени. Вы можете увидеть необходимую информацию в приложении по телефону и через специальный интерфейс брокера сообщений.

Записка содержит 58 страниц, 16 рисунков, 19 ссылок и 3 приложения

Анотація

Метою роботи було впровадження програмного та апаратного інтегрованого управління робототехнічною системою для тестування механізмів моніторингу та контролю технологічних процесів. Було розроблено, запрограмовано та встановлено програмний та апаратний стенд. Після цього серверна частина була реалізована. Однією з переваг проекту є можливість додавання або видалення окремих модулів платформи, що робить його функціонально гнучким і дозволяє налаштувати. Система збирає та обробляє технологічні дані та працює в режимі реального часу. Ви можете побачити необхідну інформацію в додатку по телефону та через спеціальний інтерфейс брокера повідомлень.

Записка містить 58 сторінок, 16 рисунків, 19 посилань та 3 додатки

Abstract

The aim of the work was to introduce software and hardware integrated control of the robotic system for testing the mechanisms of monitoring and control of technological processes. A software and hardware stand was developed, programmed and installed. After that, the server part was implemented. One of the advantages of the project is the ability to add or remove individual modules of the platform, which makes it functionally flexible and allows you to configure. The system collects and processes technological data and works in real time. You can see the necessary information in the application by phone and through a special interface of the message broker.

The note contains 58 pages, 16 figures, 19 links and 3 appendices

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

TCP/IP – Transmission Control Protocol/Internet Protocol

UART – Universal Asynchronous Receiver/Transmitter

ROS – Robot Operating System

MQTT – Message Queuing Telemetry Transport

BLDC – BrushLess DC motor

TTL – Transistor–transistor logic

ВСТУП

Розробка програмно-апаратного комплексу, стійкого до відмов, який відповідатиме високим вимогам до роботизованих систем, потребує значних зусиль у проектуванні та моделюванні майбутньої системи. Окрім загальних вимог щодо безпеки та надійності, до такої системи пред'являються додаткові вимоги щодо швидкості та простоти управління. Крім того, установка та обслуговування системи не повинні вимагати заміни значної частини елементів системи та велику кількість висококваліфікованих фахівців.

Найбільші труднощі при розробці такої системи - це проектування систем зв'язку елементів, з можливістю подальшого розвитку системи. Системи з малою здатністю до адаптації можуть стати непорушними монолітами у разі зміни навколишнього середовища. Протокол та методи зв'язку в таких системах повинні забезпечувати гнучкість та легкість адаптації різних програмних та апаратних засобів для виконання цілей, поставлених перед системою.

Найбільш ефективним рішенням проблеми монолітної системи є концепція багатоагентної системи. Це системи що розподіляють обов'язки між агентами та делегують усю відповідальність за успіх та прийняття рішень на агентів. Єдиною відмінністю цієї системи є присутність базової платформи що виконуватиме функції транспортування та живлення системи, її наявність є наперед відомою для внутрішніх агентів робота і забезпечує внутрішню та зовнішню комунікацію. Завдяки такому підходу до проектування системи, забезпечується стабільність системи, у якій помилка під час роботи одного інтелектуального агенту мінімально впливає на роботу усієї системи.

Виходячи з цього, було запропоновано дослідити існуючі роботизовані платформи, технології розробки їх програмного забезпечення, а також їх користувачів, зокрема методи створення мульти-агентних систем, та, у результаті, створити гнучкий програмно-апаратний комплекс системи віддаленого управління роботизованою платформою.

Було поставлено завдання: проаналізувати відкриті рішення, спланувати систему, використовуючи концепцію мульти-агентних систем, розробити програмну та апаратну частини системи, розробити інтерфейс користувача, який дозволить користуватись системою.

Для реалізації даного програмного продукту було обрано мову C++ та апаратну платформу Arduino, для комунікації було обрано фреймворк ROS та брокер повідомлень MQTT.

1АНАЛІЗ МУЛЬТИАГЕНТНОЇ РОБОТИЗОВАНОЇ СИСТЕМИ

На сьогоднішній день популярною є заміна людини на небезпечній, тривіальній та повторюваній роботі роботизованими системами, які розроблені так, щоб бути стійкими до навколишнього середовища та успішно виконувати завдання. Зараз майже всі важкі промислові гіганти використовують роботів з людьми або замість них, оскільки це призводить до більш високої швидкості, кращої якості, нижчих витрат і, як результат, підвищення ефективності та більший дохід[1].

Метою цієї роботи є створення такого програмно апаратного комплексу що дозволить у різних небезпечних сферах мінімізувати людську діяльність і, як наслідок, помилки та ризик для людського життя.

Сучасні системи спостереження та управління складаються з набору датчиків і актуаторів, які можуть керувати всіма системами в будівлі за умови співпраці з людиною оператором. Є ще ризиковані робочі місця, які займають люди, такі як охоронець та прибиральник. Ідея застосування інтелектуальної системи на основі агентів полягає в тому, щоб покрити всі потреби будівлі в таких дрібних і повторюваних діях, а крім того вивести людей із зон ризику. Наприклад, є звичайною практикою забезпечення ризикованих робочих місць страхуванням, однак при застосуванні такої системи компанія зменшить бюджет страхування, а працівник займе безпечнішу роботу[1-2].

Різні завдання з технічного обслуговування легко розкладаються на базові сценарії, такі як патрулювання, локалізація, навігація, звіт тощо. Хоча для деяких з них, таких як згаданий обов'язок звітності, потрібен лише ПК та програмне забезпечення для звітності, більшість обов'язків ґрунтуються на постійному русі до місць, які потребують уваги,(патрулювання), або пошуку речей(локалізація та патрулювання). Тут необхідно використовувати мобільну робототехніку.

Щоб повноцінно замінити людину як працівника спостереження, робот повинен задовольняти таким умовам:

1. Робот повинен мати однакові або більші якісні характеристики у порівнянні з людиною

2. Робот повинен мати стабільне з'єднання з оператором, щоб підтримувати передачу керування у разі неприємностей

3. Він повинен бути розроблений відповідно до середовища використання.

4. Робот повинен мати базові датчики та актуатори, які замінюють людські органи взаємодії на рівні, необхідному для виконання цього обов'язку.

Що стосується системи, яка має підтримувати місцезнаходження та здійснювати спостереження, то існують такі принципи:

- Система повинна бути пристосованою до нових пристроїв, таких як датчики, роботи, камери та консолі оператора
- Мати захищений зв’язок усіх вузлів, які беруть участь у процесі.
- Стандартизований API для різних підсистем, щоб вільно інтегруватися з іншими системами, такими як система контролю доступу або система роботи з клієнтами.
- Вона повинна базуватися на стандартній архітектурі як для апаратного, так і для програмного забезпечення та мати мінімальні відхилення від неї.
- Оскільки такі технології дуже популярні серед ентузіастів, ця система повинна мати надійне співтовариство професіоналів, зацікавлених у її розвитку підтримці.[2-4]

1.1 Аналіз існуючих програмних засобів

У процесі пошуку інформації, та аналізу існуючих рішень, було виявлено, що на даний момент такі системи є досить складними у використанні, або реалізують поставлену задачу не в повному обсязі:

— “project STOP” — це мультиагентна система розроблена компанією Ingeniarius LTD за підтримки університету Коімбри, Португалія. Система надає моливість менеджементу поверхів будівлі за допомогою агентів роботів-патрулювальників. Недоліком даної системи для поставленої задачі можна вважати її негнучкість у плані

масштабування(агенти є віртуальними та використовують лише один сервер обробки даних), непопулярність рішень щодо апаратної частини, і як наслідок її нестабільність, та відсутність можливостей додання нових функцій до алгоритмів патрулювання;

— “Milestone” — це реалізація архітектури гнучких систем информатизації будівлі. Дане програмне забезпечення також не реалізує поставлену задачу буквально а лише дозволяє за стандартизованим API підключати модулі(агенти) до мережі даних.

Також недоліком цих систем є їх громіздкість. Для початківців ці системи можуть здатися недолугими та дуже специфічними в плані налаштування.

Так як автоматизація людської діяльності за допомогою алгоритмів машинного навчання набуває широкого розповсюдження, то розробка зручної, інтуїтивно зрозумілої, гнучкої та невимогливої до апаратного забезпечення системи є актуальною задачею.

1.2 Опис багатоагентної структури

Багатоагентна структура – є системою що розподіляє задачі між ресурсо незалежними агентами. Ці агенти не повністю знають систему або нічого не знають про структуру системи, в якій вони працюють. Розробка у області багатоагентних систем включають такі теми:

Прийняття рішень: як агент приймає рішення? Який зв’язок між вашими сприйняттями, ідеями та впливом?

- Контроль: які ієрархічні зв’язки існують між агентами? Як вони синхронізуються?
- Спілкування: Які повідомлення вони надсилають один одному? Який синтаксис цих повідомлень?[1]

Багатоагентні системи можуть використовуватися для штучного інтелекту. Вони спрощують вирішення проблем, обмінюючись необхідними знаннями про підрозділи, до яких підключений один незалежний інтелектуальний агент, і координують діяльність агентів. Тому ми маємо на увазі розподілений штучний інтелект.

Агенти можуть обмінюватися знаннями, використовуючи будь-яку узгоджену мову. У системі протоколів зв'язку підхід може призвести до загальних удосконалень. Прикладами мови є KQML або ACL [3]. Ця система має значні переваги перед іншими, оскільки забезпечує безперервну роботу, навіть у разі проблем з одним із системних агентів. Інший спосіб побудови передбачає існування ієрархічної структури, в якій функціональність всієї системи залежить від головного агента, який обробляє всі зв’язки як зображено на рисунку 1.1. Завдяки своїй ієрархічній структурі система дуже чутлива до зриву основного агента. У разі несподіваних фатальних помилок програміста вся система перестає працювати, а тому необхідно забезпечити певні методи моніторингу стану системи, що надасть можливість підтримувати систему[2-3].

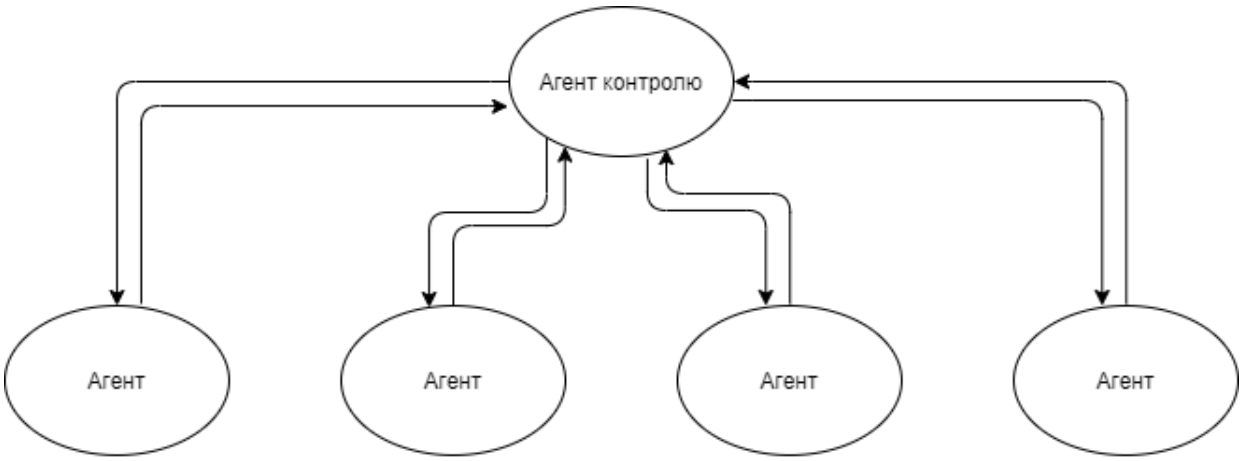


Рисунок 1.1 – Централізована система.

Проводяться дослідження що показують вищу продуктивність у багатьох галузях промисловості для багатоагентних та розподілених систем. Це свідчить про те, що галузь готова прийняти технологію [4].

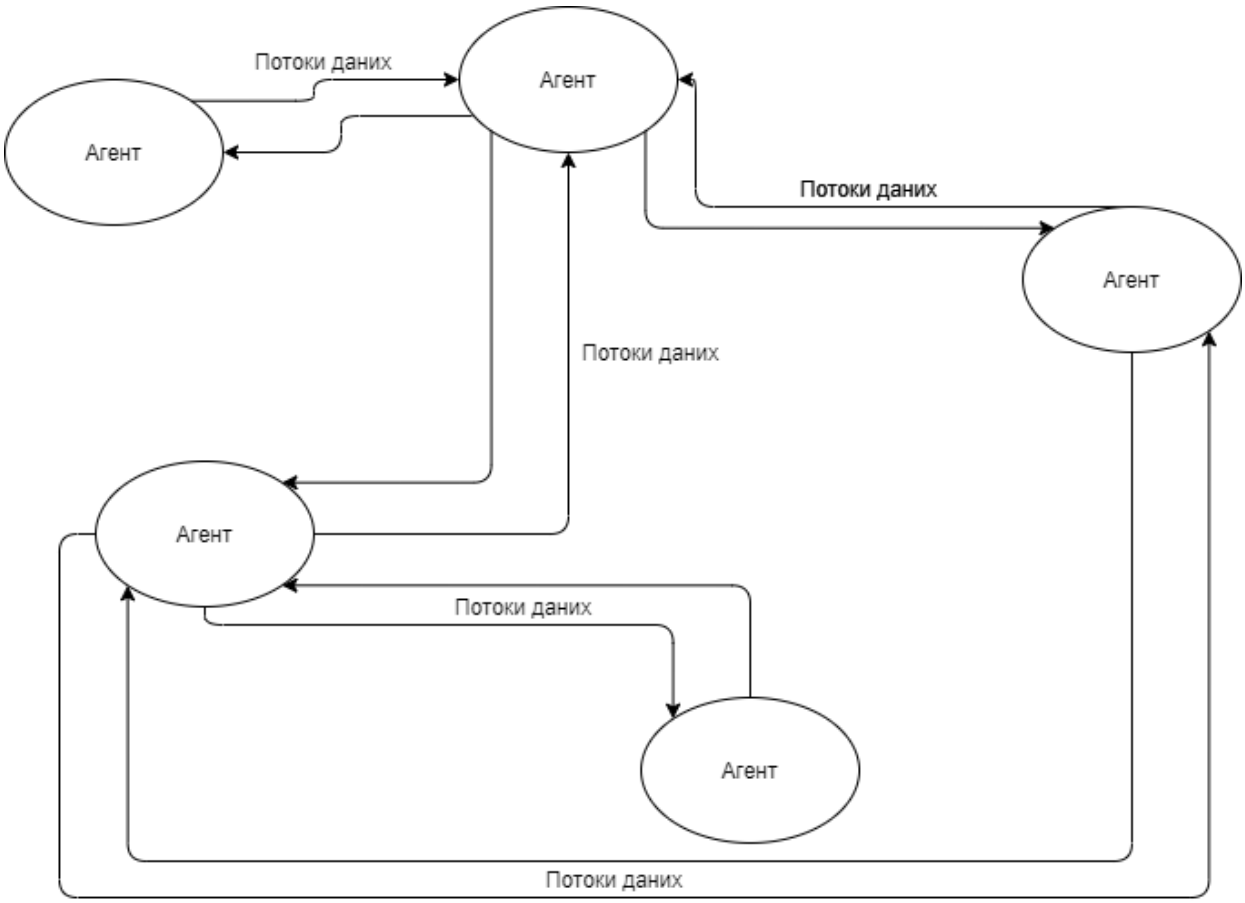


Рисунок 1.2 – Розподілена система

На рисунку 1.2 зображено розподілену систему сформовану з п’яти незалежних учасників. Кожен з них надсилає інформацію про свої стосунки або прохання змінити контакти іншого агента. Дослідження мультиагентних систем в основному зосереджуються на архітектурі системи, алгоритмах консенсусу, розподіленій оптимізації та програмному забезпеченні для моделювання та впровадження. Очікується, що в майбутньому будуть розвиватися технології, що мають більше агентів. Крім того, інші галузі намагаються використовувати ці системи, показуючи, що агентська технологія досягла значних успіхів у комерційному використанні. Процес розробки програмного забезпечення для багатьох системних агентств вимагає технічної підтримки обраної платформи. Важливо зрозуміти детальну документацію та дружній інтерфейс. Ефективність розробки може бути значно поліпшена, якщо у вас є гарний комплект розробки програмного забезпечення (SDK). Після застосування багатоагентних систем послуги стають довгостроковою проблемою для складних середовищ та географічних вузлів розподілу. Професійні служби підтримки

розподілених апаратних та програмних компонентів мають велике значення в майбутньому[4-6].

1.3 Опис агенту мультиагентної системи

Під час розробки агенту керуються такими принципами:

- Самоврядування: агенти є автономними одиницями.
- Вузький напрямок: агент не має повного уявлення про структуру системи.
- Децентралізація: в системі управління немає основного предмета; агенти рівні.

Тобто агент бачить лише його зв’язок і стан. У разі зміни в роботі будь-якого іншого з'єднання агент повинен надіслати агенту запит на цю зміну. Агенти, відповідальні за свою територію, будуть задовольняти запити та, залежно від ситуації, що склалася, відповідуть на запит або відхилять його.

Наприклад, регулятор вентиляційної системи (пристрій 1) повинен представляти електромагнітний струм, підключений до генератора струму (пристрій 2). Агент 1 надсилає запит на передачу електроенергії та її з'єднань та передає її всім мережевим агентам. Агент 2, як і всі інші агенти, отримав запит на відправку та обробку. Перевіряючи поточний стан, Агент 2 виявляє опорний струм, який неможливо надати, тому що через опорний потік абсолютно немає струму. Агент 2 відхиляє запит і повідомляє агента контролю про аварію.

Одже вимогами до агентів можна назвати:

- Можливість надійного обміну даними між агентами.
- Зчитування системних даних (зчитування датчиків та системних значень).
- Обробка та зберігання отриманих даних.
- Система звітування про стан[6].

2АРХІТЕКТУРА АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ

Аналізуючи поставлену задачу та методи її вирішення, було вирішено розроблювати програмний комплекс на основі брокерів повідомлень. Головною перевагою таких систем є гнучкість для масштабування та універсальність між вузлової взаємодії і можливість використання на будь-яких пристроях без портування на цільову операційну систему.

2.1.1 Архітектура мульти-агентної системи

Для реалізації поставленої задачі було вирішено використовувати розподілену архітектуру[4-5], яка складається з таких компонентів: брокер повідомлень, шлюз даних, та так звані ноди (вузли обробки та генерації даних). Схема даної архітектури зображена на рисунку 3.1.

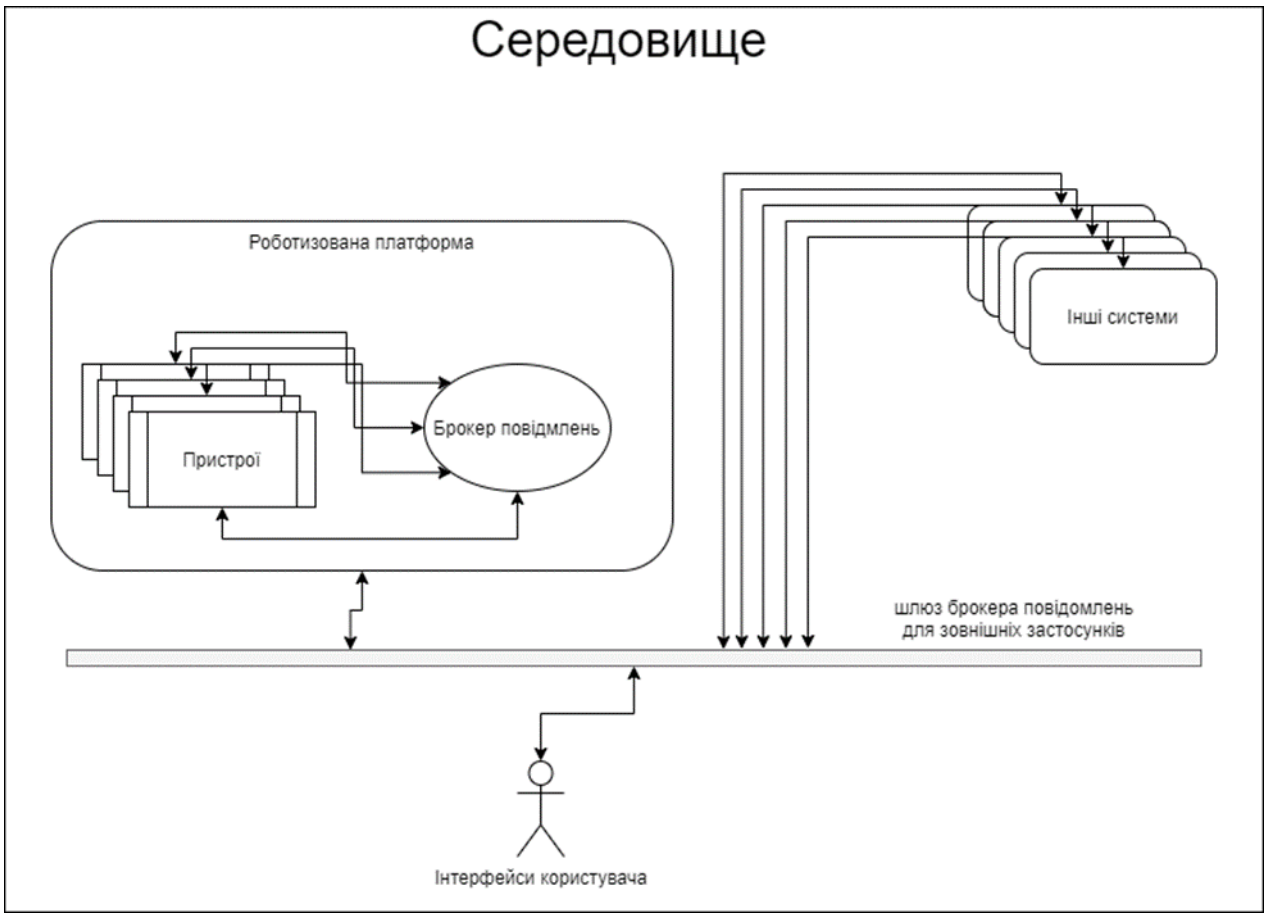


Рисунок 3.1 – розподілена архітектура програмного комплексу

Головним центром програмного комплексу є брокер повідомлень. Він відповідальній за менеджмент повідомлень та розсилання їх між нодами. Для того,

щоб користуватися програмою, потрібно бути авторизованим у системі, для запобігання неконтрольованого доступу до даних.

Під час користування системою, користувач взаємодіє з клієнтським додатком, яким є мобільній застосунок в даному випадку. На рівні користувача реалізований інтерфейс, за допомогою якого відбувається налаштування підключення та керування шасі робота. Також на користувацькому рівні відбувається попередня обробка даних перед відправленням на сервер і також опрацювання результатів від сервера (реальна швидкість робота). Ще на цьому рівні відбувається перший етап автентифікації користувача для обмеження неконтрольованого доступу до програми[2].

2.1.2 Архітектура агенту клієнтського застосунку

Для реалізації клієнтського застосунку було використано фреймворк Flutter, в основу якого було покладено шаблон розподіленого проектування (рисунок 3.2).

Такий підхід полегшує відокремлення розробки різних елементів додатку та легко взаємодіяти з ними[3]. Програма складається з віджетів та класів що частково нагадують MVC за своєю семантикою. Віджети відповідають за представлення інтерфейсу та передачу даних у класи-обробники (Viev). Вони в свою чергу відповідають за перевірку даних та їх передачу у систему(модель) через шлюз брокера повідомлень[12].

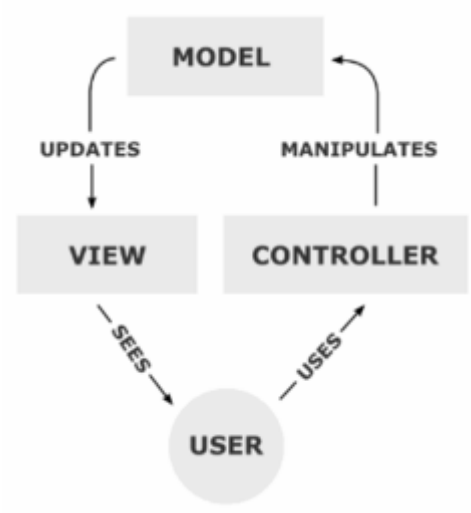


Рисунок 3.2 – Схема роботи додатку

Додаток ділиться на три частини:

• Шлюз (Model), реалізує доступ та синхронізацію з системою.

• Представлення (View) як і в класичному шаблоні MVC, Вигляд — це графічний інтерфейс, тобто вікно, кнопки тощо.

• Обробники (Controller), аналогічно до представлення відповідають класичному елементу MVC а саме обробка та перевірка даних між представленням та системою.

2.1.3 Архітектура серверу

Для реалізації системи було обрано мікросервісну архітектуру, що дозволяє розподіляти задачі та абстрагуватися від “заліза” вона складається з брокеру повідомлень та нод[11].

Брокер повідомлень — це сервери повідомлень що реалізують MPI взаємодію між програмами(нодами) з розділеними апаратними ресурсами. Іншими словами кожна програма незалежно від того де вона виконується має можливість отримати повідомлення з вказівками до виконання а також повідомити про результат роботи.

Нода — Окрема програма або підпрограма що має можливість підключатись до брокера та підписуватись на певні данні та отримувати або публікувати їх у узгодженому форматі.

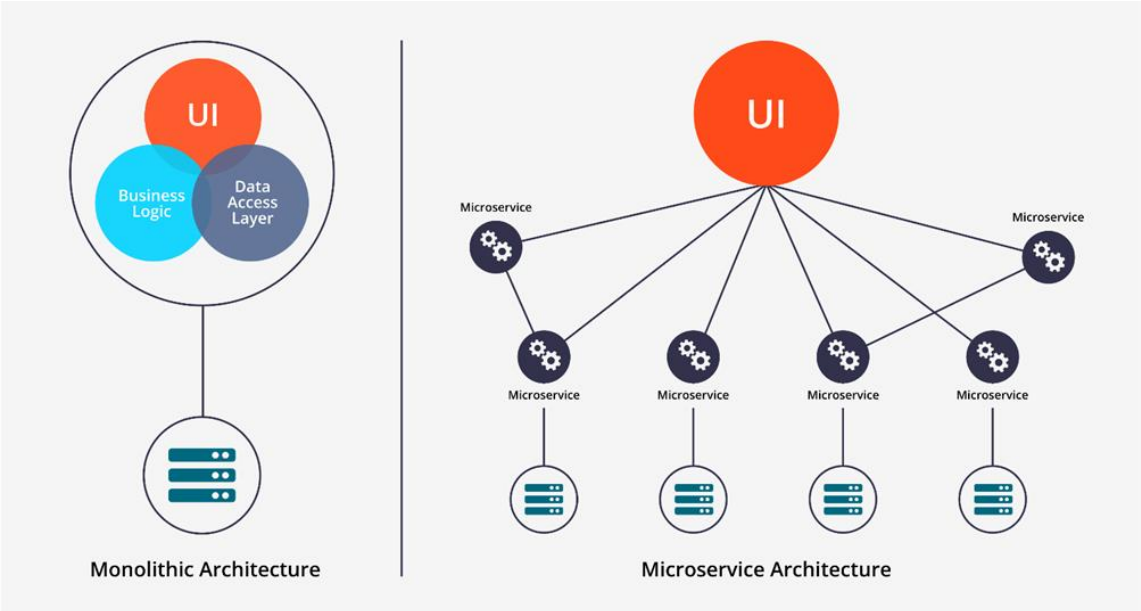


Рисунок 2.1.3.1 – Схема порівняння монолітних систем з мікросервісними

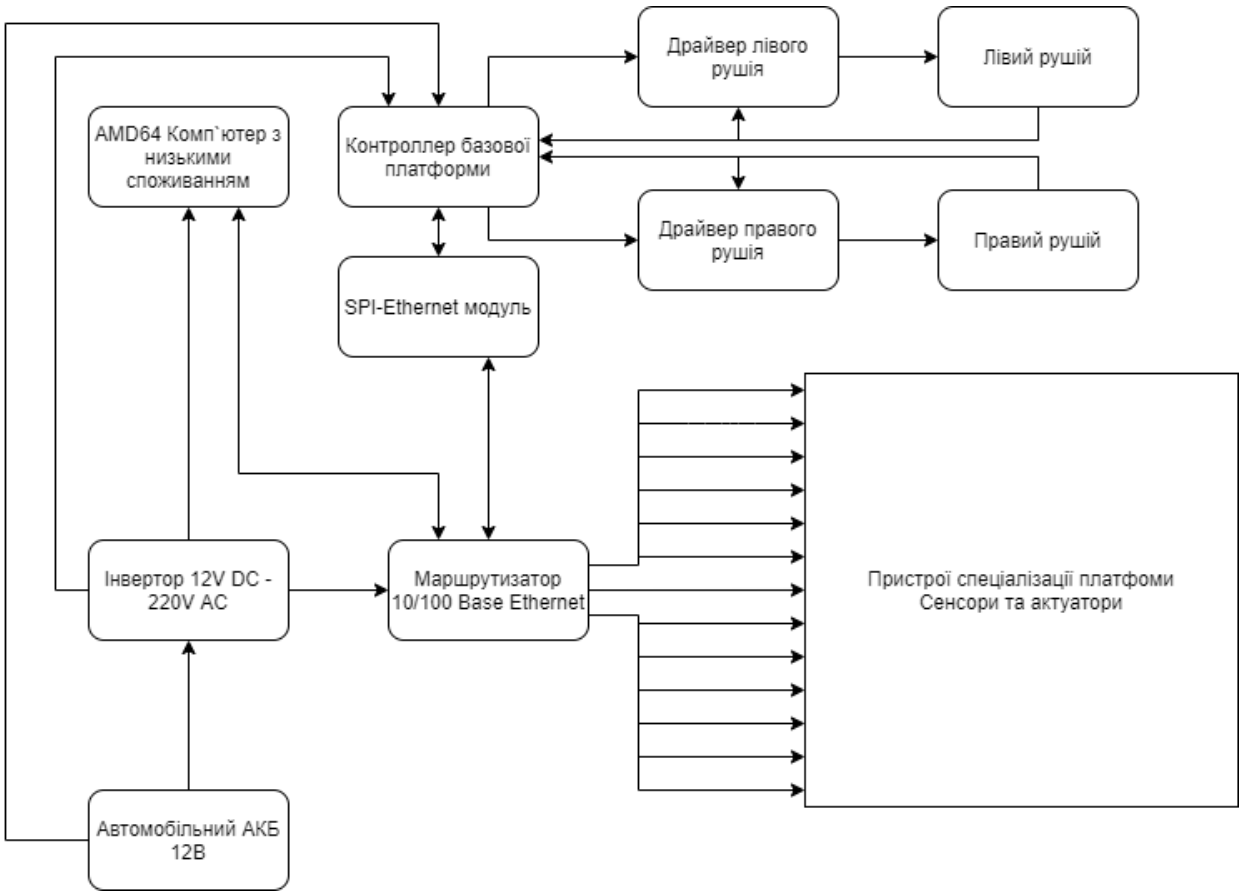
Основна мета застосування цієї концепції полягає в розподіленні та абстрагуванні від джерел даних та зосередженні на їх обробці. За рахунок такого розподілу підвищується можливість повторного використання та гнучкість системи. Також для великих проектів, застосування мікросервісів дозволяє застосувати тестування різноманітних компонентів програмної системи[1].

2.2 Архітектура апаратної частини системи

Під час проектування апаратної частини системи увага приділялась таким вимогам[1]:

- Наявність підтримки високорівневих мов програмування.
- Легка конфігурація інтерфейсів з можливістю абстракції від окремих елементів системи.
- Надійність, доступність та дешевизна кінцевої системи.

Тому була запропонована архітектура зображена на рисунку 2.2.1



Основними елементами системи можна виділити:

- Контроллер базової платформи – є структурною одиницею що відповідає за агентів пересування, та взаємодії з базовими актуаторами та сенсорами платформи
- Ethernet модуль – усі данні з контролера є доступними у внутрішній мережі агентів платформи та публікуються за допомогою ROS
- AMD64 Комп’ютер з низькими споживанням – э обчислювальним центром платформи. Вибір саме такого обчислювального модуля зумовлений широким використанням архітектури, підтримкою більшості операційних систем та надійністю. Відповідає за агентів локалізації навігації та сторонніх агентів користувача. Також використовується як сервер брокерів повідомлень ROS та MQTT.

2.3 Архітектура програмної частини системи

Система буде складатися з модулів(нод) що виконуватимуть незалежно один від одного задачі системи. Головним модулем буде сервер, який одночасно буде елементом компонування системи. Така структура, дозволить легко та інтуїтивно користуватися системою за рахунок того, що модулі є окремими сутностями не тільки абстрактно але й фізично [5].

На рисунку 2.3.1 наведена схема структури системи, на якій розташовані всі програмні модулі.

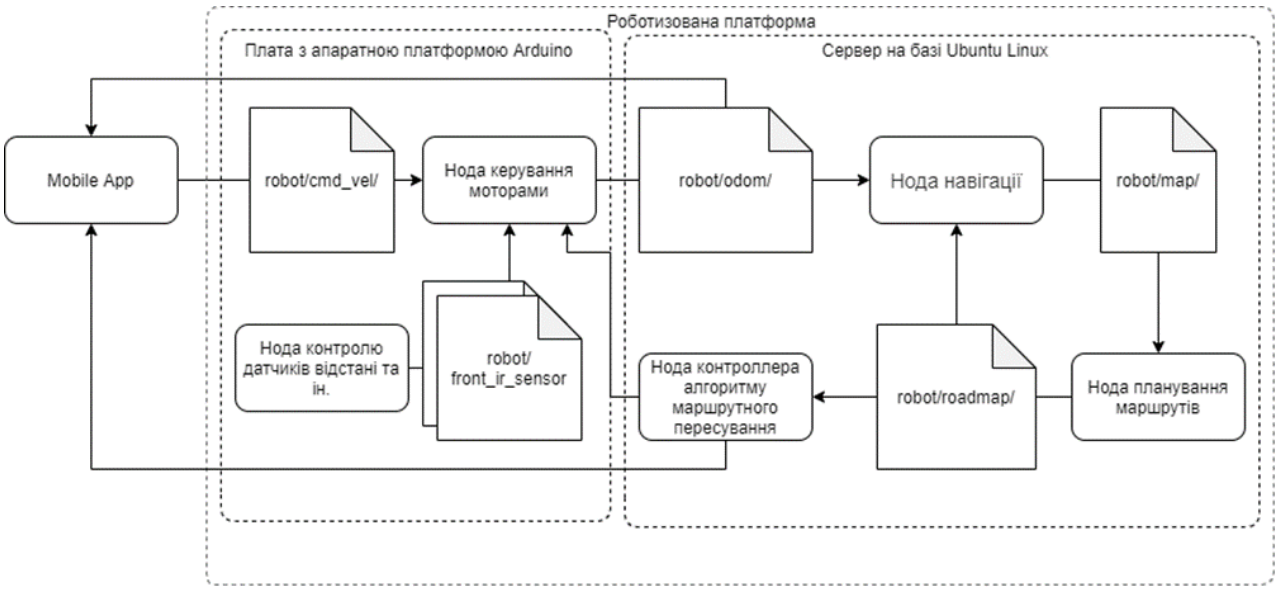


Рисунок 2.3.1 — Схема структури системи

Месседж брокери це сервери повідомлень що реалізують MPI взаємодію між програмами(нодами) з розділеними апаратними ресурсами. Іншими словами кожна програма незалежно від того де вона виконується має можливість отримати повідомлення з вказівками до виконання а також повідомити про результат роботи.

Наприклад: існує система з датчиків температури та сервера що зберігає їх дані. необхідно влаштувати систему взаємодії між датчиком та сховищем таким чином щоб у разі зміни налаштування мережі, це не вплинуло на роботу системи.

Можна звичайно використовувати HTTP запити та сподіватися що систему не доведеться масштабувати, але Технологія месседж брокерів дозволяє розробнику відкинути налаштування транспортного рівня та працювати над бізнес-логікою застосунку в той час як брокер займатиметься налаштуванням комунікацій між додатками-клієнтами та оптимізацією трафіку.

Для додатків він виглядатиме як додаток Telegram, є чати та групові конференції(topics) до яких можна приєднатися щоб читати та відписувати інформацію, також є чат боти(services) через які деякі з програм можуть надавати послуги.

З такою структурою датчики зможуть через месседж брокер публікувати дані температури у чатах з префіксом /temperature(безперечно назва чатів завжди залежить від розробника але існують стандарти, дотримання яких робить систему більш

зрозумілою) а у серверів сховищ даних буде можливість підписатись на ці чати та слідкувати за повідомленнями. Ця аналогія максимально добре описує роботу месседж брокерів а тому має сенс враховувати реальність роботи з мессенджерами під час проектування своїх систем. Наприклад один чат у якому будуть флудити всі кому не лінь не дасть можливості читачам устигати за кожним повідомленням і деякі можуть бути втрачені, в той час як окремі індивідуальні чати для кожного датчика призводять до того що зросте навантаження на мережу та обладнання. Золота середина це розділення чатів за типами повідомлень та семантичною приналежністю: наприклад температуру в приміщенні 503 слід публікувати в темі /temperature/503 а вологість у темі /humidity/503.

Цікавий момент що з курсу фізики ми пам’ятаємо що вологість неможливо визначити без температури а тому датчики вологості у 503 аудиторії мають підписатися на тему температури та використовувати дані з неї для обрахунків.

Така абстракція дуже зручна не тільки для вбудованих систем але і для мікро сервісних технологій адже реалізує ту необхідну ізольованість споживачів, постачальників та даних.

Слід зазначити що так само як сучасні месенджери, так і меседж брокери гарантують end to end шифрування. І хоча шифрування підтримують далеко не всі контролери, що підтримують брокерські мережі, не слід забувати що така можливість є.

Клієнт-додаток використовується як візуалізація отриманих даних від агентів, та спосіб комунікації користувача з агентами. Для збереження цілісності архітектури системи, клієнт-додаток також використовує месседж брокери для обміну повідомленнями[1].

3 АРХІТЕКТУРА АПАРАТНО ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Вступ

При проектуванні системи було вивчено та проаналізовано предметну область та вимоги замовника. Після ретельного аналізу було вирішено розроблювати програмний продукт, який заснований на розподілених технологіях.

Технології, які використовуються у системі, були обрані за принципом зручності у використанні, відкритості вихідних кодів, актуальності в наш час та можливістю виконання на будь-якій операційній системі. Мова C++ дає можливість розробки високопродуктивних програмних продуктів для великої кількості пристроїв з або без операційних систем. Брокер повідомлень надає моливість абстрагуватися від внутрішньої організації інших частин системи та взаємодіяти з даним які вони продукують. Також перевагою використання цих технологій є те, що їх можна запустити у контейнерах та побудувати на цьому швидке та надійне розгортання серверів з подальшим масштабуванням без створення при цьому спеціальної архітектури програмного забезпечення. Особливої уваги заслуговує моливість проведення симуляційних тестів з підміною інших нод тестовими процесами що генеруватимуть схожі до натуральних дані. Це надає змогу зосередити всю увагу на створенні якісної та протестованої бізнес-логіки.

На клієнтському рівні було обрано технології, які задовольнють такі ж умови як і серверні, але з поправкою на виконання в додатку. Фреймворк Flutter було обрано через те, що за допомогою нього можливо створювати складні графічні інтерфейси, які легко модифікувати, тестувати та розширювати в подальших циклах розробки програмного забезпечення а головне – використовувати як на Android так і на IOS. Мова Dart була обрана через те, що вона забезпечує типизацію та підтримку модулів. Також ця мова програмування надає безліч різноманітного “синтаксичного цукру”, який дозволяє скоротити кількість зайвого коду[4].

Дані технології в сукупності дають змогу збудувати якісний та надійний продукт, який захищений від патентних позовів з боку розробників, бо всі ці технології покриті ліцензіями, які виключають таку можливість і надають доступ до вихідних кодів даних проектів.

3.2 Проектування системи

Система буде складатися з модулів(нод) що виконуватимуть незалежно один від одного задачі системи. Головним модулем буде сервер, який одночасно буде елементом компонування системи. Така структура, дозволить легко та інтуїтивно користуватися системою за рахунок того, що модулі є окремими сутностями не тільки абстрактно але й фізичнох[5].

На рисунку 4.1 наведена схема структури системи, на якій розташовані всі програмні модулі.

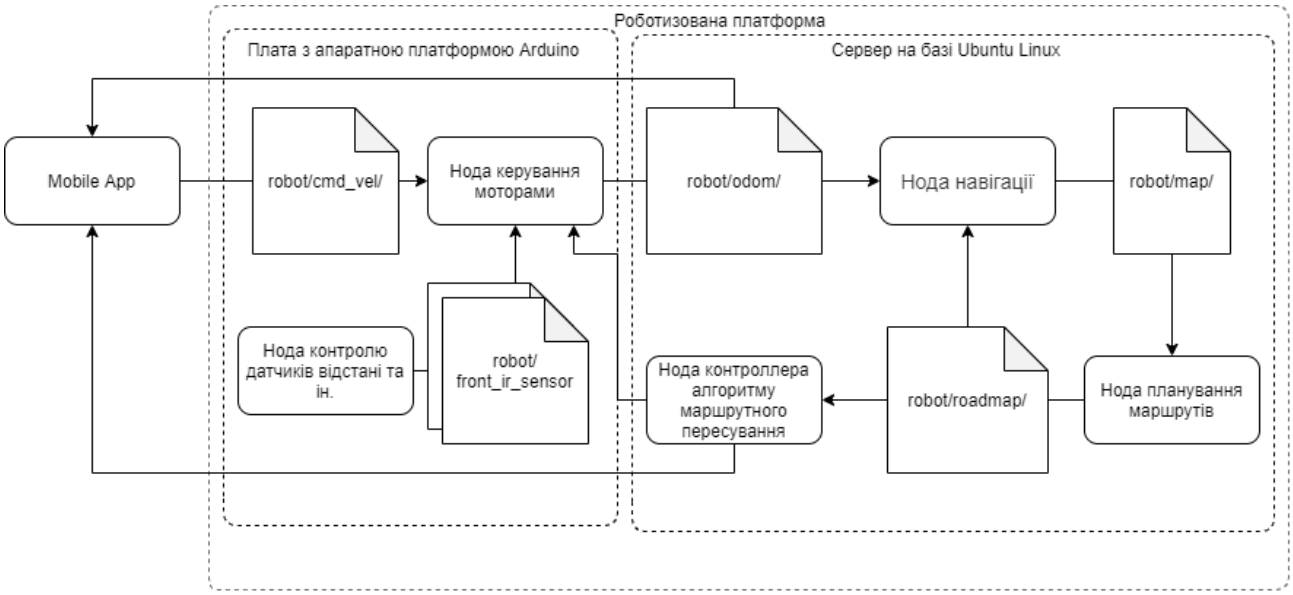


Рисунок 4.1 — Схема структури системи

3.3 Вибір мікроконтролера

Згідно вимог мікроконтроллер що відповідає за базову периферію повинен мати можливість зчитувати вірні покази з датчиків та контролювати швидкість рушіїв.

Для проекту було відібрано два контроллери що є часто використованими та мають велику кількість документації. Нижче наведено порівняльну таблицю 3.2.1.

Таблиця 3.2.1 – Порівняльна характеристика контролерів Atmega та ESP32

Характеристика	Atmega2560	ESP32
Тактова частота	16MHz	160MHz+
Кількість пінів GPIO	70	18
Flash	256KB	16MB
EEPROM	8KB	-
SRAM	4KB	512KB
WIFI	NO	YES
SHIELDS Compatibility	YES	NO
Operating voltage	5V	3V3
USB Port	YES	NO

Хоча Atmega2560 на базі апаратної платформи Arduino MEGA поступається іншим платформам, проте має простий інтерфейс програмування та достатню кількість бібліотек тому

для роботи було обрано апаратну платформу Arduino MEGA.

Живлення плати Arduino MEGA може здійснюватися двома способами:

— Через USB type B порт;

— Через вхід зовнішнього живлення .

Стабілізація зовнішнього джерела виконується за допомогою схеми AMS113-5.0. Спеціальна схема захищає компоненти компьютера. Схеми обох типів харчування наведені на рисунку.

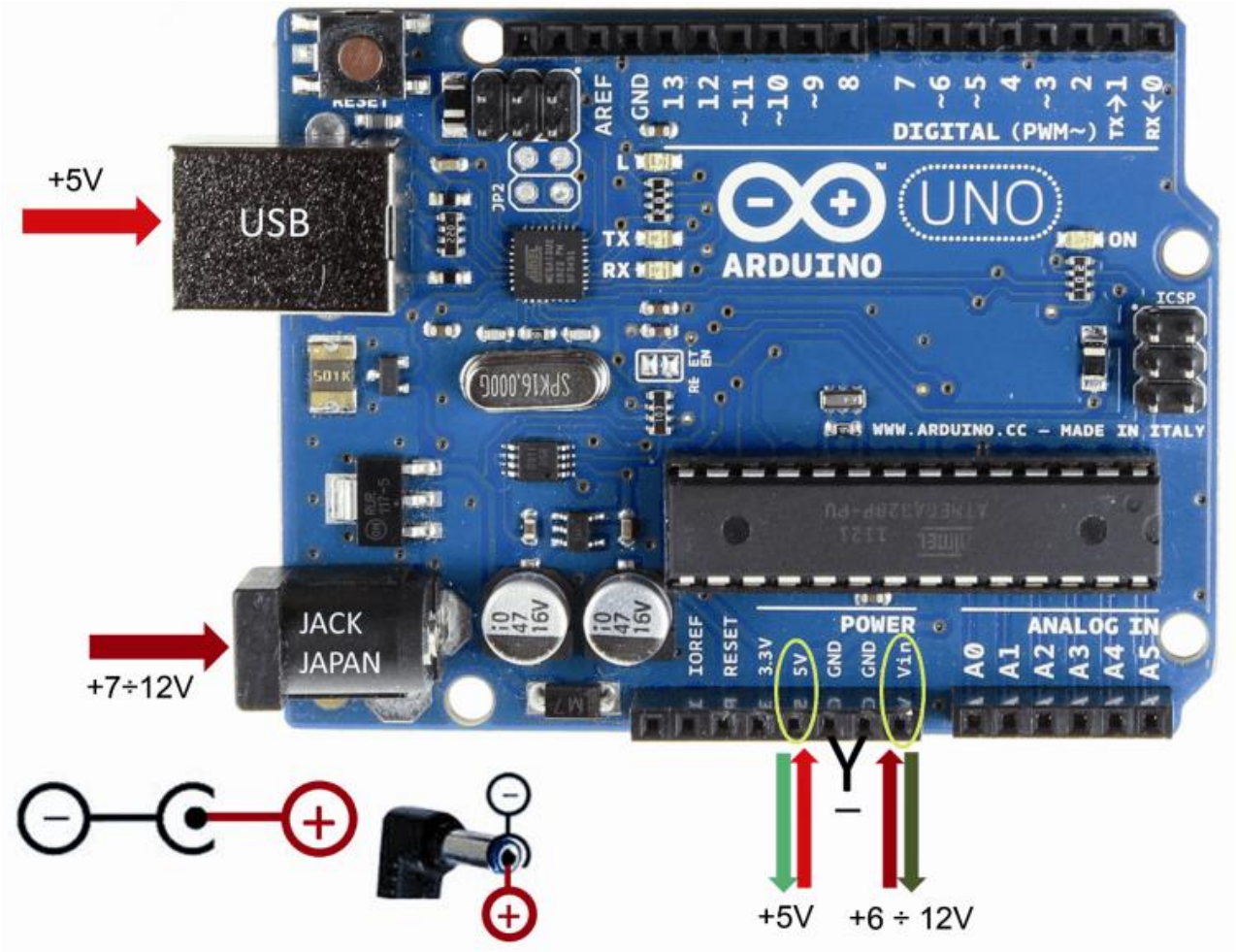


Рисунок 3.2.2 Схема живлення апаратної платформи Arduino Mega/Uno

Плата Arduino має 4 світлодіоди, які показують стан сигналу:

— Tx та Rx (загораються коли рівень сигналу низький, і показують, що Tx та Rx активні).

— PWR (загорається при напрузі 5 В)

— L (Світлодіод загального призначення)

Зокрема мікроконтролер має 6 виходів зовнішніх переривань, що дозволяє моментально реагувати на зміну станів важливої периферії.

3.4 Програмування мікроконтролера

У цій частині звіту описані способи та методи, які використовувались під час створення мікропрограмного забезпечення робота. Як було сказано в попередній частині, управління низьким рівнем забезпечується шляхом підключення

периферійних пристроїв до Arduino Mega 2560 на базі контролера AVR Atmega 2560 відповідно. Тому для цього завдання є лише три компілятори, це:

AVRA - компілятор асемблера (занадто складний і важкий для реалізації необхідної архітектури)

Компілятор AVR-GCC - C (класичне рішення, забезпечує швидкий, але вкладений і повторний код через складні рішення ООР)

AVR-G ++ - компілятор C ++ (забезпечує рішення ООР, необхідну швидкість та простоту)

Ми вирішили розробити прошивку, використовуючи C ++ та AVR-G ++ для її компіляції. це загальне рішення для проектів на базі Ардуїно, тому наявна спільнота, а також ООП та численні бібліотеки.

Архітектура програми відображається в Об'єктно-орієнтованій діаграмі в наданому архіві та буде описана в частині архітектури звіту.

Код зберігається у src-папці проекту та розділяється на класи, які зберігаються у заголовкових та вихідних файлах, один від одного, структури зберігаються у заголовку DataStruct.h, а всі визначення надаються з Definitions.h.

Скомпільовані файли мікропрограмного забезпечення (* .hex) зберігаються у папці проекту в папках їх платформи.

Цілями цієї частини були:

- Реалізувати можливість прийому та перетворення даних із ІЧ-датчиків та кодерів у звичайні блоки.
- Забезпечте періодичний виклик функції з частотою 10 Гц для контролю ардуїно.
- Використовуючи надані кінематичні закони та формули, обчислюйте роботів реальні кутові (w) та лінійні (v) швидкості.
- Обчислити одометрію робота та кутові швидкості кожного колеса.
- Вкажзати PID-регулювання швидкості колеса на основі реальної та бажаної кутових швидкостей.
- Переконатися, що звичайний сценарій відповідає 10 Гц частоті для виклику функції

3.4.1 Середовище розробки

Eclipse CDT[14] забезпечує повністю функціональне середовище інтегрованого розвитку C та C ++ на базі платформи Eclipse. Особливості включають: підтримку створення проекту та керовану збірку для різних ланцюжків інструментів, стандартна збірка виготовлення, навігація по джерелах, різні інструменти знань джерела, такі як ієрархія типу, графік викликів, включають браузер, браузер визначення макросу, редактор коду з підсвічуванням синтаксису, складання та гіперпосилання навігація, рефакторинг вихідного коду та генерування коду, засоби візуальної налагодження, включаючи перегляд пам'яті, регістрів та мікрокоду.

Eclipse CDT - плагін Eclipse, який перетворює Eclipse у потужний ID / C + C ++. Він був розроблений, щоб принести багато чудових функцій Eclipse, якими користуються розробники Java, розробникам C / C ++, таких як управління проектами, інтегрована налагодження, майстри класів, автоматизовані побудови, забарвлення синтаксису та завершення коду. Коли Eclipse використовується як Java IDE, він використовує та інтегрується з JDK. Аналогічно, CDT використовує та інтегрується зі стандартними C / C ++ інструментами, такими як g ++, make та GDB. Це призвело до того, що він стає дуже популярним у Linux, де ці інструменти легко доступні та використовуються для більшості C ++ розробок. CDT можна налаштувати в Windows для використання тих же інструментів. Постійно намагаються змусити CDT працювати з інструментами C ++ Microsoft, щоб зробити його ще більш привабливим для розробників Windows C ++.

3.4.2 З'єднання з мережею

Для підключення мікроконтролера до брокера повідомлень та до локальної мережі насамперед, через відсутність апаратних мережевих функцій, було вирішено підключити модуль ENC28j60 що дозволяє підключатись до локальної мережі через ETHERNET.

Модуль ENC28J60 Ethernet для підключення до мережі Ethernet схем на базе Arduino, AVR, PIC, ARM и др.. Напруга питания - 5 или 3.3 В.

ENC28J60 module	Arduino Uno/Due	Arduino Mega
CS	D10	D53
SI	D11	D51
SO	D12	D50
SCK	D13	D52
RESET	RESET	RESET
INT	D2	D2
VCC	3V3	3V3
GND	GND	GND

Рисунок 3.3.3.1 Схема підключення мережевого модуля

3.5 Вирішення проблем кінематики

Кінематика - це галузь класичної механіки, яка займається точками, тілами та системою, що складається з різних тіл без урахування сил, що діють на кожне.

Використовуються два основні методи:

Кінематика вперед: Вимірювання або пошук поточного положення робота.

Зворотна кінематика: використовуючи координати положення цілі, знайдіть дії, необхідні для досягнення мети.

Для чого тут використовується кінематика?

Локалізувати жорстке тіло (будучи нашим роботом тут) у тривимірному просторі, маючи при цьому пов'язану контрольну координату.

Тут посилаються координати самого робота.

Опорна система координат визначається трьома значеннями координат, положенням по осі x, положенням по осі y та його кутом від центральної лінії робота.

Щоб знайти своє положення в декартовій площині, ми використовуємо поняття під назвою Одометрія.

Одомертія- це спосіб оцінки переміщення за допомогою даних, отриманих із сенсорів руху.

Зазвичай використовують енкодери для вимірювання колеса або кута повороту, а також орієнтації колеса.

Тут використовуються два рівняння для пошуку переміщень лівого колеса та правого колеса:

$$DL = 2r / C * NL$$

$$DR = 2r / C * NR$$

У цьому випадку для пошуку лінійного переміщення робота достатньо знайти середнє зміщення обох коліс, тобто.

$$D = (DR + DL) / 2$$

3.6 Вирішення проблем контролю

Інженер з систем управління відповідає за розробку, розробку та впровадження рішень, що керують динамічними системами. Динамічні системи - це системи, які постійно змінюються. Мета інженера систем управління - забезпечити стабільність цих постійно мінливих систем, щоб отримати бажаний результат.

Використовуються два основні типи систем управління:

Системи з відкритим контуром: де контролер керує приводом, використовуючи задане вхідне значення, без подачі зворотного зв'язку на вхід. Прикладом може бути пральна машина або вимикачі світла.

Закриті системи циклу: Контролер керує приводом, використовуючи постійно змінюване вхідне значення через зворотний зв'язок, який враховує помилку. Приклад - кондиціонер.

Ця система керується в замкнутому циклі, використовуючи дані з кодерів двигунів, і застосовує силу, обчислену ПІД-контролером підрахунку в пропорційній, інтегральній та похідній помилках.

3.6.1 ПІД Контроллер

PID-контролер - це програмна абстаркція, яка використовується в промислових контрольних програмах для регулювання температури, витрати, тиску, швидкості та інших змінних технологічних процесів. Контролери PID (пропорційна інтегральна

похідна) використовують механізм зворотного зв’язку циклу управління для управління змінними процесів і у співпраці з нечіткою логікою забезпечує стабільний і простий спосіб управління.

Пропорційний (P) контроль: Це збільшує швидкість реакції системи та додає стабільності до певної точки. Крім того, вона зменшує стійку похибку.

Інтегральне управління (I): дестабілізує перехідну область та усуває помилку стаціонарного стану. Крім того, він вносить стійкість до зовнішніх порушень.

Похідне (D) управління: Це підвищує стійкість за рахунок зменшення коливань у системі, яка в основному гасить дію. Недоліком управління похідними є те, що воно не впливає на похибки стійкого стану.

У цій системі ми використовували правило Циглера-Ніколса для налаштування PID-контролю. Цей метод дозволяє знайти пропорційне, інтегральне та похідне посилення на основі перехідної реакції рушія, який в даному випадку є нашим Роботом. Після декількох експериментів нам вдалося отримати значення PID-контролера, що дало бажаний результат.

Недоліками цього методу Зіглера-Нікольса полягали в тому, що його дуже трудомісткі та введені коливання в систему піддавали небезпеці двигуни робота.

3.7 Архітектура ПО мікроконтролера

Для початку було обрано реалізацію змішаної моделі архітектури коду, яка займає повторне використання коду від ООР, з одного боку, і дозволяє мати функціональну програму, орієнтовану на переривання, з іншого.

Щоб уникнути динамічного розподілу пам’яті, кожен необхідний екземпляр структури даних, об'єкта, змінної створюється та ініціалізується як глобальний на початку програми. Це було зроблено через відомі проблеми динамічного розподілу пам'яті на AVR-контролерах. Клас "Енкодер" іноді забезпечує помилку сегментації помилок під час компіляції через динамічне розподілення всередині класу.

Структура алгоритму:

функція setup () void, що складається з ініціалізації апаратної частини та програмних класів, екземплярів кодера, конфігурації Serial та Timer5. Це частина ініціалізації.

функція void loop (), яка поки що порожня, оскільки весь код виконується в ISR Timer5 кожні 100 мс.

void TimerISR () функція, яка утримує процедуру управління обома колесами, вона використовує void encUpdate () для оновлення структур з даними кодера.

Детальний опис проекту див. У діаграмі класів та наданих вихідних файлах.

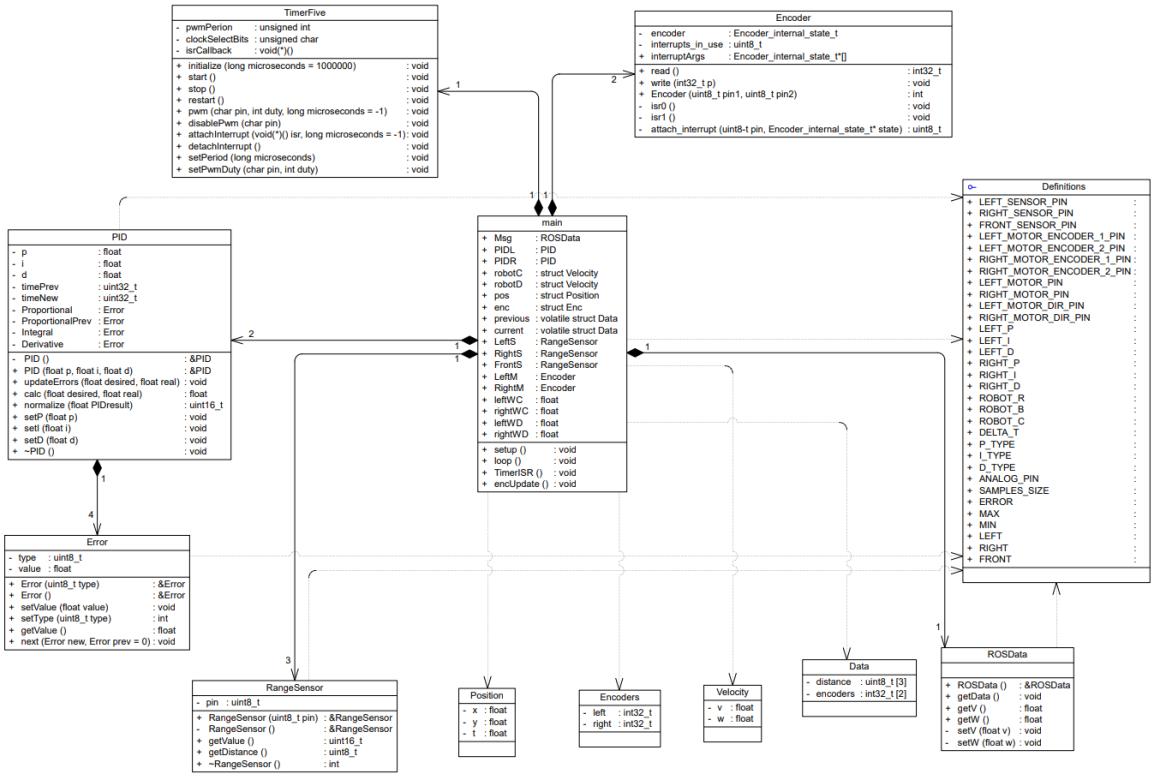


Рисунок 3.4.1 діаграма класів системи керування платформою

3.8 Програмування AMD64 Комп'ютера

Для розробки та роботи була обрана операційна система Ubuntu.

Ubuntu - це платформа базової операційної системи для комп’ютерів, на яку можуть покладатися інші проекти. Це легкий дистрибути, що спеціалізується на системах з архітектурою AMD64.

Перевагами такого вибору є:

— Для системи існує широкий вибір програмного забезпечення розробника

— SSH та SFTP сервери налаштовані за замовчуванням (порт 22)

На цю операційну систему було встановлено та налаштовано ROS та MQTT що здійснюють внутрішню та зовнішню комунікацію системи відповідно.

3.9 Розробка мобільного додатку

Програмний застосунок для керування роботизованою системою містить у собі одного головного актора – користувач системи;

На рисунку 4.2 представлена діаграма прецедентів, яка описує функції та дії актора у системі.

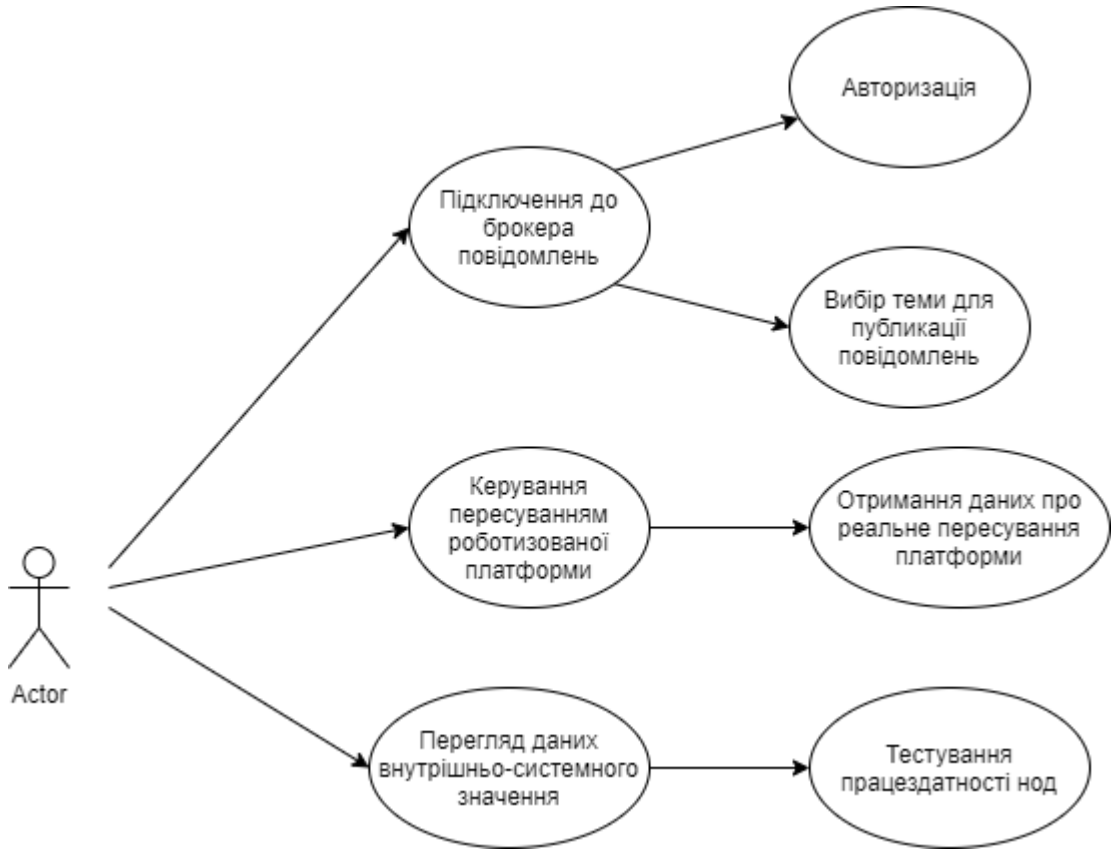


Рисунок 4.2 — Діаграма прецедентів системи

3.10 Архітектура повідомлень системи

Для між програмної взаємодії система використовує повідомлення[6]. Вони публікуються у відповідних темах що класифікують види повідомлень за призначенням та даними що вони містять.

Повідомлення однієї теми мають однаковий вид що спрощує обробку вхідних та формування вихідних даних.

Кожне повідомлення окремо від даних користувача містить необхідну для брокера службову інформацію що не має значення для системи а необхідна для успішної доставки.

Для кожного вхідного повідомлення у ноді існує метод – обробник що розпаковує данні та передає їх далі у ноду.

Детальна інформація про їх структури (ім’я, тип і розмір поля, опис поля) приведена у таблицях 4.1 — 4.4.

Таблиця 4.1. Структура повідомлення “cmd_vel”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Linear::x	double	Значення проекції вектора лінійної шидкості робота на вісь OX правої трійки векторів(ПТВ)
Linear::y	double	Значення проекції вектора лінійної шидкості робота на вісь OУ ПТВ
Linear::z	double	Значення проекції вектора лінійної шидкості робота на вісь OZ ПТВ
Angular::x	double	Значення проекції вектора кутової шидкості робота на вісь OX ПТВ
Angular::y	double	Значення проекції вектора кутової шидкості робота на вісь OУ ПТВ

Angular::z	double	Значення проекції вектора кутової шидкості робота на вісь OZ ПТВ

Таблиця 4.2. Структура повідомлення “odom”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
PoseWithCovariance::Pose::Point	double[3]	Координата робота на площині(x,y,z) ПТВ
PoseWithCovariance::Pose::Quaternion	double[4]	Кватерніон – структура що описує напрям робота
PoseWithCovariance::Covariance	Float[36]	Масив що характеризує можливу розбіжність даних з реальністю (похибка)

Таблиця 4.3. Структура повідомлення “map”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
MapMetaData::width	Uint32_t	Ширина карти у пікселях
MapMetaData::height	Uint32_t	Висота карти у пікселях
MapMetaData::resolution	Uint32_t	Масштаб карти
MapMetaData::origin	Pose	Координати першого пікселя карти та її напрям ПТВ

Таблиця 4.4. Структура повідомлення “distance”

Ім’я поля	Тип і розмір поля	Опис поля
RadiationType	Uint8_t	Тип датчика (ультразвук або інфрачервоний)
FieldOfView	Float	Поле зору датчика у радіанах
MinRange	Float	Мінімальне значення
MaxRange	Float	Максимальне значення
Range	Float	Поточне значення

3.11 Розробка додатку користувача

Додаток користувача – це додаток для операторського нагляду та контролю над станом системи. Він включає в себе інші підмодулі, які виконують основні функції системи, та є елементом компонування в системі.

Підмодулі додатку користувача:

До цих модулів належать наступні:

— модуль клієнта MQTT брокера;

— модуль керування роботизованою платформою;

— модуль взаємодії з внутрішньо системними даними.

3.11.1 Опис клієнта MQTT брокера

Даний модуль відповідальний за підписування та створення тем а також двосторонньої обробки повідомлень з них.

— Налаштування з’єднання з брокером та авторизація;

— Контроль цілісності даних;

— Пакування та розпакування повідомлень;

3.11.2 Модуль керування роботизованою платформою

Даний модуль виконує всю роботу пов’язану з генерацією вхідних даних для повідомлень cmd_vel, а саме:

— Отримання даних з інтерфейсу;

— Перетворення даних у правильний формат;

— Оновлення значень інтерфейсу даними зворотного зв’язку.

3.11.3 Модуль взаємодії з внутрішньо системними даними

Даний модуль дозволяє моніторинг та генерація повідомлень у системні теми такі як /roadmap. Це спрощує роботу при розширенні та тестуванні системи.

3.12 Розробка друкованої плати

Визничивши компоненти та принципи їх заємодії було побудовано принципову схему зображену на рисунку 3.8.1.

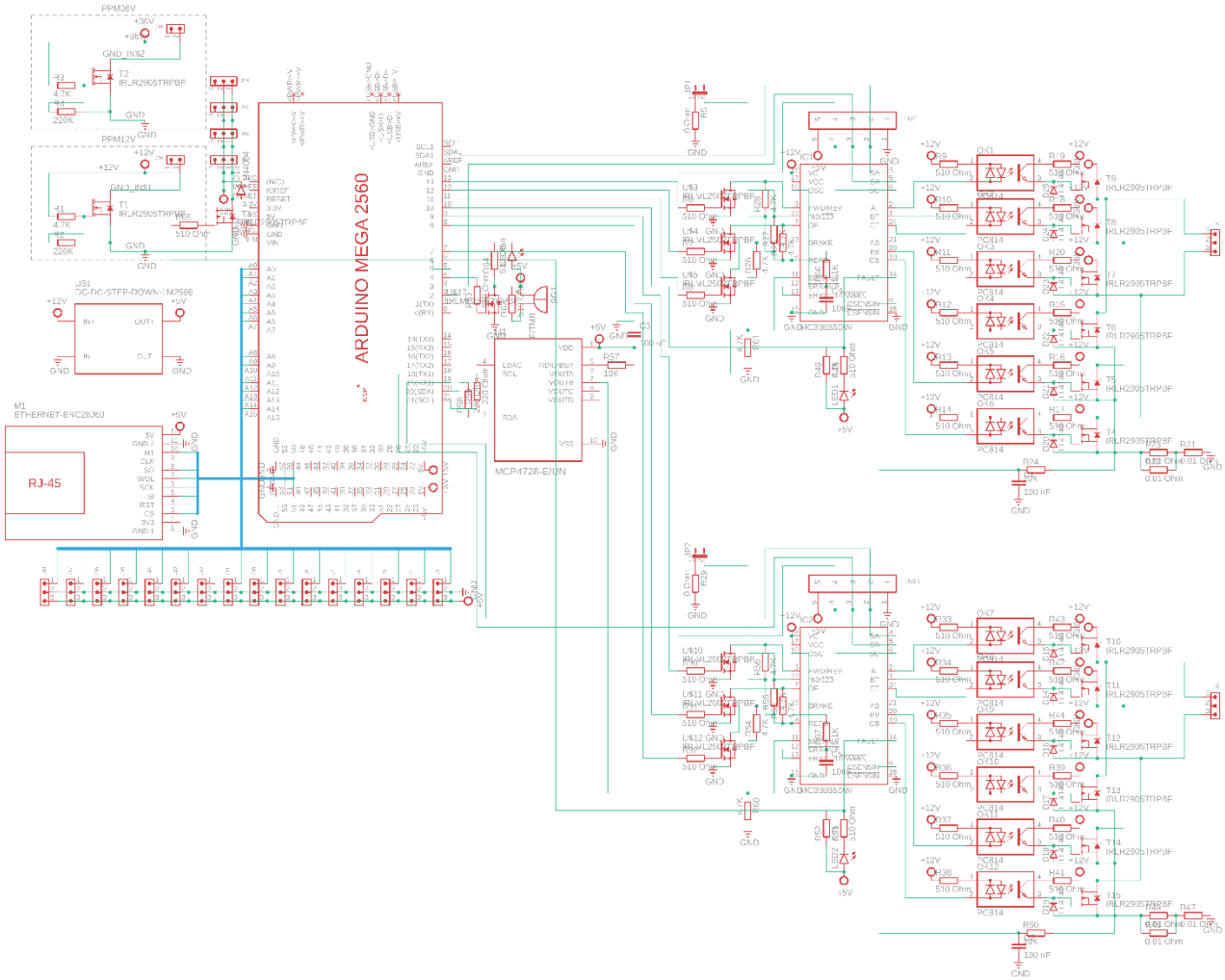


Рисунок 3.8.1 Принципова схема плати керування платформою

На основі цієї схеми було роведено та виготовлено друковану плату кервання роботизованою платформою що зображена на рисунках 3.8.2-3.8.4

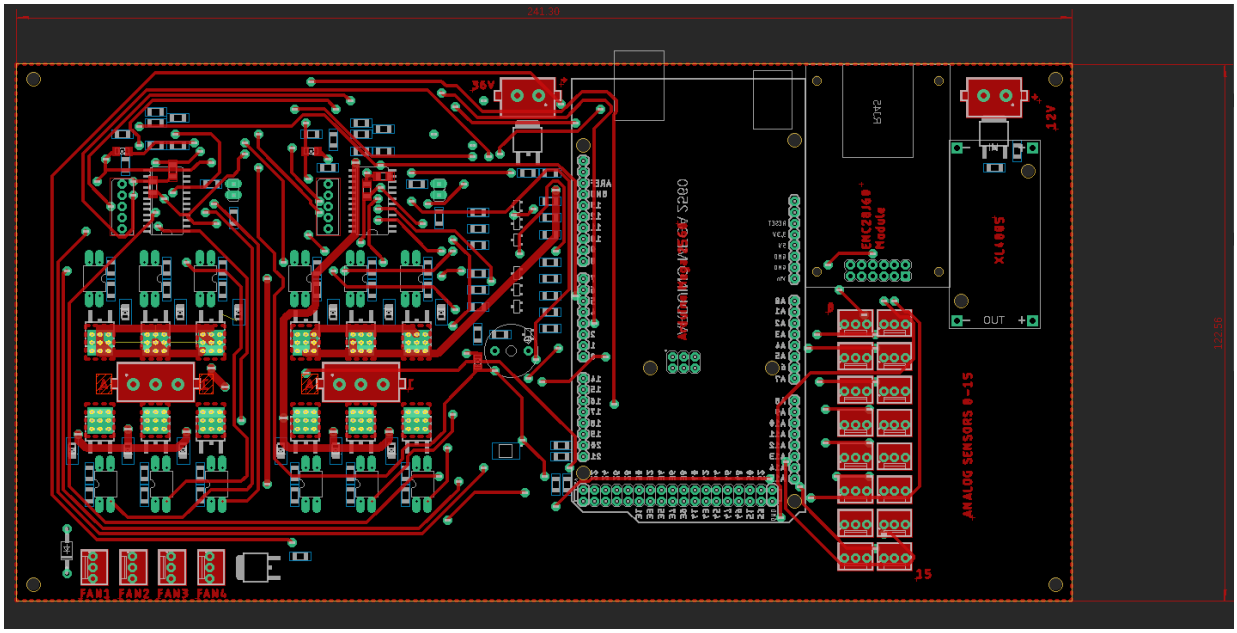


Рисунок 3.8.2 Верхній шар плати

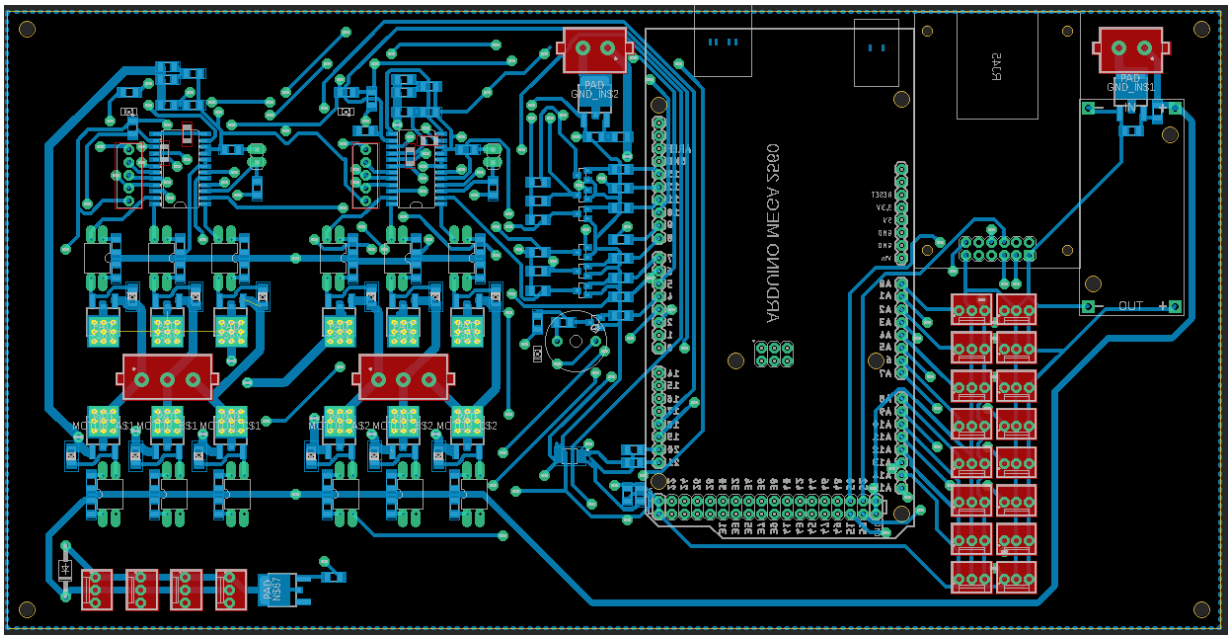


Рисунок 3.8.3 Нижній шар плати

Слід відмітити що полігон землі що вкриває всю невикористану площу є схованим та представленим у вигляді пунктиру навколо плати. Він використовується для зменшення вартості виробництва плати та підвищення її більшої екранованості від зовнішніх перешкод.

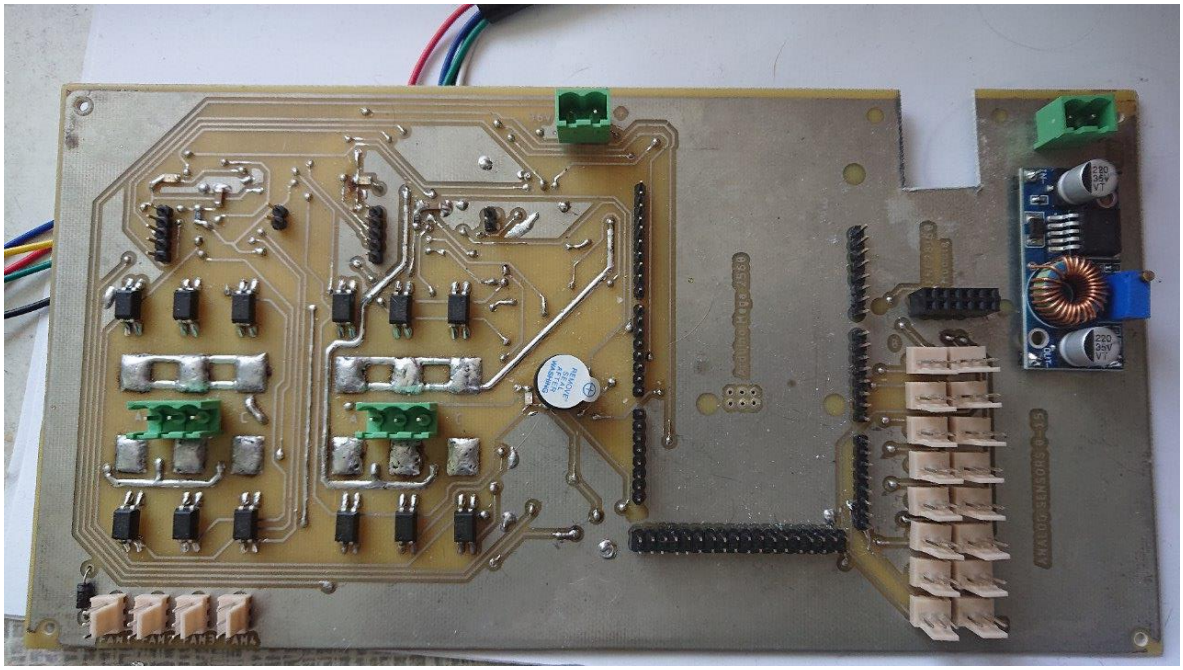


Рисунок 3.8.4 Вид готової плати зверху

3.13 Висновки до розділу

У цьому розділі було розглянуто усі види робот проведені для створення продукту:

— Вибір апаратних частин

— Створення коду для мікроконтролера

— Налаштування комунікацій для частин системи

— Розробка схем та друкованих плат

— Вирішення математичних задач керування системами руху з замкнутим циклом керування

4 ЗАСОБИ РОЗРОБКИ

Програмний продукт для інструментів та методів розробки можна розділити на Відповідно до інструментів та методів розробки програмний продукт можна розділити на три частини: розробка програми Andruino, розробка серверного додатка та розробка програми інтерфейсу.

Додаток Andruino було розроблено в C ++. Середовище розробки "Eclipse for C / C ++" було використано для розробки програми управління контролером для розробки програми мікроконтролерів AVR Atmega2560 та компілятора "AVR GCC" для створення програми мікроконтролерів AVR Atmega 2560. Цей додаток було розроблено в програмі Arduino IDE, яка дозволяє вбудувати мікропрограмне забезпечення.

Додаток для серверної частини агента також було розроблено на C ++. Для зручності розробки встановлена операційна система з відкритим кодом Ubuntu 18.04. Існує безліч корисних додатків, включаючи редактор коду Geani та компілятор G ++ для створення програм на комп'ютері. Для розвитку зв’язку з інтерфейсом програми та мікроконтролером ми використовували бібліотеку (бібліотека з інструментами конфігурації UART) та SEFL SDK (бібліотека була розроблена лабораторією кіберсистем, яка включає в себе мережеві інструменти мереж TCP та базову модель інтелектуального агента).

4.1 Arduino IDE

Arduino IDE[15] – це середовище розробки дя швидкого проектування коду для вбудованих систем. Включає у себе фреймворк Ардуіно, середовище, та монітор серійного порта.

Він працює на платформі Java, яка постачається із вбудованими функціями та командами, які відіграють важливу роль у налагодженні, редагуванні та компілюванні коду в оточенні.

Головний код, також відомий як мініатюра, створена на платформі IDE, врешті-решт генерує файл Hex, який потім передається та відправляється контролеру на

дошці. В основному IDE містить дві основні частини: редактор і компілятор, де спочатку відбувається написання а потім компіляція та завантаження коду в Arduino.

Це середовище підтримує мови C та C ++.

4.2 SPI - зв’язок

SPI (4-wire) – популярний інтерфейс для послідовного обміну даними між мікросхемами. Інтерфейс SPI відноситься до найбільш використовуваних інтерфейсів для з'єднання мікросхем.

Найменування «Serial Peripheral Bus» відображає його призначення – шина для підключення зовнішніх пристроїв.

Шина SPI організована за принципом «ведучий – ведений (підлеглий)". В якості ведучого шини зазвичай виступає мікроконтролер, але ним також може бути програмована логіка, DSP-контролер або спеціалізована IC.

На відміну від I2C і UART, SPI вимагає більше сигналів для роботи, але може працювати на більш високих швидкостях.

Підключенні до ВЕДУЧОГО шини зовнішніх пристроїв утворюють підлеглі шини. В якості підлеглих виступають різного роду мікросхеми, зокрема, запам'ятовуючі пристрої (EEPROM, Flash-пам'ять, SRAM), годинник реального часу (RTC), АЦП/ЦАП, цифрові потенціометри, спеціалізовані контролери тощо.

Інтерфейс використовує 4 лінії для обміну даними:

- SCLK - Serial Clock: тактовий сигнал (від ведучого) Інші позначення: SCK, CLK Arduino: pin 52
- MOSI - Master Output, Slave Input: дані від ведучого до веденого Інші позначення: SDI, DI, SI Arduino: pin 51
- MISO - Master Input, Slave Output: дані від веденого до ведучого Інші позначення: SDO, DO, SO Arduino: pin 50
- SS - Slave Select: вибір веденого; встановлюється провідним Інші позначення: nCS, CS, CSB, CSN, nSS, STE Arduino: за замовчуванням pin 53

Периферійний пристрій (Slave) взаємодіє з ведучим (Master) тоді, коли на виводі SS присутній низький рівень сигналу. У протилежному випадку дані від Master-пристрою будуть ігноровані. Така архітектура дозволяє взаємодіяти з декількома SPI-пристроями, підключеними до однієї та тієї самої шини: MISO, MOSI та SCK.

Лінія SS зазвичай для кожного веденого своя, але де-яких ведених можливо підключити до однієї SS – такий спосіб використовується для каскадного підключення пристроїв.

Незалежне підключення до шини SPI

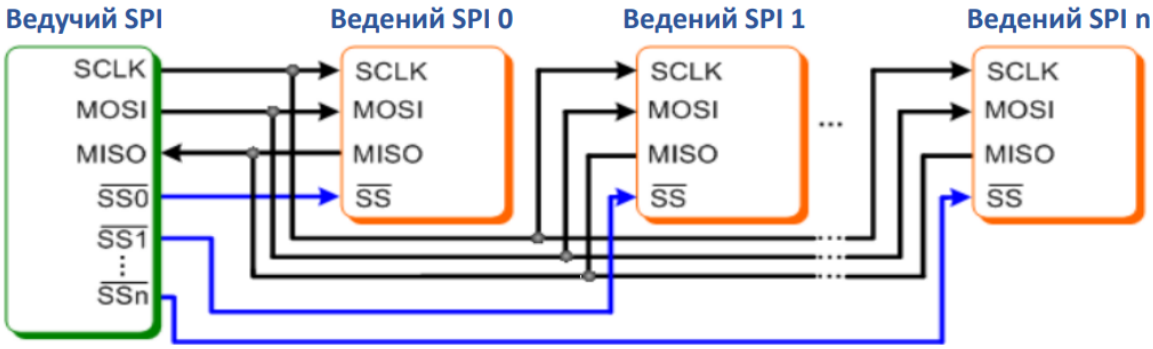


Рисунок 4.2.1 Незалежне підключення до шини SPI

Незалежне підключення більш поширене, тому що дозволяє використання будь-яких SPI-сумісних мікросхем. Головним недоліком такого підключення є необхідність в додаткових лініях для адресації підлеглих мікросхем (загальна кількість ліній зв'язку дорівнює 3+n, де n-кількість підлеглих мікросхем).

Для забезпечення одностороннього зв’язку з одним пристроєм, достатньо використовувати SCLK, MOSI (у випадку, якщо ведений пристрій лише приймає дані) або SCLK, MISO (у випадку, якщо ведений пристрій лише передає інформацію). На вході SS веденого пристрою повинен бути встановленим логічний нуль, інакше ведений не буде працювати.

SPI має чотири режими (0, 1, 2, 3), які засновані на поєднанні «полярності» тактового сигналу (clock polarity , CPOL – тактова полярність) та фази синхронізації (clock phase, CPHA – тактова фаза) (Рисунок 4.2.2) .

CPOL – початковий рівень сигналу синхронізації:

- якщо CPOL = 0, то лінія синхронізації до початку циклу передачі і після його закінчення має низький рівень (тобто перший фронт зростаючий, останній - спадаючий),
- інакше, якщо CPOL = 1, - високий (тобто перший фронт спадаючий, а останній - зростаючий).

CPHA – фаза синхронізації; від цього параметра залежить в якій послідовності виконується установка та вибірка даних:

- якщо CPHA = 0, то за переднім фронтом у циклі синхронізації буде виконуватись вибірка даних, а потім, за заднім фронтом, - встановлення даних;
- якщо ж CPHA = 1, то встановлення даних буде виконуватись по передньому фронту в циклі синхронізації, а вибірка – по задньому).

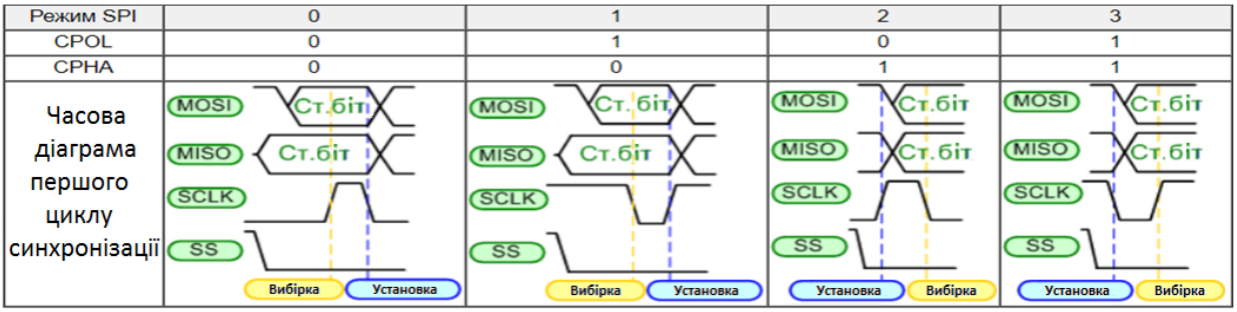


Рисунок 4.2.2 Режими SPI

У всіх моделях Ардуіно на базі мікроконтролерів AVR є SS, який використовується в режимі роботи Slave (наприклад, при керуванні Ардуіно від зовнішнього пристрою). Однак, у бібліотеці реалізовано лише режим роботи Master, тому в цьому режимі вивід SS повинен бути сконфігурованим як вихід. У протилежному випадку SPI може апаратно перемкнутись у режим Slave, що призведе до непрацездатності функцій бібліотеки. Для управління виводом SS периферійних пристроїв можна використовувати будь-який із доступних виводів. Наприклад, на платі розширення Arduino Ethernet для взаємодії з вбудованою SD-картою і контролером Ethernet по SPI використовуються виводи 4 та 10 відповідно.

4.3 TCP/IP-з’єднання

TCP / IP (Інтернет-протокол) - це протокол управління передачею даних. TCP - це основний протокол віртуального транспорту для Інтернету. TCP забезпечує надійну та послідовну доставку повнодуплексних октетних потоків (8-бітових байтів). Використовувані програми TCP вимагають надійної транспортної послуги, орієнтованої на з'єднання, наприклад, пошта (SMTP), передача файлів (FTP) та служба віртуального терміналу (Telnet).

Весь спектр протоколів Інтернету - набір правил і процедур - зазвичай називається TCP / IP, хоча задіяні й інші.

TCP / IP визначає, як пристрої спілкуються через Інтернет, забезпечуючи всебічне спілкування, яке визначає, яким чином вони повинні бути пакетовані, адресовані, передані, маршрутизовані та отримані в пункті призначення. TCP / IP не потребує центрального управління і призначений для того, щоб зробити мережі надійними, з можливістю автоматичного відновлення після відмови будь-якого пристрою в мережі.

Два основних протоколи в наборі протоколів Інтернет виконують певні функції. TCP визначає, як програми можуть створювати канали зв'язку по мережі. Він також контролює, як повідомлення збираються у менші пакети перед тим, як пересилатись через Інтернет та повторно збиратись у правильному порядку на адресу призначення.

IP визначає, як адресувати та пересилати кожен пакет, щоб переконатися, що він досяг бажаного пункту призначення. Кожен передавальний комп'ютер у мережі перевіряє цю IP-адресу, щоб визначити, куди пересилати повідомлення.

Основна особливість протоколу TCP / IP - це перевірка даних. Для кожного пакету даних, що надсилається з одного пристрою на інший, завжди існує пакет відповідей, що підтверджує отримання цього пакету. І протокол перевіряє ці відповіді та порівнює відправлені пакети, так що у випадку втрати пакету даних протокол повторює свою передачу.

TCP / IP-адресація базується на таких параметрах:

- MAC-адреса унікальна для кожного пристрою, та є ідентифікатором пристрою.

- IP-адреса пристрою - це непряма адреса, яка формується мережевими пристроями та спрощує навігацію у мережі.

4.4 Мова програмування C++

C++[16] - мова програмування високого рівня, створена як розширення мови програмування C, або "C з класами". Мова значно розширилася з часом, а сучасний C ++ тепер має об’єктно-орієнтовані, загальні та функціональні можливості, на додаток до засобів для маніпулювання низьким рівнем пам'яті. Вона майже завжди реалізовується як компільована мова, і багато виробників надають компілятори C ++, включаючи Фонд вільного програмного забезпечення, LLVM, Microsoft, Intel, Oracle та IBM, тому він доступний на багатьох платформах. C++ був розроблений з ухилом до системного програмування та вбудованого, обмеженого ресурсами програмного забезпечення та великих систем з продуктивністю, ефективністю та гнучкістю використання, як підкреслює його дизайн. C ++ також виявився корисним у багатьох інших контекстах. Основними перевагами цього є програмна інфраструктура та додатки, обмежені ресурсами, включаючи настільні програми, відеоігри, сервери (наприклад, електронна комерція, веб-пошук або SQL-сервери) та високопродуктивні програми.

4.5 Mosquitto

Mosquitto[12] - це брокер повідомлень з відкритим кодом (ліцензія на EPL / EDL), який реалізує протоколи MQTT версій 5.0, 3.1.1 та 3.1. Mosquitto має невелику вагу і підходить для використання на всіх пристроях, від однопанельних комп'ютерів низької потужності до повних серверів.

Протокол MQTT забезпечує легкий метод здійснення обміну повідомленнями за допомогою моделі публікації / підписки. Це робить його придатним для обміну повідомленнями Інтернету речей, таких як датчики низької потужності або мобільні пристрої, такі як телефони, вбудовані комп'ютери або мікроконтролери[6-7].

Проект Mosquitto також надає бібліотеку C для впровадження клієнтів MQTT, а також дуже популярні клієнти MQTT командного рядка mosquitto_pub та mosquitto_sub.

Mosquitto[10] є частиною Фонду Eclipse, є проектом iot.eclipse.org і фінансується cedalo.com.ASP.NET Core MVC — це фреймворк для створення веб-додатків та API-інтерфейсів за допомогою застосування шаблону проектування MVC. ASP.NET Core MVC надає заснований на моделях спосіб побудови динамічних веб-сайтів, що дозволяє точно розділити рівні логіки програмного застосунку. Він надає повний контроль над розміткою, підтримує принципи TDD та використовує найновіші веб-стандарти[7].

4.6 Arduino

Для реалізації роботи з сенсорами та актуаторами було вирішено використати фреймворк Arduino з апаратною платформою Arduino Mega 2560 що створена на базі контролера Microchip (Atmel AtMega2560). Було використано такі програмні продукти я к ArduinoJson[17] та AsyncMQTTclient що дозволили уникнути проміжного шару програмного та апаратного забезпечення для публікації та отримання даних.

Arduino[13] - апаратна обчислювальна платформа для аматорського конструювання, основними компонентами якої є плата мікроконтролера з елементами вводу/виводу та середовище розробки Processing/Wiring на мові програмування, що є спрощеною підмножиною C/C++. Arduino може використовуватися як для створення автономних інтерактивних об'єктів, так і підключатися до програмного забезпечення, яке виконується на комп'ютері (наприклад: Processing, Adobe Flash, Max/MSP, Pure Data, SuperCollider). Інформація про плату (рисунок друкованої плати, специфікації елементів, програмне забезпечення) знаходяться у відкритому доступі і можуть бути використані тими, хто воліє створювати плати власноруч.

4.7 ROS

ROS -Robot Operating System(не слід плутати з поняттям операційних систем), це фреймворк до складу якого входять велика кількість інструментів для програмування робототехнічних комплексів. Всі вони так чи інакше пов’язані або використовують вбудований меседж брокер для обміну разноманітною інформацією з роботом та між собою.

Відразу слід зазначити що ця система не є універсальною та має ряд недоліків що унеможливлюють ії використання у інших сферах діяльності крім робототехнічної, але завдяки вищезгаданим інструментам та додаткам вона дозволяє познайомитись з самою концепцією такого виду мереж.

Для початку роботи з ROS необхідно налаштувати лінукс-подібну систему(найкраще підходить Ubuntu) за цим планом:

Встановити Ubuntu(не важливо чи це буде віртуальна машина чи реальна) та оновити до найсвіжішої версії shell-командами:

```
sudo apt-get update
sudo apt-get upgrade
```

Пройти всі пункти 1.1-1.6 з цієї сторінки `http://wiki.ros.org/melodic/Installation/Ubuntu`

(дуже важливо читати все що там написано а не копіювати всі команди що виділені у блоки)

Виконати пункт 3 з цієї сторінки: `http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment#Create_a_ROS_Workspace`

На цьому етапі якщо все встановлено правильно команди

```
cd catkin_ws
catkin_make
```

мають не повертати жодних помилок

Встановити розширення для фреймворку що надає можливість комунікувати з брокером додаткам що працюють на малопотужних контролерах:

```
sudo apt-get install ros-melodic-rosserial
```

```
sudo apt-get install ros-melodic-rosserial-server
```

Не обов’язково але можливо також встановити аналогічне розширення для комунікації через вебсокети:

sudo apt-get install ros-melodic-rosbridge-server

Вивантажити бібліотеку для arduino-eclipse для роботи з ROS з гіт-репозиторія https://github.com/VladislavHolets/arduino_libs/tree/master/ros_lib/0.8.0

Для користувачів Eclipse помістити бібліотеку у папку

```
~/.arduinocdt/libraries
```

Після виконання цих пунктів з’явиться можливість підключити бібліотеку у середовищі

Перед початком роботи з фреймворком рекомендується ознайомитись з інструкціями що знаходяться на сторінці

```
http://wiki.ros.org/ROS/Tutorials
```

Для комфортної роботи фреймворк дозволяє створити файли запуску (launch files) що запускають систему з вказаними додатками та їх параметрами. Файл запуску потрібен для запуску системи та додатків однією командою у одному вікні термінала (зазвичай кожен додаток та ядро використовують по екрану термінала для відображення статусу);

Класичним файлом запуску для старту системи з rosserial та rosbridge буде файл з вмістом:

```
<launch>
  <include file="$(find rosbridge_server)/launch/rosbridge_websocket.launch"/>
  <include file="$(find rosserial_server)/launch/socket.launch"/>
</launch>
```

Збережений з розширенням .launch

Запуск системи відбувається командою:

```
roslaunch /[path_to_your_file].launch
```

Це запустить систему з “вікном” для підключення на 11411 порту.

Для запуску ноди на мікроконтролері існує велика кількість прикладів та матеріалів на сторінках `http://wiki.ros.org/` .

А також репозиторій що містить робочий приклад.

https://github.com/VladislavHolets/ArduinoEthernetRos

4.8 MQTT

Один з найбільш розповсюджених протоколів що працюють над TCP/IP для забезпечення мікро сервісних/мультиагентних та інших peer to peer систем комунікацією.

Перевагою MQTT є підтримка у майже кожній мові програмування та велика кількість брокерів з підтримкою від багатьох компаній.

У цьому курсі матеріали та приклади будуть базуватися на прикладі Mosquitto (продукт Eclipse foundation) для операційної системи Ubuntu 18.04 з локальним брокером.

Для початку роботи з ним необхідно:

Додати до індексу репозиторій з актуальними версіями командою:

sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa

Оновити індекс командою:

```
sudo apt-get update
```

Завантажити сервіс командами:

```
sudo apt-get install mosquitto
sudo apt-get install mosquitto-clients
sudo apt-get install mosquitto-auth-plugin
sudo apt-get install mosquitto-dbg
```

Тепер сервіс брокера працює та готовий до використання. За допомогою команд mosquitto_sub та mosquitto_pub можна перевірити його роботу написавши повідомлення у тему, у разі успіху у вікні sub відобразиться повідомлення надіслане командою pub.

На відміну від ROS, MQTT вже є професійним інструментом а тому чудово документований та більш зручний для впевнених користувачів.

4.9 ArduinoJSON

Для Arduino теж є бібліотека для роботи з JSON:

https://arduinojson.org/

Її можна завантажити з менеджера пакетів Eclipse або з сайту. На самому сайті бібліотеки присутній пакет документації та зручний калькулятор-генератор функцій для пакування та розпакування JSON-строки.

У бібліотеці тим не менш є недолік, через оптимізацію роботи з символьними масивами вона погано індексується а тому команда Organize includes (Ctrl+Shift+O) підключає бібліотеку частково а не повністю що перешкоджає компіляції.

бібліотеку слід підключати одним заголовковим файлом “ArduinoJson.h”.

4.10 Фреймворк Flutter

Dart[19] - мова програмування, створена Google. Dart позиціонується в якості заміни / альтернативи JavaScript. Один з розробників мови Марк Міллер (Mark S. Miller) написав, що JavaScript «має фундаментальні вади», які неможливо виправити. Тому і був створений Dart. Переваги над JavaScript:

- можливість явного визначення типів (статична типізація).
- підтримка використання повноцінних класів.
- підтримка підключення модулів.

Flutter[19] - це набір для розробки програмного забезпечення з відкритим кодом, створений Google. Він використовується для розробки програм для Android, iOS, Windows, Mac, Linux, Google Fuchsia та Інтернету. Програми Flutter написані мовою Dart і використовують багато вдосконалених можливостей мови. У Windows, macOS та Linux за допомогою офіційного проекту Flutter Desktop Embedding Flutter працює у віртуальній машині Dart, яка має функціональний механізм виконання. Під час написання та налагодження програми Flutter використовує компіляцію Just In Time, що забезпечує «гаряче перезавантаження», за допомогою якого модифікації вихідних файлів можна вводити в запущену програму. Flutter поширює це на підтримку сталого гарячого перезавантаження, де в більшості випадків зміни вихідного коду можуть бути відображені негайно в запущеній програмі, не

вимагаючи перезавантаження або втрати стану. Ця функція, реалізована у Flutter, отримала широку оцінку.

Версії випуску програм Flutter складені заздалегідь (AOT) на Android та iOS, завдяки чому можливі високі показники роботи Flutter на мобільних пристроях.

4.11 EAGLE

Для розробки апаратної частини було використано спеціалізоване програмне забезпечення що дозоляє проектувати друковані плати.

EAGLE[18] - це прикладний електронний додаток для автоматизованого дизайну (EDA) із схематичним захопленням, схемою макета друкованої плати (PCB), автоматичним маршрутизатором та автоматизованим виробництвом (CAM). EAGLE розшифровується як редактор графічного макета, що легко застосовується (німецька: Einfach Anzuwendender Grafischer Layout-Editor) та розроблений CadSoft Computer GmbH.

EAGLE містить схематичний редактор для проектування схем. Схеми зберігаються у файлах з розширенням .SCH, частини визначаються в бібліотеках пристроїв з розширенням .LBR. Частини можна розмістити на багатьох аркушах і з'єднати між собою через порти.

Редактор макета PCB зберігає файли плати з розширенням .BRD. Це дозволяє зворотній анотації до схематичного та автоматичного маршрутизації для автоматичного з'єднання слідів на основі з'єднань, визначених на схемі.

EAGLE зберігає файли макета Gerber і PostScript, а також файли свердлівок Excellon і Sieb & Meyer. Це стандартні формати файлів, прийняті компаніями-виробниками PCB, але, враховуючи типову базу користувачів користувачів невеликих дизайнерських фірм та любителів EAGLE, багато виробників і монтажної майстерні друкованих плат також приймають файли плати EAGLE (з розширенням .BRD) безпосередньо для експорту оптимізованих виробничих файлів та і розміщуйте самі дані.

EAGLE забезпечує багатовіконний графічний інтерфейс користувача та систему меню для редагування, управління проектами та для налаштування параметрів

інтерфейсу та дизайну. Системою можна керувати за допомогою клавіш миші, клавіатури або введення певних команд у вбудованому командному рядку. Кілька команд, що повторюються, можна комбінувати у файли сценаріїв (з розширенням .SCR). Також можливо досліджувати файли дизайну, використовуючи специфічний для EAGLE об'єктно-орієнтований мову програмування (з розширенням .ULP).

4.12 Висновки до розділу

Цей розділ описує усі використані програмні та апаратні засоби для розробки системи:

- Фреймворки для розробки мобільних додатків
- Фреймворки комунікаціїї
- Середовища розробки та проектування
- Бібліотеки
- Брокер повідомлень
- Інтерфейси зв’язку

5 ІНТЕРФЕЙС КОРИСТУВАЧА

При вході на клієнтський додаток, користувачеві необхідно авторизуватися в системі щоб почати працювати в системі. На рисунку 5.1 зображена форма авторизації.

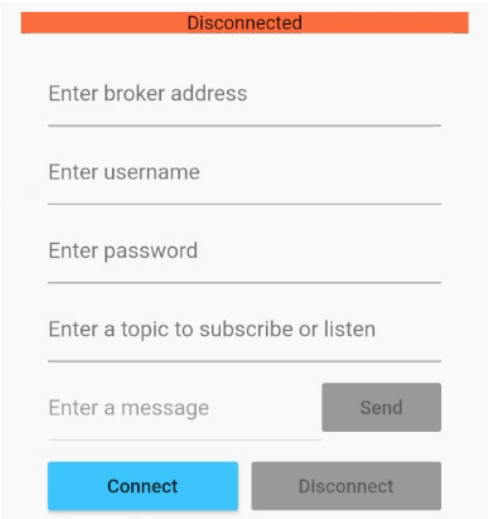


Рисунок 5.1 — Форма авторизації

Після того, як користувач авторизувався в системі, він отримує доступ до основного функціоналу системи. Користувач має доступ до усіх сторінок системи, або має можливість вийти з свого профілю. Також він може перейти до свого основного робочого простору – панелі керування платформою. На рисунку 5.3 зображене головне меню системи. Червоним виділено данні що уже надходять в систему у обрану тему.

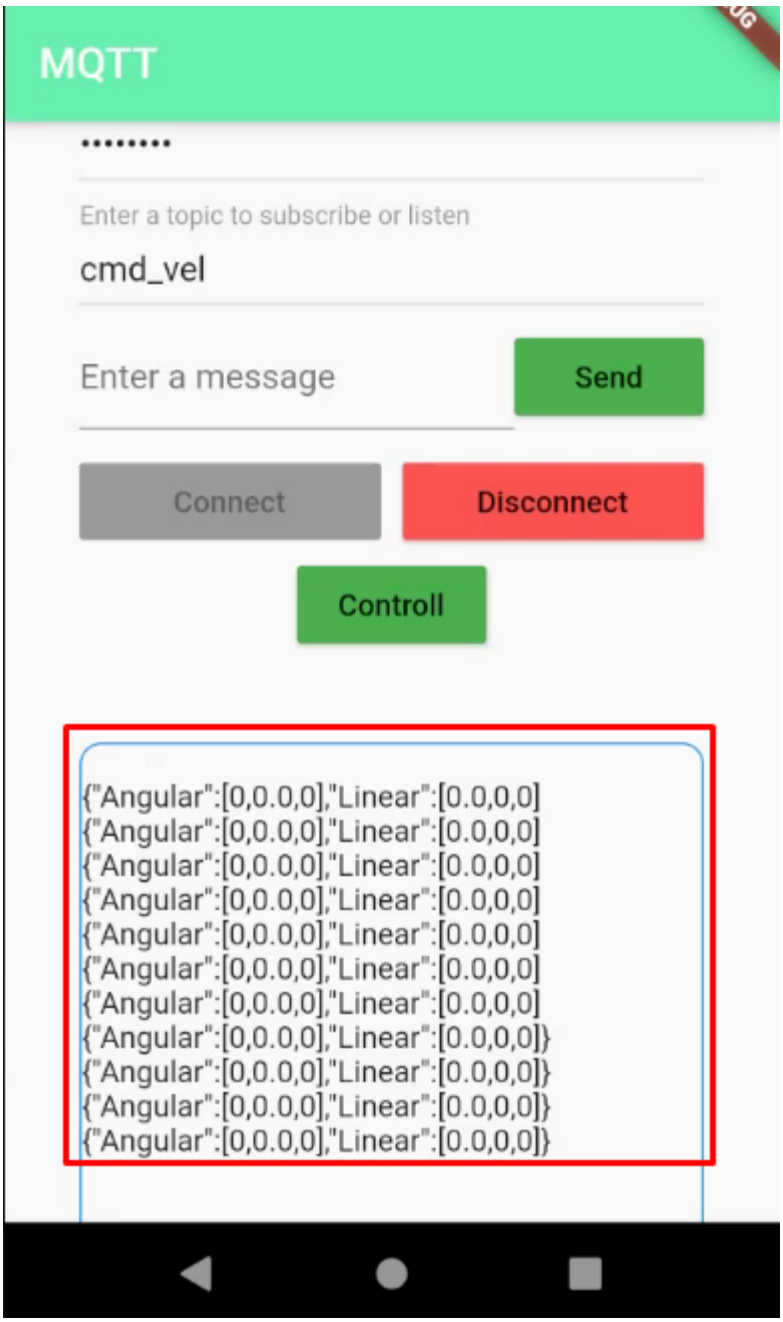


Рисунок 5.3 — Головне меню системи

Одним із головних компонентів кабінету користувач е є меню керування платформою. За його допомогою користувач має змогу задавати швидкість на напрям руху платформи. На рисунку 5.4 зображене меню керування роботизованою платформою.

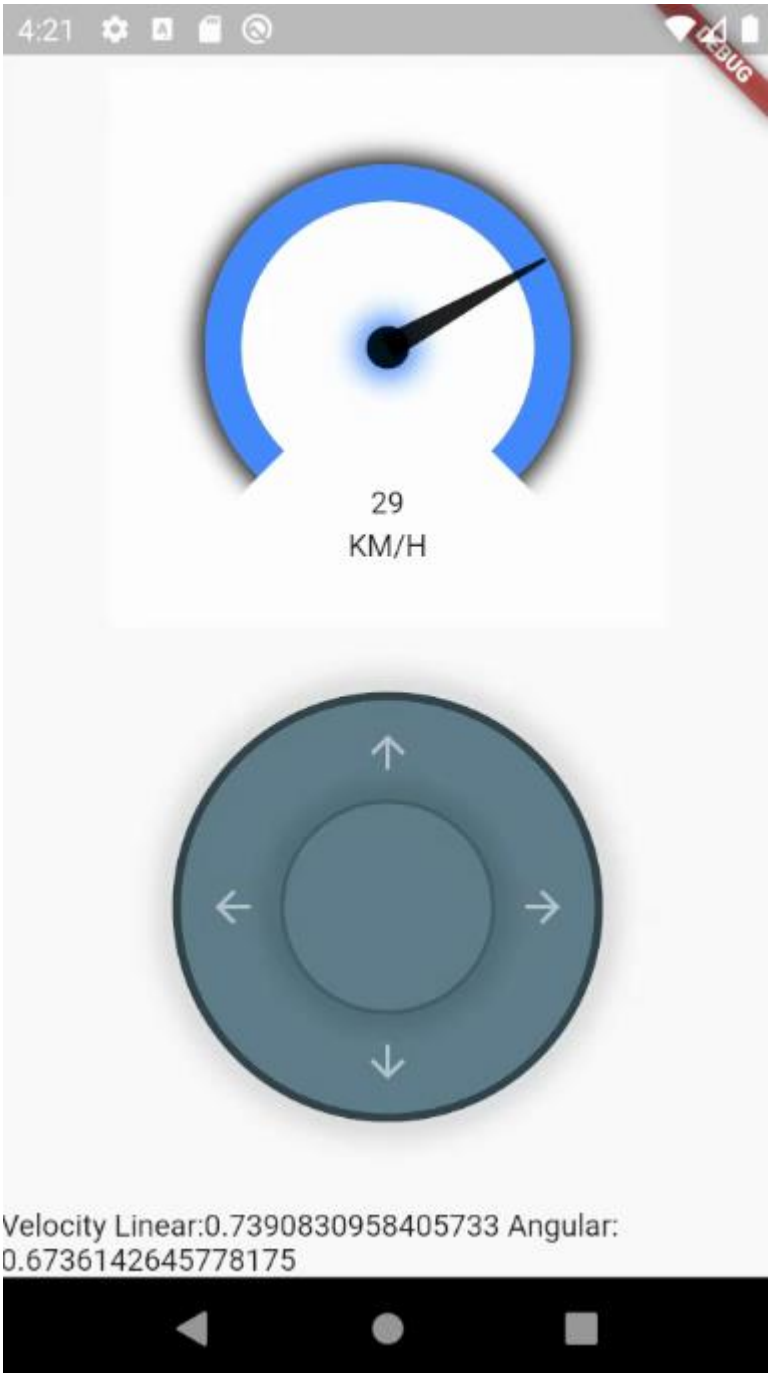


Рисунок 5.4 — Меню керування роботизованою платформою

ВИСНОВКИ

Під час аналізу існуючого програмного забезпечення для управління розподіленою системою були досліджені багатоагентні та мікромережеві системи. Аналіз показав, що існуючі системи не вирішують проблему повністю, вони громіздкі та мають високі вимоги до апаратних засобів для користувачів пристрою.

Розроблений програмно-апаратний комплекс є актуальним, оскільки він був розроблений як частина більшої системи, яка функціонує без втручання людини. Розроблений інтелектуальний агент легко інтегрувати в розроблену систему управління кількома агентами. Розроблений програмний продукт дозволяє автоматизувати роботу та допомагає повідомляти про проблеми чи несправності в системі.

Огляд методів та інструментів розробки програмних систем. Вибір створити програмну систему на основі агентів обміну повідомленнями, а також побудовану на розподіленій архітектурі, виправданий. Це дозволяє збільшити гнучкість та зручність системи, як при розробці, так і в обслуговуванні та використанні.

Результати випробувань програмно-апаратного комплексу підтверджують правильність роботи, що означає відповідність вимогам.

Користувачів системи можна працевлаштувати в передпокої із вбудованими робототехнічними платформами, а також розробників та сервісного персоналу.

Робота над дипломним проектом вдосконалила знання різних технологій, що використовуються при розробці програмного забезпечення. Були також вивчені різні методи та алгоритми машинного навчання, були визначені різні сфери, до яких ці методи можна застосувати. Також було виготовлено декілька програмних прототипів, які вирішили різні аспекти проблеми, а також основу розробленого програмного забезпечення.

5.1 Висновки до розділу

Цей розділ описує інтерфейс мобільного додатку користувача системи. Тут приведено базовий функціонал та основні методи взаємодії з ним.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Holets V. “ROBOTIC PLATFORM FOR SMART SURVEILLANCE AND MANAGEMENT”, International Competition of Student Scientific Works Black Sea Science 2020, Odessa, ONAFT 2020
2. Arun Hampapur, Lisa Brown, Jonathan Connell, Sharat Pankanti, Andrew Senior and Yingli Tian Smart Surveillance: Applications, Technologies and Implications Conference Paper, Jan 2004.
3. Predictive Maintenance 2020 Forum Brochure.
4. Intelligent Maintenance Conference 2020 Program, Sep 2020.
5. Shanghai International Smart Maintenance Engineering Exhibition.
6. Project “STOP” presentation, Aug 2019.
7. Тарасов В.Б. От многоагентных систем к интеллектуальным организациям. Философия, психология, информатика. М., Эдиториал. 2002
8. Wooldridge M.J. An introduction to multi-agent systems. Wiley, 1996.
9. Ghallab M., Nau D., Traverso P. Automated planning: Theory & Practice. Morgan Kaufmann, 2004
10. Bellifemine F, Caire G, Greenwood D (2007) Developing multi-agent systems with JADE. Wiley, London
11. Thomas Bräunl EMBEDDED ROBOTICS
12. https://mosquitto.org/ Mosquitto broker from Eclipse foundation main page.
13. https://www.arduino.cc/reference/en A resource dedicated to Arduino project.
14. https://www.eclipse.org/cdt/documentation.php Eclipse foundation main page.
15. https://www.arduino.cc/reference/en Arduino reference
16. https://www.cplusplus.com/info/description/ C++ language reference page.
17. Benoît Blanchon, The Ultimate Guide to Master ArduinoJson 6.2020.
18. https://knowledge.autodesk.com/support/eagle?sort=score EAGLE CAD reference page
19. https://flutter.dev/docs Flutter reference page

ДОДАТОК 1

Програмно-апаратний комплекс віддаленого управління роботизованою платформою.

Специфікація

КР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6157_20Б

Аркушів 1

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ” КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6160_20Б	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ” КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6160_20Б 12-1	ROSData.h TimerFive.h Encoder.h Error.h PID.h RangeSensor.h	Основні компоненти
УКР.НТУУ” КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6160_20Б 12-2	Додаток 2.doc	Опис програмного модуля

ДОДАТОК 2

Програмно-апаратний комплекс віддаленого управління роботизованою платформою.

Лістинг програмного модулю

КР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6160_20Б

Аркушів 7

Київ 2020

```
#include <Arduino.h>
#include "DataStruct.h"
#include "Definitions.h"
#include "Encoder.h"
#include "PID.h"
#include "RangeSensor.h"
#include "ROSData.h"
#include "TimerFive.h"

ROSData Msg;

PID PIDL(LEFT_P, LEFT_I, LEFT_D);
PID PIDR(RIGHT_P, RIGHT_I, RIGHT_D);
struct Velocity  robotC = Velocity(), robotD = Velocity();
struct Position pos = Position();
struct Enc enc;

volatile struct Data previous;
volatile struct Data current;

RangeSensor LeftS(LEFT_SENSOR_PIN);
RangeSensor RightS(RIGHT_SENSOR_PIN);
RangeSensor FrontS(FRONT_SENSOR_PIN);

Encoder LeftM(LEFT_MOTOR_ENCODER_2_PIN, LEFT_MOTOR_ENCODER_1_PIN);
Encoder RightM(RIGHT_MOTOR_ENCODER_1_PIN, RIGHT_MOTOR_ENCODER_2_PIN);

float leftWC = 0, rightWC = 0, leftWD = 0, rightWD = 0;

void encUpdate();
void TimerISR();
//TODO: create class Motor, compose Encoder object into Motor Object so that it
will be possible to create setPosition() function.
void setup() {
      LeftM.write(0);
      RightM.write(0);

      Serial.begin(115200);

      delay(2000);

      pinMode(RIGHT_MOTOR_DIR_PIN, OUTPUT);
      pinMode(LEFT_MOTOR_DIR_PIN, OUTPUT);

      Timer5.initialize(100000);
      Timer5.attachInterrupt(TimerISR);

}

void loop() {

}

void TimerISR() {
      current.distance[FRONT] = FrontS.getDistance();
      current.distance[LEFT] = LeftS.getDistance();
      current.distance[RIGHT] = RightS.getDistance();

      previous.encoders[LEFT] = current.encoders[LEFT];
      previous.encoders[RIGHT] = current.encoders[RIGHT];
```

```
	current.encoders[LEFT] = LeftM.read();
	current.encoders[RIGHT] = RightM.read();

	encUpdate();

	robotC.v = TWO_PI * ROBOT_R * (enc.left + enc.right) / ROBOT_C / 2.0
			/  DELTA_T;
	robotC.w = TWO_PI * ROBOT_R * (enc.right - enc.left) / ROBOT_C / 2.0
			/ DELTA_T;

	pos.t = atan2(sin(pos.t + robotC.w * DELTA_T),
			cos(pos.t + robotC.w * DELTA_T));
	pos.x = pos.x + robotC.v * DELTA_T * cos(pos.t);
	pos.y = pos.y + robotC.v * DELTA_T * sin(pos.t);

	leftWC = (robotC.v - (ROBOT_B )/ 2 * robotC.w) / ROBOT_R;
	rightWC = (robotC.v + (ROBOT_B )/ 2 * robotC.w) / ROBOT_R;

	leftWD = (robotD.v - ROBOT_B / 2 * robotD.w) / ROBOT_R;
	rightWD = (robotD.v + ROBOT_B / 2 * robotD.w) / ROBOT_R;

	int16_t forceL = PIDL.normalize(PIDL.calc(leftWD, leftWC));
	digitalWrite(LEFT_MOTOR_DIR_PIN, forceL < 0);
	analogWrite(LEFT_MOTOR_PIN, abs(forceL));

	int16_t forceR = PIDR.normalize(PIDR.calc(rightWD, rightWC));
	digitalWrite(RIGHT_MOTOR_DIR_PIN, forceL < 0);
	analogWrite(RIGHT_MOTOR_PIN, abs(forceR));
	Serial.println(
			"X:" + String(pos.x) + " Y:" + String(pos.y) + " T:"+
String(pos.t));

}
void encUpdate() {
	enc.left = current.encoders[LEFT] - previous.encoders[LEFT];
	enc.right = current.encoders[RIGHT] - previous.encoders[RIGHT];
}
#include "PID.h"

#include <Arduino.h>
#include <stdint.h>

PID::PID(float p, float i, float d) {
	this->p = p;
	this->i = i;
	this->d = d;
	this->ProportionalPrev.setType(P_TYPE);
	this->ProportionalPrev.setValue(0);
	this->Proportional.setType(P_TYPE);
	this->Proportional.setValue(0.0);
	this->Integral.setType(I_TYPE);
	this->Integral.setValue(0.0);
	this->Derivative.setType(D_TYPE);
	this->timeNew = millis();
	this->timePrev = millis();
}

float PID::calc(float desired, float real) {
	this->timePrev = this->timeNew;
	this->timeNew = millis();
	//TODO: make one time stamp for all of the objects
```

```
	this->updateErrors(desired, real);
	float result = this->p * this->Proportional.getValue()
			+ this->i * this->Integral.getValue() * DELTA_T
			+ this->d * this->Derivative.getValue() / DELTA_T;
	return result;
}

int16_t PID::normalize(float PIDresult) {
if (abs(PIDresult)>255){
	PIDresult=PIDresult/abs(PIDresult)*255;
}
	return (int16_t) PIDresult;
//TODO:function to normalize this value to analogWrite();
}

void PID::updateErrors(float desired, float real) {
	this->ProportionalPrev.setValue(this->Proportional.getValue());
	this->Proportional.setValue(desired - real);
	this->Integral.next(this->Proportional);
	this->Derivative.next(this->Proportional, this->ProportionalPrev);
//	Serial.println(
//			"ProportionalPrev:" + String(ProportionalPrev.getValue()) +
//			" Proportional:" + String(Proportional.getValue()) +
//			" Integral:" + String(Integral.getValue()) +
//			" Derivative:" + String(Derivative.getValue()));
}

void PID::setD(float d) {
	this->d = d;
}

void PID::setI(float i) {
	this->i = i;
}

void PID::setP(float p) {
	this->p = p;
}

PID::~PID() {
	// TODO Auto-generated destructor stub
}

#include "RangeSensor.h"

#include <Arduino.h>
#include <stdint.h>

RangeSensor::RangeSensor() {

	//Serial.println("No port in RangeSensor object -> while(1){}");
//	while (1) {
//	};
this->pin=0;
}

uint16_t RangeSensor::getValue() {
	uint16_t samples[SAMPLES_SIZE];
	uint16_t average;
	for (int i = 0; i < SAMPLES_SIZE; i++) {
		//samples[i] = analogRead(this->pin);
```

```
            //average += samples[i];
            average += analogRead(this->pin);
      }
      average /= SAMPLES_SIZE;
      /*    if (average > MAX || average < MIN) {
       *          return 0;
       *    }
       */
      /*    for (int i = 0; i < SAMPLES_SIZE; i++) {
       *          if ((samples[i] > (average + ERROR))
       *                      || (samples[i] < (average - ERROR))) {
       *                //dangerous part TODO: safetify
       *                // return getValue();
       *                return 0;
       *          }
       *    }
       */
      return average;
}

uint8_t RangeSensor::getDistance() {
      uint16_t value = this->getValue();
      if (value != 0) {
            return (float) (272600 * pow(this->getValue(), -1.631));
      } else {
            return 0;
      }
}

RangeSensor::RangeSensor(uint8_t pin) {
      if (ANALOG_PIN(pin)) {
            this->pin = pin;
      } else {
            //Serial.println("Wrong port in RangeSensor object -> while(1){}");
            while (1) {
            };
      }
}

RangeSensor::~RangeSensor() {

}
#ifndef DEFINITIONS_H_
#define DEFINITIONS_H_

//RangeSensors pins
#define LEFT_SENSOR_PIN A2
#define RIGHT_SENSOR_PIN A4
#define FRONT_SENSOR_PIN A3

//Encoders pins
#define LEFT_MOTOR_ENCODER_1_PIN 18
#define LEFT_MOTOR_ENCODER_2_PIN 19
#define RIGHT_MOTOR_ENCODER_1_PIN 2
#define RIGHT_MOTOR_ENCODER_2_PIN 3

//Motors pins
#define LEFT_MOTOR_PIN 9
#define RIGHT_MOTOR_PIN 4
#define LEFT_MOTOR_DIR_PIN 6
#define RIGHT_MOTOR_DIR_PIN 5
```

```
//Left wheel PID parameters
#define LEFT_P (40)
#define LEFT_I (350)
#define LEFT_D (2)

//Right wheel PID parameters
#define RIGHT_P (40)
#define RIGHT_I (350)
#define RIGHT_D (2)

//Robot parameters
#define ROBOT_R (1.7/100.0)
#define ROBOT_B (9.5/100.0)
#define ROBOT_C (298.0*12.0*2.33)

//Common delta t
#define DELTA_T (0.1)

//Error
#define P_TYPE 0
#define I_TYPE 1
#define D_TYPE 2

//RangeSensor
#define ANALOG_PIN(pin) (true)
#define SAMPLES_SIZE 10
#define ERROR 10
#define MAX 900
#define MIN 20

#define LEFT 0
#define RIGHT 1
#define FRONT 2

#endif /* DEFINITIONS_H_ */
#include "Error.h"

#include <stdint.h>

Error::Error() {
	// TODO Auto-generated constructor stub
	this->type = P_TYPE;
	this->value = 0;
}

void Error::setType(uint8_t type) {
	this->type = type;
}

Error::~Error() {
	// TODO Auto-generated destructor stub
}

Error::Error(uint8_t type) {
	this->type = type;
	this->value = 0;
}

void Error::setValue(float value) {
this->value=value;
```

```
}

float Error::getValue() {
	return this->value;
}

void Error::next(Error now, Error prev) {
	if (this->type == P_TYPE) {
		this->value = now.getValue();
	} else if (this->type == I_TYPE) {
		this->value+=now.getValue();
	}else if(this->type==D_TYPE){
		this->value=now.getValue()-prev.getValue();
	}
}

#ifndef DATASTRUCT_H_
#define DATASTRUCT_H_

#include <stdint.h>


struct Data{
	uint8_t distance[3];
	int32_t encoders[2];
};
struct Velocity {
	Velocity() {
		v = 0;
		w = 0;
	}
	float v, w;
};
struct Position {
	Position() {
		x = 0;
		y = 0;
		t = 0;
	}
	float x, y, t;
};
struct Enc {
	int32_t left, right;
};
#include "ROSData.h"

#include <Arduino.h>

ROSData::ROSData() {
	// TODO Auto-generated constructor stub

}

float ROSData::getV() const {
	return v;
}

void ROSData::setV(float v) {
	this->v = v;
}
```

```
void ROSData::getData() {
	while (Serial.available()) {
		char temp[2];
		temp[1] = Serial.read();
		temp[2] = Serial.read();
		if (temp[1] == 'v') {
			if (temp[2] == ':') {
				this->setV(Serial.parseFloat());
				while (Serial.available() && Serial.peek() == ' ') {
					Serial.read();
				}
			}
		} else if (temp[1] == 'w') {
			if (temp[2] == ':') {
				this->setW(Serial.parseFloat());
				while (Serial.available() && Serial.peek() == ' ') {
					Serial.read();
				}
			}
		} else {
			while (Serial.available()
					&& (Serial.peek() != 'v' || Serial.peek() != 'w')) {
				Serial.read();
			}

		}
	}
}

float ROSData::getW() const {
	return w;
}

void ROSData::setW(float w) {
	this->w = w;
}

ROSData::~ROSData() {
	// TODO Auto-generated destructor stub
}
```

ДОДАТОК З

Програмно-апаратний комплекс управління роботизованою платформою.

Опис програмного модулю

КР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6160_20Б

Аркушів 8

Київ 2019

Анотація

Розділ містить опис програмної частини, яка слугує для забезпечення керування роботизованою платформою. Цей модуль відповідає за авторизацію користувача та внесення змін до системи.

ЗМІСТ

Загальні відомості

У додатку міститься частина коду що забезпечує підключення користувача до платформи та зміну її характеристик.

Додаток було написано мовами C++ та Dart у середовищах Eclipse та Android Studio відповідно.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

У даній частині програми реалізуються такі функції:

1. Прийом даних авторизації користувача
2. Обробка даних авторизації
3. Перегляд даних платформи
4. Зміна швидкості руху платформи та тестування периферії

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Модуль реалізований у формі набору класів. Він забезпечує зв’язок між користувачем і модулем авторизації та модулем керування рушіями, отримуючи дані, реагуючи на них та відправляючи відповідні команди у реальному часі. Тобто реалізує контроль над платформою.

ТЕХНІЧНІ ЗАСОБИ ЩО ВИКОРИСТОВУЮТЬСЯ

Модуль розроблено у середовищі розробки Eclipse CDT. При роботі програми використовуються такий пристрій як апаратна роботизована платформа, що складається з таких пристроїв як комп’ютер, мікроконтролер Atmega 2560, та периферійні модулі.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Система забезпечує контроль роботи роботизованоъ платформи та може працювати паралельно з нею. Запуск платформи выдбувається на декілька хвилин раніше від додатку через необхідну ініціалізацію.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними є інформація, яку надсилає користувач при авторизації. Це може запит, що містить у собі дані авторизації користувача.

Вихідними ж є вплив на характеристичні данні платформи (наприклад, зміна швидкості) у разі якщо цього дозволяє регламент доступу.