МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

(повне найменування назва факультету)

дискретного аналізу та інтелектуальних систем

(повна назва кафедри)

ДИПЛОМНА РОБОТА

РОЗРОБКА ХМАРНОГО ЗАСТОСУНКУ ДЛЯ КЕРУВАННЯ РОБОТИ РОБОТІВ

Виконала: студентка групи ПМІ-45с

спеціальності 122 «Комп'ютерні науки»

(шифр і назва спеціальності)

Панас Ю.О.

(підпис)(прізвище та ініціали)

Керівник доцент Коковська Я.В.

(підпис) (прізвище та ініціали)

Рецензент

(підпис) (прізвище та ініціали)

Львів – 2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики

Кафедра Дискретного аналізу та інтелектуальних систем

Спеціальність 122 «Комп’ютерні науки»

(шифр і назва)

«ЗАТВЕРДЖУЮ»

Завідувач кафедри Притула М.М.

"31 "серпня 2022 року

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

ПАНАС ЮЛІЇ ОЛЕКСАНДРІВНИ

(прізвище, ім'я, по батькові)

1 **.Тема роботи**

Розробка хмарного застосунку для керування роботи роботів

керівник роботи Коковська Я.В. ,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені Вченою радою факультету від **" 13" вересня 2022 року № 15**

2. Строк подання студентом роботи **13.06.2023р.**

3. Вихідні дані до роботи

Мови програмування Python, TypeScript. Середовище розробки Visual Studio Code.

4. Зміст дипломної роботи (перелік питань, які потрібно розробити)

1. Розробити мікросервісну архітектуру
2. Теоретичні основи роботи з MQTT брокером
3. Вивчення протоколу CAN Open
4. Теоретичні основи роботи з Jetson Nano

5. Розробка зручного графічного інтерфейсу.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Діаграма архітектури програмного забезпечення. Скріншоти всіх сторінок застосунку. А саме сторінок для контролю моторами, сенсорами, гальмами та девайсами. Сторінка для перепрошивки девайсів та контролю повідомлень з шини. Зображення повідомлень

6. Консультанти розділів роботи

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

7. Дата видачі завдання **31 серпня 2022 р.**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Архітектура мікросервісів та створення відповідних сервісів	01.10.2022 - 01.11.2022	

2	Дослідження контейнеризації та використання Docker для ефективної роботи з сервісами	01.11.2022-17.11.2022	
3	Дослідження протоколу Canopen для взаємодії з мережевим обладнанням	17.11.2022 - 15.12.2022	
4	Обробка та аналіз даних шини	15.12.2022 - 31.12.2022	
5	Отримання та відображення даних	10.01.2023 - 01.02.2023	
6	Опис та розробка зручного графічного інтерфейсу	01.02.2023 - 01.04.2023	
7	Перепрошивка девайсу	01.04.2023 - 20.04.2023	
8	Jetson Nano	20.04.2023 - 25.04.2023	
9	Передача повідомлення та комунікація між сервісами через MQTT брокер	25.04.2023 - 30.04.2023	

ЗМІСТ

Вступ 4

Використані технології 5

Розділ 1 Архітектура мікросервісів та створення відповідних сервісів 7

Розділ 2 Дослідження контейнеризації та використання Docker для ефективної роботи з сервісами 8

Розділ 3 Дослідження протоколу Canopen для взаємодії з мережевим обладнанням 9

Розділ 4 Обробка та аналіз даних шини 14

Розділ 5 Отримання та відображення даних 16

Розділ 6 Опис зручного графічного інтерфейсу 17

Розділ 6.1 Зміна Node id 17

Розділ 6.2 Доступ SDO 17

Розділ 6.3 Unmute 18

Розділ 6.4 Reset, Command Mode, Auto Command Mode 18

Розділ 6.5 Сторінки контролю даних 19

Розділ 7 Перепрошивка девайсу 22

Розділ 8 Jetson Nano 26

Розділ 9 Передача повідомлення та комунікація між сервісами через MQTT брокер 28

Висновки 30

Додатки 32

Список Літератури 41

ВСТУП

Актуальність даної роботи та оцінка проблеми :

Сучасний світ наповнений різноманітними технологіями, що стали важливою частиною нашого життя, починаючи від телефонів, що в наших кишенях та й до великих систем. Щоб все працювало правильно, без перебоїв та завжди було актуальним, оновлення програмного забезпечення є надзвичайно важливою частиною цього процесу і навіть не потрібно згадувати якою важливо є сумісність версій в цьому процесі.

Дана дипломна робота пропонує рішення розробки хмарного застосунку, використовуючи мікросервісну архітектуру для контролю версій програмного та апаратного забезпечення, стану девайсу та можливості оновлення програмного забезпечення .

Проблема:

Правильне використання роботів вимагає розуміння конструкції, а саме, як працюють мотори, сенсори, світло та гальма. Якщо використовувати різні версії моторів або сенсорів можемо зіткнутись з проблемою сумісності версій або невідповідальністю конфігурацій, що може створити багато труднощів або навіть небезпеки у використанні.

Мета роботи :

Створити багатофункціональний хмарний застосунок, для роботи з роботом. Передбачити можливість отримання програмного, апаратного забезпечення, назви девайсу, перепрошивки девайсів одного типу. Застосувати контейнери для створення мікросервісів. Створити сторінку для відображення доступних пристроїв.

ВИКОРИСТАНІ ТЕХНОЛОГІЇ ТА ПРИСТРОЇ

1. Екосистема контейнерів:
   - Docker
2. Мови програмування:
   - Python
   - TypeScript
3. Фреймворки:
   - Django
   - FastAPI
   - React
4. Черга асинхронних завдань:
   - Celery
5. Протоколи обміну інформацією:
   - WebSocket
   - MQTT
   - SocketCAN
   - PDO
   - SDO
6. Сховища:
   - PostgresSQL
   - Redis
7. Мікрокомп’ютер в роботі:
   - Jetson Nano
8. Хмарні сервіси:

- Bitbucket
- Docker Hub

РОЗДІЛ 1. АРХІТЕКТУРА МІКРОСЕРВІСІВ ТА СТВОРЕННЯ ВІДПОВІДНИХ СЕРВІСІВ

Для створення легкого та простого у використанні хмарного застосунку, продумана архітектура - є важливою частиною процесу. Оскільки комунікація з даними робота буде відбуватись на пряму, це вимагає програмі знаходитись в самому роботі, але знаходження цілого проекту в роботі немає абсолютно ніякого сенсу, таким чином монолітний спосіб написання, не тільки в даному випадку несумісний, так ще й зовсім не актуальний. Робота з мікросервісами - це саме те, що потрібно, зручно просто та безпечно, і якщо один з сервісів перестане відповідати, це не зупинить роботу системи. Отже, створюємо сервіс для самого фронтенду, для детектора існування девайсу, для створення об'єкту перепрошивання і отримання даних та для самої комунікації з девайсом. Наступний крок - вибір протоколу обміну інформації. Одним з найкращих та найпопулярніших протоколів обміну інформації в системах розумних будинків, промислової автоматизації , всюди, де потрібен надійний зв'язок між програмами є MQTT (Message Queuing Telemetry Transport). Отже, наші сервіси будуть публікувати та підписуватись на теми. Для обробки багатьох даних одночасно використовуємо Celery, для створення черги асинхронних завдань та Redis, як брокер повідомлень між Django та Celery. Для швидкого отримання даних використовуємо WebSocket, відкриваючи канали зв'язку.

Важливо розуміти, що всі сервіси бекенду мають з'єднання з MQTT. Таким чином розглянути цілу архітектуру можна в додатку 1 , де детально показано як сервіси комунікують між собою і що використовують.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ КОНТЕЙНЕРИЗАЦІЇ ТА ВИКОРИСТАННЯ DOCKER ДЛЯ ЕФЕКТИВНОЇ РОБОТИ З СЕРВІСАМИ

За допомогою Docker ми "обгортаємо" нашу програму. Контейнер вміщує вихідний код програми з усіма залежностями, такими як системні бібліотеки , двійкові файли, а також зовнішні пакети, фреймворки та інше. Робота з контейнерами дозволяє працювати з ними абсолютно всюди : на будь-якому комп'ютері, інфраструктурі чи хмарі без проблем із сумісністю.

Кожен сервіс розміщений в контейнері збережений в сховищі DocketHub для подальшого використання. Також я користуюсь вже готовими шаблонами контейнерів, такі як Redis, PostgresSQL та Mosquitto. Також мікрокомп'ютер на роботі теж складається з багатьох контейнерів, які комунікують між собою.

Отже, Docker допомагає створити з програм маленькі служби з своїм функціоналом, що полегшує роботу з сервісами, оскільки ми концентруємо свою увагу тільки на одному функціоналі, якщо сервіс перестає працювати, це не суттєво впливає на всю програму. Також ми можемо легко перезапускати, видаляти та додавати сервіс, не впливаючи на інші сервіси на нашій машині.

РОЗДІЛ 3. ДОСЛІДЖЕННЯ ПРОТОКОЛУ CAN ДЛЯ ВЗАЄМОДІЇ З МЕРЕЖЕВИМ ОБЛАДНАННЯМ

Мережевий драйвер CAN надає загальний інтерфейс для встановлення, налаштування та моніторингу пристроїв CAN.

CAN – Control Area Network. Шина була створена для спілкування між пристроями за допомогою повідомлень. Система стійка до електричних перешкод і електромагнітних перешкод - ідеально підходить для важливих для безпеки додатків (наприклад, транспортних засобів або роботів в нашому випадку). Кадри CAN мають пріоритет за ідентифікатором, щоб дані з найвищим пріоритетом отримували миттєвий доступ до шини, не викликаючи переривання інших кадрів або помилок CAN. Дані містять байти даних, також корисне навантаження, яке включає сигнали CAN, які можна витягнути та декодувати для отримання інформації. CRC це циклічна перевірка надмірності використовується для забезпечення цілісності даних.

Як декодувати необроблені дані CAN у “фізичні значення”? Якщо ви переглянете додаток 2 необроблених даних шини CAN, ви, ймовірно, помітити що, необроблені дані є нечитабельними.

Щоб отримати фізичне значення сигналу CAN, потрібна така інформація:

- Початок біта: з якого біта починається сигнал
- Довжина в бітах: довжина сигналу в бітах
- Зсув: Значення, на яке зміщується значення сигналу
- Масштаб: Значення, на яке потрібно помножити значення сигналу

Щоб отримати сигнал CAN, ви «поділяєте» відповідні біти, берете десяткове значення та виконуєте лінійне масштабування: physical_value = offset + scale * raw_value_decimal.

CANopen протокол широко використовується у вбудованих програмах керування, в тому числі наприклад промислова автоматизація. Він заснований на CAN, тобто реєстратор даних шини CAN також може реєструвати дані CANopen. Це ключове значення, наприклад діагностика машин або оптимізація виробництва.

Загалом CANopen забезпечує надійну та стандартизовану комунікаційну структуру для промислової автоматизації, забезпечуючи бездоганну інтеграцію пристроїв і сприяючи ефективному обміну даними в складних мережевих середовищах.

РОЗДІЛ 4. ОБРОБКА ТА АНАЛІЗ ДАНИХ ШИНИ

Щоб зрозуміти комунікацію CANopen, необхідно розбити структуру CANopen CAN:

- 11-бітний CAN ID називається ідентифікатором об’єкта зв’язку (COB-ID) і розділений на дві частини:

Щоб зрозуміти, як працює COB-ID, давайте почнемо з попередньо визначеного розподілу ідентифікаторів, які використовуються в простих мережах CANopen

Ідентифікатори COB (наприклад 381, 581, ...) пов'язані зі службами зв'язку (передати PDO 3, передати SDO, ...). Додаток 3.

За замовчуванням перші 4 біти дорівнюють коду функції, а наступні 7 бітів містять ідентифікатор вузла. Таким чином, COB-ID деталізує, який вузол надсилає/отримує дані та яка послуга використовується.(Додаток 2)

Давайте дослідимо додаток 3 та додаток 4. На додатку 3 ми можемо побачити різні комунікаційні об’єкти такі як PDO (Process Data Object), SDO (Service Data Object), NMT (Network management), та інші. Що ж нам важливо, та як власне відбувається комунікація. Давайте розглянемо Додаток 4.

Повідомлення з ідентифікатором 705, 706 - це серцебиття, саме це повідомлення відображає, що девайс активний, якщо ми повернемось до додатку 3 то можемо побачити, останній рядок “HEARTBEAT” з CAN ID 701-77FF. Все працює доволі просто.

NMT періодично надсилає (наприклад, кожні 100 мс) повідомлення Heartbeat (наприклад, з CAN ID 705 для вузла 5) із «станом» вузла в 1-му байті даних.

«Споживач» повідомлення Heartbeat (наприклад, головний NMT і за бажанням, будь-який інший пристрій) потім реагує, якщо повідомлення не отримано протягом певного ліміту часу.

Тому на Додатку 7 за відображення активності девайсів відповідають саме ці повідомлення.

Тепер важливо згадати про словник об’єктів.

Усі вузли CANopen повинні мати словник об’єктів (OD).

Словник об’єктів — це стандартизована структура, що містить усі параметри, що описують поведінку вузла CANopen. Записи OD шукаються за допомогою 16-бітного індексу та 8-бітового субіндексу. Наприклад, індекс 1008 (субіндекс 0) CANopen-сумісного вузла OD містить ім’я пристрою вузла.

Зокрема, запис у словнику об’єктів визначається атрибутами:

- Індекс: 16-бітна базова адреса об'єкта
- Назва об’єкта: назва пристрою виробника
- Об'єктний код: масив, змінна або запис
- Тип даних: наприклад VISIBLE_STRING, UNSIGNED32 або назва запису

Дуже важливими повідомленнями є PDO та SDO.

SDO дозволяє вузлу CANopen читати/редагувати значення словника об’єктів іншого вузла через мережу CAN. Та «Моделі зв’язку», служби CANopen SDO використовують поведінку «клієнт/сервер». Зокрема, "клієнт" SDO ініціює зв'язок з одним виділеним "сервером" SDO.

У нашому випадку один з клієнтів є наш контейнер, що читає та обробляє інформацію та іншим є девайс. Наша мета полягає в тому, щоб прочитати запис («завантажити SDO»).

Давайте розглянемо Додаток 5. Повідомлення з ідентифікатором 59B є SDO, так як воно містять інформацію, яка має бути оброблена та прочитана. 61B це повідомлення, яке містить команду до завантажувача, у протоколі CANopen, називають PDO.

Давайте подивимось уважно, що 5 рядок - це відповідь на запит отримання апаратного забезпечення, що вже може бути нами прочитана, оскільки hex(05)=5 . Тобто апаратне забезпечення нашого девайсу дорівнює 5. Це доволі простий та зрозумілий приклад. Якщо спробувати отримати назву девайсу або програмне забезпечення, інформацію буде неможливо зрозуміти в чистому вигляді.

PDO використовується для ефективного обміну робочими даними в реальному часі між вузлами CANopen. Наприклад, PDO буде передавати дані про тиск від датчика тиску або дані про температуру від датчика температури.

Можливо виникає питання чому SDO не може цього робити.

В принципі, для цього можна використовувати послугу SDO. Але одна відповідь SDO може містити лише 4 байти даних через накладні витрати (командний байт і адреси OD). Крім того, скажімо, головному вузлу потрібні два значення параметра (наприклад, «SensTemp2» і «Torque 5») від вузла 5 — щоб отримати це через SDO, йому знадобиться 4 повних кадри CAN (2 запити, 2 відповіді).

Повідомлення PDO може містити 8 повних байтів даних і може містити кілька значень параметрів об’єкта в одному кадрі. Таким чином, те, що вимагає принаймні 4 кадрів із SDO, потенційно може бути зроблено з 1 кадром у службі PDO.

PDO часто розглядається як найважливіший протокол CANopen, оскільки він переносить основну частину інформації.

Для PDO використовується термінологія споживач/виробник. Таким чином, виробник «виробляє дані», які він передає «споживачу» (основному) за допомогою PDO передачі (TPDO). І навпаки, він може отримувати дані від споживача через отримання PDO (RPDO).

Розглянемо Додаток 6. Зверніть увагу, як байти даних упаковані з 3 значеннями параметрів. Ці значення відображають дані в реальному часі про певні записи OD вузла 5. Вузли, які використовують цю інформацію (споживачі), звичайно, повинні знати, як інтерпретувати байти даних PDO.

Якщо говорити про програмну реалізацію комунікації з шиною, то

Розроблено :

1. CANManager для комунікації з шиною.

Реалізовано методи : add_connection (для з'єднання з шиною) , subscribe (підписка на конкретні повідомлення), send_messages(відправлення повідомлень на шину), run(постійно прослуховує повідомлення, та викликає відповідні зворотні виклики).

2. CANSubscription(context manager), який містить вхід та вихід з підписки повідомлень.

Після отримання ідентифікатора девайсу, ми починаємо збір інформації, а саме ім'я девайсу, апаратне забезпечення та програмне забезпечення.

Створюємо об'єкт GenericClient що містить важливі нам методи, такі як read_device_name(), read_hw_id(), read_sw_version(). Підписуємось на повідомлення,

що надсилає девайс, викликаємо один із цих методів GenericClient, в методі створюємо об’єкт SDO з потрібною нам командою та надсилаємо на шину.

Тепер слід отримати та обробити повідомлення.

Створюю клас CanOpenListener, де власне отримую всі повідомлення, та базуючись на сервері функціонального коду призначаю, який саме це тип, та додаю в чергу повідомлень.

Як тільки ми підписуємось ми починаємо новий потік з прослуховуванням та обробкою абсолютно всіх повідомлень, що потрапляють в чергу з конкретним Functional code(PDO або SDO).

РОЗДІЛ 5. ОТРИМАННЯ ТА ВІДОБРАЖЕННЯ ДАНИХ

В попередніх розділах ми детально обговорили, роботу шини та розглянули повідомлення, які надсилаються та отримуються. Обговорили важливість PDO та SDO та головну різницю. Тому перейдемо до візуального відображення даних.

На Додатку 7 можна звернути увагу, що девайси кутових сенсорів та моторів містять детальну інформацію. Девайси гальм та контролю батареї є недоступними, оскільки вони знаходяться на іншій шині до якої ми не приєднані.

За серцебиття, як ми раніше згадували відповідають повідомлення з повідомленням [05]. Якщо розглянути додаток 8, то це сторінка відображення девайсів, які доступні та мають конкретне програмне забезпечення і не знаходяться в режимі завантажувача. Тобто інформація яких в сирому вигляді виглядає [00 00 00 00 00 00 00 00] або [00 00 00 00 00 00 80].

Щоб фронтенд відображав ці повідомлення, сервіс, що комунікує з девайсами, публікує всі дані на конкретну тему MQTT брокера і фронтенд просто зчитує ці повідомлення , підписуючись на цю тему.

Для отримання даних таких як тип девайсу, апаратне та програмне забезпечення, нам слід клацнути на девайс. Frontend відкриє WSS з’єднання з одним з наших сервісів, який проаналізує дані та відправить в асинхронну чергу, яка відправляє запит на сервіс, що знаходиться на роботі, через MQTT брокера. Як тільки, ми отримаємо потрібну нам інформацію, все повернеться в тому ж порядку тільки навпаки, спочатку в асинхронну чергу і потім через канал зв’язку до frontend . Канал відкритий до тих пір поки наш девайс вибраний.

РОЗДІЛ 6. ОПИС ЗРУЧНОГО ГРАФІЧНОГО ІНТЕРФЕЙСУ

В попередньому розділі, ми почали перегляд графічного інтерфейсу відображення даних, щоб продовжити цю тему, пропоную пройтись детально по функціоналу, який ми бачимо на додатках 7 та 8.

6.1 Зміна Node Id

Перше на що я хочу звернути увагу це частина з правої сторони Change Node id. Node id це ідентифікатор нашого девайсу, який ми можемо побачити з лівої сторони девайсу(05, 06, 1B, 1C, 1D, 1E). Ця функція потрібна для зміни, наприклад правого мотору на лівий або переднього на задній, чи будь-яка інша комбінація, так само це працює з кутовими сенсорами та гальмами. Для цього потрібно, просто вказати поточний ідентифікатор на новий і натиснути кнопку “Change Id”. Це корисно, як і для тестування так і для самої розробки.

6.2 Доступ SDO

Наступною частиною є SDO Access, потрібно для отримання даних в сирому форматі, якщо сервіс не відповідає. Щоб отримати дані нам потрібно задати ідентифікатор девайсу та index та subindex, які відомі зазвичай тільки розробникам, а саме embedded engineers або robotics engineers.

Індекс: індекс представляє категорію або групу вищого рівня, що містить пов’язані дані або функції. Слугує способом організації та групування кількох субіндексів. У випадку завантажувача індекс може посилатися на сам об’єкт завантажувача або певну категорію в межах функціональності завантажувача.

Наприклад, BOOTLOADER_INDEX у моєму коді представляє основний індекс об’єкта завантажувача.

Субіндекс — це більш конкретний ідентифікатор або адреса в індексі. Він надає спосіб доступу або посилання на окремі фрагменти даних або певні функції в індексі. Субіндекси зазвичай використовуються, коли з певним індексом пов’язано кілька елементів даних або функцій. У випадку завантажувача субіндекси можуть представляти різні аспекти роботи чи конфігурації завантажувача. Наприклад, BOOTLOADER_STATE_SUBINDEX представляє підіндекс, який вказує на стан завантажувача.

Як тільки ми задали дані натискаємо кнопку “Read” та отримаємо в нашому вікні відповідь на запит. Також ми можемо у вікні і написати команду та натиснути кнопку “Write” та відправити команду на самий девайс.

6.3 Unmute

Дальше розглянемо кнопку “Unmute”, кожного разу, коли ми перепрошиваємо девайси, ми заглушуємо ті девайси, які нам не є важливими, для уникнення надто великої кількості повідомлень, що можуть призвести до помилки. Тож після успішного оновлення програмного забезпечення, ми натискаємо кнопку “Unmute”, що в свою чергу відправляє повідомлення нашому девайсу про те, що ми готові знову прослуховувати всі повідомлення.

6.4 Reset, Command Mode, Auto Command Mode

Наступними є кнопки Reset, Command Mode, Auto Command Mode(Disable).

Reset – це перезавантаження нашого девайсу.

Command mode – девайс в режимі завантажувача.

Найпростіше пояснення, якщо ви раніше не стикались, з завантажувачем то це програма, що виконує завантаження операційної систем.

Якщо забігти трішки наперед, то саме в цьому режимі і відбувається завантаження нового програмного забезпечення.

Отже, щоб мати цілу картину, того для чого використовується завантажувач, на роботі:

- Перетворення та очищення даних: Завантажувач може надавати можливості для трансформації та очищення даних перед завантаженням. Може включати фільтрацію, об'єднання, розрахунок похідних значень та інші операції, що забезпечують якість та готовність даних для аналізу.
- Планування та автоматизація: Завантажувач може мати можливості планування та автоматичного запуску процесу завантаження в задані часи або за певних умов. Дозволяє забезпечити регулярне та автоматизоване оновлення даних.
- Моніторинг та керування: Завантажувач може надавати інструменти для моніторингу процесу завантаження, слідкування за успішністю, виявлення помилок та можливості керування процесом.

Ще одна кнопка Auto Command Mode (Disabled), на якій видно що ми в режимі завантажувача, чи ні (Disable or Unable). Якщо так, то ми можемо перейти до режиму самої програми натиснувши цю кнопку.

6.5 Сторінки контролю даних

Також з лівої сторони, можна побачити список сторінок, перша з яких це Overview, додаток 8. Раніше вже згадували, що на цій сторінці відображаються всі

девайси, які надсилають повідомлення з конкретним ідентифікатором. Це девайси знаходяться в режимі програми, а не просто завантажувача.

Наступною сторінкою є Operations Control (додаток 9) . Базуючись на повідомленнях, які ми отримуємо з шини, можемо прослідкувати швидкість наших моторів, що видно на сторінці.

Power Control(додаток 10) . Інформацію про батарею ми отримуємо, коли з'єднання налаштовано з конкретною шиною. Ми можемо побачити кількість батареї, напругу, відносну вологість та температуру.

Також на сторінці є інформація про потужність колес.

Angel sensors – кутові сенсори(додаток 11). Можемо побачити кут переднього та заднього сенсорів, та їхні різниці в поточному стані. Контроль - дає можливість тестувати, якщо натиснути кнопку Calibrate, також є магнітне поле.

Motor control - управління моторами (додаток 12). На даній сторінці, можна проаналізувати дані, що надходять з наших моторів, це пришвидшення, вісь обертання, струм, напруга та інше для кожного двигуна. Переглянути швидкість та ідентифікатор. Також можна відправити запит на швидкість обертання. Можна обрати конкретний девайс або всі разом. Якщо переглянути додаток (13.1), то можна зрозуміти, що дуже багато різної інформації надходить для аналізування.

Ultrasounds, Brakes and Lights - Ультразвук, гальма та світло(додаток 13, 13.1) . На цій сторінці можна також отримати дані, такі як статус, дистанція, статус гальм, світловий режим, режим гальм. Ми можемо почати залучати або вилучати гальмівний режим, почати передачу або скинути до загальний базових налаштувань.

Bus Traffic - додаток 14. На цій сторінці можна проаналізувати та переглянути швидкість надходження повідомлень з інформацією від кожного девайсу, а саме тих які є доступними .

“Devices”, ми вже розглядали(додаток 7).

ROS Overview - додаток 15. На даній сторінці, ще нічого не має, але планується додатись контроль роботизованої операційної системи.

Sidearm test – тестування. Сторінка показує, що мотори підключені, а гальма ні, оскільки вони знаходяться на різних шинах. Так як тільки робот починає створюватись і ще все не налаштовано правильно, ми використовуємо цю сторінку тільки для перевірки з’єднання.

РОЗДІЛ 7. ПЕРЕПРОШИВКА ДЕВАЙСУ

До цього розділу ми вже ознайомились з роботою CAN протоколу, протоколів PDO та SDO, розглянули які дані шина нам надсилає та що ми можемо робити з цими даними. Також ознайомились з девайсами та загальним функціоналом нашого додатку. Залишилось кілька деталей про які ми ще не згадували і один з них це перепрошивання девайсу.

Оновлення програмного забезпечення девайсів, це надзвичайно важлива частина застосунку і одна з головних. Як працює перепрошивання девайсів, та як воно виглядає на сторінці?

Вигляд сторінок під час перепрошивки до та після можна побачити в додатках 17, 17.1 і 17.2.

Першим кроком перепрошивання є вибір девайсів, які ми плануємо перепрошити. Це мають бути або контролери, або кутові сенсори, або гальма чи просто батарея. Важливо дочекатись отримання даних, щоб бути впевненим, яке апаратне забезпечення ми використовуємо та для підтвердження типу девайсу.

Тож після того, як ми отримали інформацію про девайс, ми заповнюємо поля, перше це потрібно вибрати архівований файл з новим програмним забезпеченням, наступним є апаратне забезпечення та тип девайсу. Та натискаємо кнопку “Flash Multiple”, якщо це кілька девайсів, або якщо це один девайс “Flash”. Перед тим як відправити дані на backend, рахуємо контрольну суму, та відправляємо її разом з іншими даними.

Контрольна сума - це спосіб гарантувати, що файл не містить помилок.

Також ми перераховуємо контрольну суму файлу на сервісі, для того, щоб бути впевненими, що файл не було зіпсовано, та він не містить помилок і перевіряємо з контрольною сумою, яка була надіслана.

Програмна реалізація методу прошивання:

Функція приймає кілька параметрів: список об'єктів пристрою, клієнт для трансляції, шлях до файлу прошивки, ідентифікатор помічника, ідентифікатор обладнання, тип пристрою, об'єкт обробника CAN і черга для зберігання відсотків виконання, що ініціалізує змінну прогресу та додає її початкове значення (0,0) до черги зберігання відсотків.

Потім програма готує файл прошивки, створюючи тимчасовий двійковий файл, перетворюючи його на байтовий масив і визначаючи його розмір .

Дальше ми переводимо усі пристрої в командний режим завантажувача та зчитуємо версію програмного забезпечення кожного пристрою. Залежно від версії програмного забезпечення він встановлює затримку блимання .

Виконується цикл, щоб очікувати, поки всі пристрої переходять у стан завантажувача, або поки не буде досягнуто тайм-ауту в 5 секунд.

Функція надсилає повідомлення PDO про початок завантаження на кожен пристрій, щоб почати процес завантаження мікропрограми.

Інший цикл виконується, щоб очікувати, поки пристрої завершать завантаження мікропрограми або виявлять помилку. Перевіряємо кожного пристрою та додаємо пристрої в стані очікування до списку waiting_devices. Цикл переривається, якщо всі пристрої очікують додаткових даних мікропрограми або досягнуто тайм-ауту в 5 секунд.

Якщо пристроїв немає або всі пристрої завершили завантаження мікропрограми, прогрес встановлюється на 9.0, ставиться в чергу і функція завершує виконання.

Якщо все ще є пристрої, які очікують додаткових даних мікропрограми, функція продовжується. Ми реєструємо повідомлення про те, що завантажувач готовий отримувати дані.

Функція готується до надсилання сегментів прошивки шляхом ініціалізації змінних для відстеження прогресу.

Виконує ітерацію по масиву байтів файлу, буферизуючи 7 байтів за раз і надсилаючи їх як повідомлення PDO сегмента завантаження на пристрої. Прогрес оновлюється залежно від кількості оброблених байтів.

Після надсилання всіх сегментів функція обчислює CRC (перевірка циклічної надлишковості) для файлу прошивки.

CRC надсилається на кожен пристрій як повідомлення PDO про завершення завантаження.

Інший цикл виконується для очікування, поки всі пристрої завершать процес мигання, або поки не буде досягнуто тайм-ауту в 10 секунд.

Якщо всі пристрої завершили процес мигання, прогрес встановлюється на 100,0, ставиться в чергу в percentage_queue, і цикл завершує свою роботу.

Якщо процес мигання не вдається протягом часу очікування, прогрес встановлюється на 95,0, реєструється як помилка, ставиться в чергу , і цикл завершується.

Як тільки перепрошивка завершилась, ми бачимо “Completed” в колонці Bootloader додатку 17.2.

Після цього потрібно натиснути кнопку Unmute, щоб знову отримувати всі дані заглушених об’єктів.

Вся робота з шиною відбувається на сервісі capra-can. Щоб отримувати деталі девайсу, надсилати команди, читати та публікувати повідомлення з шини та для того, щоб перепрошити девайс, я використовую багатопоточність.

РОЗДІЛ 8. JETSON NANO

Цей розділ присвячений, розуміти, де саме знаходиться сервісів, що відповідає за перепрошивка, отримання даних та й в загальному за комунікацію з шиною.

Почнемо з загальновідомих фактів. Jetson Nano — це невелика малопотужна комп’ютерна плата, розроблена компанією NVIDIA спеціально для периферійних обчислень і додатків ШІ. Він є частиною сімейства NVIDIA Jetson, яке складається з потужних платформ розробки з підтримкою ШІ.

Архітектура: Jetson Nano побудовано на основі архітектури GPU NVIDIA Maxwell із 128 ядрами CUDA, що забезпечує високопродуктивні обчислювальні можливості для робочих навантажень штучного інтелекту.

ЦП: чотириядерний процесор ARM Cortex-A57 із тактовою частотою 1,43 ГГц забезпечує загальну обчислювальну потужність для запуску програм і виконання системних завдань.

Пам'ять: плата має 4 ГБ оперативної пам'яті LPDDR4, що забезпечує ефективне керування пам'яттю та багатозадачність.

Прискорення AI: Jetson Nano містить спеціальний апаратний кодер і декодер, відомий як Video Engine, який забезпечує прискорену обробку відео та AI. Він підтримує різні фреймворки AI, включаючи TensorFlow, PyTorch і Caffe.

Підключення: плата пропонує кілька варіантів підключення, включаючи Gigabit Ethernet, USB 3.0, USB 2.0, HDMI і DisplayPort.

Зберігання: для зберігання Jetson Nano має слот для карт microSD, що дозволяє користувачам використовувати зовнішню карту microSD для операційної системи та зберігання даних.

Розширення: плата має 40-контактний роз'єм GPIO (вхід/вихід загального призначення), що дозволяє підключати додаткові датчики, периферійні пристрої та плати розширення для розширення можливостей пристрою.

Програмне забезпечення: Jetson Nano підтримує NVIDIA JetPack SDK, який надає комплексне програмне забезпечення, включаючи платформу паралельних обчислень CUDA, бібліотеки та інструменти розробки для розробки та розгортання ШІ.

Jetson Nano популярний серед розробників, аматорів і дослідників, які цікавляться додатками штучного інтелекту та робототехніки. Він пропонує баланс між продуктивністю, енергоефективністю та доступністю, що робить його доступним для широкого кола проектів і випадків використання.

Отже, цей маленький могутній комп'ютер на роботі, містить в собі велику базу різних контейнерів, з якими працюють інженери робототехніки. Один з цих контейнерів містить моє зображення, тобто мій Docker container, який є стягнути з DockerHub. Та налаштований, для роботи. Також важливим кроком, є перенесення шини в контейнер, для того, щоб сервіс міг комінікувати з шиною. Для цього вже готовий скрипт, який запускається зразу після запуску контейнера.

РОЗДІЛ 9. ПЕРЕДАЧА ПОВІДОМЛЕНЬ ТА КОМУНІКАЦІЯ МІЖ СЕРВІСАМИ ЧЕРЕЗ MQTT БРОКЕР

Якщо ми уважно розглянемо архітектуру проекту(додаток 1), побачимо, що комунікація між асинхронною чергою завдань та сервісом capra-can відбувається через MQTT брокер. Ми публікуємо інформацію на конкретні теми і підписуємося на ці теми з інших сервісів. Це означає, що для отримання даних з девайсу ми публікуємо інформацію на 4 різні теми, водночас підписуючись на ці ж теми, де має бути надіслана відповідь. Інший сервіс, з своєї сторони, підписується на ці теми і, як тільки отримує дані, починає їх обробку та відправляє відповідь на іншу тему, на яку вже підписаний інший сервіс.

Щоб підключитись до брокера, потрібно знати host та port.

Host - це мій комп’ютер, port дефолтний - 1883.

Для того щоб підключитись до брокера з робота що знаходиться в мене на комп’ютері, при налаштуванні контейнера capra-can я задаю broker host, як ip свого локального комп’ютера та порт 1883. Таким чином, я отримую доступ до mqtt broker.

Давайте трішки розглянемо головні функції брокера MQTT в загальному.

MQTT (Message Queuing Telemetry Transport) - це легкий протокол обміну повідомленнями, призначений для ефективного зв’язку між пристроями з обмеженими ресурсами та ненадійними мережами. Брокер MQTT діє як центральний центр або посередник, який полегшує спілкування між клієнтами MQTT.

- Модель публікації-підписки. MQTT дотримується шаблону обміну повідомленнями публікації-підписки. Публіцисти— це клієнти MQTT, які надсилають повідомлення брокеру, а підписники — це клієнти, які отримують

повідомлення від брокера. Підписники виявляють інтерес до певних тем і отримують повідомлення, опубліковані на ці теми.

- Якість обслуговування (QoS). MQTT підтримує різні рівні QoS для доставки повідомлень. Зазначений видавцем рівень QoS визначає надійність і гарантію доставки повідомлень передплатникам. Рівні QoS включають «щонайбільше один раз» (QoS 0), «принаймні один раз» (QoS 1) і «рівно один раз» (QoS 2).

- Збережені повідомлення. Брокери MQTT можуть зберігати збережені повідомлення, тобто повідомлення, які зберігаються на брокері та доставляються новим передплатникам одразу після підписки. Збережені повідомлення дозволяють передплатникам отримувати найновішу інформацію по темі, навіть якщо вони щойно приєдналися до системи.

- Безпека та автентифікація. Брокери MQTT часто надають такі функції безпеки, як автентифікація, контроль доступу та шифрування, щоб захистити цілісність та конфіденційність повідомлень, якими обмінюються клієнти. Це гарантує, що лише авторизовані клієнти можуть публікувати або підписуватися на певні теми.

- Масштабованість і висока доступність. Бокери MQTT можуть бути розгорнуті в масштабованих і розподілених архітектур для обробки великої кількості клієнтів і повідомлень. Налаштування високої доступності гарантують, що зв’язок MQTT залишається безперебійним навіть за наявності збоїв брокера.

ВИСНОВКИ

У дипломній роботі було проведено дослідження мікросервісної архітектури та технології Docker, що дозволяють ефективно організувати та масштабувати додатки. Мікросервісна архітектура дозволяє поділити додаток на окремі сервіси, кожен з яких виконує певну функцію та може працювати незалежно від інших сервісів. Технологія Docker дозволяє упаковувати сервіси в контейнери, що дозволяє запускати та управляти ними на різних середовищах.

Дослідження протоколу SocketCAN та CANopen дозволило встановити його ефективність у взаємодії з мережевим обладнанням, що використовує CAN-інтерфейс. Також було досліджено Process Data Object (PDO) та Service Data Object (SDO) протоколи, що дозволяють передавати дані в режимі реального часу. MQTT брокер, у свою чергу, дозволило організувати зв'язок між сервісами та передавати дані у форматі, який є зрозумілим для кожного сервісу.

Аналіз роботи завантажувача та розробка команд для запиту даних дозволили встановити принцип роботи обладнання та спосіб, яким можна отримати дані з нього. Після отримання даних у сирому форматі, було розроблено алгоритми їх обробки та перетворення у зрозумілі дані для кожного сервісу.

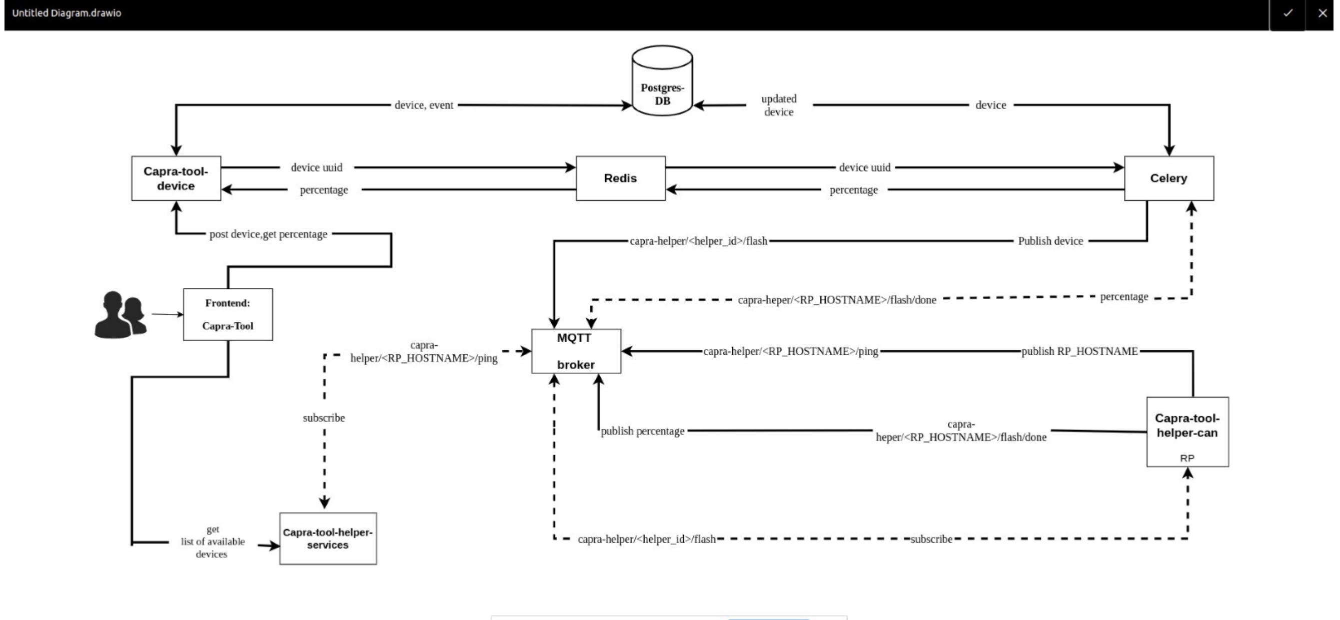
Було розроблено зручний графічний інтерфейс для роботи з даними, використовуючи React.

Отже, у ході дипломної роботи було проведено дослідження мікросервісної архітектури та технології Docker, комп'ютера що знаходиться в роботі Jetson Nano, протоколів SocketCAN, PDO та SDO, MQTT брокера та роботи завантажувача. Розроблено зручний графічний інтерфейс для роботи з отриманими даними, розроблено алгоритми обробки та перетворення отриманих даних, завантаження нового програмного забезпечення, у зрозумілу форму для кожного сервісу. Отримані

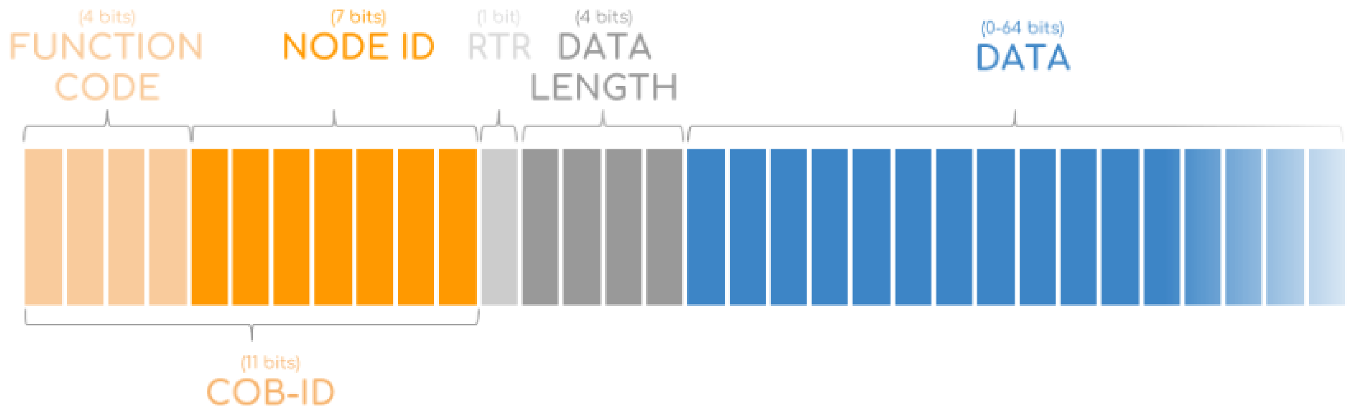
результати можуть бути використані у подальших проектах зі схожою тематикою та застосуванні для розв'язання практичних завдань у різних галузях, таких як автомобільна промисловість, промислова автоматизація та інші

ДОДАТКИ

Додаток 1



Додаток 2



Додаток 3

Example +

	COMMUNICATION OBJECT	FUNCTION CODE (4 bit, bin)	NODE IDs (7 bit, bin)	COB-IDs (hex)	COB-IDs (dec)	#
1	NMT	0000	0000000	0	0	1
2	SYNC	0001	0000000	80	128	1
3	EMCY	0001	0000001-1111111	81 - FF	129 - 255	127
4	TIME	0010	0000000	100	256	1
5	Transmit PDO 1	0011	0000001-1111111	181 - 1FF	385 - 511	127
	Receive PDO 1	0100	0000001-1111111	201 - 27F	513 - 639	127
	Transmit PDO 2	0101	0000001-1111111	281 - 2FF	641 - 767	127
	Receive PDO 2	0110	0000001-1111111	301 - 37F	769 - 895	127
	Transmit PDO 3	0111	0000001-1111111	381 - 3FF	897 - 1023	127
	Receive PDO 3	1000	0000001-1111111	401 - 47F	1025 - 1151	127
	Transmit PDO 4	1001	0000001-1111111	481 - 4FF	1153 - 1279	127
	Receive PDO 4	1010	0000001-1111111	501 - 57F	1281 - 1407	127
6	Transmit SDO	1011	0000001-1111111	581 - 5FF	1409 - 1535	127
	Receive SDO	1100	0000001-1111111	601 - 67F	1537 - 1693	127
7	HEARTBEAT	1110	0000001-1111111	701 - 77F	1793 - 1919	127

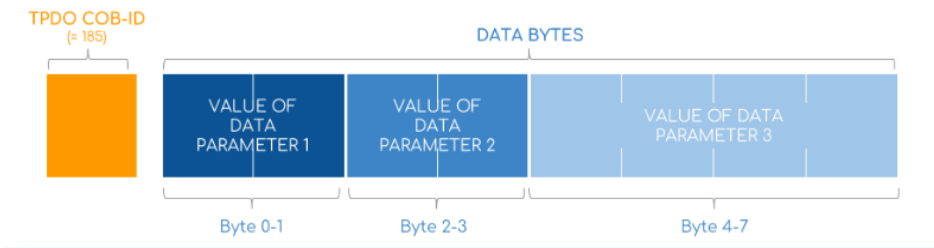
Додаток 4

```
can0  19B   [8]  00 00 00 00 00 00 00 80
can0  29B   [8]  94 41 FF 5E 00 00 80 FF
can0  19D   [8]  00 00 EA FF FF FF 00 00
can0  705   [1]  05
can0  19C   [8]  00 00 00 00 00 00 00 00
can0  185   [8]  4C 00 00 00 00 00 00 80
can0  19E   [8]  00 00 00 00 00 00 00 00
can0  49D   [8]  FC FF 6B 43 1E 17 00 00
can0  19B   [8]  00 00 00 00 00 00 00 00
can0  19D   [8]  00 00 EA FF FF FF 00 80
can0  19C   [8]  00 00 00 00 00 00 00 80
can0  49E   [8]  EA FF C4 43 1F 17 00 00
can0  49B   [8]  09 00 14 44 1E 19 00 00
can0  19E   [8]  00 00 00 00 00 00 00 80
can0  19B   [8]  00 00 00 00 00 00 00 80
can0  19D   [8]  00 00 EA FF FF FF 00 00
can0  19C   [8]  00 00 00 00 00 00 00 00
can0  706   [1]  05
can0  186   [8]  43 18 00 00 00 00 00 80
can0  19E   [8]  00 00 00 00 00 00 00 00
can0  19B   [8]  00 00 00 00 00 00 00 00
can0  19D   [8]  00 00 EA FF FF FF 00 80
can0  19C   [8]  00 00 00 00 00 00 00 80
can0  19E   [8]  00 00 00 00 00 00 00 80
can0  19B   [8]  00 00 00 00 00 00 00 80
can0  19D   [8]  00 00 EA FF FF FF 00 00
can0  19C   [8]  00 00 00 00 00 00 00 00
can0  19E   [8]  00 00 00 00 00 00 00 00
```

Додаток 5

```
root → / $ candump can0,61B:7FF,59B:7FF
  can0  61B   [8]  40 09 10 00 00 00 00 00
  can0  59B   [8]  41 09 10 00 08 00 00 00
  can0  61B   [8]  60 00 00 00 00 00 00 00
  can0  59B   [8]  00 05 00 00 00 00 00 00
  can0  61B   [8]  70 00 00 00 00 00 00 00
  can0  59B   [8]  1D 00 00 00 00 00 00 00
```

Додаток 6



Додаток 7

Capra Tool

RP_HOSTNAME
Overview
HIRCUS BOARDS
Operations Control
Power Control
Angle Sensor
Motor Control
Ultrasounds, Brakes and Lights
DIAGNOSTICS
Bus Traffic
Devices
ROS
Overview
TESTS
Sidearm test

Devices

Choose a file...

Please select firmware.

Hardware Id | Device Type | Flash | Flash Multiple

Select All | Select None

Device	State	Bootloader	Type	Hardware	Software	Commit
05 - Angle Sensor (front)	Operational		HAS_APP	1	[0,6,2]	
06 - Angle Sensor (rear)	Operational		HAS_APP	1	[0,6,2]	
07 - Brake (left)						
08 - Brake (right)						
1B - Motor (front)	Operational		HMC_APP	5	[0,7,2]	
1C - Motor (rear)	Operational		HMC_APP	5	[0,7,2]	
1D - Motor (left)	Operational		HMC_APP	5	[0,7,2]	
1E - Motor (right)	Operational		HMC_APP	5	[0,7,2]	
40 - Power Controller						

Selected 6 device(s).

Unmute

Do not click anything below this point, unless you know what you are doing.

Reset | Command Mode | Auto Command Mode (Disabled)

CHANGE NODE ID
Old Node ID | New Node ID | Change
Old Node ID | New Node ID | Change

SDO ACCESS
Node ID 0x01 | Index 0x0000 | Sub Index 0x00
Read | Write
00 00 00 00 00 00 00 00

ERRORS
30.04.2023 16.52.21
Motor (front)
noErrorOrReset
Count: 00
reset

30.04.2023 16.52.21
Motor (right)
noErrorOrReset
Count: 00
reset

Додаток 8

Capra Tool

RP_HOSTNAME

Overview

HIRCUS BOARDS

Operations Control

Power Control

Angle Sensor

Motor Control

Ultrasounds, Brakes and Lights

DIAGNOSTICS

Bus Traffic

Devices

ROS

Overview

TESTS

Sidearm test

Overview

HOC Offline

HPC Offline

Front HMC Online

Right HMC Online

Rear HMC Online

Left HMC Online

Right HUL Offline

Left HUL Offline

Front HAS Online

Rear HAS Online

CHANGE NODE ID

Old Node ID	New Node ID	
		Change

Old Node ID	New Node ID	
		Change

SDO ACCESS

Node ID	Index	Sub Index
0x(	0x(	0x(

Read Write

```
00 00 00 00 00 00
00 00
```

ERRORS

Empty

Додаток 9

Capra Tool

RP_HOSTNAME

Overview

HIRCUS BOARDS

Operations Control

Power Control

Angle Sensor

Motor Control

Ultrasounds, Brakes and Lights

DIAGNOSTICS

Bus Traffic

Devices

ROS

Overview

TESTS

Sidearm test

Operations Control

Remote Control

	Front	Rear	Left	Right
Speed	0 mm/s	0 mm/s	0 mm/s	0 mm/s

Додаток 10

Capra Tool

RP_HOSTNAME
Overview
HIRCUS BOARDS
Operations Control
Power Control
Angle Sensor
Motor Control
Ultrasounds, Brakes and Lights
DIAGNOSTICS
Bus Traffic
Devices
ROS
Overview
TESTS
Sidearm test

Power Control

State	Value
Battery Count	0
Battery Voltage (±0.2 V)	0.0 V
Relative Humidity (±1 %)	0 %
Temperature (±0.5 °C)	0.0 °C

Wheel Power

All Disabled | All Low Power | All High Power

Wheel
State
Error
Control

Batteries

Battery	Voltage	Status

CHANGE NODE ID
Old Node ID | New Node ID | Change
Old Node ID | New Node ID | Change
SDO ACCESS
Node ID | Index | Sub Index
0x01 | 0x000(| 0x00
Read | Write
00 00 00 00 00 00 00 00
ERRORS
Empty

Додаток 11

Capra Tool

RP_HOSTNAME
Overview
HIRCUS BOARDS
Operations Control
Power Control
Angle Sensor
Motor Control
Ultrasounds, Brakes and Lights
DIAGNOSTICS
Bus Traffic
Devices
ROS
Overview
TESTS
Sidearm test

Angle Sensor

State	Front	Rear	Difference
Angle	0.329 rad	-0.324 rad	-0.005 rad
Control	Calibrate	Calibrate	
Magnetic Field	0 G	0 G	

CHANGE NODE ID
Old Node ID | New Node ID | Change
Old Node ID | New Node ID | Change
SDO ACCESS
Node ID | Index | Sub Index
0x01 | 0x000(| 0x00
Read | Write
00 00 00 00 00 00 00 00
ERRORS
Empty

Додаток 12

Capra Tool

RP_HOSTNAME
- Overview

HIRCUS BOARDS
- Operations Control
- Power Control
- Angle Sensor
- Motor Control
- Ultrasounds, Brakes and Lights

DIAGNOSTICS
- Bus Traffic
- Devices

ROS
- Overview

TESTS
- Sidearm test

Hircus Motor Controllers

Select All	Select Front	Select Rear	Select Left	Select Right
Brake				
Speed Request [mm/s]				
Speed [mm/s]	0	0	0	0
Id	13	13	15	14
Acceleration X [dm/s²]	22	20	-18	9
Acceleration Y [dm/s²]	-1	-2	6	0
Acceleration Z [dm/s²]	95	96	-108	-108
Gyro X [crad/s]	-1	0	1	1
Gyro Y [crad/s]	-1	0	0	0
Gyro Z [crad/s]	-2	-2	1	-2
Toggle Debug All	Toggle Debug Front	Toggle Debug Rear	Toggle Debug Left	Toggle Debug Right
Current [A]	-0.004	-0.040	-0.013	-0.004
Voltage [V]				

CHANGE NODE ID

Old Node ID	New Node ID	
		Change

Old Node ID	New Node ID	
		Change

SDO ACCESS

Node ID	Index	Sub Index
0x01	0x000(	0x00

Read Write

00 00 00 00 00 00 00 00

ERRORS

Empty

Додаток 12.1

Toggle Debug All	Toggle Debug Front	Toggle Debug Rear	Toggle Debug Left	Toggle Debug Right		
Current [A]	-0.009	-0.009	0.018	-0.009		
Voltage [V]	17.436	17.321	17.259	17.348		
Temperature [degC]	31	32	32	32		
Humidity [%]	25	22	23	23		
Placement	Unknown	Unknown	Unknown	Unknown	Read	Correct
Radius [mm]					Read	Write
Speed Gain					Read	Write
Current Gain						
Integral Gain					Read	Write
Feedforward Gain					Read	Write
Controller Min					Read	Write
Controller Max					Read	Write
Calibrate All	Calibrate Front	Calibrate Rear	Calibrate Left	Calibrate Right		
Hall Sensor CCW 1					Read	Write

Empty

Додаток 13

Capra Tool

RP_HOSTNAME
Overview
HIRCUS BOARDS
Operations Control
Power Control
Angle Sensor
Motor Control
Ultrasounds, Brakes and Lights
DIAGNOSTICS
Bus Traffic
Devices
ROS
Overview
TESTS
Sidearm test

Ultrasounds, Brakes and Lights

State	Left ♥	Right ♥
Distance	0 mm	0 mm
Brake State		
Light Mode (%)	5	5
Brake Mode	Engage	Engage
	Start Transmitting	Start Transmitting
	Reset	Reset

Upload happens instantly when you select a file, **make sure the right slot is selected.**

0

Choose a file...

CHANGE NODE ID

Old Node ID | New Node ID | Change

Old Node ID | New Node ID | Change

SDO ACCESS

Node ID	Index	Sub Index
0x01	0x000(	0x00

Read | Write

00 00 00 00 00 00 00 00

ERRORS

Empty

Додаток 13.1

Capra Tool

RP_HOSTNAME
Overview
HIRCUS BOARDS
Operations Control
Power Control
Angle Sensor
Motor Control
Ultrasounds, Brakes and Lights
DIAGNOSTICS
Bus Traffic
Devices
ROS
Overview
TESTS
Sidearm test

Ultrasounds, Brakes and Lights

State	Left ♥	Right ♥
Distance	0 mm	0 mm
Brake State		
Light Mode (%)	5	5
Brake Mode	Disengage	Disengage
	Start Transmitting	Start Transmitting
	Reset	Reset

Upload happens instantly when you select a file, **make sure the right slot is selected.**

0

Choose a file...

CHANGE NODE ID

Old Node ID | New Node ID | Change

Old Node ID | New Node ID | Change

SDO ACCESS

Node ID	Index	Sub Index
0x01	0x000(	0x00

Read | Write

00 00 00 00 00 00 00 00

ERRORS

Empty

Додаток 14

Capra Tool

RP_HOSTNAME
Overview
HIRCUS BOARDS
Operations Control
Power Control
Angle Sensor
Motor Control
Ultrasounds, Brakes and Lights
DIAGNOSTICS
Bus Traffic
Devices
ROS
Overview
TESTS
Sidearm test

Bus Traffic

1,000,000 bit

Node	Count [Hz]	Data[1] [kbit/s]	Load[1] [%]
05 - Angle Sensor (front)	20	2.320	0.2
06 - Angle Sensor (rear)	20	2.320	0.2
1B - Motor (front)	130	18.160	1.8
1C - Motor (rear)	130	18.160	1.8
1D - Motor (left)	130	18.160	1.8
1E - Motor (right)	130	18.160	1.8
Total	560	77.280	7.7

Error	Sub Error	Count

1: These are approximates, based on the worst case calculations.

CHANGE NODE ID

Old Node ID | New Node ID | Change

Old Node ID | New Node ID | Change

SDO ACCESS

Node ID	Index	Sub Index
0x01	0x000(	0x00

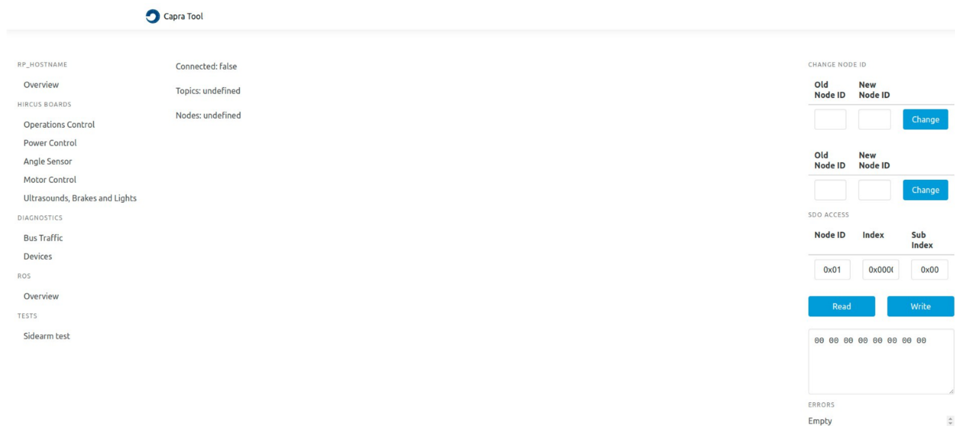
Read | Write

00 00 00 00 00 00 00 00

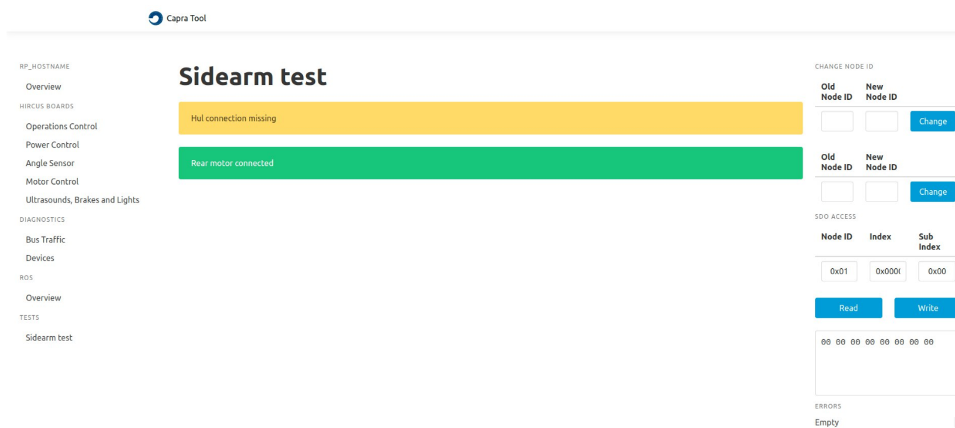
ERRORS

Empty

Додаток 15



Додаток 16



Додаток 17

Devices

Choose a file... HAS-APP-0.6.2-20220408-97.zip

Type: Hircus Angle Sensor (HAS-APP)
Version: 0.6.2 - Release - Fri Apr 08 2022 - Build 97
Size: 65352 b

1 | Hircus Angle Sensor - Bootloader | Flash | Flash Multiple

Select All | Select None

Device	State	Bootloader	Type	Hardware	Software	Commit
05 - Angle Sensor (front)	❤ Operational		HAS_APP	1	[0,6,3]	
06 - Angle Sensor (rear)	🖤 Operational		HAS_APP	1	[0,6,3]	
07 - Brake (left)						
08 - Brake (right)						
1B - Motor (front)	❤ Operational		HMC_APP	5	[0,7,2]	
1C - Motor (rear)	🖤 Operational		HMC_APP	5	[0,7,2]	
1D - Motor (left)	❤ Operational		HMC_APP	5	[0,7,2]	
1E - Motor (right)	🖤 Operational		HMC_APP	5	[0,7,2]	
40 - Power Controller						

Додаток 17.1

Devices

Choose a file...	HAS-APP-0.6.2-20220408-97.zip

Type: Hircus Angle Sensor (HAS-APP)
Version: 0.6.2 - Release - Fri Apr 08 2022 - Build 97
Size: 65352 b

1 | Hircus Angle Sensor - Bootloader | Flash | Flash Multiple

Select All | Select None

Device	State	Bootloader	Type	Hardware	Software	Commit
05 - Angle Sensor (front)	❤️ Operational	🖤 Waiting for more Firmware Data	UNDEFINED	1	[0,6,3]	
06 - Angle Sensor (rear)	❤️ Operational	❤️ Waiting for more Firmware Data	HAS_APP	1	[0,6,3]	
07 - Brake (left)						
08 - Brake (right)						
1B - Motor (front)			HMC_APP	5	[0,7,2]	
1C - Motor (rear)			HMC_APP	5	[0,7,2]	
1D - Motor (left)			HMC_APP	5	[0,7,2]	
1E - Motor (right)			HMC_APP	5	[0,7,2]	
40 - Power Controller						

Додаток 17.2

1 | Hircus Angle Sensor - Bootloader | Flash | Flash Multiple

Select All | Select None

Device	State	Bootloader	Type	Hardware	Software	Commit
05 - Angle Sensor (front)	❤️ Operational	❤️ Completed	HAS_APP	1	[0,6,2]	
06 - Angle Sensor (rear)	❤️ Operational	❤️ Completed	HAS_APP	1	[0,6,2]	
07 - Brake (left)						

СПИСОК ЛІТЕРАТУРИ

1. Oliver Hartkopp SocketCAN - The official CAN API of the Linux kernel,електронні ресурси в науці, культурі та освіті [Електронний ресурс] - Режим доступу: ”https://docs.kernel.org/networking/can.html”
2. Gaston C. Hillar MQTT Essentials - A Lightweight IoT Protocol - 2017 - С. 150 - 280.
3. Sam Newman Building Microservices: Designing Fine-Grained Systems - 2015 - 280 с.
4. Sean Walton Linux Socket Programming - 2001 - Видавничий центр: Sams - 400с.