Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

__Інститут інформатики та радіоелектроніки,__
__Факультет комп'ютерних наук і технологій__
(повне найменування інституту, назва факультету)

_____Кафедра програмних засобів_____
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
_____бакалавр_____
(ступінь вищої освіти)

на тему РОЗРОБКА ANDROID ЗАСТОСУНКУ ДЛЯ НАЛАШТУВАННЯ ПОЛЬОТНОГО КОНТРОЛЕРА КВАДРОКОПТЕРА
ANDROID APPLICATION DEVELOPMENT FOR ADJUSTING A QUADCOPTER FLIGHT CONTROLLER

Виконав: студент 4 курсу, групи КНТз-117
Спеціальності 121 Інженерія програмного забезпечення
(код і найменування спеціальності)

Освітня програма (спеціалізація)
_____Інженерія програмного забезпечення_____

_____Гордієнко В.І._____
(прізвище та ініціали)
Керівник__Федорончак Т.В._____
(прізвище та ініціали)
Рецензент__Казурова А.Є,_____
(прізвище та ініціали)

2021

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____С.О. Субботін
“____” _____20___року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Гордієнка В’ячеслава Ігоровича
(прізвище, ім’я, по батькові)

1. Тема проєкту (роботи) Розробка Android застосунку для налаштування польотного контролера квадрокоптера. Android Application Development for Adjusting a Quadcopter Flight Controller

керівник проєкту (роботи) Федорончак Тетяна Василівна., к.т.н., доцент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від “ 30 ” березня 2021 року № 103
2. Строк подання студентом проєкту (роботи) 24 травня 2021 року
3. Вихідні дані до проєкту (роботи) технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Опис програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Федорончак Т.В., доцент		
Нормоконтроль	Дейнега Л.Ю., ст. виклаадч		

7. Дата видачі завдання “ 28 ” лютого 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-3 тижні	Розділ 1
3	Опис програми	4 тиждень	Розділ 2
4	Розробка програми	5-6 тижні	Розділ 3
5	Експлуатація, тестування та експериментальне дослідження програми	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Додатки
7	Захист роботи	9 тиждень	

Студент _____ Гордієнко В.І.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Федорончак Т.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 121 с., 2 табл., 35 рис., 3 дод., 11 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, НАЛАШТУВАННЯ, КВАДРОКОПТЕР, ПОЛЬОТНИЙ КОНТРОЛЕР, JAVA, ANDROID, USB, MSP

Об’єкт дослідження — налаштування польотного контролера квадрокоптера в польових умовах.

Предмет дослідження — мобільний застосунок для налаштування польотного контролера.

Мета роботи — розробка застосунку для взаємодії мобільного пристрою на базі операційної системи Android з польотним контролером квадрокоптера.

Матеріали, методи та технічні засоби: середовище розробки Android Studio, мобільний телефон с операційною системою Android, польотний контролер квадрокоптера з програмною прошивкою Betaflight.

Результати: розроблено застосунок для мобільних телефонів з операційною системою Android для налаштування польотного контролера квадрокоптера в польових умовах за допомогою з’єднання через інтерфейс USB.

Висновки: розроблено мобільний застосунок який дозволяє підключити польотний контролер квадрокоптера до мобільного телефону для подальшого налаштування.

Галузь використання: може використовуватися оператором квадрокоптера для налаштування польотного контролера в будь-якому місці.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 121 pages, 2 tables, 35 figures, 3 appendixes, 11 sources.

MOBILE APPLICATION, SETTINGS, QUADCOPTER, FLIGHT CONTROLLER, JAVA, ANDROID, USB, MSP

The object of work is the process of changing the flight controller's settings of a quadcopter.

The subject of research is a mobile application to set up a quadcopter's flight controller.

The purpose of the work is to develop an application for the interaction of a mobile device based on Android operating system with a flight controller of a quadcopter.

Materials, methods and tools: Android Studio development environment, mobile phone with Android operating system, quadcopter flight controller with Betaflight firmware.

Results: The application was developed for mobile phones with Android operating system to configure a quadcopter's flight controller in a field using a USB connection.

Conclusions: developed a mobile application that allows users to connect a flight controller of a quadcopter to a mobile phone for further configuration.

Scope of using: can be used by a quadcopter operator to configure a flight controller anywhere.

ЗМІСТ

С.

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ПК — Персональний комп'ютер

IDE — Integrated development environment (Інтегроване середовище розробки)

Android —Операційна система, що розроблюється компанією Google

Java — Мова програмування, розроблюється компанією Oracle

XML — Extensible Markup Language

USB — Universal Serial Bus

Betaflight — Прошивка для польотного контролеру

Betaflight Configurator — Програма для налаштування прошивки Betaflight на персональному комп'ютері

FPV — First person view (вид від першої особи)

MSP — Multiwii Serial Protocol

PID — Proportional–Integral–Derivative controller

ESC — Electronic Speed Control

GPS — Global Positioning System

CLI — Command line interface

ВСТУП

В повсякденному житті все частіше можна зустріти різного типу дрони. Найпопулярнішими є літаючі мультироторні дрони, а з них — квадрокоптер (дрон з 4 роторами).

Квадрокоптери широко застосовуються для відеозйомки та картографічної розвідки. Ще одна, відносно нова сфера використання квадрокоптерів — перегони на дронах за допомогою технології FPV (First Person View, з англ. - вид від першої особи).

Типовий квадрокоптер складається з таких основних компонентів: рама (корпус), мотори, пропелери, регулятори обертів мотора, акумуляторна батарея, радіо приймач управління, камера та відеопередавач. Ще одним, найголовнішим, компонентом є польотний контролер. Польотний контролер приймає сигнал від радіоприймача, оброблює його різними алгоритмами, враховуючи показники вбудованих в нього датчиків, та передає сигнал на регулятори обертів для подальшого керування моторами.

Існую багато різних виробників польотних контролерів, такі як Matek, Flyduino, Diatone, iFlight, Emax та інші.

Польотний контролер має програмне забезпечення — прошивку (англ. Firmware). Основними прошивками є Betaflight, iNav, KISS. Кожна з них має багато налаштувань, та окреме програмне забезпечення для персонального комп'ютера (далі - ПК) для управлінням ними. Однак доволі часто виникає необхідність змінити деякі налаштування безпосередньо перед польотом — наприклад, в полі, де немає доступу до ПК. Тому тема цієї роботи є розробка програмного забезпечення у вигляді застосунку для налаштування польотного контролера квадрокоптера.

На вибір було взято прошивку Betaflight, як найпопулярнішу серед пілотів, що приймають участь у перегонах на дронах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Необхідність мобільного застосунку

Предметом аналізу даної роботи є польотний контролер дрону. Дрон може мати 4 (quadcopter), 6 (hexacopter) або 8 (octocopter) моторів. Надалі розглядатися буде варіант дрону з 4 моторами — квадрокоптер, з прошивкою Betaflight.

Квадрокоптер (рис. 1.1) під управлінням прошивки Betaflight — це дрон невеликого розміру, переважно від 65 до 300 мм діаметру між моторами (аналог розмаху крил у літака).

Прошивка Betaflight має багато налаштувань. Зазвичай оператор квадрокоптеру вносить зміни в налаштування вдома, зі свого персонального комп'ютеру, завдяки спеціальній програмі Betaflight Configurator.

Однак, доволі часто виникає потреба змінити налаштування, чи просто перевірити їх, безпосередньо перед початком польоту. Також, польотний контролер має декілька вбудованих в нього, чи під’єднаних до нього, датчиків, стан та показники яких теж потрібно перевіряти. Звісно, можна взяти із собою ноутбук, який дозволить виконувати необхідні дії з контролером. Але ноутбук, попри те, що він не великий, займає багато місця серед інших комплектуючих, які потрібні для управління квадрокоптером, це:

– пульт управління;

– відеоокуляри (з двома дисплеями, по одному для кожного ока) чи шлем з одним дисплеєм та лінзою;

– акумуляторна батарея.

Цей список можна ще доповнити запасними пропелерами, різними інструментами, та декількома батареями, бо однієї вистачає лише на декілька хвилин (залежить від манери польоту, температури середовища та інших факторів).

Тому розробка застосунку для мобільного телефону, може заощадити чимало місця та 1-3 кг ваги, адже телефон значно менше та легше у порівнянні з ноутбуком. Тим паче, в наш час, кожна людина бере мобільний телефон із собою всюди.

Рисунок 1.1 — Квадрокоптер [1]

1.2 Аналіз існуючих рішень

Speedy Bee (рис. 1.2) — мобільний застосунок для телефонів з операційною системою Android. Розробник застосунку компанія RunCam Co.,Ltd. Застосунок дає можливість змінювати налаштування прошивки Betaflight шляхом з’єднання мобільного телефону з контролером за допомогою USB кабелю або Bluetooth (для контролерів, що не мають вбудованого Bluetooth, потрібно докупити спеціальний адаптер).

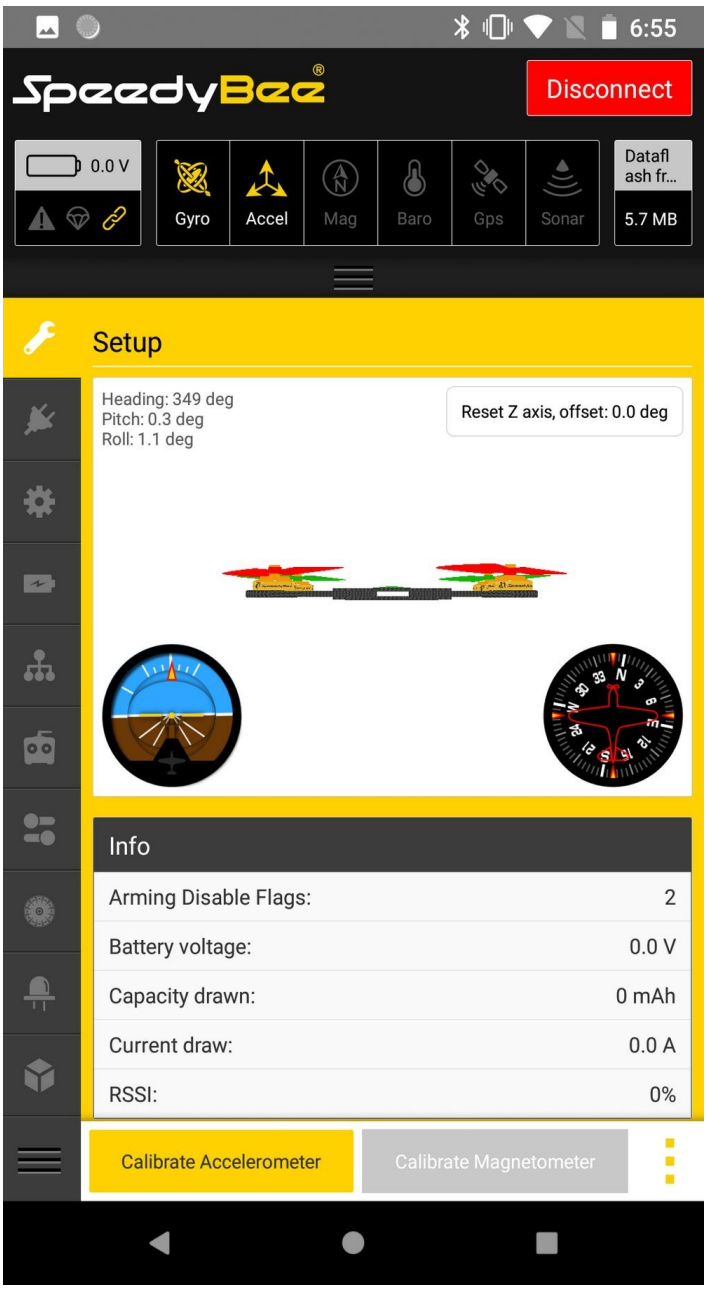


Рисунок 1.2 — Інтерфейс застосунку Speedy Bee [2]

Speedy Bee має всі ті можливості що має і програма для ПК Betaflight Configurator, серед них:

а) головний екран зі списком доступних до підключення пристроїв;

б) перегляд поточних даних вбудованих в контролер датчиків:

1) напруга акумуляторної батареї;

2) напруга живлення мікросхем;

3) стан гіроскопу [3] та його покази;

4) стан компасу та його покази;

5) покази барометру (якщо такий датчик є);

6) покази GPS (якщо такий датчик є);

7) рівень зв’язку з пультом управління;

в) налаштування PID [4] та фільтрів даних гіроскопу;

г) вибір протоколу обміну даними з ESC;

ґ) перегляд поточного сигналу від пульта управління по кожному каналу;

д) перегляд та зміна режимів польоту;

е) встановлення необхідних дій у разі обриву зв’язку;

є) налаштування сітки даних, які будуть накладатися на трансльоване зображення з камери квадрокоптера в відеоокуляри оператора.

Але деякі параметри не доступні для редагування лементами інтерфейсу застосунку. Застосунок також має окремий екран консольного терміналу (англ. CLI), де можна змінювати всі інші параметри.

Інших існуючих рішень не було знайдено, або вони мали застарілу кодову базу, або мали замало функцій.

1.3 Постановка завдання роботи

За результатом доказу необхідності застосунку та аналізом функцій аналогу Speedy Bee, були визначено вимоги до його функцій.

Розглянемо загальну схему роботи та необхідні функції програми.

За допомогою USB кабелю користувач повинен під’єднати польотний контролер до мобільного телефону та запустити застосунок. Програма повинна просканувати під’єднані контролери (а їх може бути більше одного, завдяки тому, що USB реалізує послідовний принцип з’єднання) та показати їх списком з назвами.

Далі користувач обирає необхідний контролер зі списку та програма ініціює з’єднання з контролером тепер вже на програмному рівні.

Розробити просте та інтуїтивно зрозуміло меню навігації між екранами (кожен екран буде представляти певний набір налаштувань контролера, споріднених між собою).

В меню кожен пункт повинен представляти окремий екран навігації застосунку. Для кожного екрану розробити кнопки для зчитування та запис даних у пам’ять контролера.

1.4 Висновки до розділу 1

У першому розділі було розглянуто основні причини необхідності застосунку для мобільних телефонів. Було розглянуто існуюче рішення у вигляді застосунку Speedy Bee. Також було поставлено завдання до роботи з коротким описом необхідної схеми роботи застосунку та його основних функцій.

2 ОПИС ПРОГРАМИ

2.1 Функціональні вимоги до програми

Головна функція програми це зчитування та запис даних в пам’ять контролера. У обмін даними з контролером на прошивці Betaflight здійснюється з використанням протоколу MSP (Multiwii Serial Protocol) [5].

Меню повинно бути створено з використанням популярного для Android шаблону Navigation Drawer (укр. навігаційна панель), така панель схована за замовчуванням, показати можна по кліку на спеціальну кнопку, або ж жестом від лівого боку екрану у напрямку до правого.

Зверху на кожному екрані розмістити кнопки “зчитати” та “зберегти”. Ці кнопки будуть відповідати за зчитування даних з контролеру та, відповідно, збереження даних в контролер.

На першому екрані програми (після успішного з’єднання) треба показати основні дані контролеру та його датчиків. Можна також візуалізувати дані з датчику гіроскоп — це дані про положення квадрокоптеру у просторі (тобто кути нахилу) за допомогою 3D моделі.

Обов’язково розробити окремий екран для налаштувань PID та фільтрів даних гіроскопу. Це головні алгоритмі обробки даних контролером для подальшого управління моторами.

Також треба розробити екран консольного терміналу. Цей екран дозволить користувачеві вводити команди та отримувати на них відповідь від контролеру. Таким чином користувач зможе змінювати навіть ті налаштування, для яких ще не розроблено спеціальних елементів інтерфейсу.

Ще однією потужною функцією консольного терміналу є вивід усіх налаштувань построково у вигляді *параметр = значення.*

2.2 Вибір мови програмування

Оскільки було вирішено розробляти програму для операційної Android, то мову програмування потрібно вибирати опираючись на це рішення. Основна мова програмування для Android це Java.

Java — об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems». З 2009 року розвитком Java займається компанія «Oracle», після викупу активів «Sun Microsystems».

Головна перевага мови у методі компіляції програмного коду. Java компілюється у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Таким чином один і той же байт-код може виконуватися на різних операційних системам. Це кардинально відрізняє мову від інших (наприклад C, C++), програмний код яких компілюється у бінарний для певного процесору і операційної системи.

Kotlin — відносно нова, статично типізована мова програмування, що працює поверх Java Virtual Machine. Розроблюється компанією JetBrains. Автори поставили собі за мету створити більш лаконічну та більш зручнішу мову ніж Java. Так як код, написаний на Kotlin, компілюється в байт-код, для віртуальної машини Java, він повністю сумісний з кодом на Java. Це дає змогу не переписувати проєект цілком но іншу мову, а лише доповнювати/змінювати деякі частини.

Перелік деяких можливостей, що відрізняє Kotlin від Java:

– null-беспечність;

– класи даних (англ. Data classes);

– функції-розширення;

– розумне приведення типів;

– функціональне програмування;

– лямбда функції для інтерфейсів з одним методом;

– асинхронне програмування за допомогою коротин.

Результати порівняння розглянутих мов програмування зведено до таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування Java та Kotlin

Характеристика	Java	Kotlin
Null-беспечність	Ні	Так
Розумне приведення типів	Ні	Так
Checked exception	так	Ні
Singleton objects	Так	Так (просте створення)
Функціонально програмування	Ні	Так
Функції-розширення	Ні	Так

2.3 Вибір середовища розробки

Eclipse (рис. 2.1) — вільне модульне інтегроване середовище розробки програмного забезпечення з відкритим кодом. Підтримується організацією Eclipse Foundation. Написаний в основному на Java, тому підтримка розробк програм на Java є пріорітетом. А за допомоги системи плагінів на базі Eclipse подубовано збірки для розробки програм і на інших мовах: Java, C/C++, Python, Python та інші. Графічний інтерфейс Eclipse написаний з використанням інструментарію SWT (Standard Widget Toolkit). SWT використовує графічні компоненти операційної системи, на якій запущено середовище. Eclipse була основним середовищем розробки для операційної системи Android до 2014 року, коли компанія Google випустила нове середовище Android Studio.

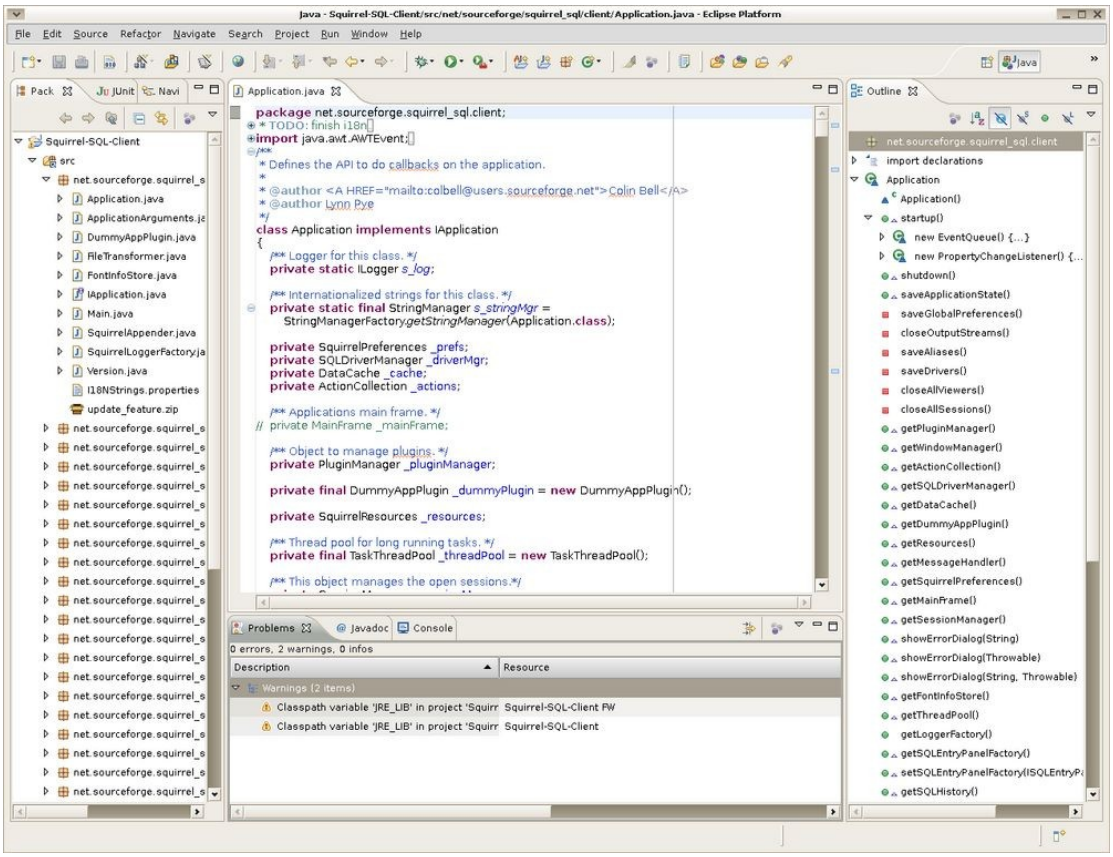


Рисунок 2.1 — Інтерфейс Eclipse [6]

Android Studio (рис. 2.2) — безкоштовне середовище розробки, розроблюється компанію Google, яка розроблює і операційну систему Android. Середовище Android Studio побудоване на базі вихідного коду продукту IntelliJ IDEA Community Edition, що розвивається компанією JetBrains. Android Studio має зручний та сучасний графічний інтерфейс. Має потужний вбудований редактор xml розмітки інтерфейсів програм, може працювати з віддаленими репозиторіями коду.

Android Studio можна запускати на усіх основних операційних системах для персональних комп'ютерів: Windows, MacOS, Linux. З середовищем також входить до комплекту Android SDK та емулятор віртуальних пристроїв.

Результати порівняння розглянутих середовищ розробки зведено до таблиці 2.2.

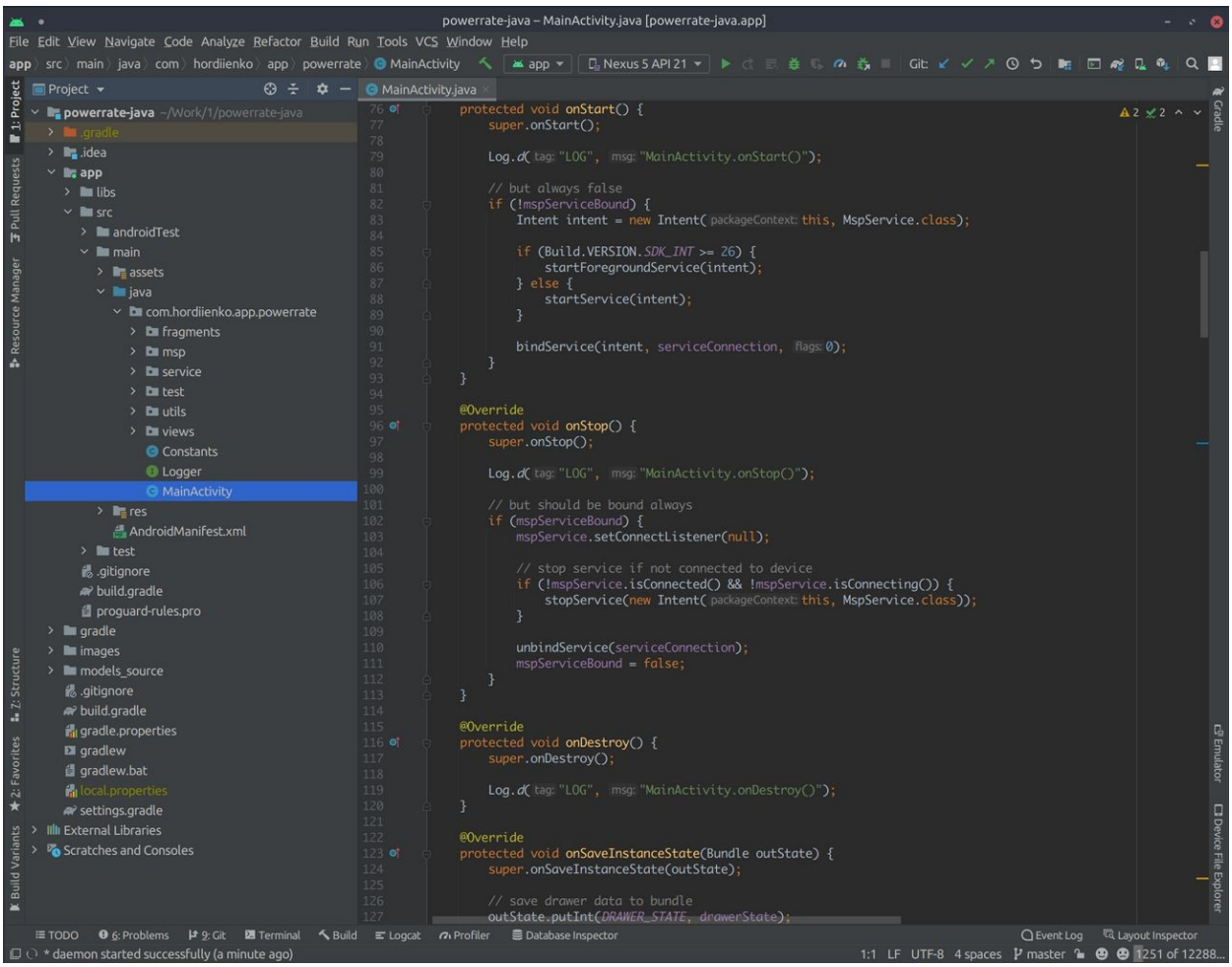


Рисунок 2.2 — Інтерфейс Android Studio, екран с файлом MainActivity.java

Таблиця 2.2 – Порівняння середовищ розробки Eclipse та Android Studio

Характеристика	Eclipse	Android Studio
Система побудови	Ant	система побудови Gradle
Підтримка мов Програмування	Java, C/C++, Python, Python та інші	Java та Kotlin
Графічний інтерфейс	Побудований на SWT	Побудований на Swing
Підключення бібліотек	Jar файли	Jar файли, Maven-сумісні репозиторії
Підтримка Cloud Platform	Ні	Так

2.4 Вибір бібліотеки для роботи з USB

Оскільки програма буде обмінюватися даними з контролером за допомогою USB інтерфейсу — потрібно вибрати необхідну бібліотеку.

UsbSerial — бібліотека з відкритим програмним кодом від іспанського розробника Felipe Herranz (felHR85). Бібліотека має низку сумісних з нею пристроїв на таких чипах: CP210X, CDC, FTDI, PL2303, CH34x, CP2130 SPI-USB. Бібліотека дозволяє, за допомогою простої ініціалізації з'єднання, обмінюватися даними з USB пристроєм (у нашому випадку контролером) потоком байтів, почергово на вхід та вихід.

2.5 Висновки по розділу 2

У другому розділі було розглянуто основні мови програмування для операційної системи Android — Java та Kotlin, та основні середовища розробки — Eclipse та Android Studio. Опираючись на досвід розробника було обрано мову програмування Java та інтегроване середовище розробки Android Studio. Також описано технічні вимоги до програми. Однією з вимоги є підключення польотного контролера до мобільного телефону за допомоги USB інтерфейсу. А також обмін даними з контролером по протоколу MSP, для цього було обрано бібліотеку з відкритим програмним кодом UsbSerial.

3 РОЗРОБКА ПРОГРАМИ

3.1 Основні компоненти програми

Кожна Android програма повинна мати файл AndroidManifest.xml. Це файл формату XML в якому реєструються основні компоненти системи, що будуть використовуватися програмою. Структура файлу AndroidManifest.xml (зміст файлу див. додаток Б.1):

– <manifest> — кореневий тег;

– <uses-permission; android:name="android.permission.FOREGROUND_SERVICE"/> — задаємо системі, що нам потрібні права для роботи сервісу у фоновому режимі;

– <uses-feature android:glEsVersion="0x00020000"/> — задаємо обмеження, для роботи програми пристрій має бути обладнаній графічним прискорювачем;

– <application> — параметри програми, задається назва, інока, тема інтерфейсу та інше;

– <activity> — компонент, що відображає графічний інтерфейс програми;

– <service> — компонент, код якого, виконується відокремлено від *activity,* також може виконуватися в фоновому режимі (коли *activity* не відображається на екрані).

Архітектуру програми було поділено на основні компоненти, діаграму яких зображено на рис. 3.1.

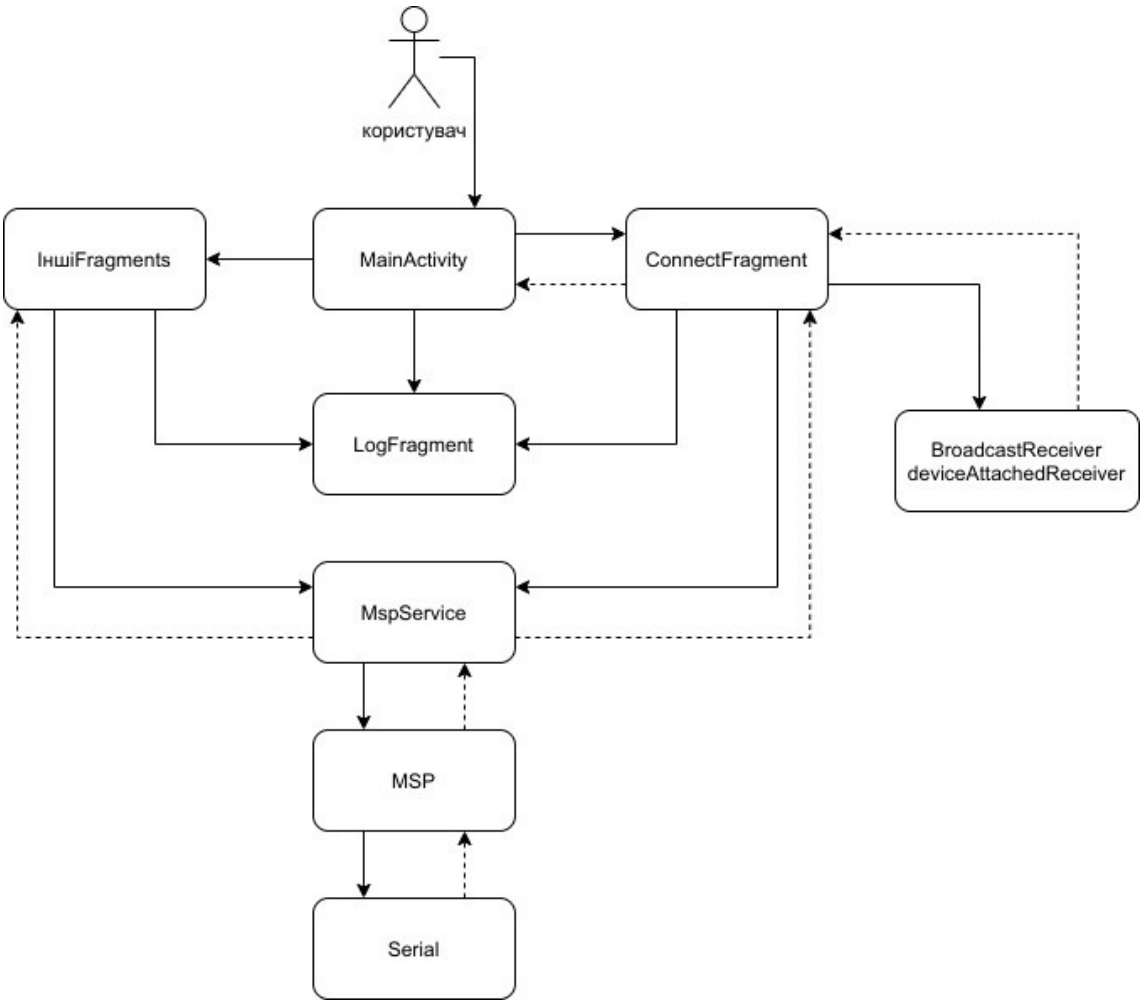


Рисунок 3.1 — Діаграма основних компонентів програми

3.2 Клас MainActivity — компонент <activity>

MainActivity — головний компонент програми. З цього класу починається робота, коли користувач запускає програму. Програму було спроектовано за шаблоном *Single Activity* — коли існує лише один Activity, а зміна інтерфейсу програми відбувається шляхом заміни фрагментів (клас Fragment, далі в тексті — фрагмент-екран). Після запуску програми, MainActivity створює перший фрагмент-екран — ConnectFragment, і показує його на весь екран, потім ініціює запуск сервісу MspService (компонент <service>).

Також, MainActivity створює навігаційне меню з лівого боку, котре не видно спочатку. Щоб відкрити бокове меню потрібно натиснути кнопку зверху зліва екрану, або ж свайпнути від лівої частини екрану в правий бік. На початку в меню є лише два пункти — Connect (укр. З'єднання) та About (укр. про програму). Після успішного з'єднання з польотним контролером, меню доповнюється іншими пунктами:

– Setup (перегляд поточної інформації);

– Ports (налаштування портів контролера);

– Configuration (загальні налаштування);

– Power & Battery (покази датчиків живлення);

– Failsafe (налаштування ситуацію обриву зв'язку);

– PID Tuning (алгоритму PID);

– Receiver (налаштування радіоприймача сигналу управління);

– Modes (режими польоту);

– Motors (перевірка моторів);

– CLI (консольний термінал).

3.3 Клас ConnectFragment — перший фрагмент-екран для MainActivity

ConnectFragment — частина графічного інтерфейсу, що показує список доступних пристроїв, та ініціює з'єднання з вибраним.

Відразу після створення, ConnectFragment звертається до MspService, котрий далі звертається до системного сервісу UsbManager, для отримання списку підключених до мобільного телефону пристроїв. Для постійного моніторингу підключення та відключення USB пристроїв (для оновлення їх списку) створюється екземпляр класу BroadcastReceiver.

BroadcastReceiver — це системний компонент, за допомогою якого, можна обмінюватися даними з іншими процесами системи. Найчастіше

використовується для отримання повідомлень від системи про певні зміни. Для функціонування, його потрібно зареєструвати за допомогою функції registerReceiver з додатковим аргументом типу IntentFilter.

Код реєстрації BroadcastReceiver для отримання повідомлень про підключення та відключення USB пристроїв:

```
  private BroadcastReceiver deviceAttachedReceiver = new BroadcastReceiver() {
      @Override
      public void onReceive(Context context, Intent intent) {
         updateDevices();
      }
  };
   IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction("android.hardware.usb.action.USB_DEVICE_ATTACH
ED");
    intentFilter.addAction("android.hardware.usb.action.USB_DEVICE_DETACH
ED");
   getMainActivity().registerReceiver(deviceAttachedReceiver, intentFilter);
```

Після вибору користувачем потрібного контролеру, ConnectFragment виконує з’єднання з контролером через USB за допомогою класу Service. Після з’єднання, створюється клас MSP, якому передається екземпляр класу Service, а екземпляр класу MSP передається до MspService, який його зберігає для подальшого використання. ConnectFragment зчитую першу порцію даних з контролеру через MspService (розділ 3.5). До цих даних входить:

– API версія прошивки контролера;

– назва прошивки та версія;

– дата релізу прошивки;

– назва контролеру та апаратна версія;

– унікальний ідентифікатор контролеру;

– ім’я оператора, що збережено у налаштуваннях контролеру.

Останньою функцією ConnectFragment (рис. 3.2) є сповіщення про успішне з’єднання класу MainActivity, щоб той виконав заміну фрагмента-екрана на інший.

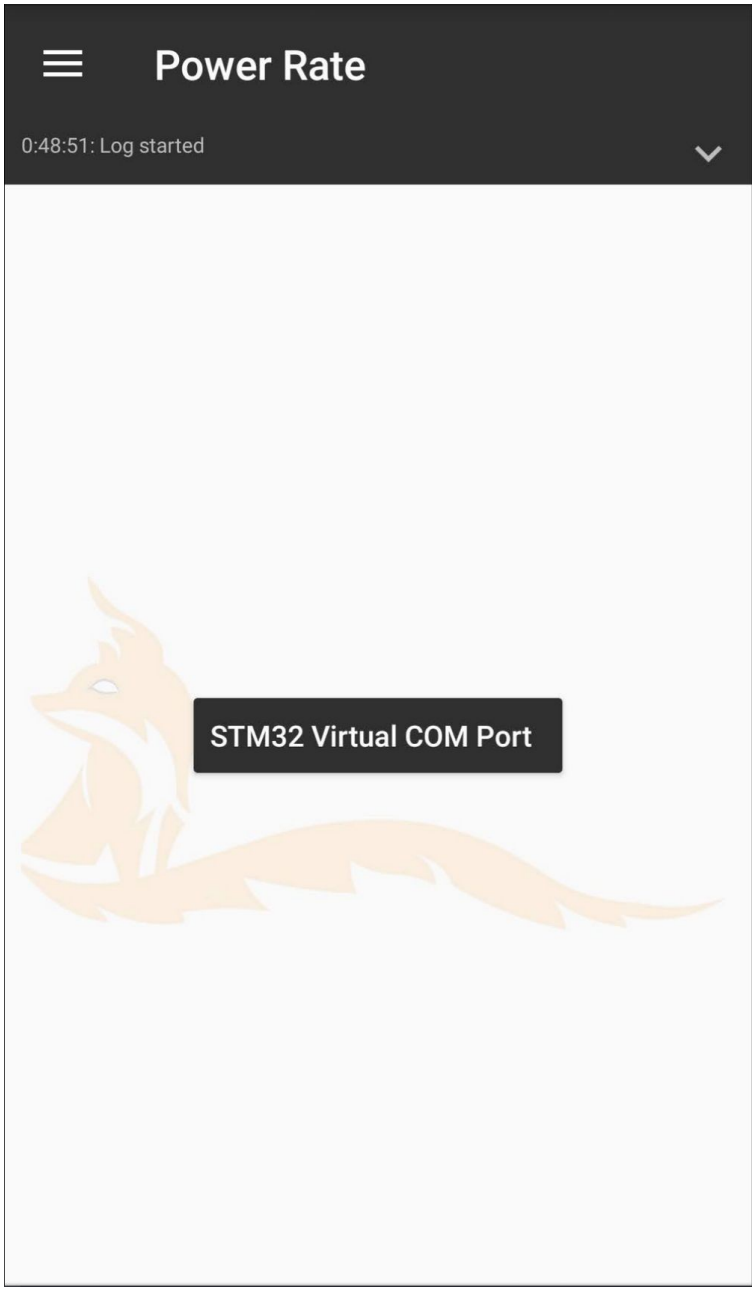


Рисунок 3.2 — Інтерфейс фрагмента-екрана ConnectFragment

3.4 Клас LogFragment

Даний клас є малим фрагментом інтерфейсу. Його призначення - відображення інформації щодо поточного з’єднання з контролером. Клас LogFragment реалізує інтерфейс Logger, через який інші фрагменти-екрани можуть відправляти до класу текстову інформацію для зберігання на час роботи з контролером. Текст інтерфейсу Logger:

```
public interface Logger {
   void log(CharSequence message);
}
```

Фрагмент LogFragment (рис. 3.3) закріплено зверху екрану. На початку фрагмент відображається невеликого розміру, де вміщається дві строки тексту. По кліку на фрагмент, він розгортається до більшого розміру, де вміщається 8-9 строк тексту. Так як фрагмент повинен зберігати весь текст що до нього надходить, в ньому реалізована система прокрутки текст по вертикалі.

Кожний раз, коли в фрагмент, додається новий текст — відбувається автоматична прокрутка до низу. До кожного тексту на початку також додається поточний час. Код додавання тексту (реалізація методу log інтерфейсу Logger):

```
   @Override
   public void log(CharSequence message) {
      if (textLog.length() > 0) {
         textLog.append("\n");
      }
```

```
        textLog.append(dateFormat.format(new Date()) + ": ");
        textLog.append(message);

        scrollDown();
    }
```

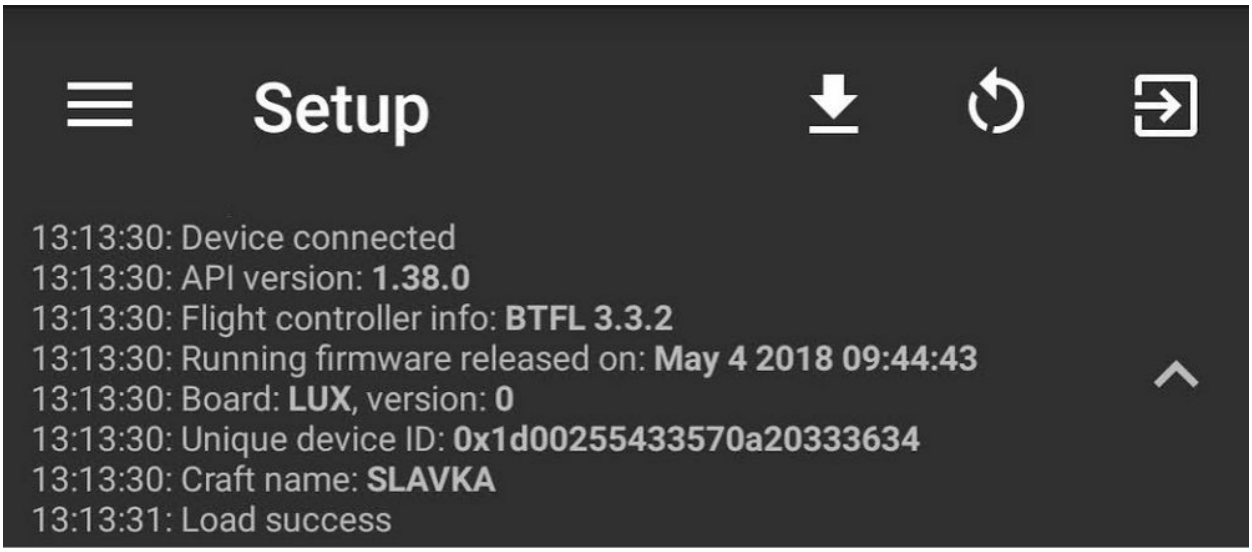


Рисунок 3.3 — Інтерфейс фрагменту LogFragment

3.5 Клас MspService — компонент <service>

Клас MspService виконує функцію сервісу, код якого виконується незалежно від графічного інтерфейсу. Одразу після запуску сервіс створює системне повідомлення (англ. Notification), яке користувач не може відмінити. Таке повідомлення потрібно для того, щоб Android система не зупиняла сервіс, коли користувач закриває графічний інтерфейс програми — MainActivity.

MspService також створює BroadcastReceiver для отримання сповіщення про відключення контролера від USB мобільного телефону. Код створення та реєстрації BroadcastReceiver:

```
private BroadcastReceiver deviceDetachedReceiver = new BroadcastReceiver() {
```

```
        @Override
        public void onReceive(Context context, Intent intent) {
UsbDevice device = intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);

            if (msp.getSerial().getUsbDevice().equals(device)) {
                disconnect();
            }
        }
    };
    private void registerDeviceDetached() {
        IntentFilter intentFilter = new IntentFilter();

intentFilter.addAction("android.hardware.usb.action.USB_DEVICE_DETACHED
");

        registerReceiver(deviceDetachedReceiver, intentFilter);
    }
```

Основна функція класу MspService — обмін даними з контролером по протоколу MSP. Для цього в класі створена система тасків (англ. Task). Алгоритм роботи тасків такий: Фрагмент-екран створює таску, наприклад для зчитування певних даних з контролеру, та передає таску до класу MspService. Усі таски мають два етапи: відправлення запиту до контролеру, та отримання від нього відповіді по протоколу MSP (детальніше принцип роботи протоколу MSP розглянуто у розділі 3.7).

Щоб уникнути одночасного виконування декількох тасків, що не передбачено архітектурою протоколу MSP, передбачено їх виконання почергово, за допомогою ScheduledExecutorService з одним потоком для виконання. Код створення екземпляру ScheduledExecutorService:

```
privateScheduledExecutorService scheduledExecutorService
...
scheduledExecutorService = Executors.newSingleThreadScheduledExecutor()
```

3.6 Serial — клас для роботи з USB з'єднанням

Клас Serial потрібен для реалізації логіки з'єднання з контролером, а також обміну даними з ним.

Клас Serial створюється через статичний метод connect:

```
public static Serial connect(Context context, int baudRate, UsbDevice usbDevice)
{
    Serial instance = new Serial();
    instance.usbManager = (UsbManager)
context.getSystemService(Context.USB_SERVICE);
    instance.baudRate = baudRate;
    instance.usbDevice = usbDevice;

    if (instance.connect(context)) {
       return instance;
    } else {
       return null;
    }
  }
```

Опис коду: створюємо екземпляр Serial, в який зберігаємо отриманий екземпляр системного сервісу UsbManager разом з baudRate та usbDevice з аргументів функції, та виконуємо метод connect.

В методі connect, для з'єднання з контролером, спочатку треба перевірити права доступу:

```
usbManager.hasPermission(usbDevice)
usbDevice — обраний пристрій, переданий класом ConnectFragment
```

Якщо прав немає, треба відправити до системи запит. Для цього створюється PendingIntent:

```
PendingIntent pendingIntent = PendingIntent.getBroadcast(context, 0, new
Intent(ACTION_USB_PERMISSION), 0);
usbManager.requestPermission(usbDevice, pendingIntent);
```

Для відстеження повідомлення про результат запиту — створюється UsbGrantReceiver (який розширює BroadcastReceiver). UsbGrantReceiver реєструється разом з фільтром IntentFilter(ACTION_USB_PERMISSION):

```
IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
usbGrantReceiver = new UsbGrantReceiver();
context.registerReceiver(usbGrantReceiver, filter);
```

UsbGrantReceiver по отриманні результату виконує метод openDevice (якщо успіх), та errorOpenDevice (якщо помилка):

```
private class UsbGrantReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
   String action = intent.getAction();
```

```
        if (action != null && action.equals(ACTION_USB_PERMISSION)) {
            if
(intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED,
false)) {
                openDevice(context);
            } else {
                errorOpenDevice(context);
            }
        }
    }
```

Якщо ж права є (метод hasPermission повернув true), то відразу виконуємо метод openDevice.

В методі openDevice створюється екземпляр класу UsbSerialDevice (бібліотека UsbSerial), якому задаються параметри з’єднання з контролером, та коллбек-метод для отримання прочитаних даних:

```
serialDevice.setBaudRate(baudRate);
serialDevice.setDataBits(UsbSerialInterface.DATA_BITS_8);
serialDevice.setStopBits(UsbSerialInterface.STOP_BITS_1);
serialDevice.setParity(UsbSerialInterface.PARITY_NONE);
serialDevice.setFlowControl(UsbSerialInterface.FLOW_CONTROL_OFF)
serialDevice.read(mUsbReadCallback)
```

3.7 MSP — клас для роботи з протоколом обміну даними MSP

Протокол MSP (MultiWii Serial Protocol) - послідовний протокол, є фактичним стандартом для взаємодії з контролером польоту MultiWii. Його

реалізація містить перелік найпоширеніших операцій, які можна очікувати з точки зору пульта дистанційного керування / телеметрії. За необхідності розробники можуть додати власну функціональність.

Обмін даними по протоколу MSP відбувається за допомогою пакетів. Пакет MSP має певну структуру. Вона складається із заголовку, розміру, типу, даних (якщо є) та контрольної суми.

Опис структури пакету протоколу MSP (рис. 3.4):

– header — заголовок з трьох байтів, завжди починається з символів “$M” (2 байти). Третій байт вказує напрям руху пакету: “<” - напрям до контролера, “>” - напрям від контролера;

– size — розмір даних, що містяться в payload;

– type — тип пакету, вказує тип даних які містить пакет. Тип даних залежить від прошивки. Приклад — MSP_ATTITUDE (108);

– payload — дані, якщо є (size не 0);

– crc - кінцевим байтом пакету MSP є контрольна сума. Контрольна сума - це результат послідовного виконання операції XOR над байтами size, type та payload. Для пакету request (запит) контрольна сума дорівнює типу.

Приклади пакетів MSP (рис. 3.4):

– command - пакет, що відправляються контролеру, яке містить певні дані у полі payload (зазвичай використовується для запису даних у пам’ять контролера);

– request - пакет, що відправляються контролеру без даних (для запиту даних з контролеру);

– response - пакет, отриманий від контролера (з даними).

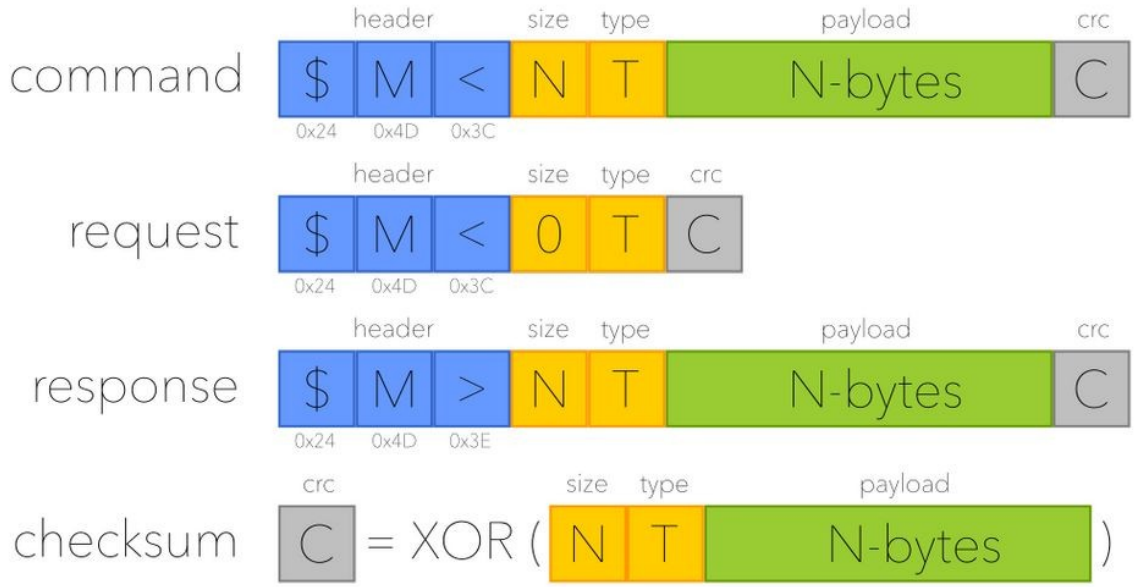


Рисунок 3.4 — Приклади та структура пакетів протоколу MSP

Клас MSP містить алгоритми для оперування пакетами протоколу MSP. Має метод для формування пакету та відправлення до контролера:

```
public void sendMessage(int code, byte[] data) {
    byte[] buffer;

    if (data != null) {
        byte checksum;
        buffer = new byte[data.length + 6];
        buffer[0] = 36; // $
        buffer[1] = 77; // M
        buffer[2] = 60; // <
        buffer[3] = (byte) data.length; // data length
        buffer[4] = (byte) code; // code
        checksum = (byte) (buffer[3] ^ buffer[4]);
        for (int i = 0; i < data.length; i++) {
```

```
            buffer[i + 5] = data[i];

            checksum ^= buffer[i + 5];
        }
        buffer[5 + data.length] = checksum;
    } else {
        buffer = new byte[6];
        buffer[0] = 36; // $
        buffer[1] = 77; // M
        buffer[2] = 60; // <
        buffer[3] = 0; // data length
        buffer[4] = (byte) code; // code
        buffer[5] = (byte) (buffer[3] ^ buffer[4]); // checksum
    }
    serial.write(buffer);
}
```

Клас MSP відстежує отримані сирі дані від класу Serial, та обробляє їх з урахуванням алгоритму протоколу MSP у методі onRead.

Клас MSP також створює два інших класи, які допомагають оброблювати пакети від прошивки Betaflight:

– MSPHandler — перетворює пакет MSP у дані, для використання у фрагменті-екрані графічного інтерфейсу;

– MSPHelper — працює навпаки, створює пакет MSP, з даних, отриманих від фрагменту-екрану графічного інтерфейсу;

Метод onRead класу MSP після закінчення обробки байтів створює об’єкт класу MSPPacket і передає його на подальшу обробку до класу MSPHandler.

Приклад обробки пакету класом MSPHandler:

```
            mspData.batteryConfig.vbatmincellvoltage = mspPacket.readU8() /
10.0f; // 10-50
            mspData.batteryConfig.vbatmaxcellvoltage = mspPacket.readU8() /
10.0f; // 10-50
            mspData.batteryConfig.vbatwarningcellvoltage = mspPacket.readU8() /
10.0f; // 10-50
            mspData.batteryConfig.capacity = mspPacket.readU16();
            mspData.batteryConfig.voltageMeterSource = mspPacket.readU8();
            mspData.batteryConfig.currentMeterSource = mspPacket.readU8();
```

Приклад створення пакету класом MSPHelper:

```
          mspPacket.push8((int) (mspData.batteryConfig.vbatmincellvoltage * 10));
            mspPacket.push8((int) (mspData.batteryConfig.vbatmaxcellvoltage *
10));
            mspPacket.push8((int) (mspData.batteryConfig.vbatwarningcellvoltage *
10));
            mspPacket.push16(mspData.batteryConfig.capacity);
            mspPacket.push8(mspData.batteryConfig.voltageMeterSource);
            mspPacket.push8(mspData.batteryConfig.currentMeterSource);
```

3.8 ІншіFragment — класи, для реалізації окремих груп налаштувань

Терміном ІншіFragment було узагальнено усі інші фрагменти-екрани, котрі відображають певні налаштування контролеру. Кожен з цих фрагментів має свій пункт у боковому меню. Кожен клас фрагмента-екрана розширює клас BaseTabFragment, який розширює клас MspTaskFragment, який в свою чергу розширює клас MspServiceFragment (рис. 3.5)

Почнемо розглядати класи з кінця, тобто з MspServiceFragment. Клас MspServiceFragment виконує підключення до сервісу MspService в системному методі onStart (метод викликається коли програма стає видимою на дисплеї). Після успішного підключення викликається абстрактний метод onServiceBound, для сповіщення свого дочірнього класу.

Клас MspTaskFragment виконує функцію посередника між BaseTabFragment та MspServiceFragment для обміну даними з сервісом MspService, шляхом передачі таски, також показує анімацію прогресу виконання операції.

Клас BaseTabFragment при створенні ініціює завантаження первинних даних з контролеру. Також виконує роботу з меню зверху екрану. Перелік пунктів меню:

– Save — зберегти зміни у пам'ять контролера;

– Load — заново зчитати дані з пам'яті контролера;

– Reboot — перезавантажити контролер;

– Disconnect — відключитися від контролера.

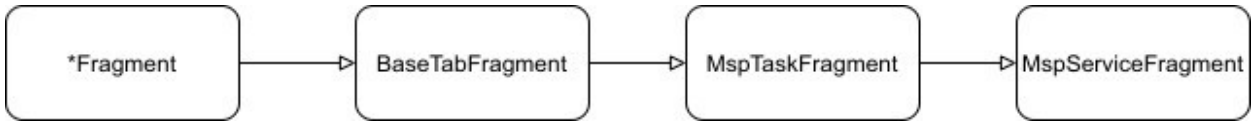


Рисунок 3.5 — Діаграма розширення класів фрагментів-екранів

3.9 Структура файлів проекту

Головними файлами програми є AndroidManifest.xml (в ньому реєструються головні компоненти програми), та MainActivity.java (клас, з якого починається робота програми). Структура файлів програми наведена на рис. 3.6.

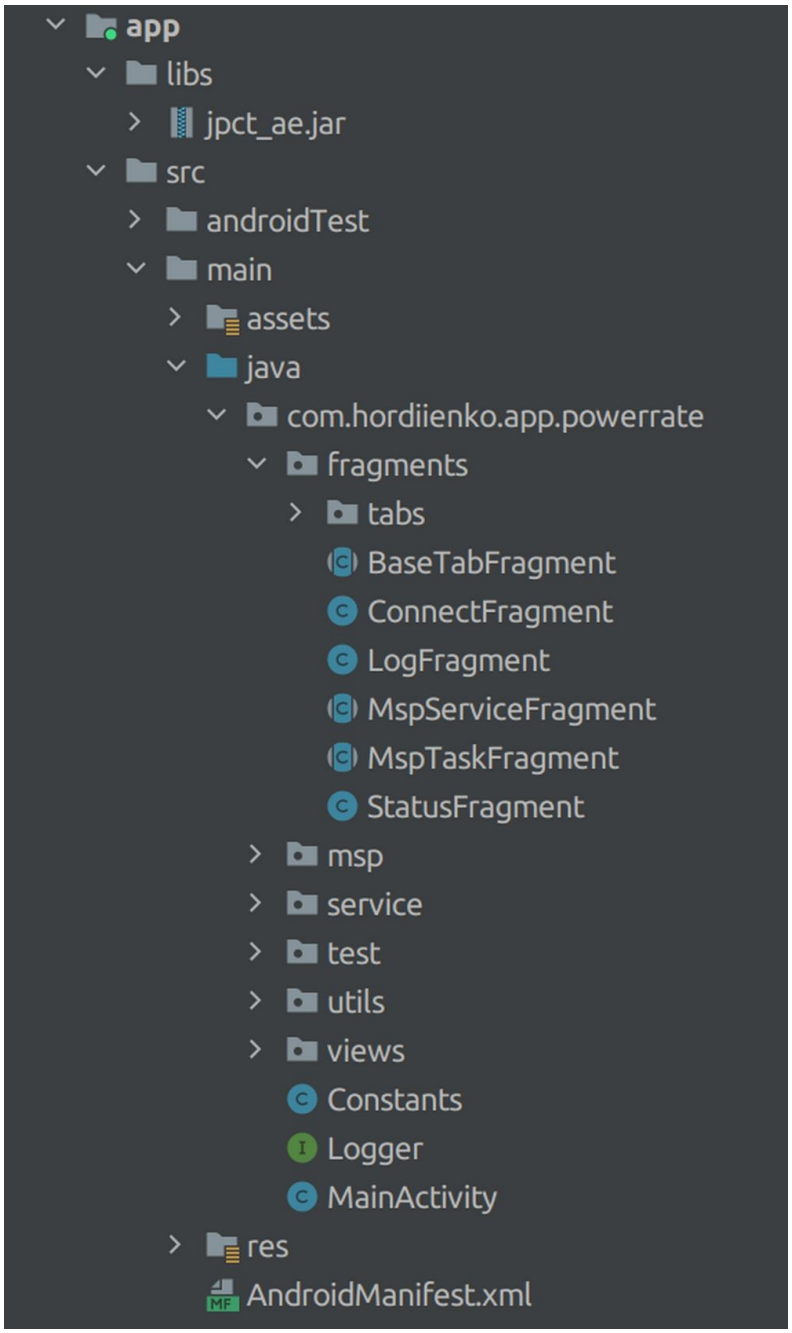


Рисунок 3.6 — Структура файлів проекту

У директорії src/main/java знаходяться файли програмного коду. У директорії src/main/res знаходяться файли з ресурсами (розмітка фрагментів у форматі xml, зображення, меню, іконки, тощо).

Розглянемо директорію src/main/java більш детально:

а) директорія fragments — тут знаходяться базові класи фрагментів-екранів та класи невеликих фрагментів:

1) ConnectFragment — фрагмент-екран пошуку USB пристроїв та підключення до них;

2) LogFragment — фрагмент відображення лог інформації зверху екрану;

3) StatusFragment — фрагмент що завжди присутній внизу екрану, відображає поточну інформацію контролеру;

б) директорія fragments/tabs — фрагменти-екрани, що відображають певні налаштування контролера, згруповані між собою;

в) директорія msp — тут знаходяться класи для обробки пакетів обміну даними з контролером по протоколу MSP;

г) директорія service — класи для роботи з USB та клас Service, для роботи незалежно від графічного інтерфейсу;

ґ) директорія utils — набір класів що містять допоміжний код;

д) директорія views — набір різних view, що розроблені для візуалізації даних контролеру, для яких немає view у стандартному наборі системи Android.

3.10 Висновки до розділу 3

У третьому розділі було розглянуто основні необхідні компоненти при розробці програми для операційної системи Android. Було розглянуто та описано реалізацію компонентів <activity> за класом MainActivity, та <service> за класом MspService. Також було описано протокол MSP, структуру його пакетів, та роботу з ними через класи MSPHandler та MSPHelper.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення на умови застосування програми

Застосунок створений для налаштування польотного контролера квадрокоптера з прошивкою Betaflight. Перевага застосунку перед офіційною програмою для персонального комп'ютера Betaflight Configurator — у економії місця серед речей оператора квадрокоптера. Для використання застосунку достатньо мати при собі мобільний телефон з операційною системою Android версії 4.0 чи новіше. Також знадобиться microUSB — USB кабель, та перехідник USB-OTG (рис. 4.1).

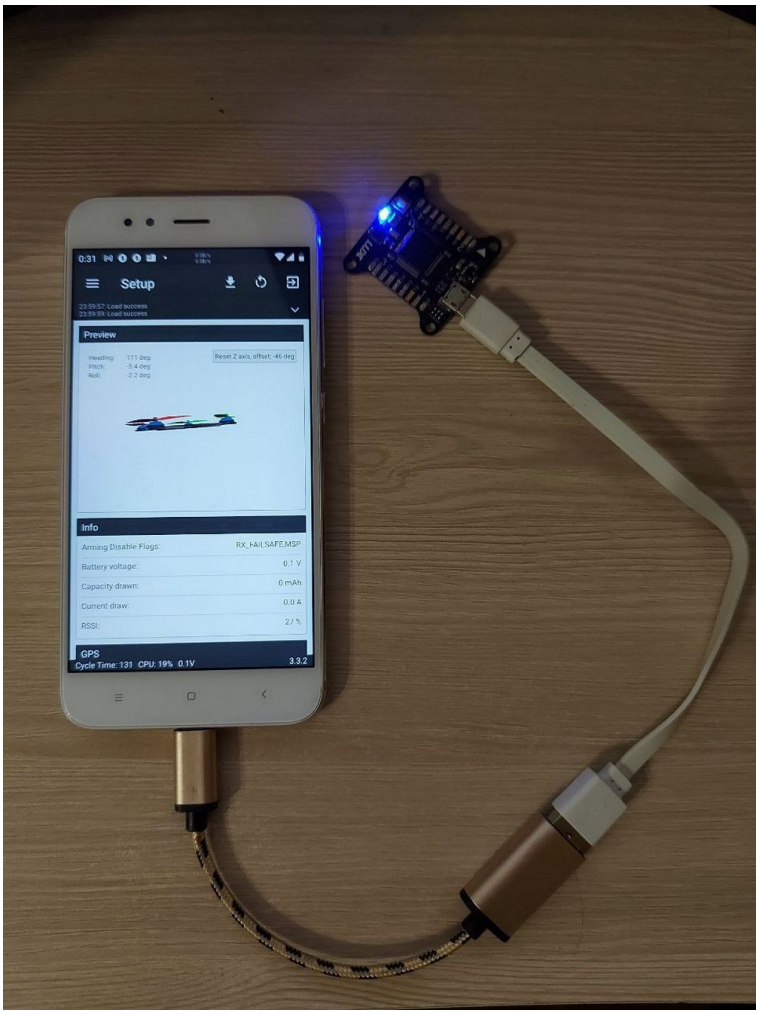


Рисунок 4.1 — Фото контролера LUX підключеного до мобільного телефону

4.2 Робота із застосунком

4.2.1 Підключення контролера до мобільного телефону

Як було вказано у розділі 4.1, для підключення контролера до мобільного телефону потрібні кабель microUSB — USB та перехідник USB-OTG.

USB-OTG (USB On-The-Go) розширює специфікації USB 2.0 для легкого з'єднання периферійних USB-пристроїв безпосередньо між собою без використання персонального комп'ютера. Прикладом застосування цієї технології є можливість під’єднання фотоапарата безпосередньо до пристрою друку. Цей стандарт виник через об'єктивну потребу надійного з'єднання особливо поширених USB-пристроїв без застосування комп'ютера, якого у потрібний момент може і не бути під рукою [7].

Після запуску застосунку першим екраном-фрагментом (далі екран) буде ConnectFragment. Екран покаже список доступних для підключення контролерів. Далі треба обрати контролер, для цього потрібно натиснути на його назву. У разі потреби застосунок може запитати дозвіл для застосунку для підключення до пристрою (рис. 4.2). Після успішного підключення екран автоматично буде змінено на екран Setup.

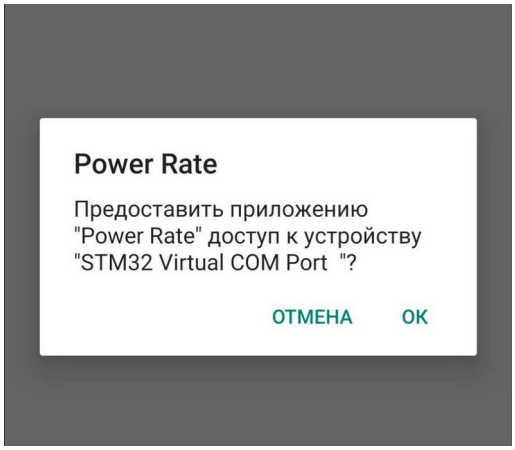


Рисунок 4.2 — Система запитую дозвіл для застосунку на підключення до контролера

4.2.2 Навігаційне меню

У застосунку є бокове навігаційне меню (англ. Navigation Drawer). Список пунктів меню залежить від того, чи виконано з’єднання з контролером. Якщо ні — в меню буде тільки два пункти:

– Connect (підключення);

– About (про застосунок).

Якщо ж з’єднання було виконано, меню зміниться на наступне (кожен пункт меню відповідає окремому екрану) (рис. 4.3):

– Setup;

– Ports;

– Configuration;

– Power & Battery;

– Failsafe;

– PID Tuning;

– Receiver;

– Modes;

– Motors;

– CLI.

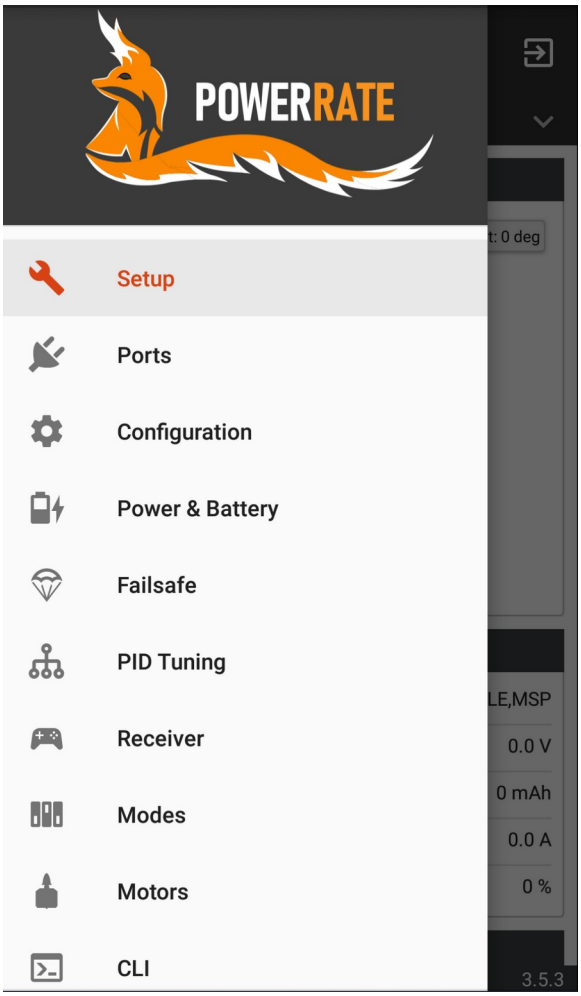


Рисунок 4.3 — Бокове меню навігації в режимі з’єднання

4.2.3 Екран Setup

Перший фрагмент-екран, який відображається після успішного підключення до контролеру. Зверху екрана показується 3D модель квадрокоптера, для візуалізації поточного положення у просторі (кути нахилу). Також екран Setup показує користувачеві основну інформацію з датчиків контролера (рис. 4.4):

– поточний статус;

– напруга батареї в Вольтах;

– сила току - навантаження на батарею в Амперах;

– кількість спожитої енергії в мА/год;

– рівень сигналу радіозв’язку в RSSI;

– кількість знайдених супутників GPS;

– висота над рівнем моря за даними GPS;

– широта за даними GPS;

– довгота за даними GPS.

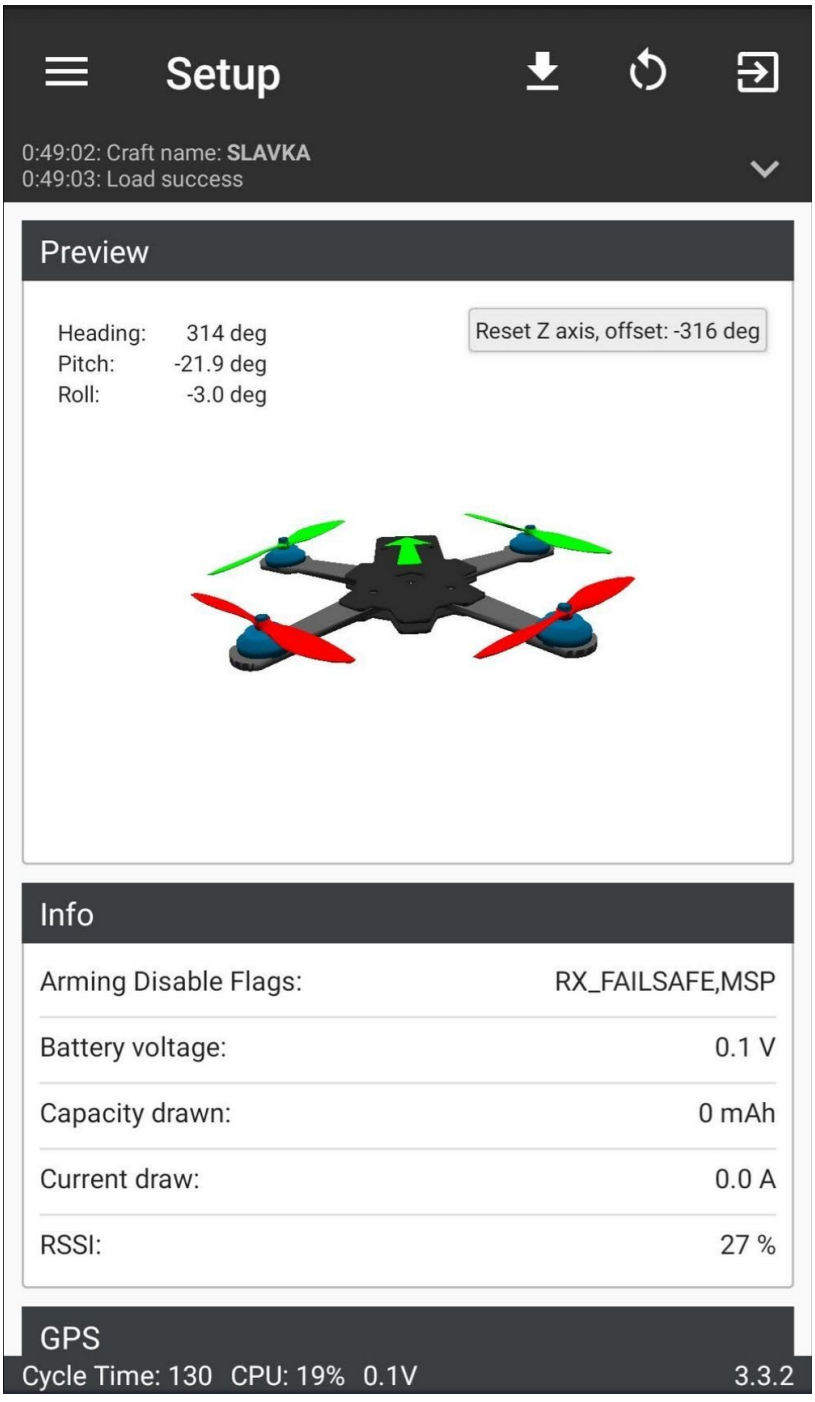


Рисунок 4.4 — Екран Setup

4.2.4 Екран Ports

На цьому екрані можна налаштувати порти контролера — установити тип телеметрії, тип сенсору чи тип додаткового датчика (рис. 4.5). Також можна налаштувати швидкість обміну даними на порту. Порт контролера — це його апаратний UART [8] інтерфейс. До UART можна підключати різні пристрої та датчики. Типовий контролер має щонайменш 3-4 порти UART. Перший завжди підключений до USB, за допомогою якого користувач з'єднує контролер за мобільним телефоном.

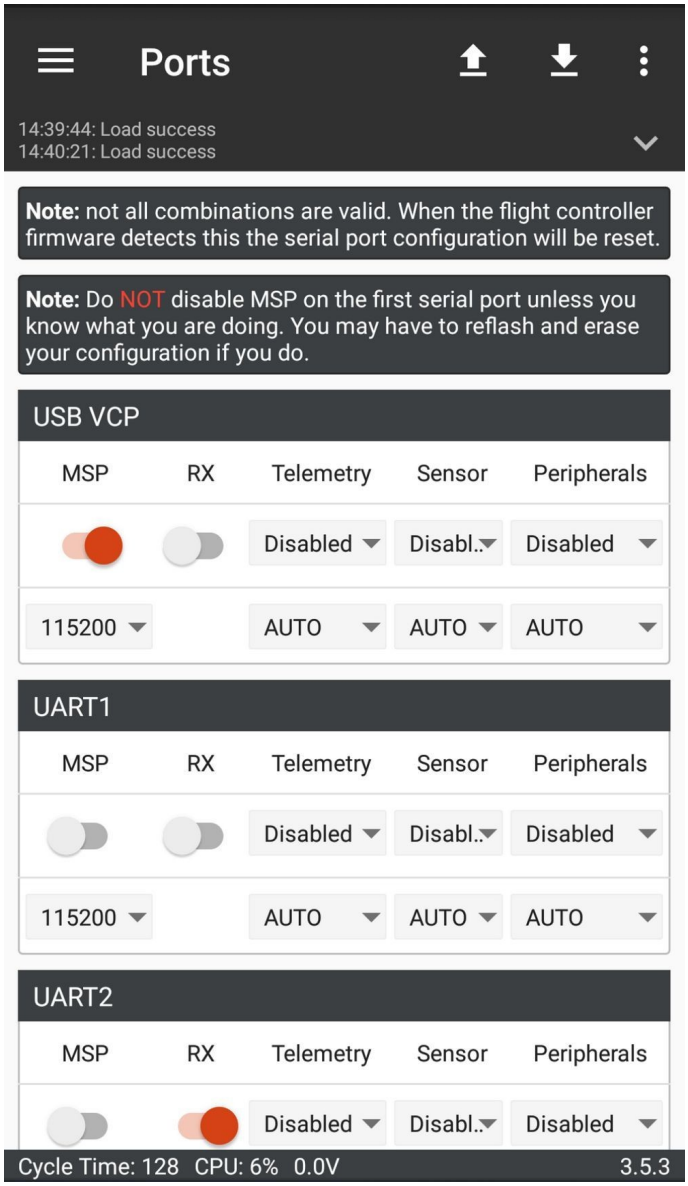


Рисунок 4.5 — Екран Ports

4.2.5 Екран Configuration

Екран Configuration містить найбільше налаштувань (рис. 4.6). Тут користувач може обрати:

– тип дрону (кількість моторів та їх розташування);

– напрям обертання моторів;

– частоту опитування гіроскопу процесором;

– частоту роботи алгоритму PID;

– ввімкнення датчиків: акселерометру, барометру, компасу;

– вибір протоколу обміну даними з ESC;

– налаштування граничних сигналів від радіопередавача;

– положення контролера на корпусі квадрокоптера;

– ввімкнення та налаштування датчику GPS;

– ввімкнення додаткових функцій;

– налаштування звукового сповіщення на певні дії контролера.

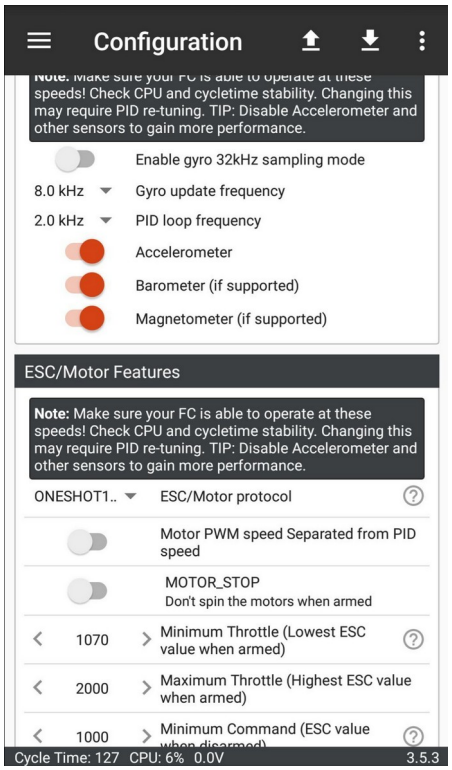


Рисунок 4.6 — Екран Configuration

4.2.6 Екран Power & Battery

На екрані Power & Battery користувач може переглядати поточну інформацію про стан батареї та виконувати деякі налаштування пов’язані з нею, серед яких (рис. 4.7):

– вибір датчику для вимірювання напруги;

– вибір датчику для вимірювання сили струму;

– встановлення мінімальної та максимальної границь напруги;

– налаштування масштабування вимірювання датчику напруги;

– налаштування масштабування вимірювання датчику сили струму.

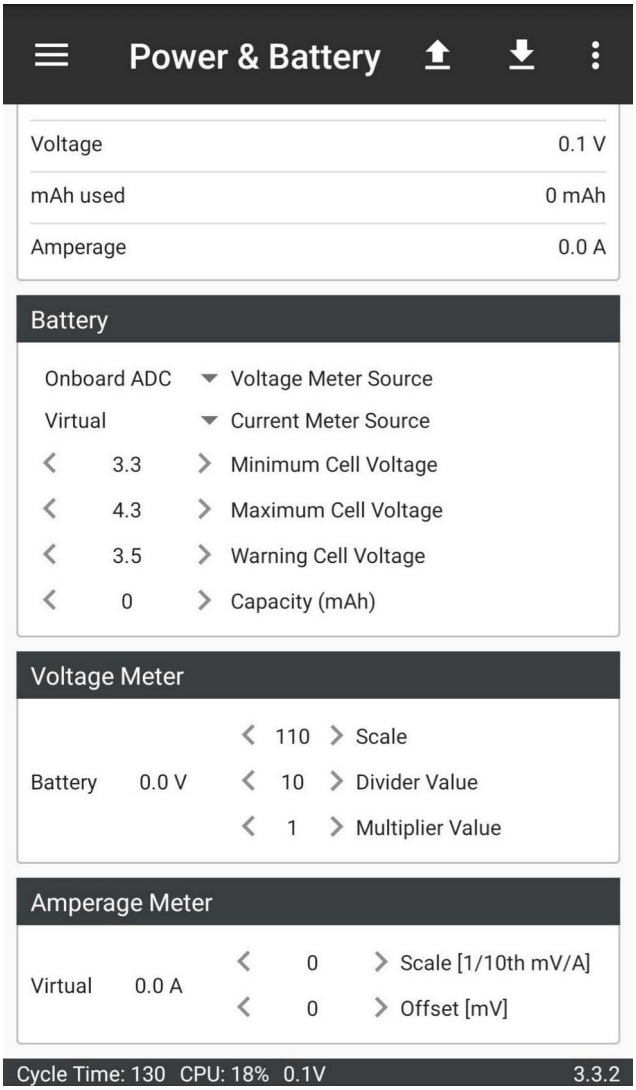


Рисунок 4.7 — Екран Power & Battery

4.2.7 Екран Failsafe

На екрані Failsafe користувач може вибрати та налаштувати процедуру, що буде виконуватися у разі виникнення ситуації обриву зв'язку з пультом управління (рис. 4.8). Це може статися через перешкодження радіосигналу, або ж коли квадрокоптер опинився занадто далеко від оператора. Також режим Failsafe може бути запущений контролером у разі виникнення помилки чи збою у його роботі, наприклад відмова двигуна. Загалом на вибір дається два варіанти процедур:

– вимкнення моторів;

– плавний спуск.

На екрані також можна вибрати канал радіозв'язку, який буде відстежуватися на виникнення обриву зв'язку.

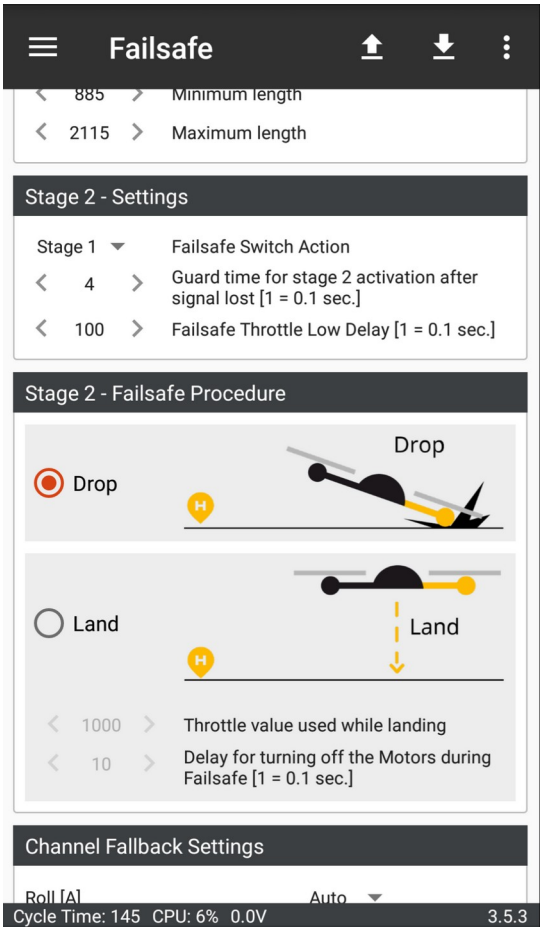


Рисунок 4.8 — Екран Failsafe

4.2.8 Екран PID Tuning

На екрані PID Tuning зосереджені налаштування PID алгоритму обробки даних з датчику гіроскопу (рис. 4.9). Кожен параметр з PID можна налаштувати окремо для кожної з осей: ROLL (крен вправо-вліво), PITCH (нахил вперед-назад) та YAW (обертання по вертикальній осі).

Також можна налаштувати чутливість сигналів від радіоприймача окремо по ROLL, PITCH та YAW, з урахуванням експоненціальної функції.

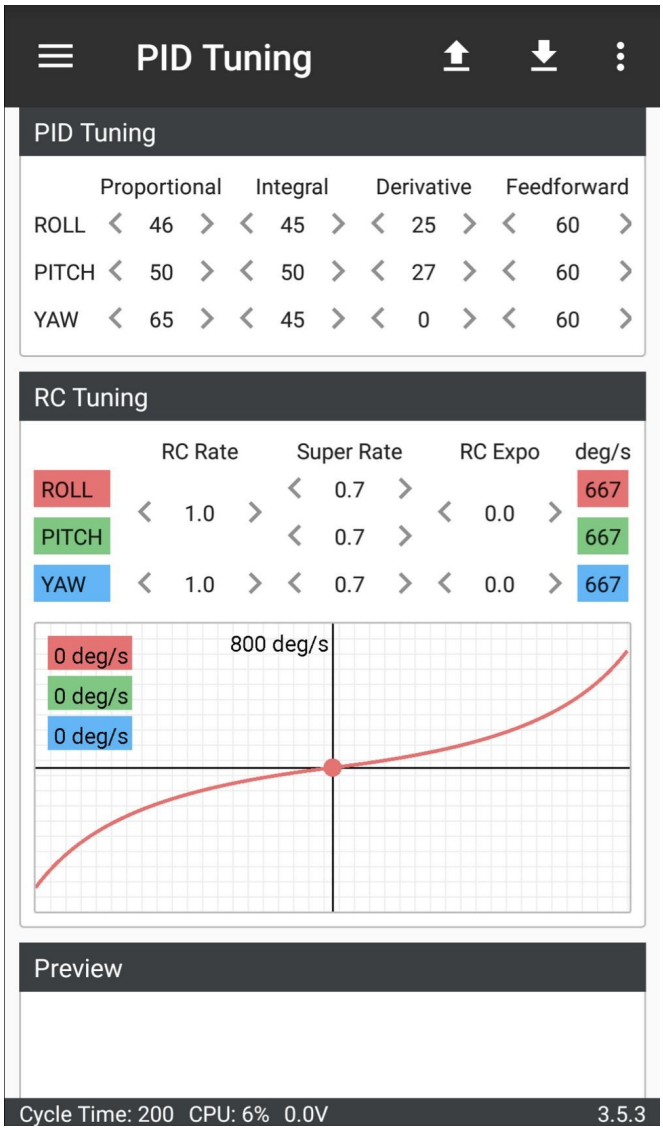


Рисунок 4.9 — Екран PID Tuning

4.2.9 Екран Receiver

Екран Receiver містить не багато налаштувань, передусім він призначений для перегляду поточного сигналу по кожному з каналів радіоприймача (рис. 4.10). Кількість каналів залежить від радіопередавача та радіоприймача. Також тут є 3D модель для візуального перегляду зміни положення моделі у просторі на зміни радіо сигналу. Це дає змогу оператору перевірити налаштування алгоритмів з екрану PID Tuning.

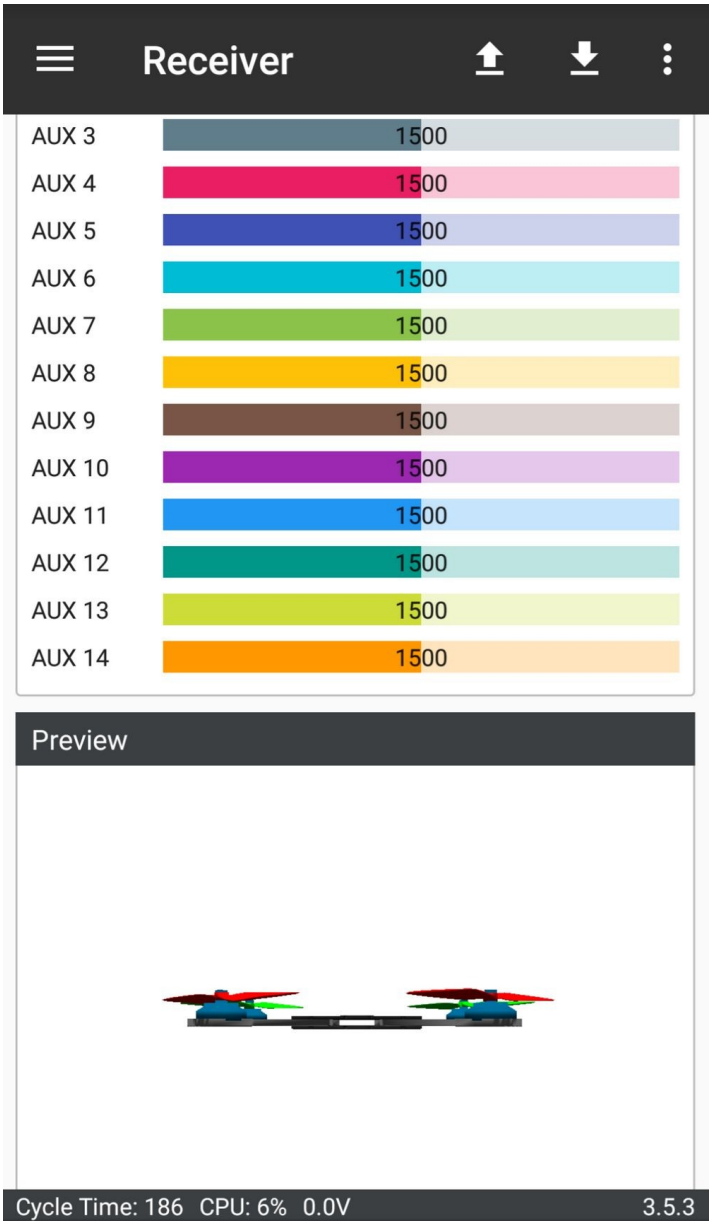


Рисунок 4.10 — Екран Receiver

4.2.10 Екран Modes

Екран Modes дає можливість налаштувати режими польоту (рис. 4.11). Перемикання режиму налаштовується на радіоканал, можна вибрати певний діапазон сигналу, в якому режим буде активовано. Таким чином на один канал можна налаштувати перемикання декількох режимів.

Для активації чи деактивації режиму треба натиснути на його назву. Після активації, праворуч від назви з'являться елементи управління режимом. Канал можна вибрати за допомогою випадаючого меню, а діапазон сигналу за допомогою двох слайдерів.

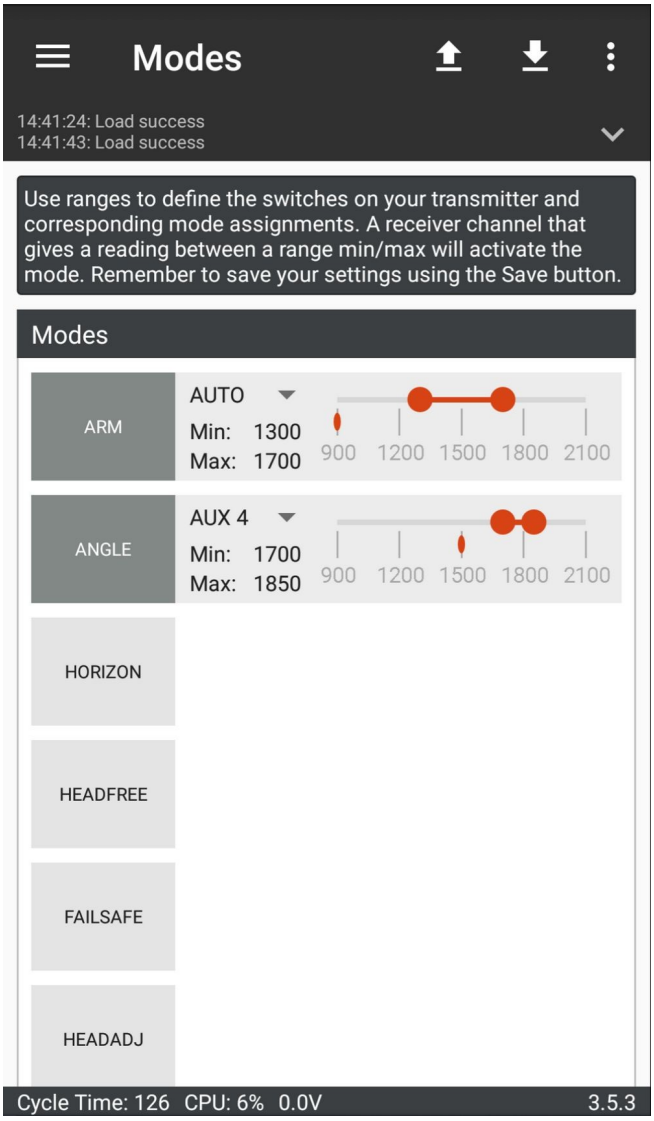


Рисунок 4.11 — Екран Modes

4.2.11 Екран Motors

На екрані Motors користувач може перевірити роботу моторів (рис. 4.12). Для цього передбачено по одному слайдеру для кожного мотора. Спочатку треба прочитати застереження та ввімкнути перемикач, потом перетаскувати слайдер потрібного мотора для його запуску на обраній швидкості, або ж слайдер Master для одночасного запуск усіх моторів.

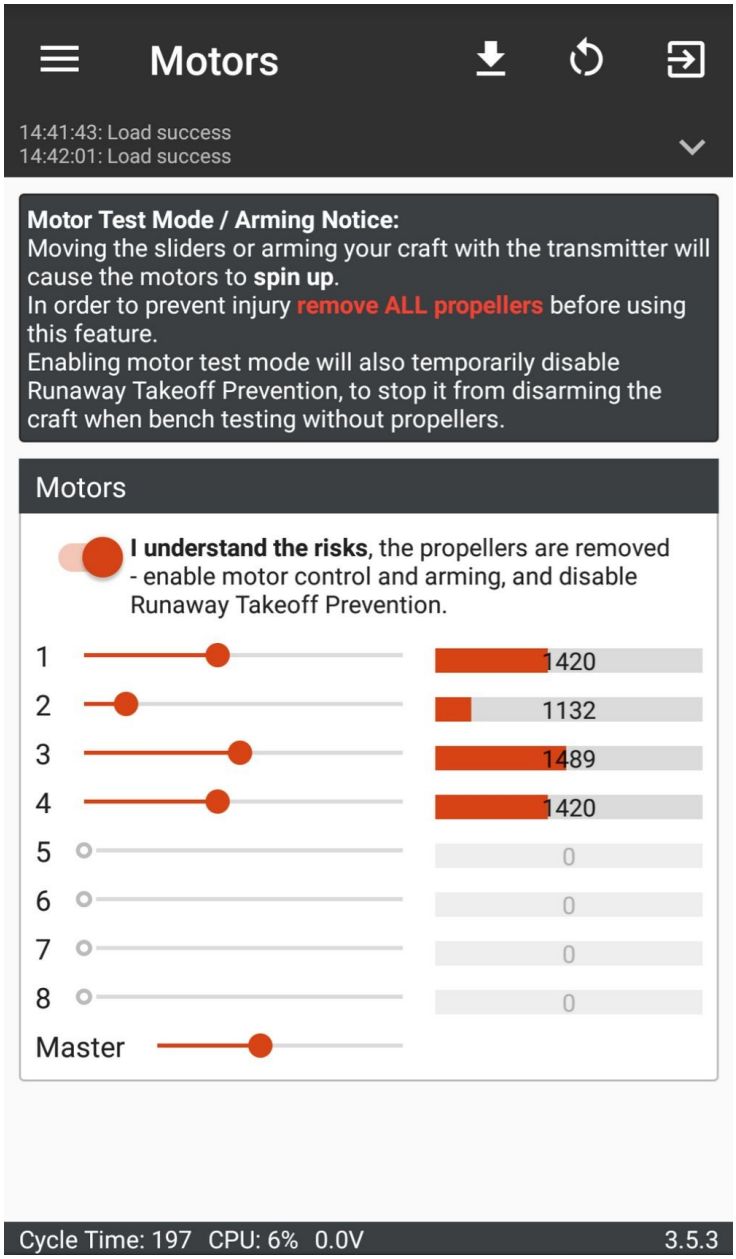


Рисунок 4.12 — Екран Motors

4.2.12 Екран CLI

Останній екран з меню — CLI (Command line interface). У даного екрану функцій не багато, а точніше — всього дві. Перша — вводити команду що буде відправлено до контролеру, друга — вивід відповіді від контролеру на екран. На примір введемо команду dump, у відповідь ми отримаємо перелік усіх параметрів контролеру, по одному на кожній із строк (рис. 4.13).

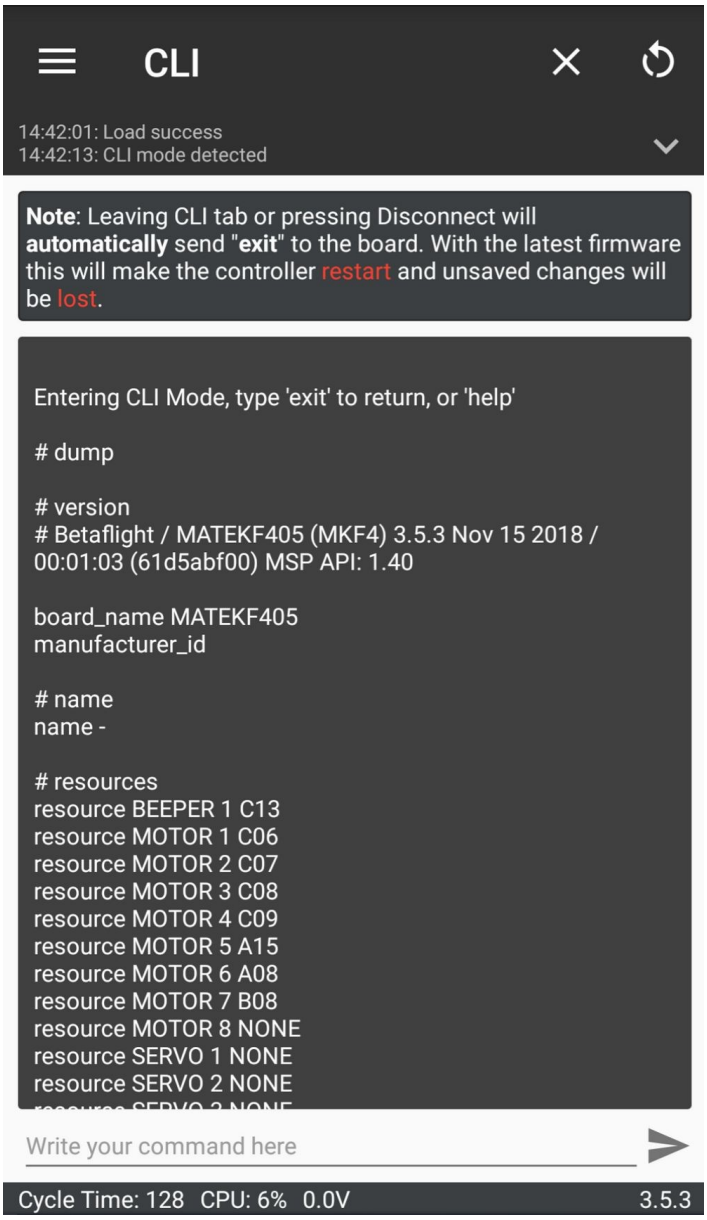


Рисунок 4.13 — Екран CLI

4.3 Висновки до розділу 4

У четвертому розділі було протестовано роботу застосунку після підключення контролеру LUX до мобільного телефону з операційною системою Android. По ходу тестування застосунку було розглянуто та описано можливості кожного екрану. По закінченні було виявлено, що застосунок відповідає усім функціональним вимогам.

ВИСНОВКИ

В результаті виконання дипломної кваліфікаційної роботи бакалавра виконано аналіз необхідності мобільного застосунку для налаштування польотного контролера квадрокоптера. На підставі проведеного аналізу існуючих рішень на функціональних вимог до програми, було сформовано технічне завдання, та структуру компонентів системи у вигляді діаграми виконаної мовою UML.

Застосунок був розроблений мовою програмування Java. Розробка велася у інтегрованому середовищі розробки Android Studio. Для розробки також була використана бібліотека для роботи з USB інтерфейсом UsbSerial від іспанського розробника Felipe Herranz (felHR85).

Було описано схему роботи протоколу MSP, зокрема розглянута структура його пакетів, за допомогою яких застосунок виконує обмін даними з контролером.

В результаті виконання завдання застосунок було розроблено з зручним та інтуїтивно зрозумілим інтерфейсом. Застосунок має декілька екранів, що дозволяють виконувати налаштування параметрів контролера у зручний для користувача спосіб. Також розроблено бокове меню для зручного перемикання екранів. Було протестовано функціональність усіх компонентів програми.

Всі завдання, поставлені у дипломній кваліфікаційній роботі, були повністю виконані.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Holybro Kopis 2 HDV 6S 5 - GeekBuying [Електронний ресурс]. - Режим доступу: https://www.geekbuying.com/item/Holybro-Kopis-2-HDV-6S-5-Inch-FPV-Racing-Drone-PNP-Version-420906.html

2. Speedy Bee – Додатки в Google Play [Електронний ресурс]. - Режим доступу: https://play.google.com/store/apps/details?id=com.runcam.android.runcambf

3. Гіроскоп [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/%D0%93%D1%96%D1%80%D0%BE%D1%81%D0%BA%D0%BE%D0%BF

4. Пропорційно-інтегрально-диференціальний закон регулювання [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BF%D0%BE%D1%80%D1%86%D1%96%D0%B9%D0%BD%D0%BE-%D1%96%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE-%D0%B4%D0%B8%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D1%96%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D0%B7%D0%B0%D0%BA%D0%BE%D0%BD_%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8E%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F

5. MSP - The MultiWii Serial Protocol — Web Archive [Електронний ресурс]. - Режим доступу: https://web.archive.org/web/20190715222846/http://www.stefanocottafavi.com/msp-the-multiwii-serial-protocol/

6. Eclipse [Електронний ресурс]. - Режим доступу: [Електронний ресурс]. - Режим доступу:

7. USB On-The-Go [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/USB_On-The-Go

8. UART [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/UART

9. Betaflight [Електронний ресурс]. - Режим доступу: https://github.com/betaflight/betaflight

10. Герберт Шилдт. Java. Руководство для начинающих [Текст]: 5-е издание / Герберт Шилдт. – М. : Вильямс, 2012. – 624 с.

11. Герберт Шилдт. Java. Полное руководство [Текст]: 8-е издание / Герберт Шилдт. – М. : Вильямс, 2013. – 1104 с.

ДОДАТОК А

Технічне завдання

Вступ

Метою проекту є розробка програми для забезпечення можливості налаштування польотного контролера квадрокоптера за мобільного телефону. Програма повинна бути створена у вигляді застосунку для операційної системи Android.

А.1 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему “Розробка Android застосунку для налаштування польотного контролера квадрокоптера”, затверджене наказом № 86 від 20 березня 2021 р. по Національному університету «Запорізька політехніка».

А.2 Призначення розробки

Програма призначена для надання користувачам можливості налаштування польотного контролера квадрокоптера за мобільного телефону.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Програма повинна надавати наступні функціональні можливості користувачам:

– показувати список підключених до мобільного телефону контролерів;

– виконувати з’єднання з контролером по протоколу MSP;

– виконувати обмін даними з контролером по протоколу MSP (зчитування та запис);

– показувати бокове навігаційне меню;

– показувати окремі екрани для кожної групи налаштувань.

А.3.2 Вимоги до надійності

Програма повинна забезпечувати оброблення наступних ситуацій:

– система не надала дозвіл на підключення контролера, потрібно запитати дозвіл у користувача;

– не вдалося виконати підключення до контролера;

– не вдалося виконати з’єднання по протоколу MSP;

– обрив підключення з контролером.

А.3.3 Умови експлуатації

Для забезпечення експлуатації програми необхідно встановити “Android застосунок для налаштування польотного контролера квадрокоптера” на мобільний телефон.

А.3.4 Вимоги до складу та параметрів технічних засобів

Для забезпечення експлуатації програми користувач повинен мати:

– мобільний телефон з операційною системою Android версії 4.0 чи вище;

– microUSB — USB кабель;

– перехідник USB-OTG.

А.3.5 Вимоги до маркування і упакування

Програма може бути записана на USB-носії або оптичному чи жорсткому магнітному диску. На упаковці має бути вказано назву роботи «Android застосунок для налаштування польотного контролера квадрокоптера».

А.3.6 Вимоги до транспортування і збереження

Вимоги до транспортування та зберігання повністю ідентичні вимогам, пропонованим до носія, на якому зберігається програма.

А.4 Стадії та етапи розробки

Для роботи над дипломною кваліфікаційною роботою бакалавра заплановано виконання наступних етапів:

– аналіз предметної області;

– опис програми;

– розробка програми;

– експлуатація, тестування та експериментальне дослідження програми;

– оформлення пояснювальної записки та документів до неї.

А.5 Порядок контролю та приймання

Дипломна кваліфікаційна робота бакалавра повинна бути виконана у відповідності з календарним планом.

Дипломна кваліфікаційна робота бакалавра має бути прийнята консультантами відповідних розділів та керівником роботи загалом, після

чого затверджена завідуючим кафедрою та захищена на засіданні екзаменаційної комісії.

ДОДАТОК Б

Текст програми

Б.1 Зміст файлу AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hordiienko.app.powerrate"
    >

    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
    <uses-feature android:glEsVersion="0x00020000"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        >
        <activity
            android:name="com.hordiienko.app.powerrate.MainActivity"
            android:launchMode="singleTop"
            android:windowSoftInputMode="adjustPan"
            >
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <service
            android:name="com.hordiienko.app.powerrate.service.MspService"
            android:enabled="true"
            android:exported="false"
            />
```

```
    </application>

</manifest>
```

Б.2 Зміст файлу MainActivity.java

```
package com.hordiienko.app.powerrate;

import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Build;
import android.os.Bundle;
import android.os.IBinder;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.View;
import android.widget.Toast;

import com.hordiienko.app.powerrate.fragments.ConnectFragment;
import com.hordiienko.app.powerrate.fragments.LogFragment;
import com.hordiienko.app.powerrate.fragments.StatusFragment;
import com.hordiienko.app.powerrate.fragments.tabs.CliFragment;
import com.hordiienko.app.powerrate.fragments.tabs.GpsFragment;
import com.hordiienko.app.powerrate.msp.data.FeatureConfig;
import com.hordiienko.app.powerrate.service.ConnectListener;
import com.hordiienko.app.powerrate.service.MspService;
import com.hordiienko.app.powerrate.test.TestFragment;
import com.hordiienko.app.powerrate.utils.AboutDialog;
import com.hordiienko.app.powerrate.msp.MspTab;
import com.hordiienko.app.powerrate.msp.MspTabList;
```

```
import com.mikepenz.materialdrawer.Drawer;
import com.mikepenz.materialdrawer.DrawerBuilder;
import com.mikepenz.materialdrawer.model.DividerDrawerItem;
import com.mikepenz.materialdrawer.model.PrimaryDrawerItem;
import com.mikepenz.materialdrawer.model.SecondaryDrawerItem;
import com.mikepenz.materialdrawer.model.interfaces.IDrawerItem;

public class MainActivity extends AppCompatActivity implements ConnectListener {
    private static final int DRAWER_STATE_DEFAULT = 0;
    private static final int DRAWER_STATE_CONNECTED = 1;

    private static final String DRAWER_CURRENT_ID = "DRAWER_CURRENT_ID";
    private static final String DRAWER_STATE = "DRAWER_STATE";

    // com.hordiienko.app.powerrate.views from layout
    private Toolbar toolbar;

    // logger
    private Logger logger;

    // drawer
    private Drawer drawer;
//    private DrawerTabManager drawerTabManager;
    private int drawerState;
    private int drawerCurrentId;
    private int aboutId;

    // service
    private MspService mspService;
    private boolean mspServiceBound;

//    private Fragment currentTabFragment;
    private StatusFragment statusFragment;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

   findViews();

   initToolbar();
   initDrawer();
   initFragments(savedInstanceState);
}

@Override
protected void onStart() {
    super.onStart();

    Log.d("LOG", "MainActivity.onStart()");

    // but always false
    if (!mspServiceBound) {
        Intent intent = new Intent(this, MspService.class);

        if (Build.VERSION.SDK_INT >= 26) {
            startForegroundService(intent);
        } else {
            startService(intent);
        }

        bindService(intent, serviceConnection, 0);
    }
}

@Override
protected void onStop() {
```

```
    super.onStop();

    Log.d("LOG", "MainActivity.onStop()");

    // but should be bound always
    if (mspServiceBound) {
        mspService.setConnectListener(null);

        // stop service if not connected to device
        if (!mspService.isConnected() && !mspService.isConnecting()) {
            stopService(new Intent(this, MspService.class));
        }

        unbindService(serviceConnection);
        mspServiceBound = false;
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();

    Log.d("LOG", "MainActivity.onDestroy()");
}

@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);

    // save drawer data to bundle
    outState.putInt(DRAWER_STATE, drawerState);
    outState.putInt(DRAWER_CURRENT_ID, drawerCurrentId);
}
```

```
    private void findViews() {
        toolbar = findViewById(R.id.toolbar);
    }

    private void initToolbar() {
        setSupportActionBar(toolbar);
    }

    private void initDrawer() {
        drawer = new DrawerBuilder()
                .withActivity(this)
                .withToolbar(toolbar)
                .withHeader(R.layout.drawer_header)
                .withOnDrawerItemClickListener(new DrawerItemClickListener())
                .build();

//        drawerTabManager = new DrawerTabManager();
    }

    private void initFragments(Bundle savedInstanceState) {
        FragmentManager fragmentManager = getSupportFragmentManager();

        if (savedInstanceState == null) {
            drawerCurrentId = 0;

            logger = new LogFragment();

            fragmentManager.beginTransaction()
                    .add(R.id.containerLog, (Fragment) logger, LogFragment.class.getSimpleName())
                    .add(R.id.containerTab, new ConnectFragment(),
ConnectFragment.class.getSimpleName())
//                    .add(R.id.containerTab, new TestFragment(),
ConnectFragment.class.getSimpleName())
                    .commit();
```

```
        } else {
            // restore drawer data from bundle
            drawerState = savedInstanceState.getInt(DRAWER_STATE);
            drawerCurrentId = savedInstanceState.getInt(DRAWER_CURRENT_ID);

//            currentTabFragment = fragmentManager.findFragmentById(R.id.containerTab);
            logger = (Logger)
fragmentManager.findFragmentByTag(LogFragment.class.getSimpleName());
        }
    }

    @Override
    public void onBackPressed() {
        // close drawer first
        if (drawer != null && drawer.isDrawerOpen()) {
            drawer.closeDrawer();
        } else {
            if (mspServiceBound) {
                if (mspService.isConnected()) {
                    Toast.makeText(this, getString(R.string.need_to_disconnect),
Toast.LENGTH_SHORT).show();
                    return;
                }
            }

            super.onBackPressed();
        }
    }

    @Override
    public void onConnect() {
        Log.d("LOG", "MainActivity.onDeviceConnect()");

        // TODO select current tab in drawer by current class from service
```

```
        // replace drawer items to pages
        initConnectedDrawer();

        // insert status fragment
        insertStatusFragment();

        // reset current fragment
//        currentTabFragment = null;

        // select first page from pages
        selectPage(0);
    }

    @Override
    public void onDisconnect(boolean reconnect) {
        Log.d("LOG", "MainActivity.onDisconnect()");

//        if (reconnect) {
//            logger.log("Reconnecting...");
//        }

        // reset drawer items to default
        drawerCurrentId = 1;
        initDefaultDrawer();

        FragmentManager fragmentManager = getSupportFragmentManager();

        ConnectFragment connectFragment = new ConnectFragment();
//        Bundle args = new Bundle();
//        args.putBoolean(ConnectFragment.RECONNECT_KEY, reconnect);
//        connectFragment.setArguments(args);

        fragmentManager.beginTransaction()
```

```
            .replace(R.id.containerTab, connectFragment,
ConnectFragment.class.getSimpleName())
            .commit();

    removeStatusFragment();

    // update app title
    setTitle(R.string.app_name);

    logger.log(getString(R.string.disconnected));
  }

  private void insertStatusFragment() {
    // insert status fragment
    statusFragment = new StatusFragment();

    FragmentManager fragmentManager = getSupportFragmentManager();
    fragmentManager.beginTransaction()
        .replace(R.id.containerStatus, statusFragment, StatusFragment.class.getSimpleName())
        .commit();
  }

  private void removeStatusFragment() {
    // find status fragment
    FragmentManager fragmentManager = getSupportFragmentManager();

    if (statusFragment != null) {
      fragmentManager.beginTransaction()
          .remove(statusFragment)
          .commit();
    }
  }

  public Logger getLogger() {
```

```
        return logger;
    }

//    public DrawerTabManager getDrawerTabManager() {
//        return drawerTabManager;
//    }

    private void onServiceBound() {
        mspService.setConnectListener(this);

        // init the desired drawer items
        if (mspService.isConnected()) {
            Fragment currentFragment = getCurrentTabFragment();
            MspTab currentTab = mspService.getCurrentTab();

            if (currentTab.getClazz() != currentFragment.getClass()) {
                // app was restarted fully
                try {
                    selectPage(mspService.getMspTabList().indexOf(currentTab));
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }

            drawerState = DRAWER_STATE_CONNECTED;
            initConnectedDrawer();
        } else {
            drawerState = DRAWER_STATE_DEFAULT;
            initDefaultDrawer();
        }
    }

    private void initDefaultDrawer() {
        // remove all items in drawer
```

```
        drawer.removeAllItems();

        // add default Connect page to drawer
        drawer.addItem(new PrimaryDrawerItem()
                .withName(R.string.drawer_connect)
                .withIdentifier(1)
                .withIcon(R.drawable.ic_usb_black_24dp)
                .withIconTintingEnabled(true));

        // add items to footer in drawer
        initFooterDrawer();

        // select current page in drawer
        drawer.setSelection(drawerCurrentId, false);
    }

    private void initConnectedDrawer() {
//        DrawerTab tab = drawerTabManager.getTab(drawerCurrentId);
        MspTabList mspTabList = mspService.getMspTabList();
        MspTab tab = mspTabList.get(drawerCurrentId);
        // update app title
        setTitle(tab.getName());

        // remove all items in drawer
        drawer.removeAllItems();
        // add pages items to drawer
//        drawerTabManager.addToDrawer(drawer, mspService.getMsp().getMspData());

        for (int i = 0; i < mspTabList.size(); i++) {
            addToDrawer(mspTabList.get(i), i);
        }

        // add items to footer in drawer
//        initFooterDrawer();
```

```
        // select current page in drawer
        drawer.setSelection(drawerCurrentId, false);
    }

    private void initFooterDrawer() {
//        settingsId = 100;

        aboutId = 100;

        drawer.addItems(
                new DividerDrawerItem(),
                new SecondaryDrawerItem().withName(R.string.footer_about)
                        .withIdentifier(aboutId)
                        .withIcon(R.drawable.ic_info_black_24dp)
                        .withIconTintingEnabled(true)
                        .withSelectable(false)
//                new
SecondaryDrawerItem().withName(R.string.footer_about).withIdentifier(aboutId).withIcon(Goo
gleMaterial.Icon.gmd_info).withSelectable(false)
        );
    }

    private void addToDrawer(MspTab tab, int key) {
        if (tab.getClazz() == GpsFragment.class) {
            if (!
mspService.getMsp().getMspData().featureConfig.getFeaturesByGroup(FeatureConfig.Group.G
PS).get(0).isEnabled()) {
                return;
            }
        }

        drawer.addItem(new PrimaryDrawerItem()
                .withName(tab.getName())
```

```
            .withIdentifier(key)
            .withIcon(tab.getIcon())
            .withIconTintingEnabled(true));
    }

//    @Nullable
//    public Class<?> getCurrentTabFragment() {
//        return currentTabFragment == null ? null : currentTabFragment.getClass();
//    }

    private void selectPage(long id) {
        MspTab currentTab = mspService.getCurrentTab();

        if (currentTab != null && currentTab.getClazz() == CliFragment.class) {
            Fragment fragment = getCurrentTabFragment();

            if (fragment instanceof CliFragment) {
                ((CliFragment) fragment).sendReboot();

                return;
            }

//            return;
        }

//        DrawerTab tab = drawerTabManager.getTab((int) id);
        MspTab tab = mspService.getMspTabList().get((int) id);

        try {
            Fragment newFragment = (Fragment) tab.getClazz().newInstance();

            FragmentManager fragmentManager = getSupportFragmentManager();
            fragmentManager.beginTransaction()
                    .replace(R.id.containerTab, newFragment, tab.getTag())
```

```
                .commit();

        mspService.setCurrentTab(mspService.getMspTabList().get(newFragment.getClass()));
//          currentTabFragment = newFragment;

        // store current page id
        drawerCurrentId = (int) id;

        // select current page in drawer
        drawer.setSelection(drawerCurrentId, false);

        // update app title
        setTitle(tab.getName());

        if (tab.getClazz() == CliFragment.class) {
            if (statusFragment != null) {
                statusFragment.stopLiveData();
            }
            mspService.getMsp().getMspData().configurator.cliActive = true;
        } else {
            mspService.getMsp().getMspData().configurator.cliActive = false;
        }
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (InstantiationException e) {
        e.printStackTrace();
    }
}

private Fragment getCurrentTabFragment() {
    return getSupportFragmentManager().findFragmentById(R.id.containerTab);
}

private class DrawerItemClickListener implements Drawer.OnDrawerItemClickListener {
```

```
        @Override
        public boolean onItemClick(View view, int position, IDrawerItem drawerItem) {
            Log.d("LOG", "onItemClick: " + position + ", drawerItem: " +
drawerItem.getIdentifier());

            if (drawerItem.getIdentifier() == aboutId) {
                AboutDialog.show(MainActivity.this);
//                drawer.closeDrawer();
            } else {
                // check if page exist for this id and this page not show currently
                if (drawerItem.getIdentifier() != drawerCurrentId) {
                    selectPage(drawerItem.getIdentifier());
                }
            }

//            // check if page exist for this id and this page not show currently
//            if (pageManager.isTabExist((int) drawerItem.getIdentifier()) &&
drawerItem.getIdentifier() != drawerCurrentId) {
//                selectPage(drawerItem.getIdentifier());
//            }

            // always close drawer
            drawer.closeDrawer();

            return true;
        }
    }

    private ServiceConnection serviceConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            Log.d("LOG", "MainActivity().onServiceConnected()");

            MspService.LocalBinder binder = (MspService.LocalBinder) service;
```

```
            mspService = binder.getService();
            mspServiceBound = true;

            onServiceBound();
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            Log.d("LOG", "MainActivity().onServiceDisconnected()");

            mspServiceBound = false;
            mspService = null;
        }
    };
}
```

Б.3 Зміст файлу MspService.java

```
package com.hordiienko.app.powerrate.service;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.hardware.usb.UsbDevice;
import android.hardware.usb.UsbManager;
import android.os.Binder;
import android.os.IBinder;
import android.support.v4.app.NotificationCompat;
import android.support.v4.app.NotificationManagerCompat;
```

```
import android.util.Log;
import android.util.Pair;

import com.hordiienko.app.powerrate.Constants;
import com.hordiienko.app.powerrate.MainActivity;
import com.hordiienko.app.powerrate.R;
import com.hordiienko.app.powerrate.msp.MSP;
import com.hordiienko.app.powerrate.msp.MspTab;
import com.hordiienko.app.powerrate.msp.MspTabList;
import com.hordiienko.app.powerrate.utils.Utils;

import java.util.HashMap;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;

public class MspService extends Service {
    public class LocalBinder extends Binder {
        public MspService getService() {
            return MspService.this;
        }
    }

    private final IBinder mBinder = new LocalBinder();

    private MSP msp;
    private UsbManager usbManager;
    private ConnectListener connectListener;

    private boolean connecting;

    private ScheduledExecutorService scheduledExecutorService;
    private Pair<String, MspTask> currentTask;

    private MspTabList mspTabList;
```

```
    private MspTab currentTab;

    @Override
    public IBinder onBind(Intent intent) {
        Log.d("LOG", "MspService.onBind()");

        return mBinder;
    }

    private Notification buildNotification() {
        Intent intent = new Intent(this, MainActivity.class);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent,
PendingIntent.FLAG_UPDATE_CURRENT);

        // TODO add disconnect action
        NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder(this,
Constants.NOTIFICATION_CHANNEL_ID)
                .setSmallIcon(R.drawable.ic_usb_black_24dp)
                .setContentTitle(getString(isConnected() ? R.string.notification_connected :
R.string.notification_running))
                .setContentIntent(pendingIntent)
                .setPriority(NotificationCompat.PRIORITY_DEFAULT);

        return notificationBuilder.build();
    }


    @Override
    public void onCreate() {
        super.onCreate();

        Log.d("LOG", "MspService.onCreate()");

        Utils.createNotificationChannel(this);
```

```
        startForeground(1, buildNotification());

        scheduledExecutorService = Executors.newSingleThreadScheduledExecutor();
        usbManager = (UsbManager) getSystemService(Context.USB_SERVICE);

        mspTabList = new MspTabList();
    }

//    @Override
//    public int onStartCommand(Intent intent, int flags, int startId) {
//        Log.d("LOG", "MspService.onStartCommand()");
//
//        // don't recreate
//        return Service.START_NOT_STICKY;
//    }

    @Override
    public void onDestroy() {
        super.onDestroy();

        Log.d("LOG", "MspService.onDestroy()");

        disconnect(false, true);
    }

    public MSP getMsp() {
        return msp;
    }

    public boolean isConnected() {
        return msp != null;
    }
```

```
    public void setConnecting(boolean connecting) {
        this.connecting = connecting;
    }

    public boolean isConnecting() {
        return connecting;
    }

    public void setConnected(MSP msp) {
        this.msp = msp;

        connecting = false;

        if (this.msp != null) {
            connectListener.onConnect();
            registerDeviceDetached();
        }

        showNotification();
    }

    private void showNotification() {
        NotificationManager notificationManager = (NotificationManager)
getSystemService(Service.NOTIFICATION_SERVICE);
        notificationManager.notify(1, buildNotification());
    }

    public void disconnect() {
        boolean reboot = false;

        if (msp != null && msp.isRebooting()) {
            reboot = true;
        }
```

```
    disconnect(reboot, false);
}

private void disconnect(boolean reconnect, boolean shutDown) {
    Log.d("LOG", "MspService.disconnect()");

    if (msp != null) {
        for (MspTab tab : mspTabList) {
            tab.setMspDataLoaded(false);
        }

        msp.getSerial().close();

        msp = null;

        unregisterDeviceDetached();

        if (connectListener != null) {
            connectListener.onDisconnect(reconnect);
        }
    }

    if (!shutDown) {
        showNotification();
    }
}

public HashMap<String, UsbDevice> getDevices() {
    return usbManager.getDeviceList();
}

public void executeTask(MspTask task) {
    scheduledExecutorService.execute(task);
}
```

```
public void storeTask(MspTask task, String taskName) {
    currentTask = new Pair<>(taskName, task);
}

public void removeTask() {
    currentTask = null;
}

public Pair<String, MspTask> getCurrentTask() {
    Pair<String, MspTask> task = currentTask;

    currentTask = null;

    return task;
}

public MspTab getCurrentTab() {
    return currentTab;
}

public void setCurrentTab(MspTab tab) {
    this.currentTab = tab;
}

public MspTabList getMspTabList() {
    return mspTabList;
}

public ScheduledExecutorService getScheduledExecutorService() {
    return scheduledExecutorService;
}

public void setConnectListener(ConnectListener connectListener) {
```

```
        this.connectListener = connectListener;
    }

    private void registerDeviceDetached() {
        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction("android.hardware.usb.action.USB_DEVICE_DETACHED");

        registerReceiver(deviceDetachedReceiver, intentFilter);
    }

    private void unregisterDeviceDetached() {
        unregisterReceiver(deviceDetachedReceiver);
    }

    private BroadcastReceiver deviceDetachedReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            UsbDevice device = intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);

            if (msp.getSerial().getUsbDevice().equals(device)) {
                disconnect();
            }
        }
    };
}
```

Б.4 Зміст файлу Serial.java

```
package com.hordiienko.app.powerrate.service;

import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
```

```
import android.content.IntentFilter;
import android.hardware.usb.UsbDevice;
import android.hardware.usb.UsbDeviceConnection;
import android.hardware.usb.UsbManager;

import com.felhr.usbserial.UsbSerialDevice;
import com.felhr.usbserial.UsbSerialInterface;

public class Serial {
   private static final String ACTION_USB_PERMISSION =
"com.android.example.USB_PERMISSION";

   public interface ReadListener {
      void onRead(byte[] data);
   }

   private ReadListener readListener;
   private UsbGrantReceiver usbGrantReceiver;
   private UsbManager usbManager;
   private UsbDevice usbDevice;
   private UsbDeviceConnection usbDeviceConnection;
   private UsbSerialDevice serialDevice;
   private int baudRate;
   private boolean deviceOpened;
   private boolean serialOpened;
   private boolean errorConnect;

   private Serial() {
   }

   public static Serial connect(Context context, int baudRate, UsbDevice usbDevice) {
      Serial instance = new Serial();
      instance.usbManager = (UsbManager) context.getSystemService(Context.USB_SERVICE);
      instance.baudRate = baudRate;
```

```
        instance.usbDevice = usbDevice;

        if (instance.connect(context)) {
            return instance;
        } else {
            return null;
        }
    }

    public void setReadListener(ReadListener readListener) {
        this.readListener = readListener;
    }

    public void write(byte[] data) {
        serialDevice.write(data);
    }

    public UsbDevice getUsbDevice() {
        return usbDevice;
    }

    public void close() {
        if (serialOpened) {
            serialDevice.close();
            usbDeviceConnection.close();
        }
    }

    private boolean connect(Context context) {
        if (!usbManager.hasPermission(usbDevice)) {
            // register broadcast receiver for grant permission
            IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
            usbGrantReceiver = new UsbGrantReceiver();
            context.registerReceiver(usbGrantReceiver, filter);
```

```
            PendingIntent pendingIntent = PendingIntent.getBroadcast(context, 0, new
Intent(ACTION_USB_PERMISSION), 0);
            usbManager.requestPermission(usbDevice, pendingIntent);
        } else {
            openDevice(context);
        }

        if (!serialOpened && !errorConnect) {
            synchronized (this) {
                try {
                    this.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }

        return !errorConnect;
    }

    private void openDevice(Context context) {
        if (usbGrantReceiver != null) {
            context.unregisterReceiver(usbGrantReceiver);
        }

        usbDeviceConnection = usbManager.openDevice(usbDevice);

        if (usbDeviceConnection == null) {
            errorOpenDevice(context);
            return;
        }

        deviceOpened = true;
```

```
        serialDevice = UsbSerialDevice.createUsbSerialDevice(usbDevice, usbDeviceConnection);

        if (serialDevice != null) {
            if (serialDevice.open()) {
                serialDevice.setBaudRate(baudRate);
                serialDevice.setDataBits(UsbSerialInterface.DATA_BITS_8);
                serialDevice.setStopBits(UsbSerialInterface.STOP_BITS_1);
                serialDevice.setParity(UsbSerialInterface.PARITY_NONE);
//                serialDevice.setParity(UsbSerialInterface.PARITY_EVEN);

                serialDevice.setFlowControl(UsbSerialInterface.FLOW_CONTROL_OFF);

                serialDevice.read(mUsbReadCallback);

                serialOpened = true;
//                mConnectListener.onConnect();

                // notify wait object
                synchronized (this) {
                    this.notifyAll();
                }
            } else {
                errorOpenDevice(context);
            }
        } else {
            // TODO check device support

            errorOpenDevice(context);
        }
    }

    private void errorOpenDevice(Context context) {
        if (usbGrantReceiver != null) {
```

```
            context.unregisterReceiver(usbGrantReceiver);
        }

        if (deviceOpened) {
            usbDeviceConnection.close();
        }

//        mConnectListener.onErrorConnect(ErrorType.ERROR_OPEN);

        // set error
        errorConnect = true;

        // notify wait object
        synchronized (this) {
            this.notifyAll();
        }
    }

    private UsbSerialInterface.UsbReadCallback mUsbReadCallback = new
UsbSerialInterface.UsbReadCallback() {
        @Override
        public void onReceivedData(byte[] data) {
            if (readListener != null) {
                readListener.onRead(data);
            }
        }
    };

    private class UsbGrantReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();

            if (action != null && action.equals(ACTION_USB_PERMISSION)) {
```

```
            if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false))
{
                openDevice(context);
            } else {
                errorOpenDevice(context);
            }
        }
    }
}
```

Б.5 Зміст файлу MSP.java

```
package com.hordiienko.app.powerrate.msp;

import com.hordiienko.app.powerrate.service.Serial;

public class MSP implements Serial.ReadListener {
    private static final int JUMBO_FRAME_SIZE_LIMIT = 255;

    private final Object monitor = this;

    private Serial serial;
    private MSPData mspData;
    private MSPHandler mspHandler;
    private MSPHelper mspHelper;

    // receive message
    private boolean received;
    private int state;
    private int code;
    private byte[] message_buffer;
    private int message_length_expected;
    private int message_length_received;
```

```
private byte message_checksum;
private boolean messageIsJumboFrame;
private boolean unsupported;

private boolean rebooting;

public MSP(Serial serial) {
   this.serial = serial;
   this.serial.setReadListener(this);

   mspData = new MSPData();
   mspHandler = new MSPHandler(mspData);
   mspHelper = new MSPHelper(mspData);
}

public Serial getSerial() {
   return serial;
}

public MSPData getMspData() {
   return mspData;
}

public MSPHelper getMspHelper() {
   return mspHelper;
}

@Override
public void onRead(byte[] data) {
   for (byte aData : data) {
      switch (state) {
         case 0:
            if (aData == 36) { // $
               state++;
```

```
    }
    break;
case 1:
    if (aData == 77) { // M
        state++;
    } else { // restart and try again
        state = 0;
    }
    break;
case 2: // direction (should be >)
    unsupported = aData == 33; // FC reports unsupported message error
    state++;
    break;
case 3:
    message_length_expected = aData & 0xFF;
    if (message_length_expected == JUMBO_FRAME_SIZE_LIMIT) {
        messageIsJumboFrame = true;
    }

    message_checksum = aData;

    state++;
    break;
case 4:
    code = aData & 0xFF;
    message_checksum ^= aData;

    if (message_length_expected > 0) {
        // process payload
        if (messageIsJumboFrame) {
            state++;
        } else {
            state = state + 3;
        }
```

```
    } else {
        // no payload
        state += 5;
    }
    break;
case 5:
    message_length_expected = aData & 0xFF;
    message_checksum ^= aData;
    state++;

    break;
case 6:
    message_length_expected = message_length_expected + 256 * (aData & 0xFF);
    message_checksum ^= aData;
    state++;

    break;
case 7:
    // setup buffer
    message_buffer = new byte[message_length_expected];

    state++;
case 8: // payload
    message_buffer[message_length_received] = aData;
    message_checksum ^= aData;
    message_length_received++;

    if (message_length_received >= message_length_expected) {
        state++;
    }
    break;
case 9:
    if (message_checksum == aData) {
        MSPPacket mspPacket = new MSPPacket(code, unsupported, message_buffer);
```

```
                    if (mspHandler != null) {
                        mspHandler.handleMessage(mspPacket);
                    }
                    onReceive();
                }
//                  else {
                    // TODO report crcError
//                  }

                // Reset variables
                message_length_received = 0;
                state = 0;
                messageIsJumboFrame = false;
                message_buffer = null;
                break;
            }
        }
    }

    public void sendMessage(int code, byte[] data) {
        byte[] buffer;

        if (data != null) {
            byte checksum;
            buffer = new byte[data.length + 6];

            buffer[0] = 36; // $
            buffer[1] = 77; // M
            buffer[2] = 60; // <
            buffer[3] = (byte) data.length; // data length
            buffer[4] = (byte) code; // code

            checksum = (byte) (buffer[3] ^ buffer[4]);
```

```
            for (int i = 0; i < data.length; i++) {
                buffer[i + 5] = data[i];

                checksum ^= buffer[i + 5];
            }

            buffer[5 + data.length] = checksum;
        } else {
            buffer = new byte[6];

            buffer[0] = 36; // $
            buffer[1] = 77; // M
            buffer[2] = 60; // <
            buffer[3] = 0; // data length
            buffer[4] = (byte) code; // code
            buffer[5] = (byte) (buffer[3] ^ buffer[4]); // checksum
        }

        serial.write(buffer);
    }

    public void sendMessage(int code) {
        sendMessage(code, null);
    }

    public void sendRequest(int code) throws MspException {
        boolean receiveTimeout = false;

        received = false;

        sendMessage(code, null);

        synchronized (monitor) {
            try {
```

```
                // TODO check InterruptedException
                monitor.wait(2000);
            } catch (InterruptedException ignored) {}

            if (!received) {
                receiveTimeout = true;
            }
        }

        received = false;

        if (receiveTimeout) {
            throw new MspException(MspException.ExceptionType.MSP_RECEIVE_TIMEOUT);
        }
    }

    public void reboot() {
        rebooting = true;

        sendMessage(MSPCodes.MSP_SET_REBOOT);
    }

    public boolean isRebooting() {
        return rebooting;
    }

    public void setArmingEnabled(boolean doEnable, boolean
disableRunawayTakeoffPrevention) {
        // apiVersion 1.37.0

        if (mspData.config.armingDisabled == doEnable ||
mspData.config.runawayTakeoffPreventionDisabled != disableRunawayTakeoffPrevention) {
            mspData.config.armingDisabled = !doEnable;
```

```
            mspData.config.runawayTakeoffPreventionDisabled =
disableRunawayTakeoffPrevention;

            sendMessage(MSPCodes.MSP_ARMING_DISABLE,
mspHelper.crunch(MSPCodes.MSP_ARMING_DISABLE));

            // TODO
//            if (doEnable) {
//                GUI.log(i18n.getMessage('armingEnabled'));
//                if (disableRunawayTakeoffPrevention) {
//                    GUI.log(i18n.getMessage('runawayTakeoffPreventionDisabled'));
//                } else {
//                    GUI.log(i18n.getMessage('runawayTakeoffPreventionEnabled'));
//                }
//            } else {
//                GUI.log(i18n.getMessage('armingDisabled'));
//            }
        }
    }

    private void onReceive() {
        received = true;
        freeMonitor();
    }

    private void freeMonitor() {
        synchronized (monitor) {
            monitor.notifyAll();
        }
    }
}
```

Б.6 Зміст файлу MspServiceFragment.java

```
package com.hordiienko.app.powerrate.fragments;

import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.util.Log;

import com.hordiienko.app.powerrate.MainActivity;
import com.hordiienko.app.powerrate.service.MspService;

public abstract class MspServiceFragment extends Fragment {
    abstract protected void onServiceBound();

    private MainActivity mainActivity;
    private MspService mspService;
    boolean mspServiceBound;

    @Override
    public void onActivityCreated(@Nullable Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        mainActivity = (MainActivity) getActivity();
    }

    @Override
    public void onStart() {
        super.onStart();

        if (!mspServiceBound) {
```

```
        Intent intent = new Intent(mainActivity, MspService.class);
        mainActivity.bindService(intent, serviceConnection, 0);
    }
}

@Override
public void onStop() {
    super.onStop();

    if (mspServiceBound) {
        mainActivity.unbindService(serviceConnection);
        mspServiceBound = false;
    }
}

protected MainActivity getMainActivity() {
    return mainActivity;
}

protected MspService getMspService() {
    return mspService;
}

private ServiceConnection serviceConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        MspService.LocalBinder binder = (MspService.LocalBinder) service;
        mspService = binder.getService();
        mspServiceBound = true;

        Log.d("LOG", "MspServiceFragment().onServiceConnected()");

        onServiceBound();
    }
```

```
        @Override
        public void onServiceDisconnected(ComponentName name) {
            mspServiceBound = false;
            mspService = null;
        }
    };
}
```

Б.7 Зміст файлу MspTaskFragment.java

```
package com.hordiienko.app.powerrate.fragments;

import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;
import android.util.Pair;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ProgressBar;

import com.hordiienko.app.powerrate.Logger;
import com.hordiienko.app.powerrate.R;
import com.hordiienko.app.powerrate.service.MspTask;

public abstract class MspTaskFragment extends MspServiceFragment {
    private ViewGroup containerTab;
    private ProgressBar progressTask;

    private MspTask mspTask;

    private boolean tabCreated;
```

```
    protected abstract View onCreateTab(LayoutInflater inflater, @Nullable ViewGroup
container);
    protected abstract void onTabCreated();

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup
container, @Nullable Bundle savedInstanceState) {
        Log.d("LOG", "MspTaskFragment.onCreateView()");

        View layout = inflater.inflate(R.layout.fragment_msp_task, container, false);

        containerTab = layout.findViewById(R.id.containerTab);
        progressTask = layout.findViewById(R.id.progressTask);

        showProgressBar();

//        containerTab.postDelayed(() -> {
//            View view = onCreateTab(inflater, containerTab);
//
//            if (view != null) {
//                containerTab.addView(view);
//            }
//
////            hideProgressBar();
//
//            tabCreated = true;
//            onTabCreated();
//        }, 200);

        return layout;
    }
```

```
@Override
public void onStop() {
    super.onStop();

    if (mspTask != null && !mspTask.isFinished()) {
        mspTask.unlink();
        getMspService().storeTask(mspTask, getClass().getName());
    }
}

public void taskFinished(final MspTask task) {
    runInUiThread(() -> onTaskFinished(task));
}

protected void onTaskFinished(MspTask task) {
    mspTask = null;
    hideProgressBar();
}

protected void executeTask(MspTask task) {
    mspTask = task;

    showProgressBar();

    getMspService().executeTask(task);
}

protected Logger getLogger() {
    return getMainActivity().getLogger();
}

protected MspTask getTask() {
    return mspTask;
}
```

```
    private void runInUiThread(Runnable runnable) {
        containerTab.post(runnable);
    }

    protected void showProgressBar() {
//        Log.d("LOG", "showProgressBar");

        containerTab.setVisibility(View.GONE);
        progressTask.setVisibility(View.VISIBLE);
    }

    protected void hideProgressBar() {
//        Log.d("LOG", "hideProgressBar");

        containerTab.setVisibility(View.VISIBLE);
        progressTask.setVisibility(View.GONE);
    }

    protected boolean isTabCreated() {
        return tabCreated;
    }

    @Override
    protected void onServiceBound() {
        // TODO delay for close drawer
        createTab();

        Pair<String, MspTask> taskPair = getMspService().getCurrentTask();

        if (taskPair != null && taskPair.first.equals(getClass().getName())) {
            mspTask = taskPair.second;

            showProgressBar();
```

```
            mspTask.link(this);
        }
    }

    private void createTab() {
        containerTab.postDelayed(new Runnable() {
            @Override
            public void run() {
//                LayoutInflater inflater = getLayoutInflater();
//                LayoutInflater.from(getContext());
                LayoutInflater inflater = getMainActivity().getLayoutInflater();
                View view = onCreateTab(inflater, containerTab);

                if (view != null) {
                    containerTab.addView(view);
                }

//            hideProgressBar();

                tabCreated = true;
                onTabCreated();
            }
        }, 200);
    }
}
```

Б.8 Зміст файлу BaseTabFragment.java

```
package com.hordiienko.app.powerrate.fragments;

import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.support.v4.content.ContextCompat;
```

```
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.LinearLayout;
import android.widget.Toast;

import com.hordiienko.app.powerrate.Logger;
import com.hordiienko.app.powerrate.R;
import com.hordiienko.app.powerrate.service.MspTask;
import com.hordiienko.app.powerrate.msp.MspTab;
import com.hordiienko.app.powerrate.msp.MspTabList;

import static android.content.Context.INPUT_METHOD_SERVICE;

public abstract class BaseTabFragment extends MspTaskFragment {
    private static final String DATA_LOADED_KEY = "DATA_LOADED_KEY";

    abstract public boolean onLoadPerform();
    abstract public boolean onSavePerform();
    public abstract void onUpdateTab();
    public abstract void onSaveTab();

//    private boolean serviceBound;
    private boolean dataLoaded;

    private boolean disableSave;
    private boolean disableLoad;

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
    setHasOptionsMenu(true);

    if (savedInstanceState != null) {
        dataLoaded = savedInstanceState.getBoolean(DATA_LOADED_KEY, false);
    }
}

@Override
public void onSaveInstanceState(@NonNull Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putBoolean(DATA_LOADED_KEY, dataLoaded);
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    menu.clear();
    inflater.inflate(R.menu.main_menu, menu);

    if (disableLoad) {
        menu.removeItem(R.id.action_load);
    }

    if (disableSave) {
        menu.removeItem(R.id.action_save);
    }
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_save:
            hideKeyboard();
            actionSave();
```

```
                break;
            case R.id.action_load:
                hideKeyboard();
                actionLoad();
                break;
            case R.id.action_reboot:
                actionReboot();
                break;
            case R.id.action_disconnect:
                hideKeyboard();
                actionDisconnect();
                break;
            default:
                return super.onOptionsItemSelected(item);
        }

        return true;
    }

    @Override
    protected void onServiceBound() {
        super.onServiceBound();

//        serviceBound = true;

//        if (isTabCreated()) {
//            onTabCreated();
//        }

//        if (!dataLoaded) {
//            loadData();
//        } else if (isUiDrawn()){
//            updateUi();
//        }
```

```
    }

    @Override
    public void onStop() {
        super.onStop();

//        serviceBound = false;

        if (dataLoaded) {
            onSaveTab();
        }
    }

    @Override
    protected void onTabCreated() {
        Log.d("LOG", "BaseTabFragment.onTabCreated()");

        if (!dataLoaded) {
            try {
//                DrawerTabManager drawerTabManager =
getMainActivity().getDrawerTabManager();
                MspTabList mspTabList = getMspService().getMspTabList();
//                int tabId = drawerTabManager.getTabId(getClass());
                MspTab tab = mspTabList.get(getClass());

                dataLoaded = tab.isMspDataLoaded();
            } catch (Exception e) {
                e.printStackTrace();
            }

//              DrawerTab tab = getMainActivity().getDrawerTabManager().getTab(getClass());
        }

        if (!dataLoaded) {
```

```
            loadData();
        } else {
            hideProgressBar();
            onUpdateTab();
        }

    }

    @Override
    protected void onTaskFinished(MspTask task) {
        super.onTaskFinished(task);

        Logger logger = getLogger();

        if (task.isSuccess()) {
            if (task instanceof LoadTask) {
                logger.log(getString(R.string.loadSuccess));
                dataLoaded = true;

                getMspService().getMspTabList().get(getClass()).setMspDataLoaded(true);

                loadCallback();
            } else if (task instanceof SaveTask) {
                logger.log(getString(R.string.saveSuccess));

                saveCallback();
            }
        } else {
            if (task instanceof LoadTask) {
                logger.log(getString(R.string.loadFailed));
                dataLoaded = false;
            } else if (task instanceof SaveTask) {
                logger.log(getString(R.string.saveFailed));
            }
```

```
        }
    }

    protected void disableSaveAction() {
        disableSave = true;
    }

    protected void disableLoadAction() {
        disableLoad = true;
    }

    protected boolean isDataLoaded() {
        return dataLoaded;
    }

    private void loadCallback() {
//        if (isUiDrawn()) {
//            updateUi();
//        }
        //updateUi();

        onUpdateTab();
    }

    private void saveCallback() {
        //updateUi();
    }

    protected void actionSave() {
        if (disableSave) {
            return;
        }

        onSaveTab();
```

```
        executeTask(new SaveTask(this));
    }

    protected void actionLoad() {
        if (disableLoad) {
            return;
        }

        loadData();
    }

    protected void actionReboot() {
        getMspService().getMsp().reboot();
//      getMainService().reconnect();
    }

    private void actionDisconnect() {
        MspTask task = getTask();

        if (task != null) {
            Toast.makeText(getMainActivity(), getString(R.string.need_to_disconnect),
Toast.LENGTH_SHORT).show();

            return;
        }

        getMspService().getMsp().setArmingEnabled(true, false);
        getMspService().disconnect();
    }

    private void loadData() {
        executeTask(new LoadTask(this));
    }
```

```
    private void hideKeyboard() {
        try {
            InputMethodManager imm = (InputMethodManager)
getMainActivity().getSystemService(INPUT_METHOD_SERVICE);

imm.hideSoftInputFromWindow(getMainActivity().getCurrentFocus().getWindowToken(), 0);
        } catch (NullPointerException ignored) {}
    }

    private class LoadTask extends MspTask {

        private LoadTask(MspTaskFragment fragment) {
            super(fragment);
        }

        @Override
        public void run() {
            setSuccess(onLoadPerform());

//            SystemClock.sleep(5000);

            finish();
        }
    }

    private class SaveTask extends MspTask {

        private SaveTask(MspTaskFragment fragment) {
            super(fragment);
        }

        @Override
        public void run() {
            setSuccess(onSavePerform());
```

```
            finish();
        }
    }

    protected View makeDivider() {
        float density = getResources().getDisplayMetrics().density;
        int marginTop = (int) (4 * density);
        int marginBottom = (int) (4 * density);

        View view = new View(getMainActivity());

        LinearLayout.LayoutParams layoutParams = new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT, (int) (1 *
density));
        layoutParams.setMargins(0, marginTop, 0, marginBottom);
        view.setLayoutParams(layoutParams);

        view.setBackgroundColor(ContextCompat.getColor(getMainActivity(),
R.color.md_grey_300));

        return view;
    }
}
```

ДОДАТОК В

Слайди презентації

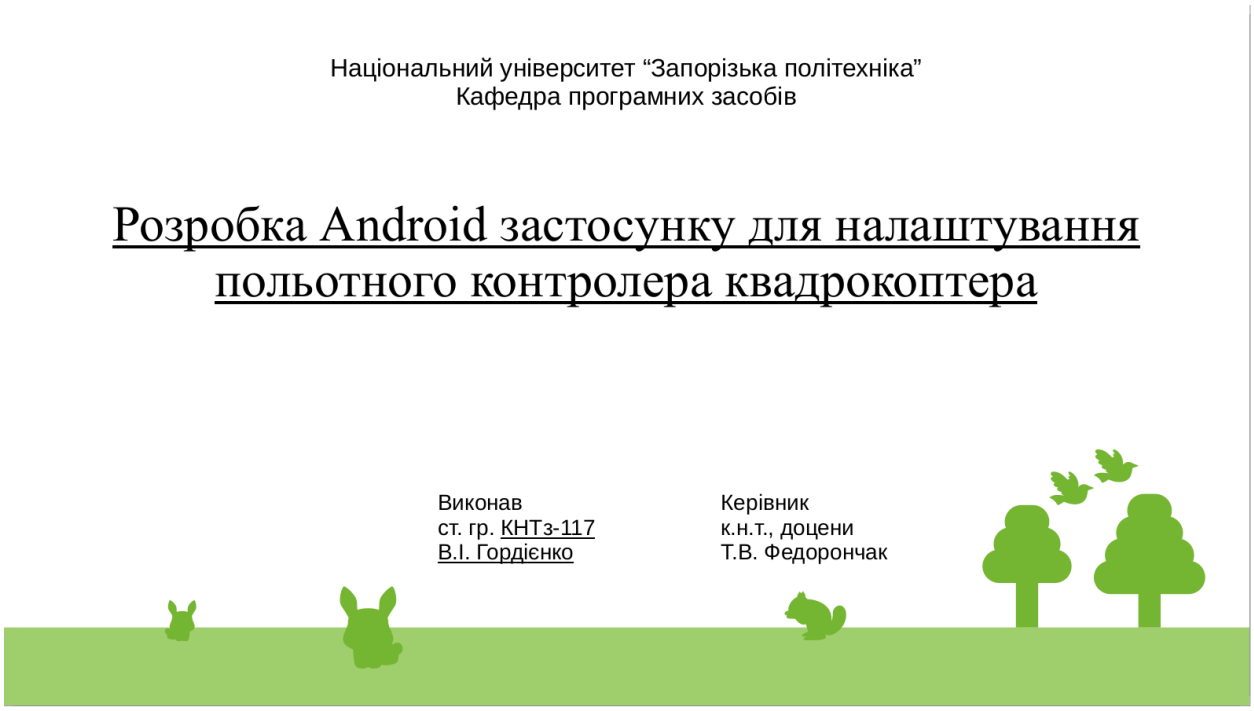


Рисунок В.1 – Слайд №1

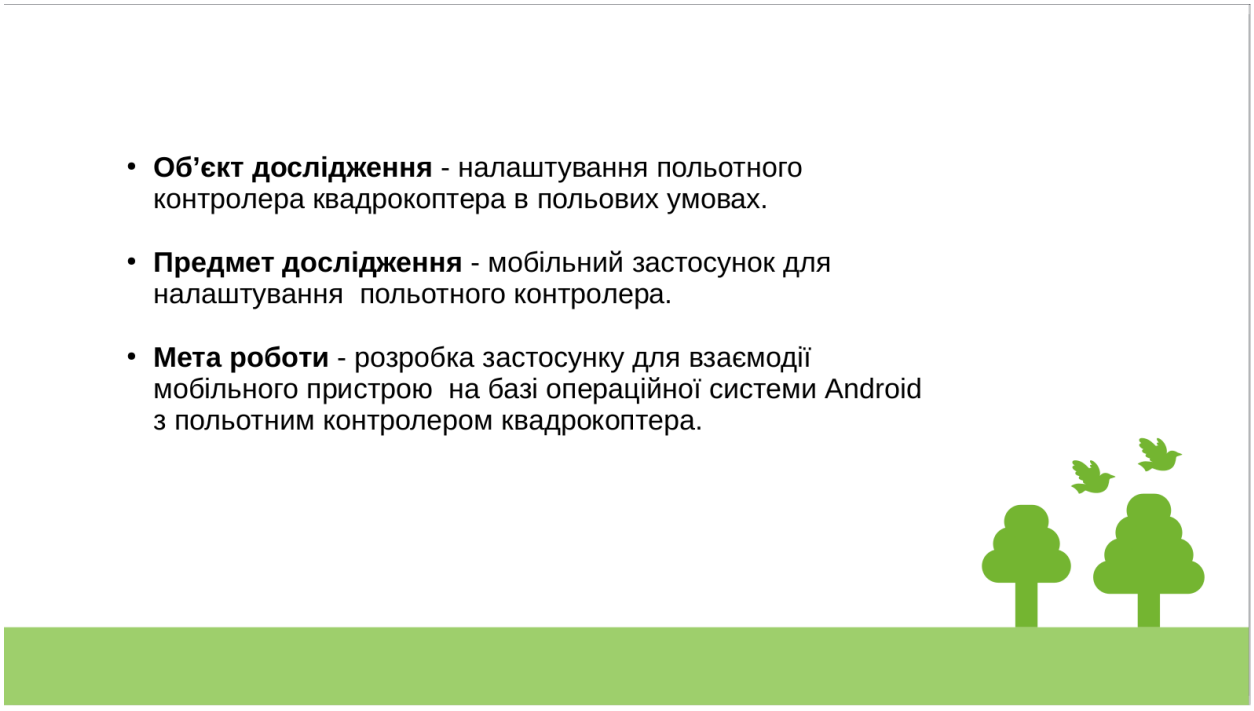


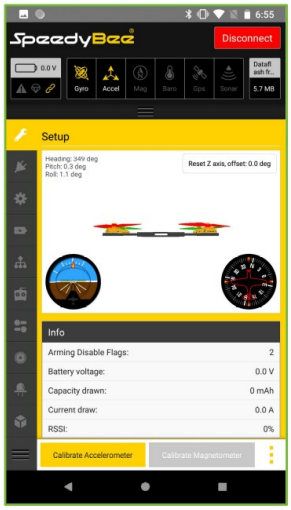
Рисунок В.2 – Слайд №2

Завдання роботи

- Аналіз необхідність мобільного застосунку
- Вибір мови програмування та середовища розробки
- Схема основних компонентів застосунку
- Розробка класу для роботи з протоколом MSP
- Тестування роботи застосунку

Рисунок В.3 – Слайд №3

Аналіз існуючого рішення - застосунок Speedy Bee



Speedy Bee — мобільний застосунок для телефонів з операційною системою Android. Розробник застосунку компанія RunCam Co.,Ltd.

Застосунок дає можливість змінювати налаштування прошивки Betaflight шляхом з'єднання мобільного телефону з контролером за допомогою USB кабелю або Bluetooth (для контролерів, що не мають вбудованого Bluetooth, потрібно докупити спеціальний адаптер).

Рисунок В.4 – Слайд №4

Порівняльні характеристики мов прогрмування

Характеристика	Java	Kotlin
Null-беспечність	Ні	Так
Розумне приведення типів	Ні	Так
Checked exception	так	Ні
Singleton objects	Так	Так (просте створення)
Функціонально програмування	Ні	Так
Функції-розширення	Ні	Так

Рисунок В.5 – Слайд №5

Порівняльні характеристики середовищ розробки

Характеристика	Eclipse	Android Studio
Система побудови	Ant	система побудови Gradle
Підтримка мов Програмування	Java, C/C++, Python, Python та інші	Java та Kotlin
Графічний інтерфейс	Побудований на SWT	Побудований на Swing
Підключення бібліотек	Jar файли	Jar файли, Maven-сумісні репозиторії
Підтримка Cloud Platform	Ні	Так

Рисунок В.6 – Слайд №6

Діаграма основних компонентів програми

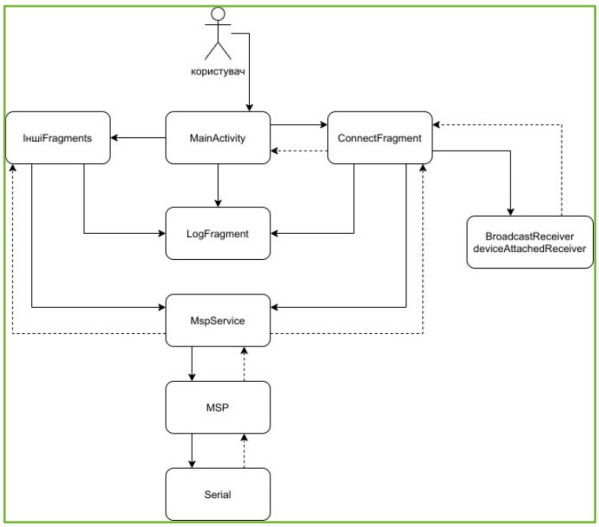


Рисунок В.7 – Слайд №7

Схема пакету протоколу MSP

header - заголовок з трьох байтів, першідва – символи “$M”, третій – напрям руху, “<” - напрям до контролера, “>” - напрям від контролера

size - розмір даних, що містяться в payload

type - тип пакету, вказує тип даних які містить пакет

payload - lані, якщо є (size не 0)

crs - Контрольна сума, операція XOR над байтами size, type та payload

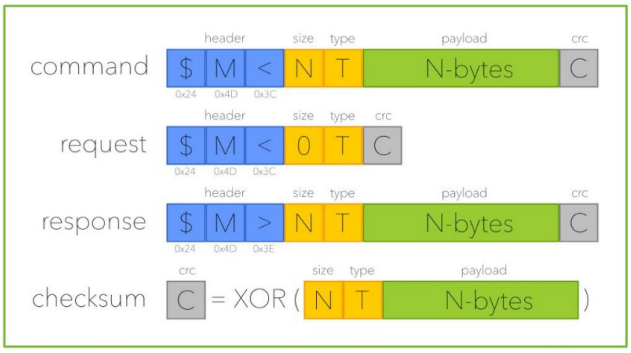


Рисунок В.8 – Слайд №8

Підключення контролеру LUX до мобільного телефону

Для підключення контролера до мобільного телефону потрібні:

- кабель microUSB — USB
- перехідник USB-OTG.

Рисунок В.9 – Слайд №9

Бокове меню навігації

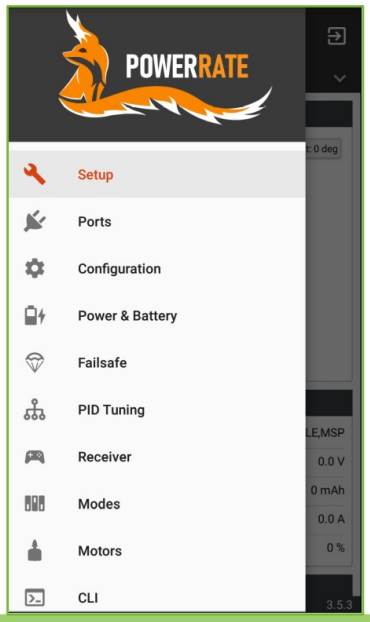


Рисунок В.10 – Слайд №10

Екран Setup

Перший екран, який відображається після успішного підключення до контролеру

Зверху екрана є 3D модель квадрокоптера, для візуалізації поточного положення у просторі (кути нахилу).

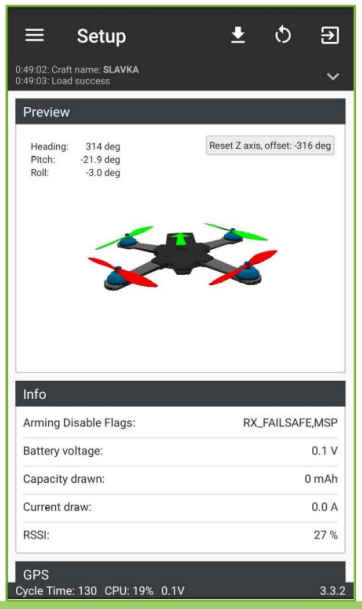


Рисунок В.11 – Слайд №11

Висновки

В результаті виконання дипломної роботи було виконано аналіз необхідності мобільного застосунку для налаштування польотного контролера квадрокоптера.

Було описано схему роботи протоколу MSP, зокрема розглянута структура його пакетів, за допомогою яких застосунок виконує обмін даними з контролером.

Застосунок дозволяє оператору квадрокоптера економити місце серед речей, яки необхідні йому у роботі. Відпадає необхідність брати із собою ноутбук.

Рисунок В.12 – Слайд №12