**Міністерство освіти і науки України**

**Чернівецький національний університет**

**імені Юрія Федьковича**

. Навчально-науковий інститут фізико-технічних та комп’ютерних наук ...

(повна назва інституту/факультету)

. Кафедра математичних проблем управління і кібернетики .

(повна назва кафедри)

**Завдання стабілізації квадрокоптера**

**Курсова робота**

**Рівень вищої освіти – перший (бакалаврський)**

Виконав:

студент . 3 .курсу, групи . 341 .

спеціальності

122 – Комп’ютерні науки .

(шифр і назва спеціальності)

. Пожога В. Г. .

(прізвище та ініціали)

Керівник . к.ф.-м.н. асистент Коцур М.П. .

(науковий ступінь, вчене звання прізвище та ініціали)

**До захисту допущено.**

**Протокол засідання кафедри** № \_ 15 **\_\_**

від “\_\_01\_\_” \_\_\_квітня\_\_\_\_\_\_\_\_ 2024 р.

Зав. кафедри\_\_\_\_\_\_\_\_\_ д. ф-м.н. **Дрінь Я.М.**

Чернівці – 2024

# 

# АНОТАЦІЯ

У даній роботі досліджується завдання стабілізації квадрокоптера у повітрі під впливом чинників оточуючого середовища, в тому числі вітру. Розроблене рішення має злітати на раніше задану висоту та утримувати її незалежно від зовнішніх впливів. Важливим аспектом є повна автономність у тому числі відсутність зовнішнього керування та живлення.

На основі проведених досліджень була зібрана відповідна апаратна база та розроблено програмне забезпечення. У якості обчислювального пристрою було обрано мікроконтролер ESP-32 S3.

Розроблений пристрій повністю задовільнив мету та вдало виконав поставлене завдання. Пристрій відповідає всім функціональним вимогам, а також вимогам по структурі та загальній архітектурі проекту.

Програмну частину продукту було реалізовано засобами середовища Arduino IDE, мовою C++ з використанням бібліотек DShotRMT, Wire.h та інших.

# **Ключові слова**: Завдання стабілізації; ПІД регулятор; Квадрокоптер .

**ANNOTATION**

In this work, studies he task of stabilizing a quadcopter in the air under the influence of environmental factors, including wind. The developed solution should fly to a previously set height and maintain it regardless of external influences. An important aspect is complete autonomy, including the absence of external control and power supply.

Based on the research, the appropriate hardware was assembled and software was developed. The ESP-32 S3 microcontroller was chosen as the computing device.

The developed device fully satisfied the purpose and successfully completed the assigned task. The device meets all functional requirements, as well as structural and general architectural design requirements.

The program part of the product was implemented to use the Arduino IDE environment, in C++ language using DShotRMT, Wire.h and other libraries.

Keywords: The task of stabilization; PID regulator; Quadcopter.

ЗМІСТ

|  |  |
| --- | --- |
| ВСТУП (Аргументація теми та її актуальність)…………………………………………. | 5 |
| ТЕОРЕТИЧНА ЧАСТИНА………………………………………………………………. | 10 |
| 1. ОГЛЯД ТЕМИ, СУЧАСНІ ПІДХОДИ ДО РЕАЛІЗАЦІЇ, РОЗШИРЕНА  ПОСТАНОВКА ЗАДАЧІ…………………………………………………………………. | 11 |
| 1.1 Постановка задачі. Сучасні підходи до реалізації ……………………………….… | 11 |
| 1.2. Призначення та область застосування……………………………………………... | 13 |
| 1.3. Загальна математична модель………………………………………………………. | 14 |
| 1.4. Опис основних алгоритмів…………………………………………………………... | 15 |
| ВИСНОВКИ ДО ТЕОРЕТИЧНОЇ ЧАСТИНИ…………………………………………… | 17 |
| ПРАКТИЧНА ЧАСТИНА………………………………………………………………… | 18 |
| 2. КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ…………….................. | 18 |
| 2.1. Загальна структура апаратно-програмного комплексу…………………………… | 18 |
| 2.2. Вимоги до програмного забезпечення……………………………………………..  2.2.1. Функціональні вимоги…………………………………………………………  2.2.2. Вимоги до складу та параметрів технічних засобів…………………………  2.2.3. Вимоги до вхідних і вихідних даних……………………………………….  2.2.4. Вимоги до програмної документації………………………………………. | 19  19  20  21  22 |
| 2.3. Функціональні можливості системи…………………………………....................... | 23 |
| 2.4. Користувацький інтерфейс…………………………………………………………. | 24 |
| 2.5. Опис класів та програмних модулів………………………………….................... | 25 |
| 2.6. Методика та результати випробувань……………………………………………… | 35 |
| ВИСНОВКИ ДО ПРАКТИЧНОЇ ЧАСТИНИ……………………………………………. | 36 |
| ЗАГАЛЬНІ ВИСНОВКИ ДО РОБОТИ………………………………………………….. | 37 |
| ВИКОРИСТАНА ЛІТЕРАТУРА…………………………………………….................. | 39 |
| ДОДАТКИ……………………………………………………………………................... | 40 |
| Додаток А………………………………………………………………………………… | 40 |
| Код програми……………………………………………………………….................... | 40 |

# ВСТУП

# Квадрокоптер – це літальний безпілотний апарат (БПЛА), що має чотири лопаті, які найчастіше розташовані перпендикулярно одна до одної в одній площині.

# На кінці кожної з них розташований двигун з повітряним гвинтом (пропелер) який створює тягу.

# Актуальність: Подібні апарати мають доволі широке застосування, починаючи з аграрної сфери діяльності людини, де за допомогою крадрокоптерів різних конфігурацій як зрошують засіяні поля так і проводять нагляд за територією. Не менш важливою сферою використання безпілотних літальних апаратів зокрема дронів-квадрокоптерів FPV(First Person View) є військова. Про те незважаючи на різноманітність завдань які подібні квадрокоптери здатні виконувати, наявні деякі обмеження у зв’язку з якими військові змушені купувати дорого-вартісні заводські рішення закордонного виробництва. Одним таким обмеженням є відсутність у квадрокоптерів FPV можливості «зависати» в одній точці та вести відеоспостереження; оскільки дані дрони не мають подібного режиму роботи і потребують постійного коригування свого курсу оператором.

**Мета роботи**: Метою цієї роботи є розробка рішення як апаратного (на основі поширених збірок FPV дронів) так і програмного для завадостійкого, автономного керування квадрокоптером. Забезпечуючи можливість стабільно вести відеоспостереження з заданої висоти у заданому напрямку.

**ТЕОРЕТИЧНА ЧАСТИНА**

1. ОГЛЯД ТЕМИ, СУЧАСНІ ПІДХОДИ ДО РЕАЛІЗАЦІЇ, РОЗШИРЕНА ПОСТАНОВКА ЗАДАЧІ

1.1. Сучасні підходи до реалізації

Системи автоматичного управління (САУ) призначені для автоматичної зміни одного або декількох параметрів об'єкта керування з метою встановлення необхідного режиму його роботи.

Ключовим елементом будь-якої системи автоматичного управління є регулятор. Регулятор це пристрій завдання якого стежити за об’єктом управління, а точніше його станом та забезпечувати необхідний закон управління.

Дослідивши питання детальніше, знаходимо перелік підходів до створення алгоритмів та систем для керування польотом квадрокоптеру. Такими наприклад є лінійні квадратичні регулятори. Проте існують істотні недоліки які роблять дані регулятори непідходящими для наших завдань. Одним суттєвим недоліком є те що дані регулятори не працюють у випадку істотних нелінійностей.

У цій роботі ми будемо використовувати інший регулятор відомий як ПІД регулятор. Рішення перевірене часом

2.2. Призначення та область застосування

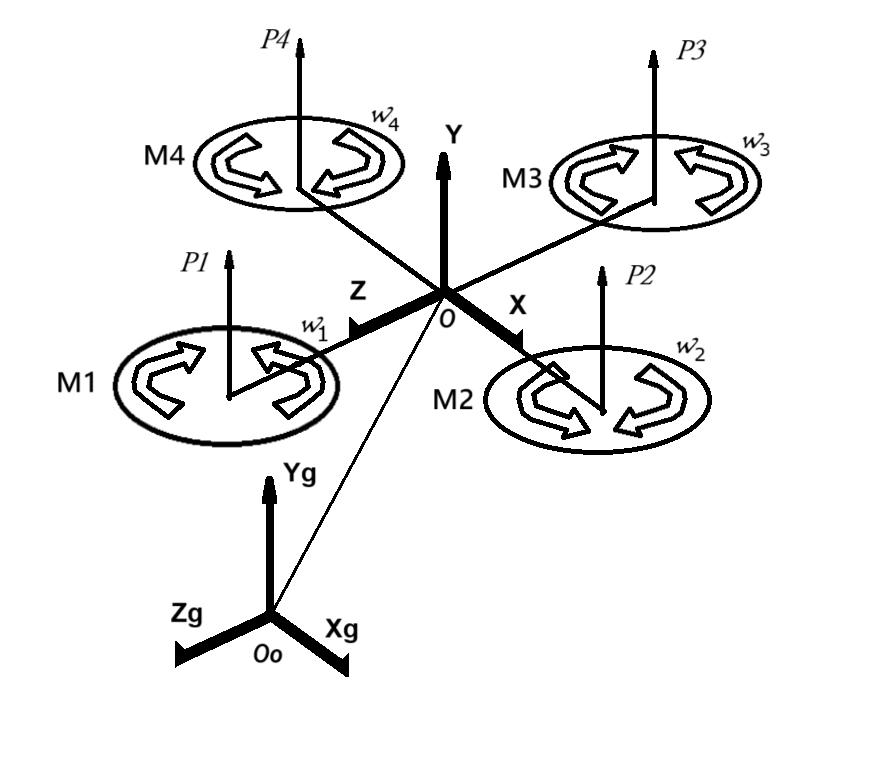
Розроблений програмний продукт призначений для стабілізації квадрокоптерів. Створений він на платформі ESP32-S3. Вбачається, що дане програмне забезпечення буде використовуватись в тих областях і галузях роботи квадрокоптерів, де необхідно тривалий час знаходитись на певній висоті та в межах певних географічних координат. Простою мовою - зависати в певній точці. Зазвичай завдання такого роду постають у випадках пов’язаних з відеоспостереженням. Зокрема коли потрібно спостерігати статично (точніше умовно статично) за якимось об’єктом або групою об’єктів. Будь то (до прикладу) трактор у полі, військова техніка на полі бою або наречені на весіллі. Призначань даній технології є багато, все залежить лише від вдалості реалізації і фантазії користувача.

2.3. Загальна математична модель

Перед початком реалізації, слід розглянути математичні принципи, що лежать в основі розробляємого програмного продукту.

Розглянемо квадрокоптер як математичну модель:

На малюнку 1 показано взаємне розміщення зв’язаної (OXYZ) і нормальної земної (OoXgYgZg) систем координат в режимі зависання і діючі на крвадрокоптер сили і моменти.



**Мал. 1** Квадрокоптер в режимі зависання

На малюнку:

*Pi* (*і* = 1,2,3,4) – сила тяги *і*-го гвинта (пропелера)

*Мі* (*і =* 1,2,3,4) – моменти опору *і-*го гвинта

*wi* (*i* = 1,2,3,4) – швидкість обертання *і­*-го гвинта

Сила тяги в зв’язаній системі координат:

(1)

Де:

– сумарна тяга, - коефіцієнт сили тяги, – щільність повітря, - коефіцієнт підйомної сили, - площа яку ометають лопаті *і-*го гвинта, - радіус *і-*го гвинта.

Сила тяги в нормальній земній системі координат:

(2)

Де:

* Матриця переходу (з зв’язаної системи координат в нормальну земну);

ψ , θ , γ – кути рискання, тангажу, крену.

Сила опору повітря і сила тяжіння:

(3)

Де: – маса квадрокоптера, – прискорення вільного падіння (прискорення сили тяжіння).

**Рівняння динаміки руху центру мас** в нормальній земній системі координат:

(4)

Враховуючи симетрію пристрою і вважаючи що центр мас розміщено в початку координат зв’язаної системи, **рівняння динаміки кутового руху** в зв’язаній системі координат можна записати у вигляді:

(5)

(6)

Де: – проєкції вектора кутової швидкості пристрою на зв'язану систему координат; – проєкції результуючого моменту; – осьові моменти інерції пристрою; – моменти, що створюють гвинти; і – гіроскопічні моменти двигунів і гвинтів.

Якщо знехтувати інерційністю гвинтів при зміні кутових швидкостей їх обертання, то вказані моменти можна виразити наступним чином:

(7)

(8)

Де: – відстань від центра мас до осі гвинта; і – моменти інерції ротора і гвинта; – коефіцієнт моменту.

Зміни кутів Ейлера визначаються через проєкції кутової швидкості кінематичними рівняннями Ейлера:

(9)

2.4. Опис основних алгоритмів

Вхідними даними для системи орієнтації і стабілізації кутового положення будуть значення кутів ψд , θд , γд , а виходами параметри кутового руху об’єкта, в нашому випадку квадрокоптера. Керуючий алгоритм системи має формувати керуючі сигнали які забезпечують створення необхідних моментів , при умові що спільна тяга гвинтів буде рівною або більшою необхідної для вертикального руху. Якщо знехтувати динамікою і обмеженнями двигунів, гіроскопічними моментами і аеродинамічними моментами частин пристрою що не створюють тягових моментів, то сигнали мають бути з точність до коефіцієнту рівними моментам . Тому моделлю об’єкта для алгоритму стабілізації є рівняння (5) і (6) при , ,

.

Суть методу «бекстепінг» полягає в представленні складної системи у вигляді ланцюжка вкладених підсистем, для кожної з яких формуються допоміжні керуючі сигнали і складається залежна від цих сигналів функція Ляпунова. Виконання критеріїв стійкості за Ляпуновим при послідовному виборі цих сигналів для кожної підсистеми забезпечує стійкість системи в цілому. Процедура має характер крокового обходу інтеграторів зворотними зв'язками, звідки походить назва «integrator backstepping», або коротко - бекстеппінг (англ. backstepping). У певних часткових випадках процедура стає регулярною і досить простою. Для кутового руху літального апарата такий випадок можливий при малих кутах тангажа і крена, коли похідні кутів γ, ψ і θ можна вважати рівними відповідним кутовим швидкостям. Тоді рівняння кутового руху (5,6) можна наближено подати у вигляді трьох підсистем:

(10)

Відповідно до наведеного в алгоритмі, введемо допоміжний керуючий сигнал для підсистеми S1:

І відповідну функцію Ляпунова:

,

похідна якої:

.

Другий допоміжний сигнал сформуємо у вигляді:

З відповідною функцією Ляпунова :   
похідна якої:

Приймаючи для системи стабілізації , , отримуємо:

Щоб підсистема була стійкою, тобто щоб при чому тільки коли , приймем:

,

Тоді керуючий сигнал буде мати вигляд:

Функція Ляпунова

Похідна функції:

Тобто замкнута підсистема – стійка.

Аналогічним чином отримуємо і інші сигнали:

де:

Вибираючи значення коефіцієнтів можна досягнути необхідної якості перехідних процесів. Значення цих коефіцієнтів наведенні в таблиці 1

**Таблиця 1.** Значення коефіцієнтів регулятора «бекстепінг»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 20 | 3 | 21 | 2.5 | 35 | 25 |

В таблиці 2 значення коефіцієнтів ПІД-регуляторів вибрані для цієї системи:

Таблиця 2. Значення коефіцієнтів ПІД-регуляторів

|  |  |  |  |
| --- | --- | --- | --- |
|  | k-P | k-I | k-D |
| γ | 2.5 | 5.2 | 10 |
|  | 5.1 | 10 | 10 |
| θ | 5.1 | 0.12 | 20 |

ВИСНОВКИ ДО ТЕОРЕТИЧНОЇ ЧАСТИНИ

Під час роботи над теоретичною частиною я дослідив сучасні підходи до реалізації поставленного завдання, обрав способи та технічну базу для подальшої реалізації. Розписав загальне призначення та область застосування продукту що розробляю. Написав об’ємну, вичерпну математичну модель до розробляємої системи з розрахунком сил які об’єкт дослідження генерує і сил які впливають на об’єкт дослідження. Також не менш важливим було дослідження алгоритмів які будуть використані у роботі. Зокрема алгоритм орієнтації і стабілізації кутового положення що використовує за основу пропорціонально-інтегрально-диференціальний регулятор відомий як ПІД регулятор.

**ПРАКТИЧНА ЧАСТИНА**

**2.1 Загальна структура розробленого апаратно-програмного продукту**

**2.1.1** **Апаратна частина**

Для початку розглянемо апаратну частину розробленого продукту.

Виріб складається:

- Просторова рама Mark 4;

- 4 двигуни безщіткові Ysido 2807 V2 1300KV;

- 4 пропелери (7 дюйм);

- Модуль керування обертами ESC (Electronic Speed Controller) Speedybee ESC 50A;

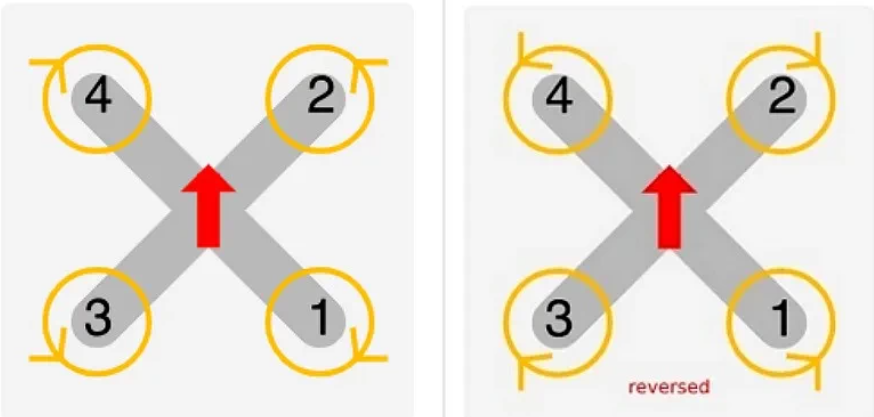
- Плата розробки ESP32 – S3 Pico Waveshare Elecronics;

- Модуль датчиків GY-87: гіроскоп, акселерометр – MPU6050; магнітометр – HMC5883L; барометр – BMP180;

- Модуль лазерного далекоміра TOF400f;

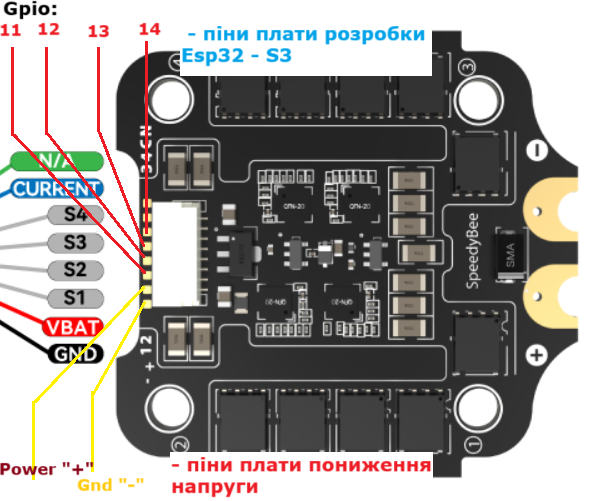
- Плата пониження напруги 24 – 3 В.

Збірка апарату починається з рами. Комплект рами складається з багатьох частин які з’єднуються за допомогою гвинтів. Далі встановлюємо модуль ESC у відведене місце в середині рами. На кінцях променів встановлюємо двигуни, провадячи по променях дроти які згодом паяєм до контактних площадок модуля регулювання обертів. Зверху на кожен двигун вкладаємо пропелер який фіксуємо комплектною гайкою. Потрібно звертати увагу на те, що пропелер має полярність і встановлюється відповідно обраної вами схеми:

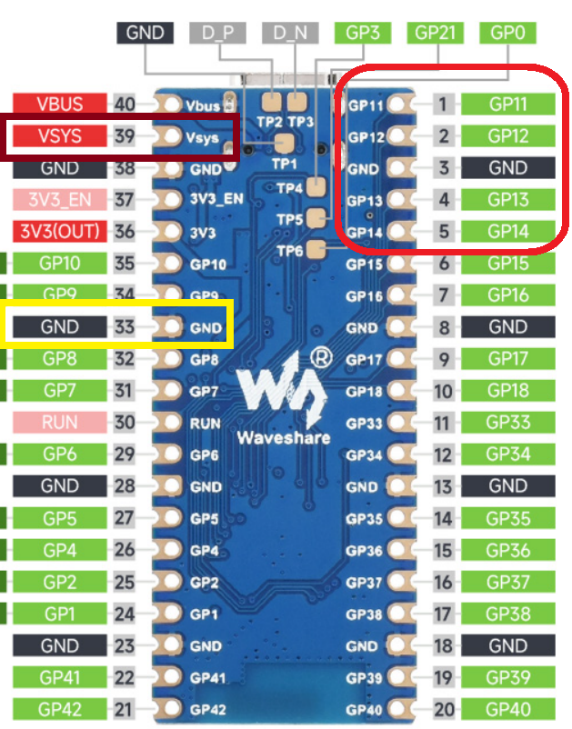


**Мал2.** Схема встановлення пропелерів

Встановлюємо плату розробки на якій розташований мікроконтролер який я буду згодом програмувати. ЇЇ слід розташувати на рамі таким чином, щоб мати доступ до USB порту на платі. Від плати розробки до модуля ESC потрібно провести 4 сигнальні дротики через які ми будемо здійснювати керування обертами двигунів. Для зручності було використано комплектний роз’єм. До самої плати розробки дроти було припаяно на 11, 12, 13 і 14 виходи. Також від модуля ESC потрібно провести ще два дроти на плату пониження напруги. А саме виходи Vbat та Gnd. Зі зворотної сторони плати пониження напруги проводимо відповідні дроти на плату розробки від’ємний(«-») вихід на вихід Gnd, а додатній («+») на вихід Vsys плати розробки. Перед цим слід налаштувати плату пониження напруги на напругу 3.3 В.



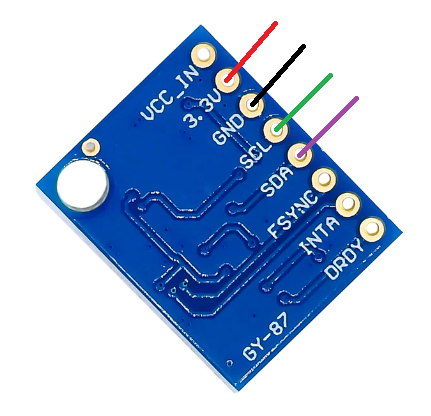
**Мал3.** Виходи модуля ESC (Power це вихід Vbat на модулі ESC)



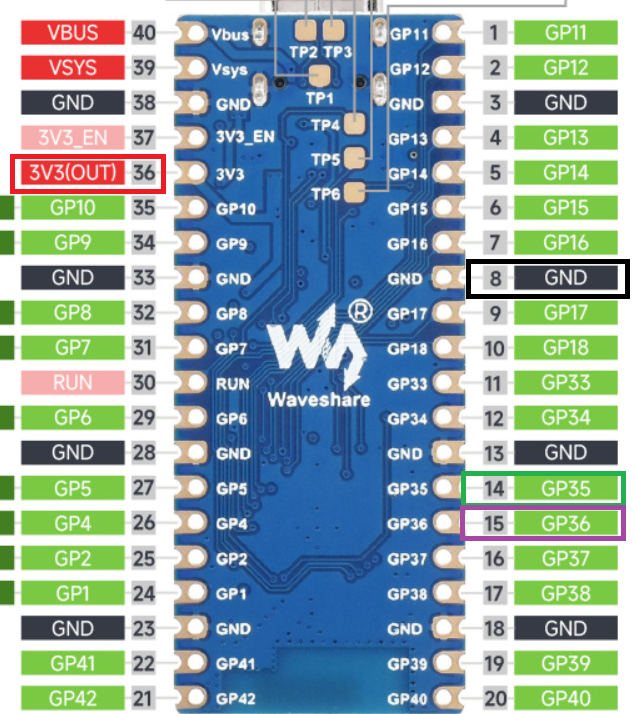
**Мал4.** Виходи плати розробки ESP32-S3 Pico

Для орієнтації у просторі нам знадобиться набір датчиків, зокрема акселерометр, гіроскоп, магнітометр (компас) та барометр. Для задоволення цих потреб було придбано модуль датчиків GY-87. Цей модуль містить 3 мікросхеми. Дві з них нам потрібні і працюють по протоколу I2C. Цей тип з’єднання передбачає двоє сигнальних дротів. Також на модуль потрібно провести живлення та заземлення.

Його схема підключення:

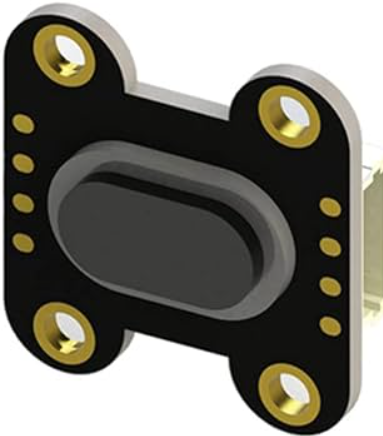


**Мал5.** Схема підключення модуля GY-87

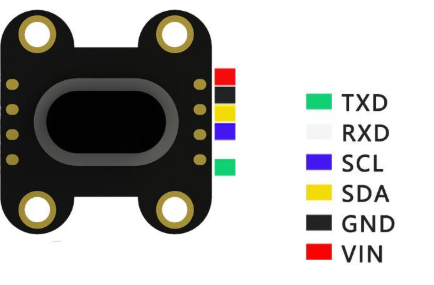


**Мал6.** Схема підключення модуля GY-87 до ESP32

Модуль лазерного далекоміра під’єднаємо послідовно на шину I2C. Кожен контакт модуля до кожного відповідного у GY-87 (VIN до 3.3V).



**Мал7.** Модуль TOF400f (вигляд знизу)

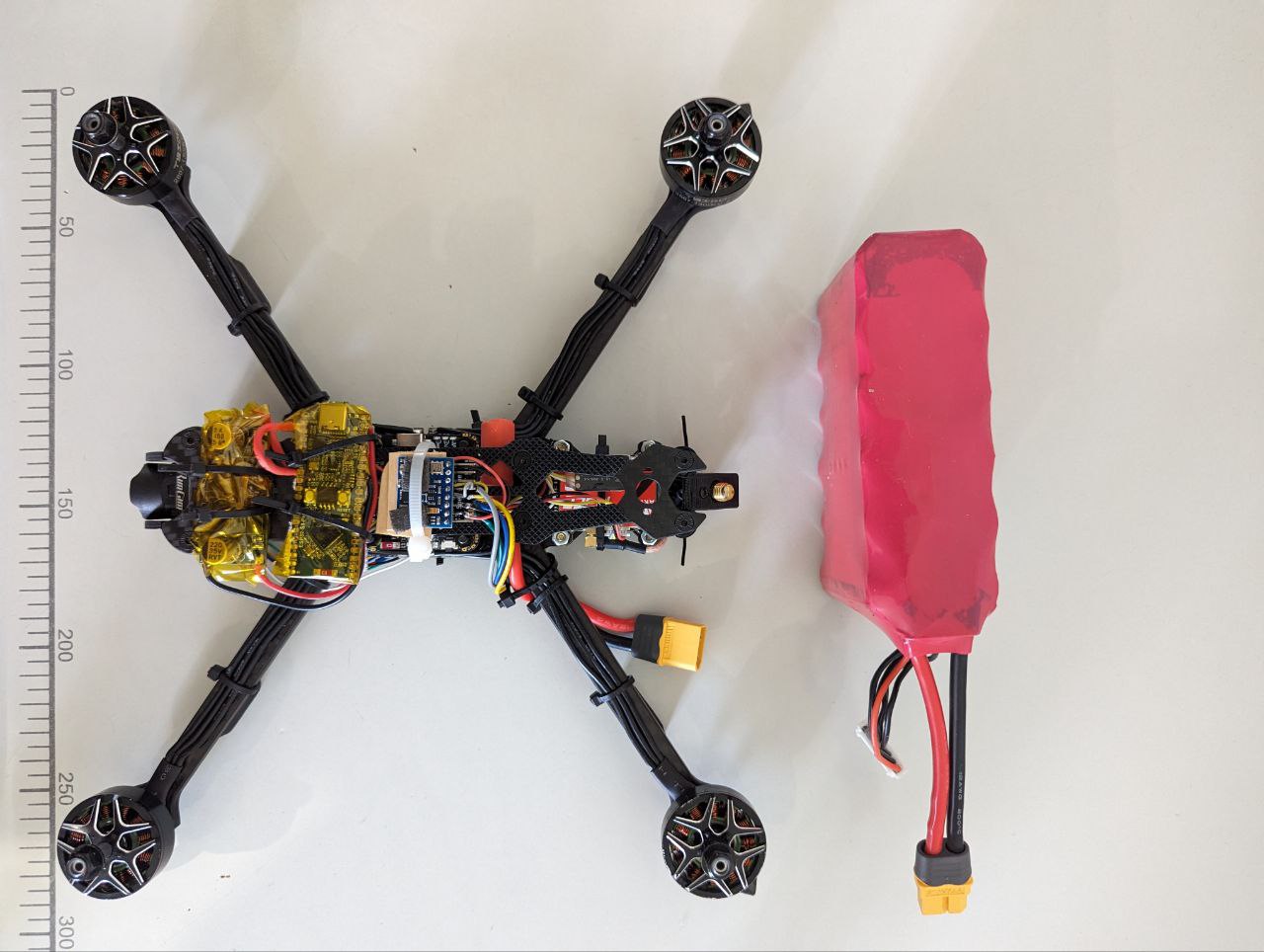


**Мал8.** Схема підключення TOF400f

TXD та RXD на постійній основі не використовувалися. Вони потрібні лише для переведення модуля у режим I2C підключення.

До проєкту було придбано ще літій-іонні акумуляторні елементи sony murata vtc6, з яких було зібрано акумуляторну батарею напругою 24 вольта та струмовіддачею від 60 до 160 (у піку) ампер.

Загальний вигляд апаратна частина проєкту такий:



**Мал9.** Зібраний квадрокоптер (вигляд зверху)



**Мал10.** Зібраний квадрокоптер (вигляд збоку)

**2.1.2** **Програмна частина**

Слід зазначити що для програмної розробки використовувалось середовище розробки ArduinoIDE а платою розробки було обрано ESP32-S3 pico. Відповідно до чого програма має певну структуру. Програма з самого початку (при створенні проєкту у середовищі) має дві функції: setup() та loop(). Розглянемо детальніше їх характеристики: setup() – це функція яка виконується один раз при виконанні програми на мікроконтролері. Найчастіше вона служить для налаштування певних параметрів мікроконтролера. loop() – це функція яка буде виконуватись на мікроконтролері періодично. Тобто один раз виконується функція setup() а потім функція loop() буде виконуватись знову і знову до тих пір поки ви не виконаєте скид мікроконтролера.

Отже, для початку імпортуємо всі бібліотеки. Нам знадобляться наступні:

* Wire.h для I2C протоколу
* VL53L1X.h для правильної взаємодії з модулем TOF400f.
* Arduino.h для правильної роботи попередньої
* DShotRMT.h стороння бібліотека для DShot з’єднання з модулем ESC
* math.h для математичних функцій мови С++
* stdio.h стандартна бібліотека вводу-виводу мови С\С++

У функції setup() я вмикаю послідовне з’єднання по СОМ порту, ініціалізую змінні бібліотек та налагоджую з’єднання з модулями зокрема гіроскопом, акселерометром і далекоміром та калібрую їх.

Я також створив декілька глобальних функцій:

upDate\_GYRO() – звертається до гіроскопа а потім до акселерометра по шині I2C та отримує данні які перетворює в градуси (градус/с для гіроскопа) і записує їх в глобальні змінні.

upDate\_BARO(bool, bool) – в залежності від параметрів робить запити до барометра або термометра для підготовки даних або для їх завантаження. Розділення підготовки та отримання даних було потрібне оскільки процес який це забезпечує в модулі BMP180 займає щонайменше 5 мілісекунд, що робить непридатним використання іншої (стандартної бібліотечної) функції у loop().

kalman\_1d() – функція яка реалізує фільтр Калмана.

pid\_equation() – функція для реалізації ПІД регулятора.

У loop() відбувається зчитування параметрів орієнтації приладу у просторі,

фільтрування отриманих з гіроскопу і акселерометру даних за допомогою фільтру Калмана. Далі обрахування необхідних керуючих сигналів за допомогою pid\_equation() яка базується на різниці між бажаною орієнтацією і тою що отримана від датчиків. Потім на отримані сигнали накладаються обмеження задля забезпечення безпеки (у тому числі і комплектуючих), та відправляються на модуль ESC.

2.2. Аналіз вимог до програмного забезпечення

2.2.1. Функціональні вимоги

**До програмного забезпечення висуваються наступні вимоги:**

- можливість зчитувати дані з датчика гіроскопа MPU6050 без затримок;

- можливість зчитувати дані з датчика барометру BMP180 без затримок;

- можливість зчитувати дані з датчика лазерного далекоміра TOF400f без затримок;

- реалізація алгоритму ПІД регулятора;

- реалізація протоколу Dshot300/600;

- реалізація фільтру Калмана;

- правильна інтерпретація та використання даних для досягнення стабілізації пристрою у повітрі;

- сумісність з модулем керування обертами (ESC).

Програмний продукт повинен бути надзвичайно ефективним та мати вражаючі швидкісні показники оскільки для поставленого завдання затримки в десяток мілісекунд критичні.

2.2.2. Вимоги до складу та параметрів технічних засобів

Розроблений продукт призначений для апаратної платформи ESP32-S3 та використовує модуль тактування RMT.

Параметри мікроконтролера:

Таблиця 2.2.2.1

|  |  |
| --- | --- |
| Параметр | Значення |
| Процесор | ESP32-S3R2 |
| ОЗП | 512KB SRAM, 384KB ROM, 2MB PSRAM |
| Частота процесора | 240 МГц. |
| Пам'ять | 16 MБ |
| Інтерфейси периферії | USB OTG, SPI, I2C, UART, ADC, PWM, DVP , LCD |

Для коректної роботи програмного продукту слід використовувати весь перелік периферійних пристроїв та власне саму плату розробки ESP32-S3-Pico

виробництва Waveshare.

Використані периферійні пристрої:

- Модуль керування обертами Speedybee ESC 50A;

- Модуль датчиків GY-87;

- Модуль лазерного далекоміра TOF400f.

При використані інших платформ або периферії з великою ймовірністю доведеться вносити зміни у програмний код для правильної його роботи. Попри це використання середовища розробки ArduinoIDE дає нам значну універсальність і можливість використання коду на платах розробки Arduino.

2.2.3. Вимоги до вхідних та вихідних даних

Вхідними даними для розробленого програмного продукту є значення фізичних параметрів оточуючого середовища:

- Тиск – не вище 1100 гПа

- Температура – не вище 180 °С

- Відстань – не більше 400 см.

- Кутова швидкість – не вище 2000°/с.

Вихідними даними є значення тангажа кожного з чотирьох двигунів. Вони мають бути числами в межах від 48 до 2000 які будуть перетворені на Dshot пакети та передані на ESC.

2.2.4. Вимоги до програмної документації

Програмне забезпечення постачається разом із супроводжувальноюдокументацією, до складу якої входить:

1. Технічне завдання.

2. Опис та обґрунтування обраної архітектури.

3. Функціональна специфікація.

4. Технічна специфікація.

5. Опис програми.

6. Програма та методика випробувань.

2.3. Функціональні можливості системи

Програмне забезпечення має такі функціональні можливості:

- зчитування**/**запис даних на периферійні пристрої;

- перетворення даних з периферії на значення фізичних величин;

- фільтрування даних акселерометра і гіроскопа;

- використання пропорціонально-інтегрально-диференціального регулятора;

- використання протоколу DShot300**/**600;

- польотний контролер для квадрокоптера;

- стабілізація квадрокоптера у повітрі.

2.4. Користувацький інтерфейс

Взаємодія користувача із програмним продуктом здійснюється шляхом підключення пристрою до СОМ порту комп’ютера через USB роз’єм. Сам пристрій оснащений роз’ємом USB type-C. Через послідовний (Serial) порт розроблений пристрій передає вибірку даних про свій стан та дані які він зчитує з наявних датчиків. Подивитись дані у послідовному порті можна у ArduinoIDE та інших.

2.5 Опис класів та програмних модулів

Розроблене програмне забезпечення складається із таких модулів:

* Модуль Гіроскопа-Акселерометра;
* Модуль Барометра;
* Модуль ПІД регулятора;
* Модуль Фільтру Калмана;
* Модуль Dshot;
* Модуль Далекоміра;
* Модуль Ініціалізації.

Модуль Гіроскопа-Акселерометра представлений функцією *upDate\_GYRO*.

За допомогою бібліотеки Wire.h робить запити по шині I2С до апаратного модуля MPU6050 та отримує данні кутових швидкостей, перетворює їх і вносить у глобальні змінні.

Модуль Барометра представлений функцією upDate\_BARO. За допомогою бібліотеки Wire.h робить запити по шині I2С до апаратного модуля BMP180 та отримує данні температури та тиску, перетворює їх і вносить у глобальні змінні.

Модуль ПІД регулятор представлений функцією pid\_equation. Реалізує алгоритм Пропорціонально-інтегрально-диференціального регулятора та записує значення у масив. Вхідні значення отримує як параметри функції.

Модуль Фільтру Калмана представлений функцією kalman\_1d. Реалізує алгоритм лінійно-квадратичного оцінювання  та записує значення у масив. Вхідні значення отримує як параметри функції.

Модуль Dshot представлений класом *DShotRMT*. Має велику кількість методів для маніпуляцій з протоколом Dshot. Призначений для створення і відправки пакетів вищевказаного протоколу.

Об’єкт класу оголошуємо так:

DShotRMT *dshot0(RTRCTL0(таймер), RTRCTL\_RMT\_CHNL0(канал RMT))*.

Для початку передачі слід викликати:

*dshot0.begin(DSHOT300(швидкість),false (рівень «нулю»))*.  
Для передачі пакети викликаємо:

*dshot0.sendThrottleValue(0 (значення від 0 до 2000) ).*

Модуль Далекоміра представлений класом VL53L1X. Призначений для налагодження і роботи з мікросхемою VL53L1 яка використовується у модулі лазерного далекоміра TOF400f який використовується в проєкті. Містить використані мною

* *TOF400f.setAddress(PFER\_ADDRESS3)*; - для встановлення адреси у мережі I2С.
* *TOF400f.setTimeout(500)*; - максимальна затримка після якої сигнал вважається втраченим.
* *TOF400f.init()*; - перевірка чи присутній пристрій і з’єднання з ним
* *TOF400f.setDistanceMode(VL53L1X::Long)*; - режим роботи мікросхеми
* *TOF400f.setMeasurementTimingBudget(50000)*; - встановлює затрачений час на одне вимірювання в мікросекундах.
* *TOF400f.startContinuous(50)*; - починає серію вимірювань через кожні х мілісекунд.
* *TOF400f.read()*; - повертає значення зчитані з мікросхеми в мм.

Модуль Ініціалізації представлений функцією setup(). Потрібний для

ініціалізації та стартового налаштування як програмних так і апаратних сутностей. Для початку у модулі запускаємо Послідовний порт: *Serial.begin(115200).*

Потім ініціалізуємо з’єднання по шині I2С:

*Wire.setClock(400000);*

*Wire.begin(SDA,SCL);*

*delay(250).*

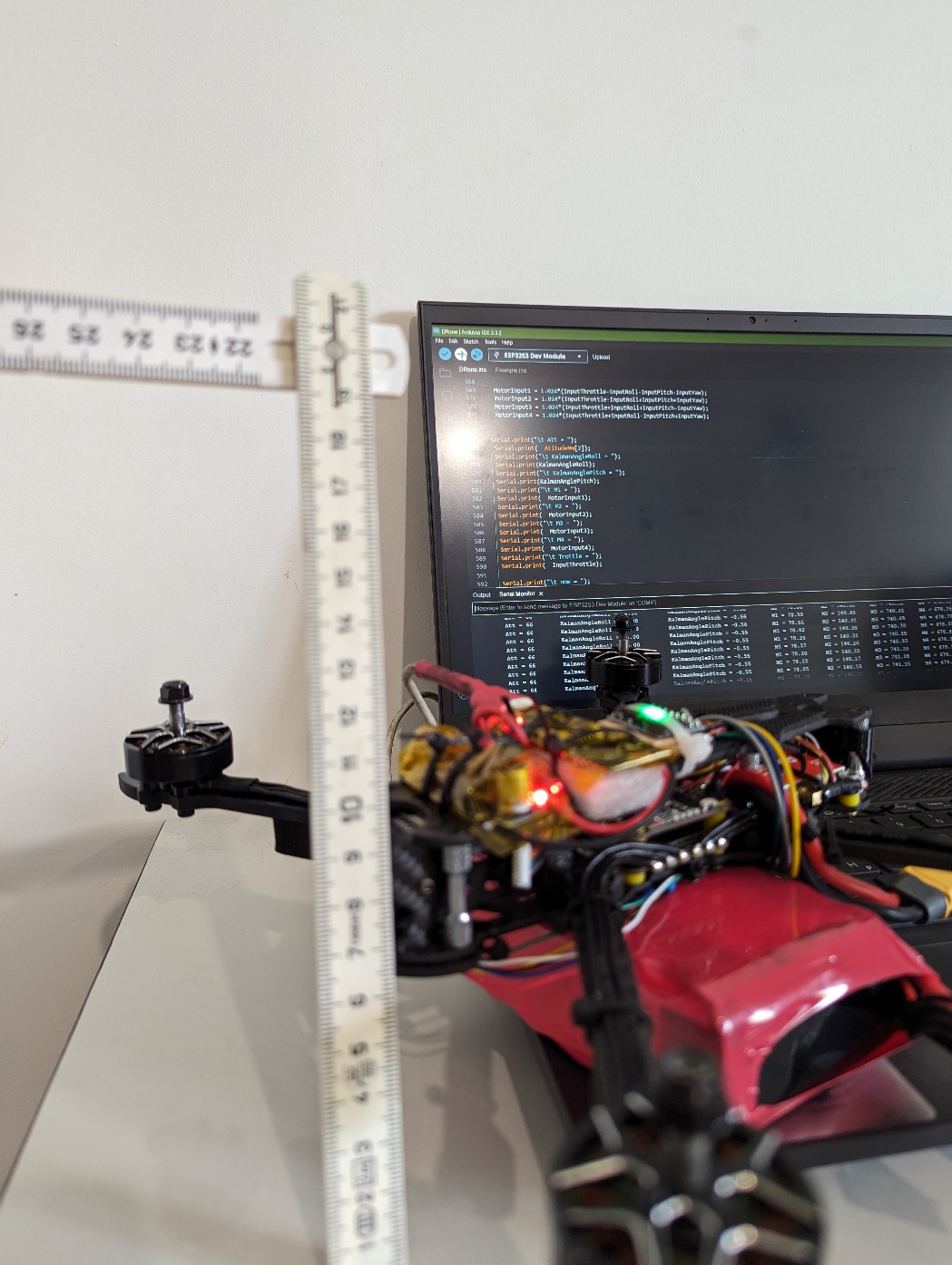
Запускаємо MPU6050, калібруємо його значення. Зчитуємо дані для калібрування барометру з мікросхеми BMP180. Записуємо адрес, потрібний режим і запускаємо періодичне вимірювання для VL53L1.

Оскільки цей модуль виконується першим і займає значний час, модуль ESC виходить у робочий режим і очікує на пакети Dshot. У випадку якщо він ці пакети не отримає за певний час ESC не ініціалізується. Тому під час калібрування Гіроскопу (найбільш коштовна операція по часу) ми також передаємо пакети Dshot з нульовим значенням.

2.6. Методика та результати випробувань

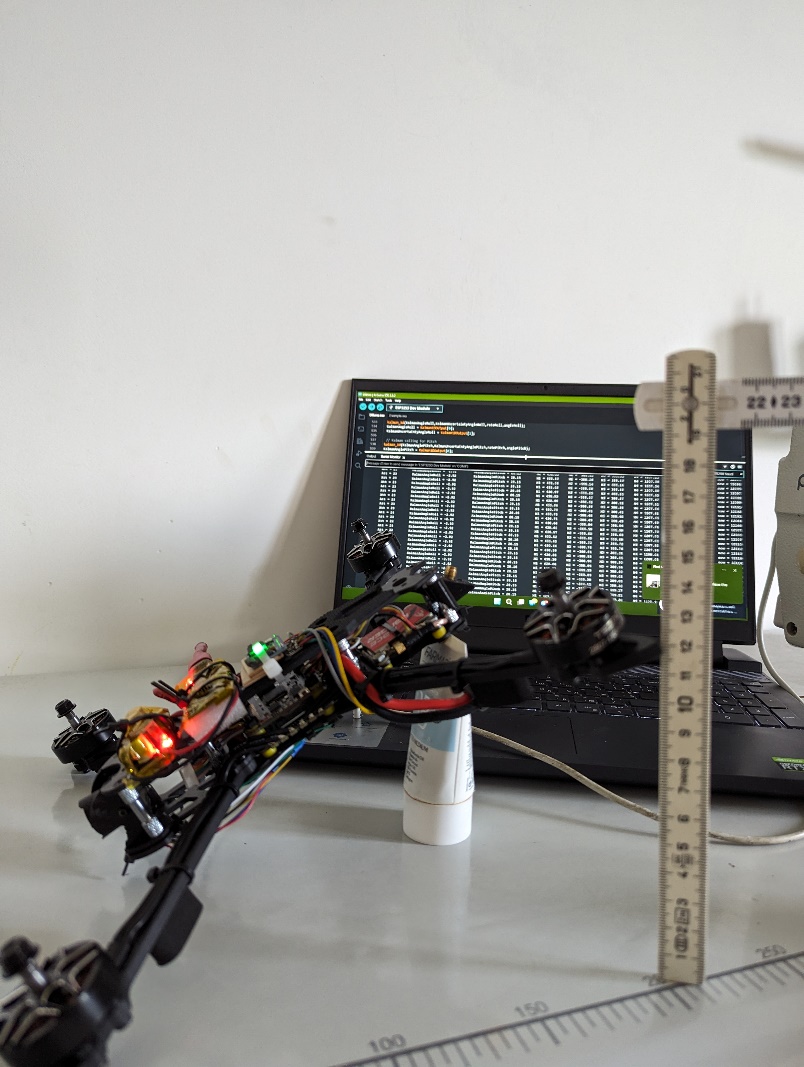
Для проведення тестування програмного та апаратного забезпечення використовувались наступні методики:

* Для перевірки лазерного далекоміра TOF400f, після встановлення на раму квадрокоптера і з’єднання з платою розробки, я під’єднав плату кабелем до комп’ютера на дисплеї відображував вікно ArduinoIDE розділ SerialMonitor куди (відповідно до виконуваної програми) ESP32 передавала значення отримані з датчика. Далі я взяв лінійку та змінюючи положення рами порівнював отримані емпіричним шляхом і виміряні програмою дані.



**Мал11.** Вимірювання висоти

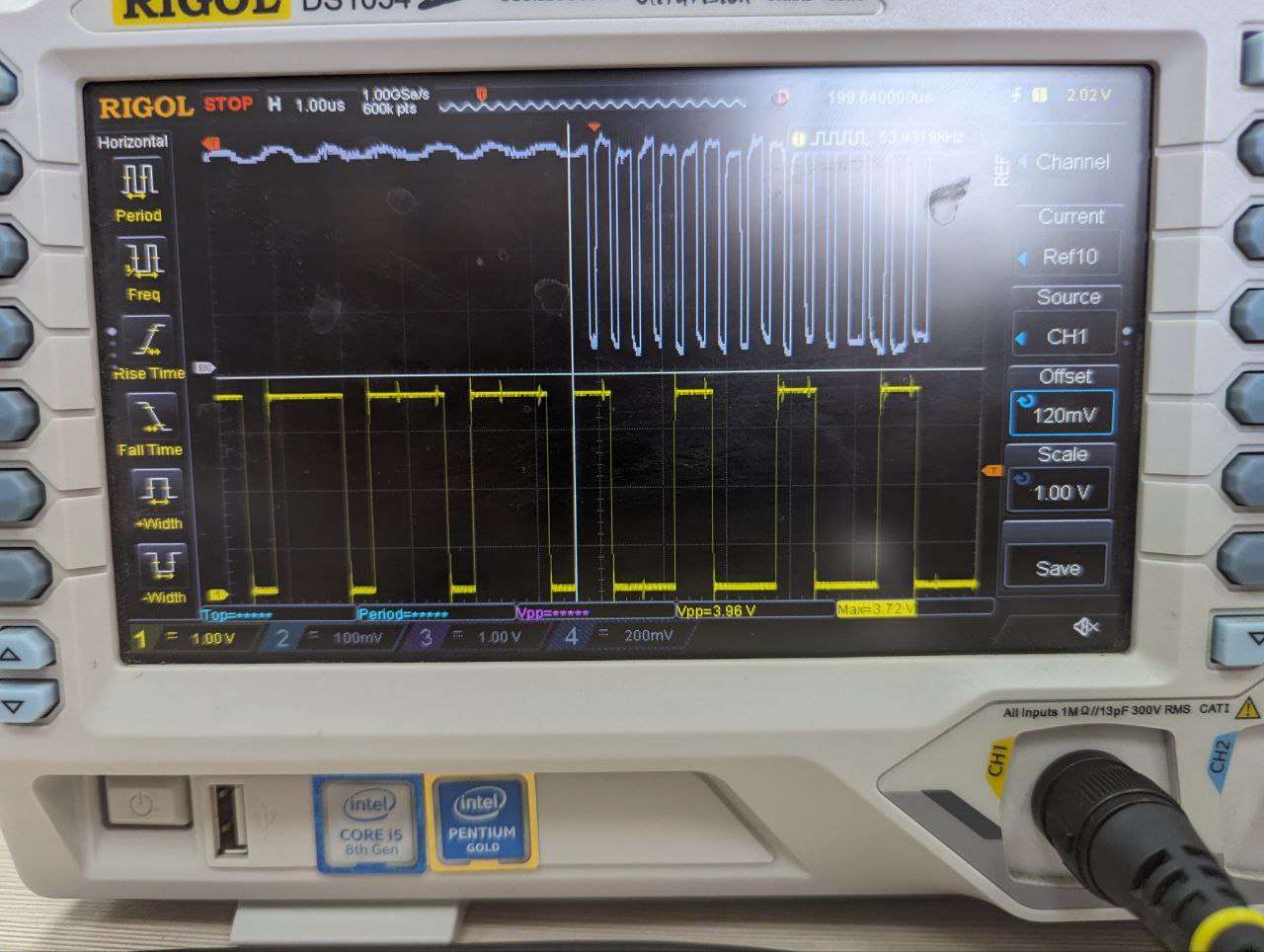
* Для перевірки акселерометра, гіроскопа та роботи фільтру Калмана, я виставив пристрій під кутом, попередньо замірявши його довжину. Довжина буде гіпотенузою прямокутного трикутника. Потім міряєм висоту в крайній піднятій точці за допомогою лінійки.



**Мал12.** Вимірювання кутів нахилу

За отриманими даними (гіпотенуза та катет прямокутного трикутника) вираховуємо синус а потім і сам кут. Порівнюємо з даними на моніторі.

* Для перевірки роботи Dshot потрібно використати осцилограф. Під’єднаємо контакти осцилографа на 10, 11, 12 або 13 піни плати розробки а також на спільне заземлення. Знімаємо осцилограму і порівнюємо її з описанням пакетів. Я мав находу порівнювати осцилограму реалізованого мною протоколу та виробничим виконанням з польотного контролеру Speedybee f405 v3.



**Мал13.** Зняття та порівняння осцилограми

Незважаючи на всі труднощі реалізації, протокол використаний мною нічим не поступається заводським рішенням та показав свою працездатність на практиці.

За результатами випробувань програмне забезпечення показало свою працездатність.

ВИСНОВКИ ДО ПРАКТИЧНОЇ ЧАСТИНИ

У ході роботи над практичною частиною було розроблено програмнезабезпечення для керування та стабілізації квадрокоптера.

У розробленому продукті реалізовані:

− модулі з’єднання з периферією;

− ПІД регулятор;

− Фільтр Калмана.

Програмний продукт розроблений мовою програмування С++ ізвикористанням середовища розробки ArduinoIDE. Продукт дозволяє керувати та стабілізувати квадрокоптер у повітрі.

Перевірка роботи програмного продукту здійснювалася як програмним так і емпіричним шляхом.

Розроблений програмний продукт відповідає поставленим вимогам тапродемонстрував свою працездатність.

ЗАГАЛЬНІ ВИСНОВКИ ДО РОБОТИ

У цій роботі нами проводилося дослідження проблеми стабілізації квадрокоптера у повітрі.

Для конструювання ПЗ нами були використані такі алгоритми:

1) Алгоритм пропорціонально-інтегрально-диференціального регулятора

2) Алгоритм лінійно-квадратичного оцінювання (фільтр Калмана)

3) Протокол передачі даних Dshot

4) Протокол передачі даних I2C

У цілому, нами був розроблений програмний продукт для розв'язаннязавдання, який за результатами випробувань показав свою працездатність.Розроблений програмний продукт можна використовувати для:

- Польотний контролер для керування квадрокоптером;

- Стабілізації квадрокоптера в умовах помірних завад.

Крім цього, програмне забезпечення можна використовувати і для більш  
загальних задач: вимірювання кутів нахилу по двом осям, вимірювання висоти, вимірювання атмосферного тиску та як повно-функціональний компас.

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Сопронюк Ф.О., Сопронюк О.Л., Коцур М.П. методичні вказівки та завдання до лабораторних робіт з дисципліни “сучасна теорія керування ”: навч. посіб. Чернівці: ЧНУ, 2022. 34 с.

2. Пропорційно-інтегрально-диференціальний закон регулювання. Вікіпедія: вебсайт. URL: <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> (дата звернення: 23.05.2024 ).

3. Лисенко С. М., Румянцев С. В. Інтелектуалізована система керування безпілотними літальними апаратами. Комп'ютерні системи та інформаційні технології. 2020. №1. С.22-27.

4. Яровий О.В. Вибір оптимальних моделей безпілотних літальних апаратів та систем управління для виконання задач щодо моніторингу наземних об’єктів. Молодий вчений. 2018. № 5. С. 190–196.

5. Мясіщев О.А., Швець В.В. Режими польоту контролерів польоту apm 2.6 і pixhawk БПЛА. Вісник Хмельницького національного університету. 2018. № 1. С. 78–82.

6.

[ESP32-S3 Series Datasheet. Espressif: вебсайт. URL: https://www.espressif.com/sites/default/files/documentation/esp32-s3\_datasheet\_en.pdf (дата звернення: 12.05.2024)](ESP32-S3 Series Datasheet. Espressif: вебсайт. URL: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf (дата звернення: 12.05.2024) )

7. What is DShot ESC Protocol. Oscarliang: вебсайт. URL: https://oscarliang.com/dshot/ (дата звернення: 15.05.2024 )

ДОДАТКИ

Додаток АКод програми

#include <Wire.h>

#include <Arduino.h>

#include <DShotRMT.h>

#include <math.h>

#include <stdio.h>

#include <VL53L1X.h>

// STATE

bool ARMED = 1;

//I2C values

//pins

#define SCL 36

#define SDA 37

//addresses

#define PFER\_ADDRESS1 0x68

#define PFER\_ADDRESS2 0x77

#define PFER\_ADDRESS3 0x29

//DShot setings

//pins

#define LED\_PIN GPIO\_NUM\_7

#define RTRCTL0 GPIO\_NUM\_11

#define RTRCTL1 GPIO\_NUM\_12

#define RTRCTL2 GPIO\_NUM\_13

#define RTRCTL3 GPIO\_NUM\_14

//chanels

#define RTRCTL\_RMT\_CHNL0 RMT\_CHANNEL\_0

#define RTRCTL\_RMT\_CHNL1 RMT\_CHANNEL\_1

#define RTRCTL\_RMT\_CHNL2 RMT\_CHANNEL\_2

#define RTRCTL\_RMT\_CHNL3 RMT\_CHANNEL\_3

//DShot variables

DShotRMT dshot0(RTRCTL0, RTRCTL\_RMT\_CHNL0);

DShotRMT dshot1(RTRCTL1, RTRCTL\_RMT\_CHNL1);

DShotRMT dshot2(RTRCTL2, RTRCTL\_RMT\_CHNL2);

DShotRMT dshot3(RTRCTL3, RTRCTL\_RMT\_CHNL3);

//Throttle

uint16\_t throttle[4] ;

//Time when program start

unsigned long start\_ms = 0;

//Orientation values

//GYRO

//raw

float rateRoll, ratePitch, rateYaw;

//calibration values

float rollCalibration, pitchCalibration, yawCalibration;

int rateCalibrationNumber;

//Accelerometer

float AccX, AccY, AccZ;

float angleRoll, anglePitch;

//Kalman filter

float KalmanAngleRoll = 0, KalmanUncertaintyAngleRoll = 2\*2;

float KalmanAnglePitch = 0, KalmanUncertaintyAnglePitch = 2\*2;

float Kalman1DOutput[] = {0,0};

//BARO

//Presure

long Presure = 0;

long Temperature = 0;

//BARO calibration values

short AC1, AC2, AC3;

unsigned short AC4, AC5, AC6;

short \_B1, \_B2;

short MB, MC, MD;

long rawPresure = 0;

long rawTemperature = 0;

bool itsRightSequation[4]{

  false,

  false,

  false,

  false

};

// Atitude variables

long AtitudeMm[]{0,0,0};

VL53L1X TOF400f;

// PID by Rate //

float DesiredRatePitch;

float DesiredRateRoll;

float DesiredRateYaw;

float InputRoll, InputPitch, InputYaw, InputThrottle = 0;

float PIDReturn[] = {0,0,0};

///  rate   ///////  rate  /////

///  rate   P  I  D  rate  /////

///  rate   ///////  rate  /////

//P - param configuration

float PRateRoll = 0.6;

float PRatePitch = 0.6;

float PRateYaw = 2;

// I - param configuration

float IRateRoll = 3.5;

float IRatePitch = 3.5;

float IRateYaw = 12;

// D - param configuration

float DRateRoll = 0.03;

float DRatePitch = 0.03;

float DRateYaw = 0;

///  end    ///////  end   /////

///   end   ///////   end  /////

float MotorInput1 = 0, MotorInput2 = 0, MotorInput3 = 0, MotorInput4 = 0;

// PID by Angle

float DesiredAtitude;

float DesiredPitch;

float DesiredRoll;

float DesiredYaw;

float ErrorRoll, ErrorPitch, ErrorYaw, ErrorAtitude;

//float InputRoll, InputPitch, InputYaw;

float PrevErrorRoll, PrevErrorPitch, PrevErrorYaw, PrevErrorAtitude;

float PrevItermRoll, PrevItermPitch, PrevItermYaw, PrevItermAtitude;

///  angle   ///////  angle  /////

///  angle   P  I  D  angle  /////

///  angle   ///////  angle  /////

//P - param configuration

float PRoll = 0.6;

float PPitch = 0.6;

float PYaw = 2;

float PAtitude = 0.02;

// I - param configuration

float IRoll = 3.5;

float IPitch = 3.5;

float IYaw = 12;

float IAtitude = 3.5;

// D - param configuration

float DRoll = 0.03;

float DPitch = 0.03;

float DYaw = 0;

float DAtitude = 0.02;

///  end    ///////  end   /////

///   end   ///////   end  /////

//Orientation functions

 void upDate\_GYRO(){

  //DigitalLowPass Filter 10Hz

  Wire.beginTransmission(PFER\_ADDRESS1);

  Wire.write(0x1A);

  Wire.write(0x05);

  Wire.endTransmission();

  //Sensivity scale factor 65.5LSB/\*/second

  Wire.beginTransmission(PFER\_ADDRESS1);

  Wire.write(0x1B);

  Wire.write(0x8);

  Wire.endTransmission();

  //Register to start reading from

  Wire.beginTransmission(PFER\_ADDRESS1);

  Wire.write(0x43);

  Wire.endTransmission();

  // READING from Module

  //request 6 times by 8 bits from Gyro

  Wire.requestFrom(PFER\_ADDRESS1,6);

  int16\_t GyroX = Wire.read()<<8 | Wire.read();

   int16\_t GyroY = Wire.read()<<8 | Wire.read();

    int16\_t GyroZ = Wire.read()<<8 | Wire.read();

  //convert LSB to digrees per second

  rateRoll = (float)GyroX/65.5;

   ratePitch = (float)GyroY/65.5;

    rateYaw = (float)GyroZ/65.5;

  // ACCELEROMETER 8g range

  Wire.beginTransmission(PFER\_ADDRESS1);

  Wire.write(0x1C);

  Wire.write(0x10);

  Wire.endTransmission();

  //Register to start reading from

  Wire.beginTransmission(PFER\_ADDRESS1);

  Wire.write(0x3B);

  Wire.endTransmission();

  // READING from Module

  //request 6 times(bytes) by 8 bits from ACCEL

  Wire.requestFrom(PFER\_ADDRESS1,6);

  int16\_t AccXlsb = Wire.read()<<8 | Wire.read();

   int16\_t AccYlsb = Wire.read()<<8 | Wire.read();

    int16\_t AccZlsb = Wire.read()<<8 | Wire.read();

  //lsb to g(9.8 m/s)

  AccX = (float) AccXlsb/4096;

   AccY = (float) AccYlsb/4096;

    AccZ = (float) AccZlsb/4096;

   //Calibration ACCEL

    AccX -= 0.02;

  //  AccY += 0.03;

    AccZ -= 0.31;

  //Calculation of angles

  angleRoll = atan(AccY / sqrt(AccX\*AccX + AccZ\*AccZ))\*1/(3.142/180);

  anglePitch = -atan(AccX / sqrt(AccY\*AccY + AccZ\*AccZ))\*1/(3.142/180);

 }

 void upDate\_BARO(bool isRequestFlag, bool tempORpres) {

  //precision mode

  int mode = 0;

  if (isRequestFlag && !tempORpres){

    Wire.beginTransmission(PFER\_ADDRESS2);

    Wire.write(0xF4);

    Wire.write(0x2E);

    Wire.endTransmission();

    // right sequence implementation

    itsRightSequation[0]=true;}

  else if (isRequestFlag && tempORpres){

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xF4);

  //if (mode==0)Wire.write(0x34);

  Wire.write(0x34);

  //else Serial.println("\t  Chenge precision mode on Baro ");

  Wire.endTransmission();

  // right sequence implementation

  if (itsRightSequation[1]) itsRightSequation[2]=true;

  else Serial.println("\t  Error sequence  ReqPres is caled");

  }

  else if(!isRequestFlag && !tempORpres){

  //here we request transmition of raw presure data from 3 registers (8bits every)

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xF6);

  Wire.endTransmission();

  Wire.requestFrom(PFER\_ADDRESS2, 2);

  rawTemperature = Wire.read()<<8|Wire.read();

  // right sequence implementation

  if (itsRightSequation[0])itsRightSequation[1]=true;

  else Serial.println("\t  Error sequence  GetTemp is caled");}

  else if(!isRequestFlag && tempORpres){

  //here we request transmition of raw presure data from 3 registers (8bits every)

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xF6);

  Wire.endTransmission(false);

  Wire.requestFrom(PFER\_ADDRESS2, 3);

  rawPresure = (Wire.read()<<16|Wire.read()<<8|Wire.read())>>(8-mode);

  // right sequence implementation

  if (itsRightSequation[2])itsRightSequation[3]=true;

  else Serial.println("\t  Error sequence  GetPres is caled");}

 }

 void calculateRTemp() {

  long X1;

  long X2;

  long B5;

  X1 = (rawTemperature \* AC6)\*AC5 / pow(2,15);

  X2 = MC \* pow(2,11)/(X1+MD);

  B5 = X1 + X2;

  Temperature = (B5 + 8) / pow(2,4);

 }

 void calculateRPress() {

  short mode = 0;

  // local calculation variables

  long X1;

  long X2;

  long X3;

  long B3;

  unsigned long B4;

  long B5;

  long B6;

  long B7;

    // calculation for final presure

  X1 = (rawTemperature \* AC6)\*AC5/pow(2,15);

  X2 = MC \* pow(2,11)/(X1+MD);

  B5 = X1 + X2;

  B6 = B5 - 4000;

  X1 = (\_B2\* (B6 \* B6 / pow(2,12) )) / pow(2,11);

  X2 = AC2 \* B6 / pow(2,11);

  X3 = X1 + X2;

  B3 = ((AC1 \* 4 + X3)<<mode + 2) / 4;

  X1 = AC3 \* B6 / pow(2,13);

  X2 = (\_B1 \* (B6 \* B6 / pow(2,12))) / pow(2,16);

  X3 = ((X1 + X2) + 2 ) / pow(2,2);

  B4 = AC4 \* (unsigned long)(X3 + 32768) / pow(2,15);

  B7 = ((unsigned long)rawPresure - B3) \* (50000 >> mode);

  if (B7 < 0x80000000){

    Presure  = (B7 \* 2) / B4;

  }else{

    Presure = (B7 / B4) \* 2;

  }

  X1 = (Presure / pow(2,8))\*(Presure / pow(2,8));

  X1 = (X1 \* 3038) / pow(2,16);

  X2 = (-7357 \* Presure) / pow(2,16);

  Presure = Presure + (X1 + X2 + 3791) / pow(2,4);

 }

//Kalman function

 void kalman\_1d(float KalmanState, float KalmanUncertainty, float KalmanInput, float KalmanMeasurement) {

  KalmanState = KalmanState + 0.004\*KalmanInput;

  KalmanUncertainty = KalmanUncertainty + 0.004 \* 0.004 \* 4 \* 4;

  float KalmanGain = KalmanUncertainty \* 1 / ( 1 \* KalmanUncertainty + 3 \* 3);

  KalmanState = KalmanState + KalmanGain \* (KalmanMeasurement - KalmanState);

  KalmanUncertainty = (1 - KalmanGain) \* KalmanUncertainty;

  Kalman1DOutput[0] = KalmanState;

  Kalman1DOutput[1] = KalmanUncertainty;

 }

// PID functions

  void pid\_equation(float Error, float P, float I, float D, float PrevError, float PrevIterm) {

  float Pterm = P \* Error;

  float Iterm = PrevIterm + I \* (Error + PrevError) \* 0.004/2;

  if (Iterm > 400) Iterm = 400;

  else if ( Iterm < -400) Iterm = -400;

  float Dterm = D \* (Error - PrevError)/0.004;

  float PIDOutput = Pterm + Iterm + Dterm;

  if (PIDOutput>400)PIDOutput  = 400;

  else if (PIDOutput < -400) PIDOutput = -400;

  PIDReturn[0] = PIDOutput;

  PIDReturn[1] = Error;

  PIDReturn[2] = Iterm;

  }

  void rset\_pid(void) {

 PrevErrorRoll = 0; PrevErrorPitch = 0; PrevErrorYaw = 0;

 PrevItermRoll = 0; PrevItermPitch = 0; PrevItermYaw = 0;

  }

void setup() {

//Serial start

  Serial.begin(115200);

  dshot0.begin(DSHOT300,false);

  dshot1.begin(DSHOT300,false);

  dshot2.begin(DSHOT300,false);

  dshot3.begin(DSHOT300,false);

  dshot0.sendThrottleValue(0);

  dshot1.sendThrottleValue(0);

  dshot2.sendThrottleValue(0);

  dshot3.sendThrottleValue(0);

//Wire settup

  Wire.setClock(400000);

  Wire.begin(SDA,SCL);

  delay(250);

//enabling MPU6050

  Wire.beginTransmission(PFER\_ADDRESS1);

  Wire.write(0x6B);

  Wire.write(0x00);

  Wire.endTransmission();

//reading Baro calibration data

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xE0);

  Wire.write(0xB6);

  Wire.endTransmission();

  delay(300);

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xAA);

  Wire.endTransmission();

  Wire.requestFrom(PFER\_ADDRESS2,2);

  AC1 = Wire.read()<<8|Wire.read();

  delay(60);

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xAC);

  Wire.endTransmission();

  Wire.requestFrom(PFER\_ADDRESS2,2);

  AC2 = Wire.read()<<8|Wire.read();

  delay(60);

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xAE);

  Wire.endTransmission();

  Wire.requestFrom(PFER\_ADDRESS2, 2);

  AC3 = Wire.read()<<8|Wire.read();

  delay(60);

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xB0);

  Wire.endTransmission();

  Wire.requestFrom(PFER\_ADDRESS2, 2);

  AC4 = Wire.read()<<8|Wire.read();

  delay(60);

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xB2);

  Wire.endTransmission();

  Wire.requestFrom(PFER\_ADDRESS2, 2);

  AC5 = Wire.read()<<8|Wire.read();

   delay(60);

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xB4);

  Wire.endTransmission();

  Wire.requestFrom(PFER\_ADDRESS2, 2);

  AC6 = Wire.read()<<8|Wire.read();

   delay(60);

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xB6);

  Wire.endTransmission();

  Wire.requestFrom(PFER\_ADDRESS2, 2);

  \_B1 = Wire.read()<<8|Wire.read();

   delay(60);

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xB8);

  Wire.endTransmission();

  Wire.requestFrom(PFER\_ADDRESS2, 2);

  \_B2 = Wire.read()<<8|Wire.read();

   delay(60);

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xBA);

  Wire.endTransmission();

  Wire.requestFrom(PFER\_ADDRESS2, 2);

  MB = Wire.read()<<8|Wire.read();

   delay(60);

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xBC);

  Wire.endTransmission();

  Wire.requestFrom(PFER\_ADDRESS2, 2);

  MC = Wire.read()<<8|Wire.read();

   delay(60);

  Wire.beginTransmission(PFER\_ADDRESS2);

  Wire.write(0xBE);

  Wire.endTransmission();

  Wire.requestFrom(PFER\_ADDRESS2, 2);

  MD = Wire.read()<<8|Wire.read();

   delay(60);

//calibrating Gyro

  for (rateCalibrationNumber = 0; rateCalibrationNumber < 2000; rateCalibrationNumber++){

    upDate\_GYRO();

    rollCalibration += rateRoll;

     pitchCalibration += ratePitch;

      yawCalibration += rateYaw;

    delay(1);

  }

  rollCalibration /= 2000;

    pitchCalibration /= 2000;

      yawCalibration /= 2000;

//enabling TOF400f

  TOF400f.setAddress(PFER\_ADDRESS3);

  TOF400f.setTimeout(500);

  if (!TOF400f.init()) {

    Serial.println("Failed to detect and initialize sensor!");

  }

  TOF400f.setDistanceMode(VL53L1X::Long);

  TOF400f.setMeasurementTimingBudget(50000);

  TOF400f.startContinuous(50);

//end

  dshot0.sendThrottleValue(0);

  dshot1.sendThrottleValue(0);

  dshot2.sendThrottleValue(0);

  dshot3.sendThrottleValue(0);

  pinMode(LED\_PIN, OUTPUT);

  digitalWrite(LED\_PIN, LOW);

  start\_ms = millis();

}

void loop() {

  const auto now = millis();

  // Baro calling

    if((now % 100 - now % 10)/10 >= 0 && (now % 100 - now % 10)/10 < 2 && !itsRightSequation[0]){

        upDate\_BARO(true,false);

    }

    else  if((now % 100 - now % 10)/10 >= 2 && (now % 100 - now % 10)/10 < 4 && !itsRightSequation[1]){

        upDate\_BARO(false,false);

    }

    else  if((now % 100 - now % 10)/10 >= 4 && (now % 100 - now % 10)/10 < 6 && !itsRightSequation[2]){

        upDate\_BARO(true,true);

    }

    else  if((now % 100 - now % 10)/10 >= 6 && (now % 100 - now % 10)/10 < 8 && !itsRightSequation[3]){

        upDate\_BARO(false,true);

    }

    else  if((now % 100 - now % 10)/10 >= 8 && (now % 100 - now % 10)/10 <=9  && itsRightSequation[3]){

        itsRightSequation[0] = false;

        itsRightSequation[1] = false;

        itsRightSequation[2] = false;

        itsRightSequation[3] = false;

    }

  //Unarmed mode

  if (now < 10000 || !ARMED ){

    dshot0.sendThrottleValue(0);

    dshot1.sendThrottleValue(0);

    dshot2.sendThrottleValue(0);

    dshot3.sendThrottleValue(0);

  }else{

    throttle[0] = (uint16\_t) MotorInput1;

    throttle[1] = (uint16\_t) MotorInput2;

    throttle[2] = (uint16\_t) MotorInput3;

    throttle[3] = (uint16\_t) MotorInput4;

  dshot0.sendThrottleValue(throttle[3]);// motor 1 //checkd

  dshot2.sendThrottleValue(throttle[0]);// motor 2 //checkd

  dshot3.sendThrottleValue(throttle[1]);// motor 3 //checkd

  dshot1.sendThrottleValue(throttle[2]);// motor 4 //checkd

  ///MOTTORS///

  //Dch///mot//

  ///0/////1///

  ///2/////2///

  ///3/////3///

  ///1/////4///

  //Atitude mesurement calling

  if (now % 50 < 10){

    AtitudeMm[0] = AtitudeMm[1];

    AtitudeMm[1] = AtitudeMm[2];

    AtitudeMm[2] = (long)TOF400f.read(false);

  DesiredAtitude = 750 ;

  ErrorAtitude = DesiredAtitude - ((AtitudeMm[0] + AtitudeMm[1] + AtitudeMm[2])/3);

  pid\_equation(ErrorAtitude, PAtitude, IAtitude, DAtitude, PrevErrorAtitude, PrevItermAtitude);

       InputThrottle=PIDReturn[0];

       PrevErrorAtitude=PIDReturn[1];

       PrevItermAtitude=PIDReturn[2];

  }

  //Gyro

  if (now % 4 == 0 ){

    upDate\_GYRO();

    rateRoll -= rollCalibration;

    ratePitch -= pitchCalibration;

    rateYaw -= yawCalibration;

  }

  // Kalman calling for Roll

  kalman\_1d(KalmanAngleRoll,KalmanUncertaintyAngleRoll,rateRoll,angleRoll);

  KalmanAngleRoll = Kalman1DOutput[0];

  KalmanUncertaintyAngleRoll = Kalman1DOutput[1];

  // Kalman calling for Pitch

 kalman\_1d(KalmanAnglePitch,KalmanUncertaintyAnglePitch,ratePitch,anglePitch);

 KalmanAnglePitch = Kalman1DOutput[0];

 KalmanUncertaintyAnglePitch = Kalman1DOutput[1];

if (fabs(KalmanAngleRoll) > 40 || fabs(KalmanAnglePitch) > 40){

 ARMED = false;

}

//PID settings

 DesiredPitch = 0.55 ;

 DesiredRoll = 0.2 ;

 DesiredYaw = 0 ;

 ErrorRoll = DesiredRoll - KalmanAngleRoll;

 ErrorPitch = DesiredPitch - KalmanAnglePitch;

 ErrorYaw = 0 ;

  pid\_equation(ErrorRoll, PRoll, IRoll, DRoll, PrevErrorRoll, PrevItermRoll);

       InputRoll=PIDReturn[0];

       PrevErrorRoll=PIDReturn[1];

       PrevItermRoll=PIDReturn[2];

  pid\_equation(ErrorPitch, PPitch, IPitch, DPitch, PrevErrorPitch, PrevItermPitch);

       InputPitch=PIDReturn[0];

       PrevErrorPitch=PIDReturn[1];

       PrevItermPitch=PIDReturn[2];

  pid\_equation(ErrorYaw, PYaw, IYaw, DYaw, PrevErrorYaw, PrevItermYaw);

       InputYaw=PIDReturn[0];

       PrevErrorYaw=PIDReturn[1];

       PrevItermYaw=PIDReturn[2];

  MotorInput1 = 1.024\*(InputThrottle-InputRoll-InputPitch-InputYaw);

  MotorInput2 = 1.024\*(InputThrottle-InputRoll+InputPitch+InputYaw);

  MotorInput3 = 1.024\*(InputThrottle+InputRoll+InputPitch-InputYaw);

  MotorInput4 = 1.024\*(InputThrottle+InputRoll-InputPitch+InputYaw);

Serial.print("\t Att = ");

 Serial.print(  AtitudeMm[2]);

 Serial.print("\t KalmanAngleRoll = ");

 Serial.print(KalmanAngleRoll);

 Serial.print("\t KalmanAnglePitch = ");

 Serial.print(KalmanAnglePitch);

 Serial.print("\t M1 = ");

 Serial.print(  MotorInput1);

 Serial.print("\t M2 = ");

 Serial.print(  MotorInput2);

 Serial.print("\t M3 = ");

 Serial.print(  MotorInput3);

 Serial.print("\t M4 = ");

 Serial.print(  MotorInput4);

 Serial.print("\t Trottle = ");

 Serial.print(  InputThrottle);

 Serial.print("\t now = ");

 Serial.println(  now );

 //calculateRPress();

// Serial.print("\t Press = ");

 //Serial.print(  Presure );

 //calculateRTemp();

 //Serial.print("\t Temp = ");

 //Serial.println(  rawTemperature );

if (MotorInput1>1000)MotorInput1 = 1000;

if (MotorInput2>1000)MotorInput2 = 1000;

if (MotorInput3>1000)MotorInput3 = 1000;

if (MotorInput4>1000)MotorInput4 = 1000;

if (MotorInput1<49)MotorInput1 = 49;

if (MotorInput2<49)MotorInput2 = 49;

if (MotorInput3<49)MotorInput3 = 49;

if (MotorInput4<49)MotorInput4 = 49;

  }

  //dshot 300us between pokets

 // delayMicroseconds(250);

}