Хмельницький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерно-інтегрованих технологій та робототехніки

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Система керування квадрокоптером
Назва теми

КвРАКІТ. 2020037.01.05

Галузь знань 15 «Автоматизація та приладобудування»
Шифр, назва

Спеціальність 151 «Автоматизація та комп'ютерно-інтегровані технології»
Шифр, назва

Освітня програма «Автоматизація та комп'ютерно-інтегровані технології»
Назва

Виконав:

студент III курсу, група АКІТс-20-1 _____ Владислав ДЗИСЬ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник _____ Денис МАКАРИШКІН
Підпис, дата Ім'я, ПРІЗВИЩЕ

Нормоконтролер _____ Людмила КОРЕЦЬКА
Підпис, дата Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
зав. кафедри автоматизації
та комп'ютерно-інтегрованих
технологій та робототехніки _____ Валерій МАРТИНЮК
Підпис, дата Ім'я, ПРІЗВИЩЕ

«16» червня 2023 р.

Хмельницький 2023

Хмельницький національний університет

Факультет *інформаційних технологій*
Кафедра *автоматизації, комп'ютерно-інтегрованих технологій та робототехніки*
Освітній рівень *перший (бакалаврський)*
Галузь знань *15 – Автоматизація та приладобудування*
Спеціальність *151 – Автоматизація та комп'ютерно-інтегровані технології*
Освітня-професійна програма *Автоматизація та комп'ютерно-інтегровані технології*

ЗАТВЕРДЖУЮ
Зав. кафедрою _____
_____ В. Мартинюк
«01» 02 2023р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дзись Владислав Сергійович

1 Тема роботи: Система керування квадрокоптером
керівник роботи Макаришкін Д.А., к.т.н, доцент
Затверджено наказом по університету від «01» березня 2023р. №5.
2 Строк подання студентом роботи на кафедру:03.06.2023р.
3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування
4 Зміст пояснювальної записки (перелік питань, які потрібно розробити) Вступ. Огляд методів розв'язання поставленої задачі. розробка схемотехнічних рішень. Розробка алгоритму роботи програмного забезпечення. Висновки.
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) 1. Принципова схема пристрою. 2. Функціональна схема. 3. Алгоритм

Завдання отримав _____

Керівник _____

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Антиплагіат	Федула М.В., доцент кафедри АКІТтаР		
Нормоконтроль	Корецька Л.О., доцент кафедри АКІТтаР		

7. Дата видачі завдання «__01__» _____02_____ 2023 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Вибір та затвердження теми кваліфікаційної роботи; розробка завдання на кваліфікаційну роботу; складання календарного графіка виконання кваліфікаційної роботи	01.03.2023	виконано
2 Вивчення предметної області, в якій планується використання системи автоматизації; аналіз вимог до системи автоматизації	15.03.2023	виконано
3 Проектування та розробка загальної архітектури і структури системи автоматизації, інтерфейсу користувача; вибір засобів реалізації системи автоматизації	29.03.2023	виконано
4 Програмна реалізація та тестування системи автоматизації	12.04.2023	виконано
5 Написання тексту пояснювальної записки та розробка графічних матеріалів	19.04.2023	виконано
6 Остаточне коригування кваліфікаційної роботи з урахуванням зауважень керівника; оформлення кваліфікаційної роботи як документа відповідно до вимог	11.04.2023	виконано
7 Отримання супровідних документів (відгуку керівника, рецензії, довідки про перевірку на плагіат); нормоконтроль	30.05.2023	виконано
8 Підготовка до захисту та захист кваліфікаційної роботи	03.06.2023	виконано

Студент _____ Владислав ДЗИСЬ
Підпис Ім'я, прізвище

Керівник роботи _____ Денис МАКАРИШКІН
Підпис Ім'я, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Система керування квадрокоптером».

Автор роботи: Дзись Владислав Сергійович.

Керівник роботи: Макаришкін Денис Анатолійович

Пояснювальна записка: 60 с., 23 рис., 0 табл., 4 дод., 41 джерело.

Графічна частина: 3 креслення.

КВАДРОКОПТЕР, УЛЬТРАЗВУКОВИЙ ДАЛЕКОМІР, РОЗПІЗНАВАННЯ ЖЕСТІВ, ПРОГРАМУВАННЯ КОНТРОЛЕРА.

Метою роботи є розробка мікропроцесорної системи керування квадрокоптером.

Наведено опис безпілотних мультироторних літальних апаратів. Розглянуто базові принципи польоту квадрокоптера. Наведена розробка системи управління ом на базі IMU та а HC -504. Проведена розробка програмного забезпечення для в руки з відеосигналу методами бібліотеки OpenCV. Запропоновано формування сигналів та передача отриманих даних. Описана передача даних через серійний порт на бортовий контролер.

Підпис студента

23.06.2023

Дата

ЗМІСТ

ВСТУП 3
1 БЕЗПІЛОТНІ МУЛЬТИРОТОРНІ ЛІТАЛЬНІ АПАРАТИ 4
1.1 Базові принципи польоту квадрокоптера 4
1.2 ПІД регулятори 8
1.3 Політний контролер 13
1.4 Фільтр Калмана 15
1.5 Технологія комп'ютерного зору 17
1.6 Висновки до першого розділу 23
2 РОЗРОБКА СИСТЕМИ УПРАВЛІННЯ КВАДРОКОПТЕРОМ НА БАЗІ IMU ТА УЛЬТРАЗВУКОВОГО ДАЛЕКОМІРА HC -504 24
2.1 Складання пристрою та код для отримання даних 24
2.2 Перетворення даних 31
2.3 Розробка програмного забезпечення для розпізнавання жестів руки з відеосигналу методами бібліотеки OpenCV 33
2.4 Постановка задачі та проектування програмного забезпечення 35
2.4 Захоплення відеосигналу 36
2.5 Обробка зображення 39
2.5 Виділення контурів 41
2.6 Знаходження дефектів обвідного контуру 45
2.7 Фільтрування крапок дефектів 47
2.8 Обробка результатів 49
2.9 Висновки до другого розділу 50

3 ФОРМУВАННЯ СИГНАЛІВ ТА ПЕРЕДАЧА ОТРИМАНИХ ДАНИХ 51
3.1 Передача даних через серійний порт 51
3.2 Передача даних на бортовий контролер 53
3.3 Висновки до третього розділу 55
ВИСНОВКИ 56
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ 57
ДОДАТОК А Програмна реалізація 61
ДОДАТОК Б Структурна схема 95
ДОДАТОК В Функціональна схема 96
ДОДАТОК Г Алгоритм 97

ВСТУП

Сьогодні безпілотні літальні апарати (БПЛА) — це галузь технологій, що швидко розвивається. Безпілотники виконують широкий спектр військових і цивільних завдань. В останні роки багатороторні дрони (мультикоптери) набули популярності завдяки своїй доступності та простоті маневрування. Мультикоптери використовуються для доставки вантажів, фото- та відеозйомки, проведення пошукових робіт, спостереження за природними явищами тощо.

Цей напрямок активно розвивають як комерційні підприємства, так і любителі. Ця робота описує проект, метою якого є автоматизація системи керування мультикоптером із чотирма роторами, що робить елементи управління інтуїтивно зрозумілими та чутливими. В результаті був розроблений пристрій, здатний передавати всю інформацію, необхідну для польоту мультикоптера «з Землі», лише жестами однієї руки. Пристрій було розроблено незалежно від дизайну до остаточного коду, використовуючи, серед іншого, відкритий код – платформу Arduino та бібліотеку комп'ютерного зору OpenCV. Пристрій є прототипом, але є перспективним для розробки та практичного використання [1-5].

1 БЕЗПІЛОТНІ МУЛЬТИРОТОРНІ ЛІТАЛЬНІ АПАРАТИ

1.1 Базові принципи польоту квадрокоптера

Квадрокоптер (різновид мультикоптера) — літальний апарат, побудований за схемою вертольота з чотирилопатевими гвинтами.

Багатогвинтові вертольоти були розроблені на початку вертольотобудування. Джордж Ботесат у 1922 році створив і випробував один із перших квадрокоптерів, який фактично злетів із землі та залишився в повітрі. Недоліком цих пристроїв є складний редуктор, який передає обертання одного двигуна декільком гвинтам. Винахід хвостового гвинта та літака повороту поклав край цим спробам. Нові розробки почалися в 1950-х роках, але далі прототипу не пішли. Контролер безпілотного квадрокоптера.

У 21 столітті мультикоптери відроджуються як безпілотні літальні апарати. Квадрокоптери часто використовують для аматорського моделювання, завдяки своїй простоті конструкції. Мультикоптери дозволяють дешево проводити аерофотозйомку і відеозйомку - громіздка камера винесена із зони дії пропелерів [1-5].

Квадрокоптери мають чотири рівновіддалені гвинти (на відміну від одно- та двогвинтових машин, які не мають автоматичного повороту). Кожен гвинт приводиться в рух власним двигуном. Половина пропелерів обертається за годинниковою стрілкою, а половина – проти, тому квадрокоптеру не потрібен хвостовий гвинт. Керуйте квадрокоптером, змінюючи швидкість обертання гвинтів. приклад:

– прискорити всі гвинти - прискорити;

– одна сторона прискорює гвинт, інша сторона уповільнює – рух убік;

– прискоріть гвинт за годинниковою стрілкою та сповільніть гвинт проти годинникової стрілки.

Мікропроцесорна система перетворює команди радіоуправління в команди для двигуна. Для забезпечення стабільного висіння мультикоптер повинен бути оснащений трьома гіроскопами для перекату стаціонарного обладнання. Як допоміжний засіб іноді використовується акселерометр, за допомогою якого процесор може встановити абсолютну горизонтальність, і барометр, щоб зафіксувати пристрій на потрібній висоті. Гідролокатор також використовується для автоматичної посадки та утримання на малій висоті, а також для обльоту перешкод.

Сучасні мультикоптери використовують безщіточні електродвигуни та літій-полімерні акумулятори як джерело живлення. Це накладає певні обмеження на його льотно-технічні характеристики: типова маса мультикоптера становить від 1 до 4 кг, час польоту від 10 до 30 хвилин (30-50 хвилин для одного екземпляра). Вантажопідйомність для середніх моделей мультикоптерів вантажопідйомністю від 500 г до 2-3 кг дозволяє піднімати в повітря невеликі фото- або відеокамери. Існують також досить великі моделі мультикоптерів з приблизно 6-8 гвинтами (гексакоптери та октокоптери), здатні піднімати в повітря вантажі до 20-30 кг. Для збільшення вантажопідйомності використовуються коаксіально розташовані несучі ротори, наприклад, у гексакоптера 12 двигунів і 12 гвинтів розташовані попарно на 6 несучих балках. Мультикоптер може літати на швидкості від нуля (фіксоване зависання в точці) до 100-110 км/год. Запаси енергії акумуляторів дозволяють окремим моделям мультикоптерів літати на відстані до 7-12 км, але на практиці дальність (максимальна відстань, яку вони можуть пролетіти і потім повернутися в точку зльоту) зазвичай обмежена прямої видимості (100- для ручного керування 200 метрів) або зони досяжності засобів радіоконтролю та відеозв’язку. При цьому в кращих зразках такого

обладнання використовуються підсилювачі потужності радіосигналу і спрямовані антенні системи, здатні забезпечувати стійкий радіозв'язок і відеозв'язок на відстані до 100 км. Тому час польоту накладає найбільше обмеження на дальність дії мультикоптера.

Квадрокоптер складається з кількох важливих елементів, без більшості з яких неможливий стабільний політ. Перший - це контролер польоту, який обробляє всю інформацію, що надходить, і перетворює її в сигнали для двигунів. Контролери найчастіше отримують вхідну інформацію у вигляді цифрових сигналів з широтно-імпульсною модуляцією. Для повного орієнтування в повітрі достатньо чотирьох каналів: газ, поворот, тангаж і крен. У деяких моделях бортконтролерів також є можливість використовувати режим польоту - тоді у вхідному сигналі з'являється додатковий канал. На вхід контролера також підключаються датчики, що описують положення крил. На основі цих даних контролер автоматично регулює вихідний сигнал.

Після цього сигнал обробляється таким чином, що на виході виходить відповідне значення напруги, пропорційне швидкості кожного з чотирьох гвинтів. З виходу контролера сигнал, такий як широтно-імпульсний модульований сигнал, надходить до так званого ПІД-регулятора, який контролює сигнал високої потужності, який подається на сам двигун [6-8].

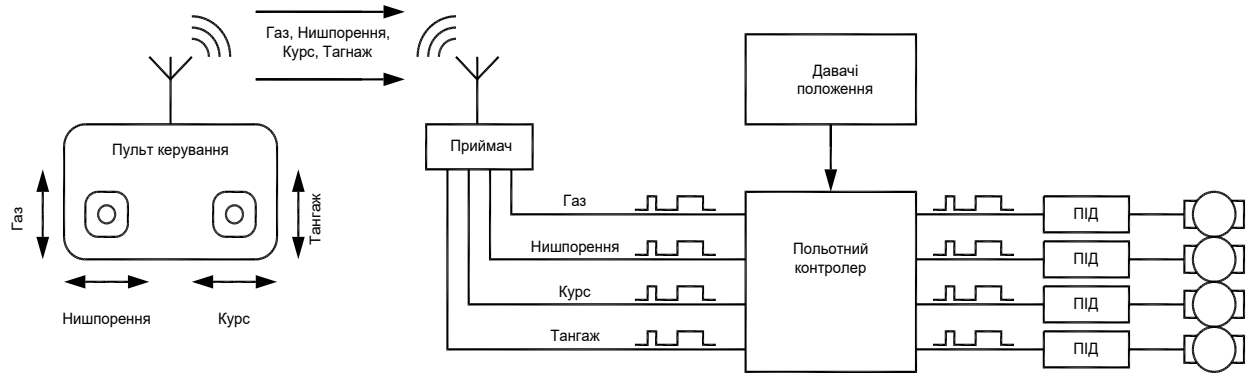


Рисунок 1.1 - Принципова схема квадрокоптера

Аеродинаміка описує основні принципи будь-якого технічного польоту, винятком є квадрокоптери. Три осі обертання абсолютно точно визначають орієнтацію квадрокоптера в просторі і напрямок, в якому він летить. Також напрямок руху не залежить від положення самого квадрокоптера в небі.

Три осі або кути, перелічені вище, зазвичай називаються нахилом, креном і поворотом. Розберемо їх докладніше.

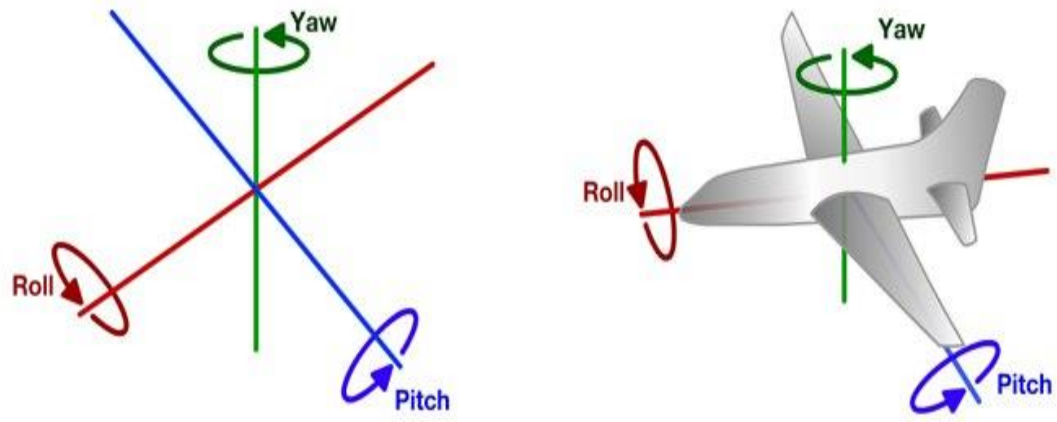


Рисунок 1.2 - Кути Ейлера – крен, тангаж та нишпорення

Тангаж визначається як поворот транспортного засобу навколо поздовжньої осі, тангаж - навколо вертикальної осі, крен - навколо поздовжньої осі.

Якщо розглядати вертоліт, то його головний гвинт впливає на тангаж і крен, хвостовий гвинт компенсує поворотний момент, а поворот залежить від того, як швидко він обертається і де розташований.

З квадрокоптерами все інакше. Гвинтів аж чотири, два з яких обертаються за годинниковою стрілкою, а інші два – проти. Отже, якщо всі пропелери квадрокоптера мають однакову швидкість обертання, то всі параметри будуть скомпенсовані. Коли швидкість обертання одного з гвинтів

квадрокоптера збільшується, баланс порушується. У той же час, якщо частота обертання гвинта, що обертається протилежно, пропорційно зменшується, поворот не зміниться, але зміниться крок або крен.

Якщо одночасно збільшити швидкість обох гвинтів, що обертаються в одну сторону, і зменшити швидкість в іншу, то кут повороту зміниться.

Управління двигунами квадрокоптера, а також частотою обертання пропелерів здійснюється за допомогою пульта дистанційного керування, а сигнали надходять на бортовий комп'ютер квадрокоптера, а необхідні корекції вносяться через гіроскопи, акселерометри тощо. до них додаються.

При проектуванні та створенні квадрокоптера слід виконати всі необхідні розрахунки, щоб знайти оптимальний баланс між вагою пристрою, потужністю встановлених на ньому двигунів і багатьма іншими факторами [9-15]

1.2 ПІД регулятори

Сигнал від контролера польоту надсилається не безпосередньо до двигуна, а до так званого ПІД-регулятора, який забезпечує відповідну напругу для двигуна на основі вхідного сигналу. ПІД-контроль є важливою технологією для керування квадрокоптером.

Наприклад, розглянемо дрон, який потрібно повернути на 60 градусів відносно його центру мас. Рух навколо центру здійснюється подачею за рахунок необхідної кутової швидкості обертання. Щоб точно повернутися до 60 градусів, необхідна кутова швидкість повинна бути застосована за певним законом. У той момент, коли різниця між поточним кутом і бажаним кутом все ще велика, кутова швидкість повинна бути високою. Коли різниця зменшується, швидкість також повинна зменшуватися. Коли різниця дорівнює 0, кутова швидкість також повинна дорівнювати 0. Однак такий закон не може

бути заданим заздалегідь, оскільки квадрокоптер має момент інерції, він може просто злетіти в потрібне положення, і в цьому випадку необхідно забезпечити швидкість у зворотному напрямку. У цьому випадку найбільш стандартним рішенням є повідомити бажану швидкість як різницю між бажаним і поточним положенням, помножену на деяку константу інтегрування. Подібний алгоритм називається ПІД-регулятором. Однак у такого алгоритму є очевидний недолік: у разі необхідного кутового положення, що обертається з певною кутовою швидкістю, дрон ніколи не може наздогнати цю точку. Рано чи пізно кутова швидкість, на яку відкалібрований регулятор, зрівняється з кутовою швидкістю обертання в потрібному положенні. Для якісного вирішення цієї проблеми пропонується до швидкості, яка розраховується як інтегрування кутової похибки за часом, помноженої на деяку невід'ємну константу, додати швидкість від регулятора. У цьому випадку необхідна швидкість буде більшою, а час, протягом якого дрон не зможе зайняти потрібне кутове положення, буде довшим. Алгоритм являє собою ПІД-регулятор. Але цей алгоритм можна вдосконалити. У момент, коли кутова похибка дорівнює 0, БПЛА полетить до цільової позиції під дією сили інерції, якщо до цього його швидкість буде відносно високою. Щоб уникнути цієї ситуації, необхідно гальмувати швидше Коли похибка все ще мала, тобто від ПІД регулятора до необхідної швидкості, додайте диференціальну похибку часу, а потім помножте на позитивне число на продовжувати. Це ПІД-регулятор [6-8].

Пропорційно-інтегральне-диференціальне управління може бути представлене наступною схемою, яка представлена на рисунку 1.3. Блок процесу містить опис об'єкта керування (його модель), сам контролер знаходиться в центральній частині схеми, розраховує сигнал помилки (розбіжності), задає сигнали та вимірювання від об'єкта керування.

Помилки в цій схемі надходять паралельно на всіх трьох блоках одночасно:

– у блоці P відбувається множення відповідного коефіцієнта пропорційної складової величину помилки;

– у блоці I відбувається інтегрування помилки та множення отриманої величини на коефіцієнт K_i;

– у блоці D відбувається диференціювання помилки з множенням на коефіцієнт K_d;

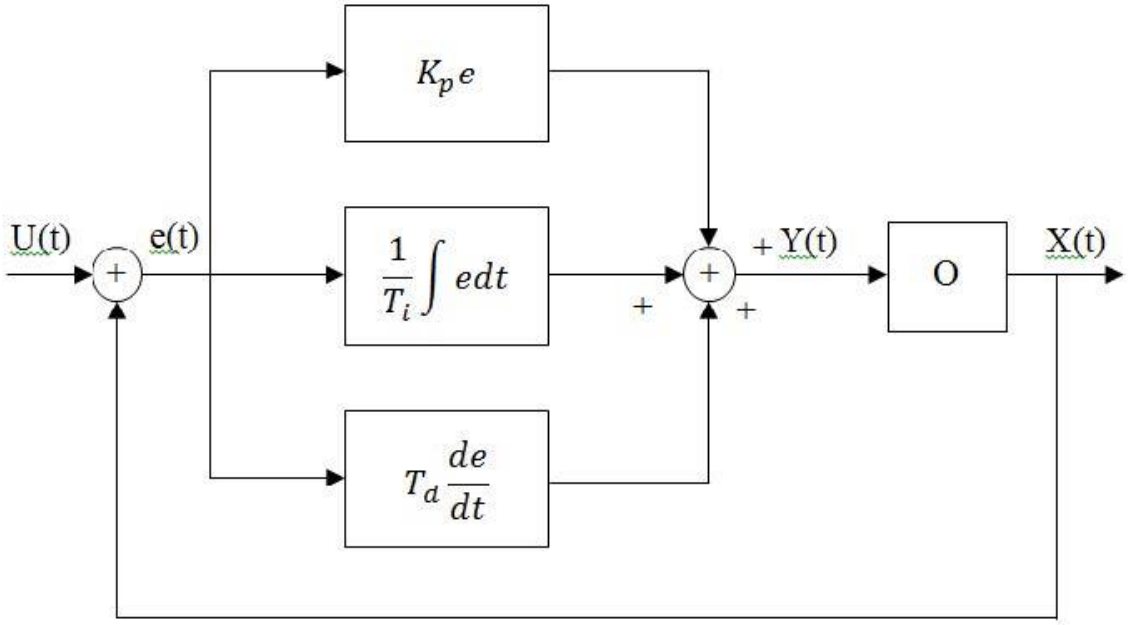


Рисунок 1.3 - Принципова схема ПІД-регулятора

Рівняння ПІД регулятора:

$$u(t) = P + I + D = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

u(t) - вихідна величина регулятора;

e(t) -сигнал неузгодженості (помилка);

P, I, D – пропорційна, інтегральна та диференційна складові;

K_p, K_i, K_d- Коефіцієнти ПІД регулятора;

Поширені також такі модифікації рівняння ПІД регулятора:

$$u(t) = K_p(e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt})$$

Коефіцієнти, що входять до виразів, корельовані, але відсутність загальноприйнятої системи параметрів майже завжди призводить до плутанини. Це необхідно враховувати при заміні одного ПІД-регулятора на інший або при налаштуванні параметрів.

Використовуючи перетворення Лапласа при нульових початкових умовах, представимо передаточну функцію ПІД-регулятора в операторній формі:

$$R(s) = K + \frac{1}{K_i s} + K_d s = K\left(1 + \frac{1}{KK_i s} + \frac{K_d s}{K}\right)$$

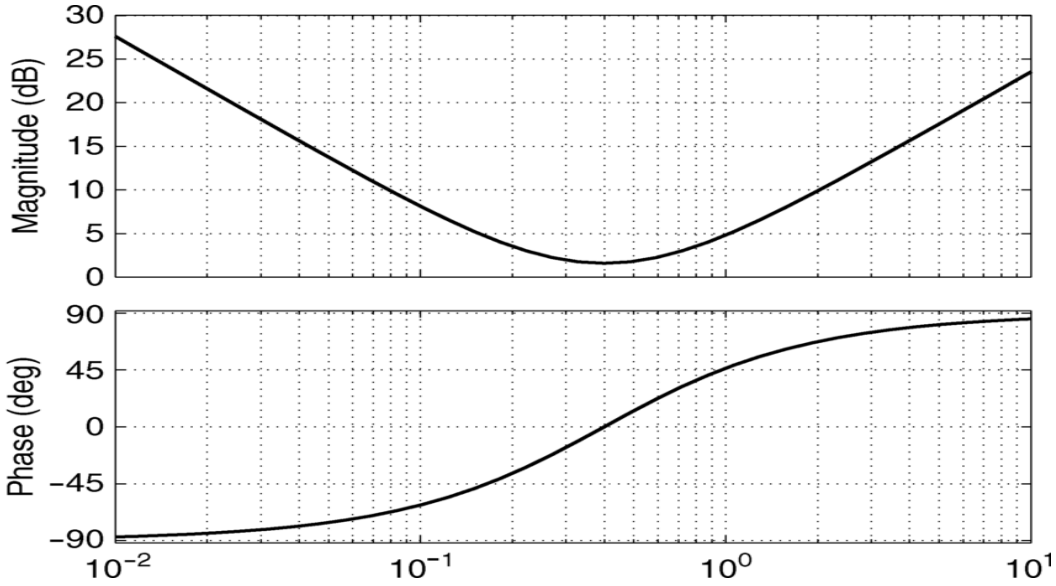


Рисунок 1.4 - АЧХ та ФЧХ ПІД-регулятора

На рисунку 1.4 наведені амплітудно-частотна (частотна) і фазочастотна (ФЧХ) характеристики операторної передавальної функції. В області низьких частот АЧХ і АЧХ визначаються інтегральним членом, в області середньої частоти - пропорційним, у області високих частот - диференціалом.

Тип частотної характеристики та частотна характеристика регулятора визначають його точність і запас стабільності. З рисунку 2 видно, що модуль підсилення регулятора зростає на нижчих частотах (тобто при наближенні до заданого режиму) із зменшенням інтегральної складової, і, таким чином, похибка «е» починає зменшуватися. Коли диференційна складова збільшується, відбувається посилення на високих частотах, що призводить до збільшення шуму вимірювання та зовнішніх завад. Виходячи з цього, диференційна складова використовується лише для покращення форми перехідного процесу в системі, тоді як її фактична реалізація зазвичай включає фільтр високих частот (ФВЧ).

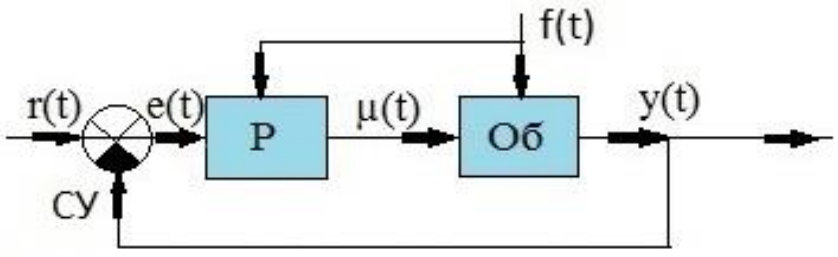


Рисунок 1.5 - ПІД-регулятор у системі з шумом та зовнішніми збуреннями f (t)

Збільшення масштабного коефіцієнта К призводить до збільшення блоку петлевого підсилення схеми керування та підвищення точності в усьому діапазоні частот, але запас фази та підсилення починає зменшуватися, що погіршує надійність і якість регулювання системи погіршується, а як пропорційна складова при подальшому збільшенні К з'являється періодичне коливання (система втрачає стійкість). f вплив вимірювальних перешкод і

шуму зменшується зі збільшенням посилення петлі та масштабного коефіцієнта [16-18].

1.3 Політний контролер

Контролер польоту — основний пульт управління, що забезпечує роботу багатогвинтового літака. Як обчислювальний центр плати керування використовується мікроконтролер, як правило, малопотужний Atmega328 або більш сучасний Atmega2560 або контролер ARM (STM32). До функцій польотного контролера відносяться:

1. Стабілізація апарату у повітрі.
2. Утримання висоти (за допомогою барометра) та позиції (за допомогою GPS).
3. Автоматичний політ за заданими заздалегідь точками (опціонально).
4. Передача на землю поточних параметрів польоту за допомогою модему або Bluetooth (опційно).
5. Забезпечення безпеки польоту (повернення до точки зльоту при втраті сигналу, автопосадка).
6. Підключення додаткової периферії: OSD, світлодіодної індикації та ін.

Кількість функцій залежить від наявності відповідної периферії на мультикоптері, деякі функції можуть бути недоступні в бюджетних контролерах. Контролер польоту ArduPilot Mega, який використовується в цьому проекті, є повноцінним рішенням для безпілотника, який, окрім радіокерованого пілотування, дозволяє автоматично керувати за попередньо створеними маршрутами, тобто польотом «точка-точка», також з борту на

наземну станцію. (мобільний телефон, планшет, ноутбук, DIY) можливість двонаправленої передачі телеметричних даних і входу у вбудовану пам'ять.

Він базується на автопілоті APM 2.x, розробленому спільнотою DIY Drones на основі проекту з відкритим вихідним кодом, який дозволяє перетворити будь-який пристрій на автономний інструмент і ефективно використовувати його не лише для розваг, але й для професійних елементи.

Особливості:

– 3-осьовий гіроскоп, акселерометр, магнітометр та високоточний барометр;

– система стабілізації з можливістю повітряної акробатики;

– утримання позиції по GPS, політ по точках та повернення на точку старту;

– можливість використання інфрачервоного датчика для обходу перешкод;

– підтримка ультразвукового датчика (s onar sensor) для автоматичного зльоту та посадки;

– автоматичне слідування маршрутними точками;

– управління двигунами за допомогою ШІМ (PWM) з використанням дешевих регуляторів швидкості (ESC);

– власна система стабілізації камери (функція контролера підвісу);

– радіозв'язок та телеметрія з борту;

– підтримка датчика рівня заряду батареї;

– світлова індикація, що настроюється при польотах;

– сумісний з багатьма радіокерованими приймачами PWM та PPM сигналів;

– передача у реальному часі телеметричних даних;

– підтримка телеметрії OSD (накладення на відеопередачу телемеричних даних) використовуючи протокол MAVLINK;

– конфігурація точок польоту за допомогою Google Maps;

– бортова флеш пам'ять 16Мбіт для автоматичної реєстрації даних;

– цифровий компас працює на HMC5883L (до версії 2.5.2);

– 6 ступенів свободи в InvenSense акселерометрі, гіроскоп MPU-6000;

– контролер Atmel ATmega2560-16AU та ATMEGA32U-2 чіп для обробки та функції USB;

– можливе завантаження оновлень вбудованого програмного забезпечення та конфігурації.

Нижче наведено деякі режими польоту, доступні контролеру:

– Stabilize - утримання обрію;

– AltHold – утримання висоти;

– Loiter – «замри і тиняйся»;

– Return-to-Launch - повернутися на точку старту;

– Auto - виконання заданого маршруту в автоматичному режимі;

– Acro – акробатика (відключення всіх стабілізаційних систем);

– Circle - обліт по колу заданого радіусу;

– Position - фіксація повітря з ручним газом зльоту;

– Land - автоматична посадка;

– Simple - "легкий" політ.

Контролер передбачає можливість самостійно програмувати режими польоту [18-22].

1.4 Фільтр Калмана

У практичній частині роботи використовується алгоритм фільтрації, який називається фільтром Калмана.

Це один із найефективніших способів фільтрації даних, особливо в робототехніці.

Фільтр Калмана використовує динамічну модель системи (наприклад, фізичні закони руху), відомі ефекти керування та численні безперервні вимірювання для формування оптимальної оцінки стану. Алгоритм складається з двох ітераційних етапів: прогнозування та коригування. Перший обчислює прогноз стану на наступний момент (з урахуванням похибок своїх вимірювань). По-друге, нова інформація від датчиків коригує прогнози (одночасно враховуючи неточності та шум у цій інформації).

Рисунок 1.6 - Схема фільтру Калмана

Рівняння представлені в матричній формі. Для однієї змінної матриця вироджується до скалярних значень.

У цих позначеннях: нижній індекс вказує момент часу: k - поточний, (k - 1) - попередній, а «знак мінус» у верхньому індексі вказує на те, що це медіанне значення прогнозу [23-24].

1.5 Технологія комп'ютерного зору

Комп’ютерний зір – це загальний термін для групи теорій і методів, які використовуються для створення машин, які можуть виявляти, відстежувати та класифікувати об’єкти.

Комп'ютерне зір - дуже молода галузь комп'ютерних технологій, яка динамічно розвивається. Початкові розробки в цій галузі відбулися лише на початку 1970-х років. Основна причина полягає в тому, що методи, які зазвичай використовуються в алгоритмах комп’ютерного зору, дуже вимогливі до обчислень для машин — на початку цифрової ери вам майже завжди доводилося мати справу з великими наборами даних. Крім того, оскільки ця техніка відносно нова і бере свій початок у багатьох інших галузях інформатики, не існує стандартного формулювання проблеми комп’ютерного зору та стандартного вирішення проблеми. Методи й алгоритми комп’ютерного зору сильно залежать від завдань і можуть сильно відрізнятися один від одного. Комп’ютерне бачення в основному походить із таких галузей, як статистика, методи оптимізації та обробка сигналів, і тісно пов’язане з ними.

Завдання комп’ютерного зору можна найбільш правильно сформулювати як отримання корисних описів із зображень або послідовностей зображень і формування корисних висновків із цих описів. Критерії та необхідні методи обробки для відбору описів корисності з конкретних завдань [5].

Основна складність техніки комп'ютерного зору і водночас причина необхідності її створення полягає в тому, що будь-яка електронно-обчислювальна машина не має абстрактного класу і може обробляти лише конкретні значення. Комп’ютер не може «розуміти», яка у нього форма, текстура, розмір чи навіть колір, його можна лише «навчити» використовувати

цю інформацію, якимось чином закодувавши її в послідовність нулів і одиниць.

Методи обробки зображень у комп'ютерному зорі. Computer Vision не розглядає методи, що використовуються для створення цифрових зображень, лише описує їх обробку, тому давайте опустимо деталі та зупинимося на пристрої захоплення (цифрова камера), яка використовує світлочутливі елементи для формування зображення (фотографії) або послідовності зображень. зображення реальних предметів (відео). "Зображення" досліджуваного комп'ютера - це певним чином закодований масив чисел.

Рисунок 1.9 - Подання зорової інформації комп'ютером

Кожен піксель цього зображення є елементом двовимірного масиву. Порядковий номер визначає його положення в двовимірному просторі зображення, а значення елемента - його яскравість. Для кольорових зображень кожен піксель повинен бути закодований принаймні трьома числами, як правило, значеннями яскравості для червоного, зеленого та синього компонентів.

Однак широкий спектр методів комп’ютерного зору приймає бінарні зображення (що складаються лише з чорних або білих пікселів) як вхідні дані. На роботі є сенс дотримуватися цих методів.

Ці алгоритми можуть виконувати різні операції - від дуже простих операцій, таких як підрахунок будь-якої функції, до більш складних операцій, пов'язаних з ідентифікацією, визначенням місцезнаходження та контролем об'єктів. Щоб сформувати бінарне зображення B з даних півтонового або кольорового зображення I, ви можете виконати операцію, яка вибирає деяку підмножину пікселів переднього плану зображення. Ці пікселі важливі для вирішення проблеми аналізу зображення. Інші пікселі ігноруються як пікселі фону. Операції вибору пікселів можуть бути простими (наприклад, пороговий оператор, який вибирає пікселі зі значеннями з заданого діапазону інтенсивності або колірного підпростору) або складними алгоритмами класифікації [2].

Будемо вважати, що двійкове зображення сформовано, і вважатимемо це зображення вхідними даними для виконання операцій аналізу зображення. Концепція бінарного зображення показана на рисунку 1.10. Приклади чотирьох бінарних зображень із рукописних символів.

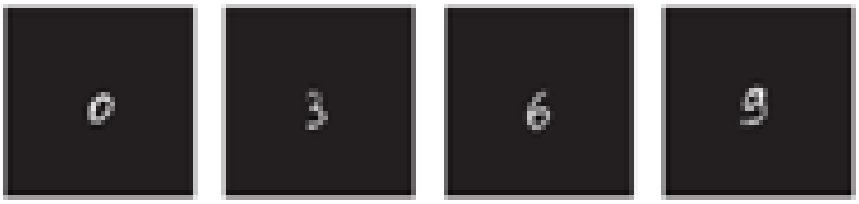


Рисунок 1.10 - Бінарне зображення рукописних символів

Пікселі бінарного зображення B приймають значення 0 або 1. Припустімо, що значення 1 відповідає пікселям переднього плану, а 0 відповідає пікселям фону. Значення пікселя на перетині рядка і стовпця масиву пікселів r позначається як B[r,c]. Зображення M × N складається з M рядків, пронумерованих від 0 до M − 1, і N стовпців, пронумерованих від 0 до N − 1. Таким чином, позначення B [0, 0] відповідає значенню лівого верхнього пікселя зображення, а B [M − 1, N − 1] — значенню правого нижнього

пікселя. У багатьох алгоритмах при обробці пікселів враховуються їх значення і значення сусідніх пікселів. При розгляді сусідніх пікселів часто використовуються два визначення околиці — околиця з чотирьох зв’язків і околиця з восьми зв’язків. Чотиризв'язкове оточення N4 [r, c] пікселя [r, c] складається з пікселів [r - 1, c], [r + 1, c], [r, c - 1] і [r, c + 1]. Їх часто називають північними, південними, західними та східними сусідами відповідно. Восьмизв'язкова околиця N8 [r, c] пікселя [r, c] містить усі пікселі чотиризв'язного околиці, а також діагональні сусідні пікселі [r − 1, c − 1], [r − 1, c + 1], [r + 1, c − 1] та [r + 1, c + 1]. Їх можна відповідно називати північно-західним (northwest), північно-східним (northeast), південно-західним (southwest) та південно-східним сусідами (southeast). Розташування сусідніх пікселів на околицях двох типів демонструється на рис. 1.11.

а *б*

Рисунок 1.11 - Чотиризв'язкові (а) та восьмизв'язкові (б) околиці пікселів

У багатьох алгоритмах околиці з чотирма або вісьмома зв’язками (або визначені іншим чином) часто називають просто околицями пікселів. Аналіз сусідства дозволяє переходити від простору пікселів до простора контурів і виконувати обробку зображення так званими скелетними методами.

Загалом методи виявлення можна умовно розділити на три основні групи:

− скелетні методи;

− методи на основі 3D моделі об'єкта;

− методи на основі 2D моделі об'єкта.

При скелетному підході вивчаються контури контурів: зазвичай є точки з високими значеннями кривизни, такі як кути, нерівності, западини тощо. Для отримання інформації про форму контуру використовуються різні зображення меж об'єктів. У методах на основі 3D-моделі руки представлена як складна 3D поверхня та класифікована за допомогою нейронної мережі. Метод 2D-розпізнавання подібний до попереднього, але використовує 2D-зображення замість 3D-моделей. Кожен метод має переваги та недоліки. Недоліком підходу на основі 3D моделі є ресурсоємність. Побудова 3D-моделі, навчання нейронної мережі та її використання може бути ресурсомістким, і не забувайте, що для використання цього підходу потрібна камера для визначення глибини.

Зупинимося на скелетних (структурних) підходах до обробки зображень з використанням топологічної інформації об’єкта. Ключовим поняттям структурного підходу є контур.

Профіль — це зовнішній контур (обрис) суб’єкта чи об’єкта. При виконанні контурного аналізу вважається, що контур містить достатньо інформації про форму об'єкта, незалежно від внутрішніх точок об'єкта. Звичайно, наведені вище положення накладають велике обмеження на обсяг аналізу контурів, що в основному пов’язано з проблемою виділення контурів на зображенні: через таку ж яскравість, як і фон, об’єкт може не мати чітких меж, або це може створювати шум через перешкоди, призводить до невдалого підсвічування контурів; об’єкти, що перекриваються або їх групування, призводить до того, що підсвічування контурів буде неправильним і не відповідає межам.

Однак перехід до розгляду лише контурів об’єктів дозволяє уникнути простору зображення до контурного простору, що значно зменшує алгоритмічну та обчислювальну складність. Таким чином, контурний аналіз

слабкий проти перешкод, і будь-яке перехрестя або лише частково видимі об’єкти можуть призвести до невиявлення або помилкових спрацьовувань, але простота та швидкість контурного аналізу дозволили цьому методу бути досить успішним. Застосований (очевидні об’єкти на контрасті фон і відсутність перешкод).

Після вибору контуру в двійковій формі він буде підданий процедурі скелетонізації (потоншення). Кожен безперервний контур, представлений скелетом, описується як набір безперервних спеціальних точок і так званого ланцюгового коду, який складається з опорної точки, деяких кодів і масиву вказівок від наступної точки до наступної точки. Спеціальними точками є кінцеві точки і точки розгалуження (тріоди), тобто точки, сусіди яких утворюють не менше трьох зв'язаних областей. на рис. На рисунку 1.12 показано зображення з двома внутрішніми петлями, однією кінцевою точкою та трьома тріодами [25-28].

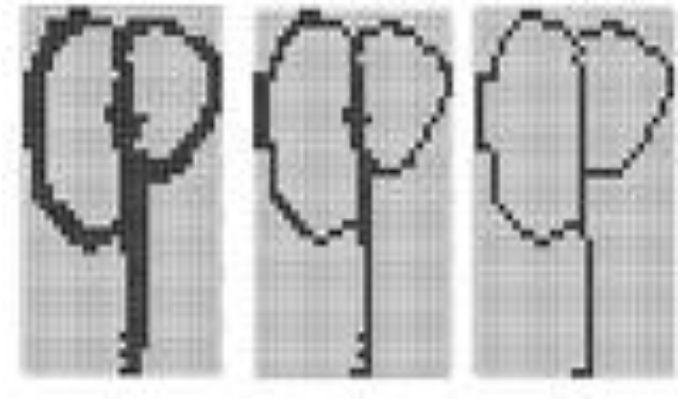


Рисунок 1.12 - Образ, що пройшов процедуру скелетизації

В отриманих описах були зроблені грубі модифікації, які включали видалення заглушок, об'єднання тугих тріодів, руйнування малих внутрішніх схем. Для зовнішніх ланцюгів існує його тип або код топології. Для цього контур записується як набір послідовних спеціальних номерів точок, що відповідають обходу за годинниковою стрілкою. Потім за допомогою перенумерації спробуйте визначити контури за одним основним типом.

1.6 Висновки до першого розділу

У розділі проведно дослідження безпілотних мультироторних літальних апаратів. Розглянуто базові принципи польоту квадрокоптера. Встановлено що для покращення роботи польотного контролера доцільне використанян ПІД регуляторі. Навдено опис польотного контролера.

2 РОЗРОБКА СИСТЕМИ УПРАВЛІННЯ КВАДРОКОПТЕРОМ НА БАЗІ IMU ТА УЛЬТРАЗВУКОВОГО ДАЛЕКОМІРА HC -504

2.1 Складання пристрою та код для отримання даних

Описаний в роботі проект виник з ідеї створення нового методу управління безпілотними дронами. Найпоширенішим способом управління сьогодні є пульт з декількома джойстиками і перемикачами, який передає всі канали управління на бортовий контролер. Хоча квадрокоптер здатний стабілізувати себе в повітрі, керування ним є нетривіальним завданням, яке потребує багатьох годин практики. Ідея проекту полягала в тому, щоб перенести всі елементи управління на інтуїтивно зрозумілу основу - жест однієї руки. Як було сказано вище, основні канали управління називаються газ, рискання, тангаж і крен. Також в польотному контролері Ardupilot виділено кілька каналів для зміни режимів під час польоту. У цій роботі зосереджується увага на управлінні шістьма каналами керування.

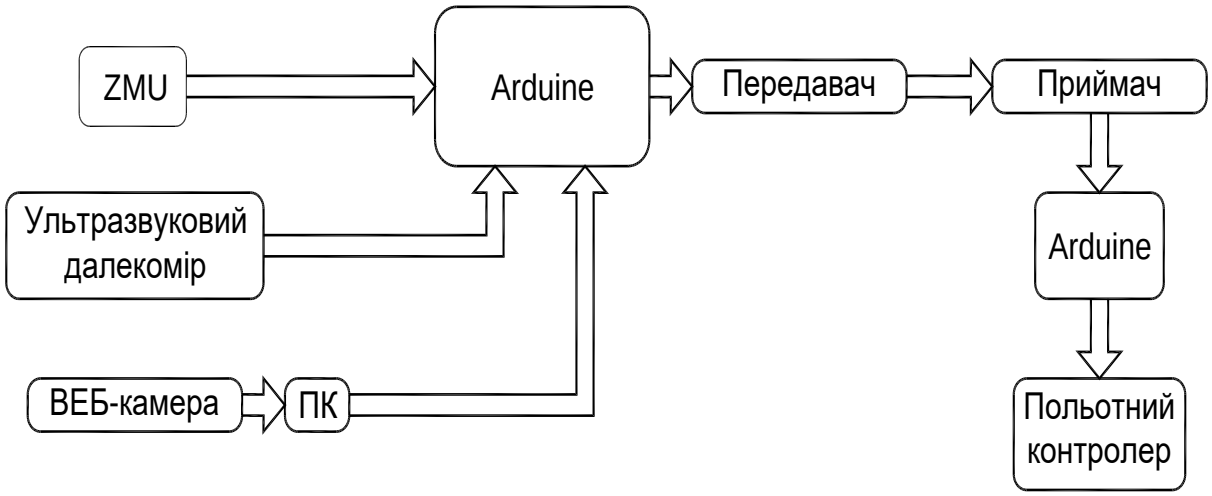


Рисунок 2.1 - Принципова схема пристрою

Принципова схема пристрою представлена на малюнку. Його ключовою частиною є мікроконтролер AVR на платформі Arduino. Він приймає сигнал від датчика, обробляє його за допомогою радіомодуля і відправляє на плату. Газовий канал контролюється ультразвуковим далекоміром - чим більше відстань від нього до руки, тим більше потужність подається на двигун. Так званий інерційний вимірювальний блок (IMU), прикріплений до руки, відповідає за поворот, кут та крен — гіроскоп, акселерометр, компас і барометр, зібрані на одній платі. Плата буде відтворювати положення пристрою в межах інтервалу, визначеного для кожного кута Ейлера.

Нарешті, канали, призначені для зміни шаблону, контролюються за допомогою методів комп'ютерного зору. Камера фіксує сигнал із зображенням жесту руки, який потім аналізується в програмі, написаній на Java. Цю мову було обрано через її кросплатформенність, що дозволяє подальше перенесення програми на пристрої, які є більш портативними, ніж настільні ПК або ноутбуки.

Почнемо збірку пристрою з маніпуляторів, які управляють першими чотирма каналами. Як логічний процесор буде використовуватися мікроконтролер AtmelATMega2560 на основі Arduino. Використання Arduino спрощує розробку та налагодження пристроїв, і в майбутньому ви можете відмовитися від нього та обмежитися мікроконтролерами [29-35].

Як було сказано вище, безпілотні дрони управляються чотирма основними каналами - дросель, поворот, крен і тангаж. Контролер польоту приймає значення кожного каналу як своє вхідне значення як 1-байтове ціле число (від 0 до 255 для дроселя, від -127 до 127 для повороту, кута та тангажу). Отже, перша мета — отримати ці значення як змінні в мікроконтролері. У ескізі ви можете одразу оголосити ці змінні та назвати їх дросель, поворот, кут та крен. Ці змінні зберігають необроблені показання

датчика. Для даних, які надсилаються після обробки, ми оголошуємо ще чотири змінні: sendThrottle, sendYaw, sendPitch, sendRoll.

Значення газу будуть отримані за допомогою ультразвукового далекоміра HC-SR04. Вам потрібно підключити його до контактів 5 В і GND для живлення, а два логічні контакти – до будь-яких цифрових контактів Arduino. Давайте підключимо вихід тригера до цифрового виводу 8 і визначимо його в програмі як TRIGGER_PIN, а відлуння – до виводу 9 і визначимо його як ECHO_PIN. Для роботи з далекоміром буде використовуватися бібліотека NewPing.h. Після імпорту введіть команду ехолота NewPing (TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE), встановивши для MAX_DISTANCE значення 200 – це максимальне вимірювання в сантиметрах. Ця команда створює об’єкт сонара з методами, які можна використовувати для визначення відстані. Тепер у функції циклу ви можете призначити змінній throttle значення sonar.ping, і воно буде пропорційним відстані до перешкоди. Це можна перевірити шляхом виведення змінних змін на монітор послідовного порту. Наприкінці функції циклу також потрібно ввести затримку 5 мс, щоб цикли опитування не «сумувалися» один з одним і не отримували помилкових значень.

У відтвореному масиві значень багато шуму, що може негативно вплинути на результати аналізу. Фільтрування для змінної дросельної заслінки має бути включено в програму. Має сенс використовувати в роботі спрощений фільтр Калмана. Як це працює описано вище. Функція фільтра приймає два параметри - стандартне відхилення значення від математичного очікування і швидкість реакції на зміни. Перший аргумент вважається експериментальним підходом - для цього потрібно створити досить великий масив даних на виході датчика без зміни зовнішніх умов, а потім знайти стандартне відхилення генеральної сукупності. Для цього можна змінити програму, але зручніше це робити в Microsoft Excel за допомогою функції стандартного відхилення. З

вибірки з приблизно 750 елементів ми отримуємо стандартне відхилення 21,47. Запишемо це значення при оголошенні константи на початку коду. Вибір швидкості зміни реакції здійснюється вручну емпіричним шляхом. Потрібно встановити досить мале значення, щоб дрон швидко відчув команду змінити газ, і встановити достатньо велике значення, щоб відфільтрувати зайві значення. Для тестування давайте додамо функцію до нашої програми зі спрощеним алгоритмом фільтра Калмана, який приймає значення та повертає його відфільтрованим. Після отримання змінної дроселя функції циклу ми негайно викликаємо функцію фільтра, параметром функції фільтра є значення дроселя, а потім призначаємо значення фільтра змінній дроселю. Також емпірично встановлено, що найкращим коефіцієнтом для цієї задачі є число 0,02.

Тепер давайте зробимо те саме для решти трьох каналів, підраховуючи та фільтруючи дані від датчика. Показники повороту, крену та тангажу зніматимуться з інерційних вимірювальних пристроїв, включаючи гіроскопи, акселерометри, барометри та компас. Він також вимагає живлення 5 В і має два логічних виходи. Дані передаються через протокол I2C, який підтримує Arduino, тому для передачі великих обсягів даних достатньо двох проводів. Давайте підключимо пристрій до I2C-сумісних портів ArduinoMega, тобто 20 і 21. Вони позначені SDA і SCL.

Щоб використовувати інерційний вимірювальний пристрій, потрібно імпортувати в скетч дві бібліотеки - Wire.h для роботи з протоколом I2C і TroykaIMU.h для роботи з самим пристроєм. Бібліотека написана за принципами об'єктно-орієнтованого програмування, тому для подальшої роботи за межами функцій необхідно створити об'єкт для кожного використовуваного модуля, тобто об'єкти класів Акселерометр, Гіроскоп і Компас. Всі отримані значення будуть проходити через фільтр Magwick, тому створіть об'єкт класу Magwick і для нього. Ми назвемо їх об’єктами «Фільтр»,

«Акселерометр», «Гіроскоп» і «Компас» — акселерометр, гіроскоп і компас відповідно.

Далі ми оголосимо змінні для даних з акселерометра, гіроскопа та компаса — кожен новачок орієнтується в тривимірному просторі, тому надайте контролеру три значення одночасно. Назвемо їх відповідно ax, ay, az, gx, gy, gz, mx, my, mz. Вам також потрібно оголосити змінну fps, яка буде зберігати частоту дискретизації фільтра. Наприкінці кожного циклу він оновлюється, але для першого циклу його потрібно ініціалізувати приблизно 100. Далі ми ініціалізуємо масив постійних значень, отриманих з прикладу калібрувальної матриці на сайті виробника [12]. Це колекції констант калібрування компаса, compassCalibrationBias — це одновимірний масив із трьох елементів, а compassCalibrationMatrix — це двовимірний масив із дев'яти елементів.

У функції налаштування необхідно ініціалізувати всі компоненти інерціального вимірювального пристрою та відкалібрувати компас. На початку функції ми відобразимо повідомлення Begin init на моніторі порту. Потім потрібно запустити метод .begin для об'єктів акселерометра, гіроскопа та компаса. Для об'єкта компаса ми також запускаємо метод calibrateMatrix із параметрами compassCalibrationBias і compassCalibrationMatrix, який ініціює калібрування компаса для цього пристрою. В кінці функції ми відображаємо повідомлення про успішну ініціалізацію на моніторі портів, яке можна побачити лише тоді, коли всі методи виконали свою роботу без помилок.

У функції циклу спочатку визначте частоту обробки фільтра, fps. Для цього у функції після всіх команд, що стосуються ультразвукового далекоміра, потрібно записати в змінну startMillis значення часу, що минув з моменту запуску контролера за допомогою функції millis, яка повертає цей час. функції, але перш ніж відкладати команду, ви повинні спочатку обчислити час, який знадобився для обробки даних, мілісекунди - startMillis, і записати

його в нову змінну deltaMillis. Потім обчислюємо частоту обробки фільтра в герцах, для цього потрібно зворотне deltaMillis помножити на 1000, оскільки час у змінній deltaMillis вимірюється в мілісекундах. Між призначенням startMillis і обчисленням fps знаходиться решта коду, який обробляє інформацію з інерціального вимірювального пристрою.

Він починається зі зчитування даних датчика. Метод readGXYZ об’єкта accel зчитує дані в G з акселерометра та записує їх у змінні ax, ay, az за допомогою вказівників. Метод readRadPerSecXYZ об’єкта гіроскопа зчитує дані в радіанах на секунду з гіроскопа та записує їх у змінні gx, gy, gz. Метод readCalibrateGaussXYZ об’єкта compass зчитує дані Гауса з компаса та записує їх у змінні mx, my, mz.

Після отримання даних бібліотека дозволяє помістити їх в об’єкт фільтра та витягти з нього конкретні значення повороту, крену та тангажу. Спочатку вам потрібно оновити коефіцієнти фільтра - це робиться за допомогою методу setKoeff відповідного об'єкта. Метод приймає два параметри - значення fps, розраховане в попередньому циклі, і числовий коефіцієнт BETA, рівний 0,22 у прикладі виробника. Це можна змінити емпірично, якщо це необхідно, поза функцією ми будемо використовувати команду #define, щоб записати це число в константу BETA.

Далі, використовуючи метод оновлення, ми оновлюємо вхідні дані для фільтра. Параметрами є отримані раніше значення g* a* і z*. Значення yaw, pitch і roll тепер можна отримати з об’єкта фільтра за допомогою методів getYawDeg, getPitchDeg і getRollDeg і записати у відповідні змінні yaw, pitch і angle. Після виведення на послідовний монітор можна побачити наступний вміст (рис.2.2).

Є ще два аспекти, які необхідно враховувати перед перетворенням даних у форму, необхідну диспетчеру польотів. По-перше, датчики не дають

правильних показань протягом певного періоду часу після запуску контролера, і неприйнятно надсилати такі дані на контролер польоту.

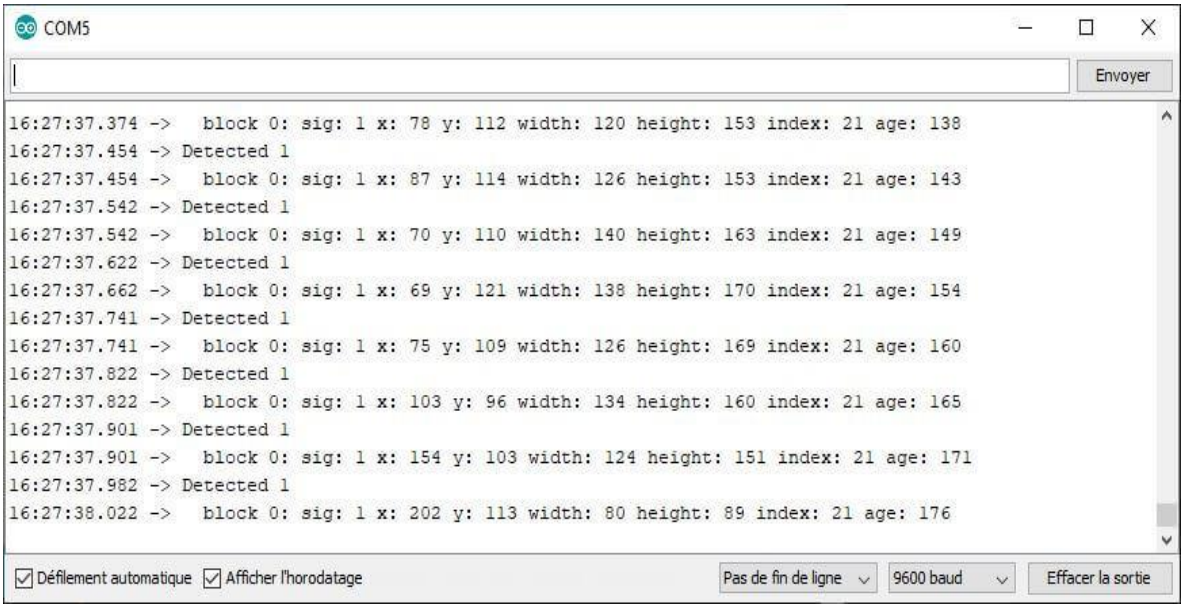


Рисунок 2.2 - Дані з монітора порту

По-друге, розгортка має приймати значення від -127 до 127, де 0 — це положення балансу без коригування розгортки. Проблема полягає в тому, що нуль зонда IMU встановлюється компасом, а відповідний напрямок – на північ. Природно, в реальному пристрої він не повинен бути прив'язаний до будь-якої сторони світла, а нульове значення має бути будь-яким зручним для користувача.

Щоб вирішити дві вищезазначені проблеми, ми додамо до програми кнопку запуску. Він повинен закоротити вихід джерела живлення 5 В і одну з логічних клем. Ми оголошуємо кількість його виходів як константу з іменем BUTTON_PIN, а функція pinMode встановить вихідне значення на INPUT_PULLUP, вхід, що містить вбудований підтягуючий резистор плати Arduino. Давайте створимо функцію, яка містить різні типи натискань кнопок. Цей код використовуватиме лише одне клацання із застосованим захистом від

брязкання шпильки, але функція також підтримує подвійне клацання, натискання й утримання тощо. Усі ці шаблони можна використовувати для подальшого розвитку програми. Ми призначаємо digitalRead(BUTTON_PIN) одній із змінних button1S, а потім викликаємо обробник кнопки. Потім потрібно задати умову, чи потрібно клацати, тобто вираз button1P є істинним. Якщо так, вам слід встановити для цієї змінної значення false і зробити те, що необхідно, тобто «інвертувати» позначку перевірки (тобто встановити для змінної перевірки значення true, якщо попереднє значення було false, і навпаки) і встановити змінну yawCenter yaw. Це нульовий рахунок для пошуку, тепер у програмі напрямок падає не на північ, а градуси, які записуються у змінній yawCenter при натисканні кнопки. Також у блок, який виводить значення на монітор порту, ми додамо умову: якщо контрольна сума false, то на початку кожного рядка друкувати знак оклику – це буде символізувати, що пристрій знаходиться в режимі очікування, а не передача даних. [36-41]

2.2 Перетворення даних

Як було сказано вище, кожне значення повинно займати один байт, газ повинен приймати значення від 0 до 255, yaw, tang і roll - від -127 до 127. Крім того, канал кроку повинен бути інвертованим, тобто він повинен мати негативне значення при нахилі вперед і позитивне значення при нахилі назад при нахилі.

Почнемо з Search Conversion. Як згадувалося раніше, його значення центрується, тобто визначається певний кут yawCenter, і вважається, що він дорівнює нулю. Отже, щоб отримати кінцевий результат, ви повинні використовувати не кут повороту, а центр повороту, який дорівнює нулю, якщо датчик знаходиться в положенні, де натиснуто кнопку запуску, і

змінюється, коли положення відхиляється від нуля. Нас цікавить зміна положення до 40 градусів в кожну сторону, а повертати руку на більші кути фізично важко. Дослідами встановлено, що при повороті ліворуч величина кута додатна, а при повороті праворуч — від'ємна. Отже потрібно скласти таку функцію sendYaw (yaw – yawCenter), щоб

При yaw - yawCenter = 0, sendYaw = 0;

При yaw - yawCenter = 40, sendYaw = -127;

При yaw - yawCenter = -40, sendYaw = 127;

Розв'язавши просту систему рівнянь, з'ясовуємо, що такою функцією є sendYaw = -3,175 (yaw – yawCenter).

Тепер дані можна змінювати як завгодно, але значення може бути більше 127 і менше -127. Ви можете виправити це за допомогою простої умови: якщо sendYaw > 127, тоді sendYaw має примусово встановити 127. Якщо sendYaw < -127, тоді sendYaw має привести до -127.

Враховуючи природу показань датчика, аналогічне перетворення має бути виконано для тангажу та крену.

Для тангажу:

При pitch = 180 або -180, sendPitch = 0;

При pitch = 140, sendPitch = -127;

При pitch = -140, sendPitch = 127;

Вирівняння, що вийшло:

sendPitch = 3,175 * pitch - 571,5, якщо -180 <= pitch <-140;

sendPitch = 3,175 * pitch + 571,5, якщо 140 < pitch < = 180;

sendPitch = 127, якщо 0> pitch> = -140;

sendPitch = -127, якщо 0 <= pitch <= 140;

Для крену:

При roll=180 або -180, sendRoll=0;

При roll = -140, sendRoll = -127;

При roll = 140, sendRoll = 127;

Вирівняння, що вийшло:

sendRoll = -3,175 * roll - 571,5, якщо -180 <= roll <-140;

sendRoll = -3,175 * roll + 571,5, якщо 140 < roll <= 180;

sendRoll = 127, якщо 0 < roll = < 140;

sendRoll = -127, якщо 0 >= roll >= -140;

Для газу потрібно довільно вибрати значення початкового значення, яке буде відповідати максимальному значенню. Також слід враховувати, що датчик не буде видавати значення, рівне нулю, тому слід вибрати іншу нульову точку. Давайте виберемо 2000 як максимальне значення і 400 як мінімальне значення, тоді умова виглядає так:

При throttle = 400, sendThrottle = 0;

При throttle = 2000, sendThrottle = 255;

Тоді ,

sendThrottle = 0.159375 * throttle - 63,75, якщо 400 < throttle < 2000;

sendThrottle = 0, якщо throttle <= 400;

sendThrottle = 255, якщо throttle >= 2000;

Тепер дані перетворені та готові до надсилання на бортовий контролер. Але перед цим слід написати програму для управління п'ятим і шостим каналом - режим літака.

2.3 Розробка програмного забезпечення для розпізнавання жестів руки з відеосигналу методами бібліотеки OpenCV

Завданням цього розділу є створення комп’ютерної програми, що реалізує розпізнавання жестів. Цю програму можна використовувати як один із модулів системи керування БПЛА. Його вихід може активувати додаткові функції та режими дрона.

У контролер, який керує дроном, повідомлення про необхідність перемикання режимів передаються по окремих каналах, які також обробляються через тимчасове опломбування. Таким чином, загальний сигнал складається з декількох сигналів з широтно-імпульсною модуляцією, розділених у часі. Спочатку імпульс передачі, тривалістю якого контролер подає напругу на двигун, а потім імпульс, тривалість якого вказує на активність того чи іншого режиму польоту.

Програма повинна якимось чином розпізнати жести, показані користувачем, повернути результат (конкретний код для кожного очікуваного жесту), а потім передати цей результат програмі, яка аналізує показання датчиків і сигнали, що передаються на квадрокоптер. Завдання досить складне з точки зору продуктивності — зняти жести у відео, а потім проаналізувати відеосигнал за допомогою складних алгоритмів, які потребують великої обчислювальної потужності. Мікроконтролери AVR для цих цілей не підходять, а мобільність персонального комп'ютера занадто низька, щоб реально використовувати пристрій. Тому мобільний телефон на базі Android стає ідеальним вибором: ми можемо створити програму, яка знімає відеосигнал з камери пристрою, обробляє його та передає результат через Bluetooth на головний модуль пристрою. Тому для розробки виберіть мову програмування - Java, яка використовується для розробки додатків Android. Перша версія програми, з метою спрощення налагодження та тестування, була створена як програма на мові JavaFX з графічним інтерфейсом для платформи IBM PC, але перенесення цієї програми на Android в майбутньому не займе багато зусиль і часу. оскільки Java є багатоплатформною мовою програмування. Функціонал першої версії програми буде обмежений підрахунком зігнутих пальців і перевіркою стану кисті (з’єднані чи розведені пальці).

2.4 Постановка задачі та проектування програмного забезпечення

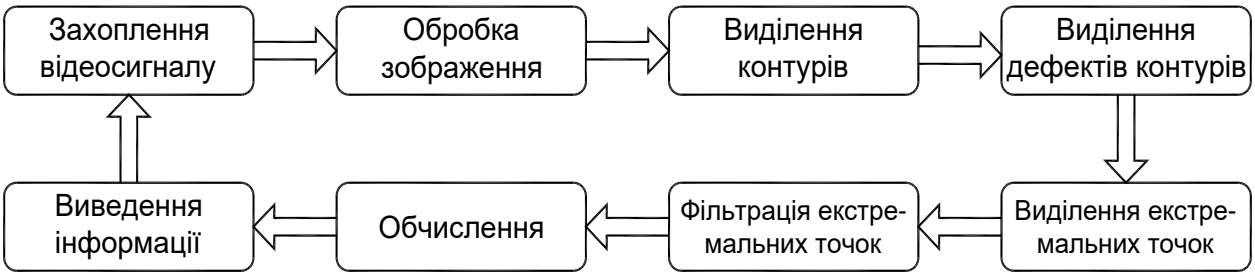


Рисунок 2.3 - Принципова схема програми

Розробку програми має сенс розділити на кілька етапів. Спочатку програма повинна отримати відеосигнал від камери і вибрати з нього зображення. Всі інші дії виконуються над аналізом цього зображення, і після завершення всіх процесів програма повертає результат і робить нове зображення. Наступним етапом після зйомки зображення є його редагування. Програма має максимально спростити завдання аналізу: зображення мають бути максимально стиснутими за роздільною здатністю та не містити зайвих кольорів. У цьому випадку потрібно трохи конвертувати в двійковий, тому що буде використовуватися двійковий метод. Далі програма повинна виділити потрібні контури зображень і знайти їх недоліки за допомогою алгоритмів комп'ютерного зору. Більш детально про це йтиметься у відповідному розділі. Далі необхідно знайти і відфільтрувати так звані крайні точки - в них міститься більшість інформації про об'єкт. Потім, залежно від того, де знаходяться ці точки, вам потрібно зробити необхідні розрахунки і прийти до невтішного висновку, який жест був продемонстрований. Крім того, вся графічна інформація повинна відображатися на екрані. Після всіх цих процесів цикл слід повторити.

Для програми в розробці IntelliJ IDEA був створений проект на платформі JavaFX. JavaFX, по суті, є платформою для створення програм Java

з графічним інтерфейсом. Для створення інтерфейсу існує файл мови розмітки FXML, який «спілкується» з програмою Java за допомогою класу controller.java. Файл FXML описує положення, розмір, графічне оформлення всіх елементів програми, а controller.java пов’язує ці елементи з кодом програми.

2.4 Захоплення відеосигналу

Перше завдання - вивести відеопотік з камери на екран комп'ютера. Це також можна зробити без використання бібліотеки OpenCV, але ми відразу скористаємося її можливостями. У класі Main ми створимо метод start, який буде виконувати операції для початкової підготовки програми. Протягом цього методу ми створюємо об’єкт класу FXMLLoader, призначаємо йому шлях до файлу FXML із тегами та призначаємо його як «батьківський». Відтепер кожен об’єкт у програмі буде «дочірнім» класом (Child) і підпорядкований файлу FXML та його контролеру. Потім створіть об’єкт класу Stage (і підоб’єкт класу Scene, яким є вікно програми в системах Microsoft Windows) і задайте його параметри – назву, розміри, можливість зміни розміру вікна. Потім скористайтеся методом Stage.show, щоб викликати вікно для взаємодії з користувачем.

Після виконання цих дій метод передає управління класу controller.java, який містить основний код програми. Тут потрібно створити об'єкти для всіх елементів програми. Це об’єкти класів Button (графічна кнопка для вмикання та вимкнення камери), ImageView (клас, призначений для відображення зображень на екрані), ScheduledExecutorService (таймер для керування відеопотоками) і VideoCapture (клас OpenCV) захоплення відео. з камери).

Давайте напишемо приватний метод Mat grabFrame, який захоплює відеопотік. Як відомо, слово перед назвою методу вказує на тип даних змінної,

яку повертає метод. В даному випадку це не стандартна змінна, а об'єкт класу Mat, який входить до бібліотеки OpenCV. У нашому випадку це буде кадровий об'єкт - зображення, взяте з відеопотоку. Отже, у приватному методі Mat grabFrame ми створюємо об’єкт frame класу Mat. Використовуючи метод this.capture.isOpened, програма перевіряє, чи походить зображення з порту камери, якщо так, викликає метод this.capture.read(frame), записує кадр в об’єкт і повертає помилку, якщо ні. Далі потрібно перевірити, чи містить фрейм будь-яку інформацію, і якщо так, виконати над ним необхідні перетворення, які ми задокументуємо пізніше. Метод повертає порожній кадр без змін.

Наступний обов’язковий метод обробляє подію, яка виникає після натискання кнопки «Запустити камеру». Назвемо це void startCamera(подія ActionEvent). Як ми бачимо, цей метод приймає натискання кнопки як параметр і не повертає значення.

Після натискання кнопки активується метод і спочатку викликається метод this.capture.open(cameraId), який включає захоплення камери з номером cameraId. Якщо захоплення ввімкнено, функція frameGrabber захоплює зображення з потоку кожні 33 мілісекунди (30 кадрів на секунду) за допомогою вищезгаданого таймера. Тут також здійснюється перенесення зображення з об'єкта класу Mat в об'єкт класу Image. Крім того, метод забезпечує виведення повідомлень про помилки, процес закриття програми, зміну напису на кнопці в залежності від стану камери (ввімкнено або вимкнено).

Тепер потрібно створити розмітку у файлі FXML: текстово або за допомогою утиліти SceneBuilder. В об’єкті програмного вікна класу Scene в операційній системі Microsoft Windows ми створимо зв’язані елементи GridPane і BorderPane. Вони використовуються для зручного розміщення елементів управління в робочій області програми. Вам також потрібно

створити об’єкт класу ImageView для відображення захоплюючих зображень на екрані та об’єкт класу Button для вмикання та вимикання камери.

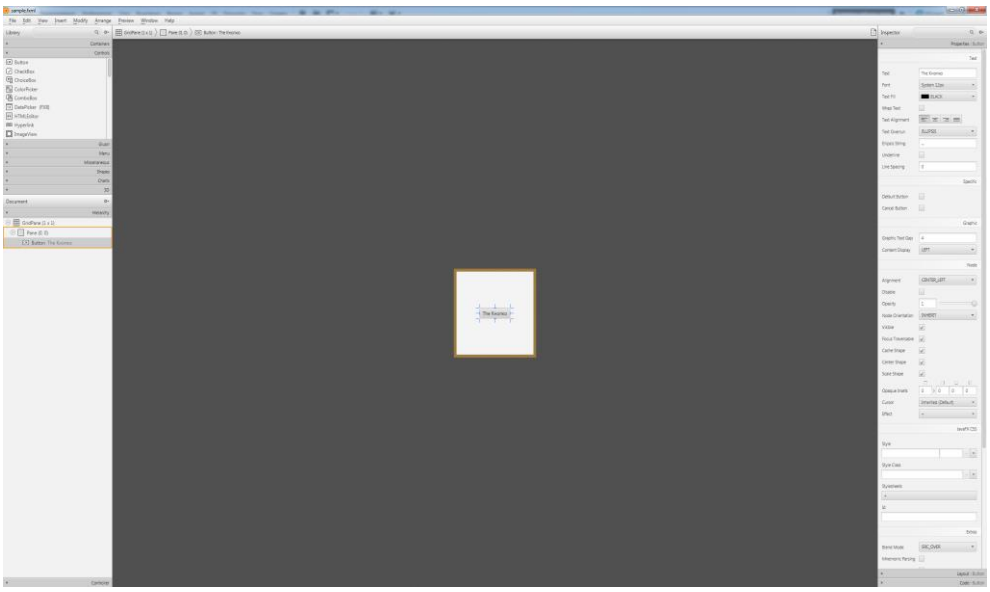


Рисунок 2.4 - Інтерфейс програми в утиліті SceneBuilder

Як показано на рисунку 2.4, процедура - результат на даний момент. Поки що вона могла лише знімати відео з камери та виводити його на екран. Також є кнопка запуску/зупинки камери, яка відповідно вмикає та вимикає камеру.

Рисунок 2.5 - Програма, що захоплює відеосигнал із камери

2.5 Обробка зображення

Перш ніж почати писати алгоритми, які аналізують зображення, вам потрібно буде трохи відредагувати їх, щоб зробити їх зручнішими для користувачів і програм.

По-перше, інформація про колір зображення є надлишковою і не вимагає обчислювальної потужності для її обробки. Крім того, це може виявитися непотрібним або навіть шкідливим, знижуючи ефективність розпізнавання об’єктів. Також, в принципі, багато методів бібліотеки OpenCV можуть обробляти лише двоколірні зображення. Тому в більшості проектів розпізнавання об’єктів зображення перетворюються на двійкові зображення (чорно-білі пікселі без відтінків сірого). Зазвичай це перетворення виконується за допомогою вибірки кольорів — у цьому випадку програма запам’ятовує певний колір і перетворює всі пікселі подібного кольору з білого на чорний, а решту — на чорний. Однак у цьому випадку такий підхід є зайвим, оскільки передбачається, що розпізнаватися будуть лише темні об’єкти на світлому фоні. Отже, як відомо, результатом захоплення відеосигналу є кадровий об’єкт класу Mat. Давайте створимо метод для перетворення його у двійкову форму та назвемо його imageEdit. В результаті всіх перетворень на зображенні воно повинно максимально повно передавати інформацію про положення об'єкта, тобто в ідеалі повністю позначати об'єкт одним кольором, а фон іншим.

Першою необхідною операцією є перетворення зображення на монохромне. Це реалізовано за допомогою методу Imgproc.cvtColor, який приймає три параметри: об’єкт вихідного зображення, об’єкт для запису перетвореного зображення та тип перетворення. В якості перших двох ми виберемо об’єкт рамки з типом перетворення COLOR_BGR2GRAY. Тепер для отримання бінарного зображення необхідне перетворення країв. За це відповідає метод Imgproc.threshold, який приймає п'ять параметрів: перші два

повторюють ті ж параметри для Imgproc.cvtColor, третій - значення порогу вибору кольору, четвертий - максимальне значення змінної і п’ятий – тип трансформації (метод підтримує різні алгоритми, з яких ми обираємо перший, перемальовуючи піксель як білий, якщо його значення яскравості менше порогового значення, інакше як чорний). Поріг не можна вибрати раз і назавжди, тому що він повинен залежати від зовнішніх умов, наприклад освітлення. Тож давайте додамо об’єкт класу Slider до класу контролера та файлу розмітки. Він встановлює максимальні, мінімальні та значення за замовчуванням для повзунка. Тепер можна вставити команду slider.getValue як третій параметр методу перетворення порогового значення та керувати пороговим значенням безпосередньо з екрана.

Тепер зображення має двійкову форму, але потрібно ще одне перетворення - розмиття, щоб уникнути перешкод. Він генерується методом Imgproc.medianBlur. Подібно до порогового перетворення, давайте створимо повзунок, який керуватиме глибиною розмиття, і емпірично виберемо найкращий тип алгоритму для завдання.

Рисунок 2.6 - Зовнішній вигляд програми після редагування зображення

2.5 Виділення контурів

Після того, як програма згенерує якісне бінарне зображення руки, можна приступати до аналізу зображення. Тепер для комп’ютера це двійковий масив, у якому чорні пікселі можна вважати справжніми, а білі пікселі — фальшивими. Наше завдання — проаналізувати інформацію, що міститься в цьому масиві, і на її основі прийняти рішення. Алгоритм має складатися з двох частин: перша частина якимось чином структурує інформацію, що міститься на зображенні, представляючи всі корисні частини інформації окремими фрагментами. Друга частина порівнює ці дані з деякими константами, заданими програмістом. Якщо оброблені дані відповідають одній із констант, програма повертає якийсь результат, інакше – повідомлення про помилку або відсутній необхідний тип зображення.

Найбільш примітивна реалізація полягає в тому, щоб повністю ігнорувати перший крок, залишаючи дані в масиві двійкових змінних і порівнюючи цей масив з будь-якими попередньо записаними зображеннями жестів у програмі. У цьому випадку використаний метод насправді є технікою кореляції: кожне значення пікселя отриманого зображення множиться на відповідне значення пікселя даного зображення. Ці значення потім підсумовуються, і з цієї суми можна зробити висновки про схожість між двома зображеннями. Цей підхід простий у програмуванні, але вкрай неефективний: через різні варіації отриманих зображень (масштаб, положення рук, індивідуальні особливості кожного користувача) цей метод може давати значні похибки. Тому в цьому проекті ми перейдемо від аналізу пікселів до аналізу контурів.

Варто обговорити, яку інформацію потрібно витягти із зображення для подальшої обробки. Почати слід з найпростіших жестів - коли площина долоні перпендикулярна камері і в якості розрахункових змінних використовується

тільки довжина пальців (натиснутий той чи інший палець чи ні) і кут між пальцями. Щоб отримати всю цю інформацію, достатньо знайти на зображенні кілька точок, які називаються точками екстремуму: кінчик кожного пальця і кожна перетинка між пальцями. У цьому випадку необхідно створити два контури: один точно окреслює руку, а інший з'єднує крайні точки в багатокутнику. У цьому випадку точки на кінчиках пальців — це точки, з яких потрібно будувати контури, а точки між пальцями — як максимальне відхилення одного контуру від іншого. Отже, спочатку потрібно знайти і намалювати ці два контури, а потім оперувати ними. Далі контур, що повторює форму руки, називається внутрішнім (зазначається в програмі як контур), а контур, який з'єднує крайні точки об'єкта в багатокутник, називається контуром (або convexHull). Природно, зображення може мати багато контурів, і всі контури того чи іншого виду об'єднуються в масив, кожен елемент якого, у свою чергу, є масивом точок або іншим типом змінної. Таким чином, як contour, так і convexHull є двовимірними масивами, і кожен окремий контур може бути названий за допомогою методу get(i), де і є номером контуру [9].

Доповнимо метод обробки зображень командою, яка знайде контури зображення та виділить їх графічно в робочій області. Спочатку напишемо команду LinkedList<MatOfPoint>contours=new LinkedList<>. Він створює масив об'єктів класу MatOfPoint – типу даних, який зберігає положення багатьох точок у просторі двовимірного зображення. Таким чином, кожен контур буде оброблений програмою як масив точок, що лежать на певній кривій. Потім за допомогою команди contours.clear ми перевіряємо, що заданий масив порожній. Далі скористайтеся методом findContours у класі Imgproc. Використовуючи певний алгоритм, він вибирає контури на білому та чорному порогах і записує їх у масив контурів. Параметри алгоритму визначаються двома властивостями: ми обираємо Imgproc.RETR_EXTERNAL,

щоб сказати програмі ігнорувати внутрішні контури та вибирати лише зовнішні. Друга властивість описує тип використовуваного алгоритму, Imgproc.CHAIN_APPROX_TC89_KCOS було емпірично визнано оптимальним.

Далі давайте відфільтруємо: для аналізу цікавий тільки один об'єкт, тому з усіх виділених контурів можна вибрати тільки об'єкт з найбільшою площею. Отже, давайте створимо новий об’єкт MatOfPoint finalContours, який буде зберігати найбільший контур, який програма намалює, і контури, над якими будуть виконуватися подальші операції. У ньому ми запишемо перший елемент початкового масиву контурів, потім будемо прокручувати кожен наступний елемент контуру, якщо значення площі, яке повертає метод Imgproc.contourArea, перевищує значення площі на один елемент finalContours, тоді finalContours прийме контури цього значення елемента. Таким чином ми отримуємо об’єкт, який містить лише контур навколо найбільшого об’єкта на зображенні.

Нарешті, ми напишемо метод, який малює всі контури в робочій області, Imgproc.drawContours. Його параметри не особливо важливі: це поля, які потрібно намалювати, масив самого контуру, колір і товщина лінії.

Знайти контури - завдання не таке вже й просте. Насправді, бібліотека OpenCV може обробляти багато змінних і масивів, і її різні методи вимагають різних типів. Тому часто потрібно застосовувати дуже значні перетворення, особливо в реалізаціях бібліотеки Java: на відміну від реалізацій в інших мовах програмування, тут багато операцій потрібно виконувати вручну.

Як нагадуємо, контур зображеного об'єкта розглядається програмою як масив класу MatOfPoint, який містить координати кожної точки цього контуру. Однак об'єкти опуклої оболонки задаються зовсім по-іншому - замість контурів метод Imgproc.convexHull знаходить кожну крайню точку зображення як елемент масиву цілих чисел MatOfInt, тоді як для графічних

зображень доступні тільки масиви MatOfPoint. Отже, для єдиного внутрішнього контуру, що залишився після фільтрації, вам потрібно створити елемент масиву за допомогою методу convexHull.add(new MatOfInt) і застосувати сам метод Imgproc.convexHull, який шукає контури навколо внутрішнього контуру, а атрибут методу повинен містять нульові елементи convexHull, виділені за допомогою методу get. В результаті масив convexHull заповнюється контурами.

Тепер конвертуємо його в потрібний формат MatOfPoint. Перетворення спочатку перетворюється з формату бібліотеки OpenCV MatOfInt у звичайний двовимірний масив змінних типу Point, а потім безпосередньо перетворюється у формат MatOfInt. Давайте створимо масив класу Point і назвемо його повними точками. Потім на кожному кроці циклу від нуля до кількості оточуючих контурів ми будемо створювати нову точку в масиві та призначати нове значення за допомогою методу convexHull.get у циклі. Ми позначаємо масив у форматі MatOfPoint як hullmop і перебираємо всі елементи hullpoints для повторного перетворення. Це відбувається за допомогою трьох команд: створюється об’єкт MatOfPoint, потім метод fromArray(hullpoints.get) передає йому значення масиву hullpoints, що відповідає лічильнику, а метод hullmop.add записує цей об’єкт у відповідний масив [5].

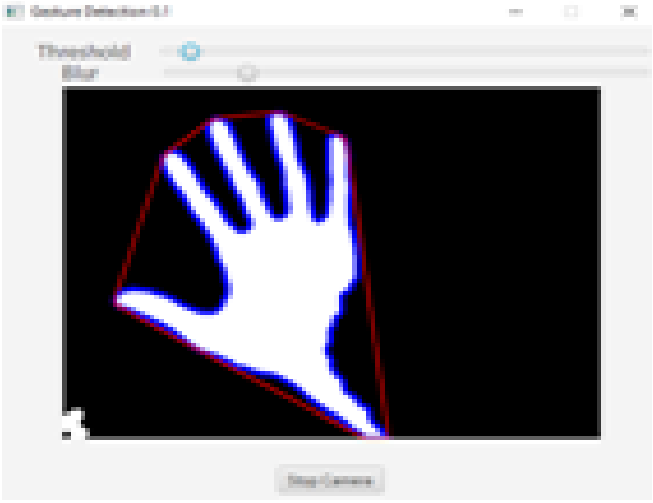


Рисунок 2.7 - Програма після виділення контурів

Тепер контури можна малювати за допомогою методу Imgproc.drawContours. Давайте виберемо інший колір і товщину лінії для його відображення. У підсумку програма виглядає так. В результаті роботи програми зображення має вигляд, показаний на рис. 2.7.

2.6 Знаходження дефектів обвідного контуру

Тепер давайте з’ясуємо недоліки так званих обертових контурів. Давайте створимо ще один масив об’єктів для зберігання інформації, цього разу використовуючи формат MatOfInt4 під назвою convDef. У масиві зберігаються дані, згруповані за чотирма числами в цілочисельному форматі. Цей формат використовується тому, що метод призначає чотири значення для опису кожного дефекту: початок, кінець, дефект, глибина.

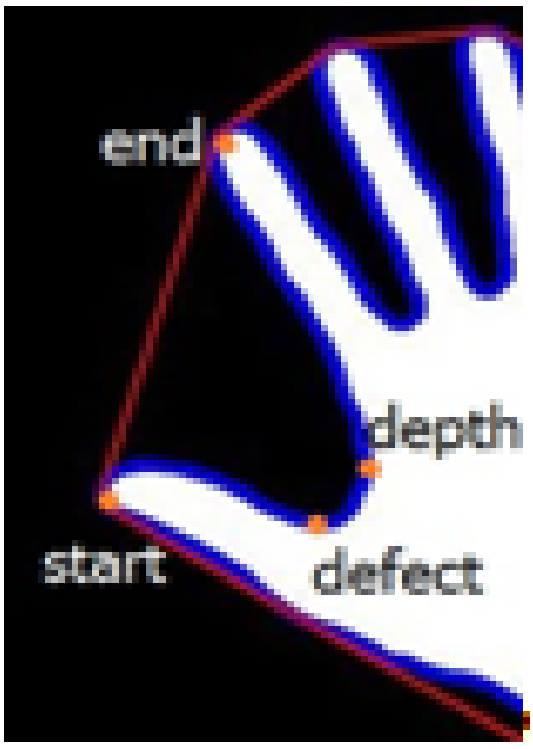


Рисунок 2.8 - Схематичне зображення значень методу, що повертаються Imgproc.convexityDefects

Для повного опису проекту достатньо точок початку, кінця та дефекту. Для зручності ми також створюємо звичайний одновимірний масив типу Integer і називаємо його cdList. За допомогою методу Imgproc.convexityDefects виберемо список дефектів контуру. Цей метод має три властивості: внутрішній, контур, навколишній контур і об’єкт convDef, до якого буде записана інформація. Але формат IntOfMat4 незручний у використанні і не приймається методами малювання, тому ми використовуємо метод convDef.get(0).toList, щоб перенести всю інформацію в звичайний масив даних. Ми також створимо масив cdList, який буде заповнено натуральними числами від нуля до кількості дефектів, помноженої на чотири. Тепер інформація записується наступним чином: в нульовому елементі стоїть число, що вказує на початок першого дефекту, в першому – кінець першого дефекту, у другому – першого дефекту, у третьому – глибину – Один недолік, на четвертому - початок другого недоліку і так далі. Тепер давайте створимо три масиви Point, один з яких зберігатиме всі початкові точки, другий — усі кінцеві точки, а третій — усі дефектні точки.

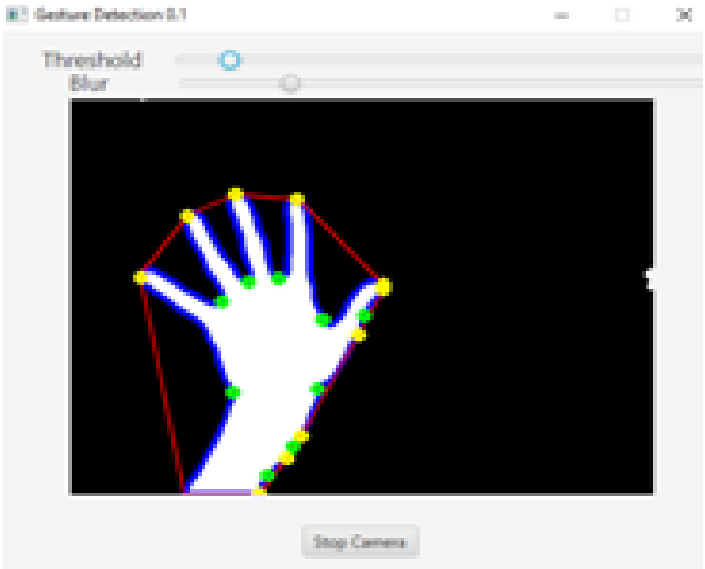


Рисунок 2.9 - Вид роботи програми зі знайденими екстремальними точками

Одразу після цього ви можете створити цикл, щоб намалювати всі точки з масиву на екрані, цього разу використовуючи метод Imgproc.circle, і побачити відповідні результати.

2.7 Фільтрування крапок дефектів

Перед початком аналізу знайдених точок результати потрібно відфільтрувати, оскільки не всі точки несуть корисну інформацію. По-перше, необхідно виключити всі дефектні точки, які утворюють надмірно великі кути із суміжними початковою та кінцевою точками, тому що навіть найбільший кут відхилення між вказівним і великим пальцями не може перевищувати π/2, і ці точки не несуть інформації про положення пальця. .По-друге, початкові точки, які знаходяться занадто близько, потрібно виключити, тому що алгоритм може помилково вибрати кілька сусідніх точок, і один палець буде вважатися двома. По-третє, необхідно виключити точки, які безпосередньо примикають до країв зображення, оскільки, як показує практика, і в цих областях алгоритм часто дає прорахунки. Щоб обробити кожен із трьох масивів точок, ми створимо ще два масиви – по одному для координат x і y. Ми назвали їх startX, startY, endX, endY, defectX, defectY і дали їм тип double. Ви можете використовувати цикл по всіх індексах точок, щоб заповнити їх інформацією про відповідні координати кожної точки. Далі ми будемо фільтрувати за допомогою умовних операторів. Давайте створимо оператор if із трьома умовами фільтра всередині циклу. Якщо координати дефекту та початкова точка з певним індексом відповідають заданим умовам, ці точки будуть записані в новий масив під назвою defectFiltered і startFiltered. Ці умови об’єднуються в логічне І, тобто призначення відбудеться, лише якщо всі три умови виконуються одночасно.

Перша умова полягає в тому, що кут між двома початковими точками і точкою дефекту, розташованої між ними, є відносно малим. Java і OpenCV не знаходять стандартної функції для трьох точок для опису кута, тому вам доведеться створити власну. Відповідно до закону косинусів, якщо відомі три сторони, можна знайти будь-який кут трикутника. Отже, давайте побудуємо уявний трикутник із початкової точки, точки дефекту та кінцевої точки та обчислимо довжини сторін, знаючи координати точки. Потім застосовуємо закон косинусів і зазначаємо значення кута змінного кута. Тепер можна створити першу умову - тільки якщо значення кута менше певного значення, точка буде записана в кінцевий масив. Тепер зупинимося на значенні π/2.

Друга умова полягає в тому, що відстань між початковою і кінцевою точками досить велика. Якщо відстань між двома точками менша за певне значення, відповідну початкову точку потрібно записати в остаточний масив, тому ми будемо використовувати цикл для порівняння кожної початкової [i] точки та кінцевої [i] точки. Якщо сума квадратів координат більша за певне значення, запишіть точку в кінцевий масив.

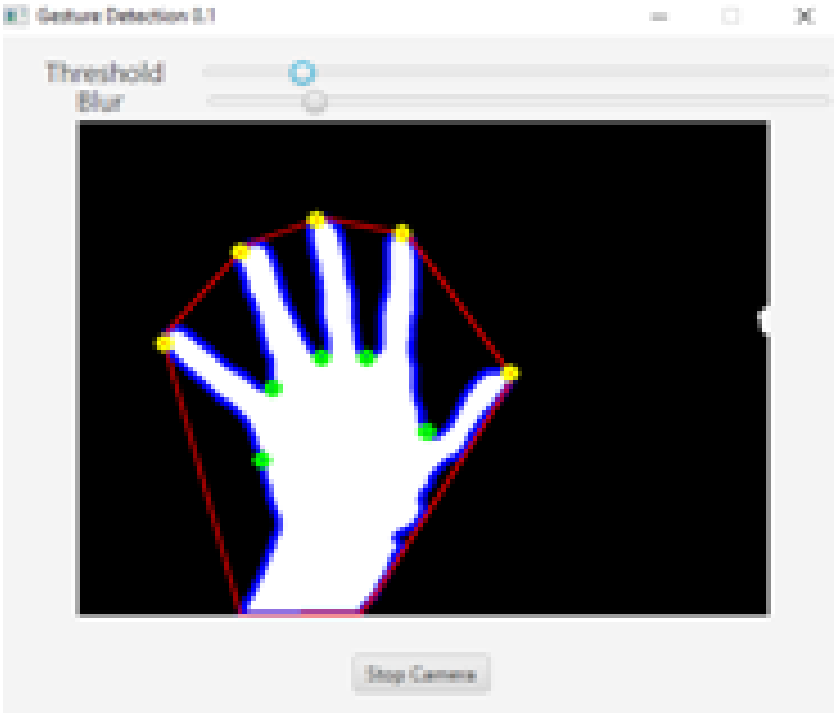


Рисунок 2.10 - Вид роботи програми після фільтрації точок дефектів

Третя умова найпростіша для виконання: кожна координата повинна знаходитися в певному інтервалі, що досягається за допомогою оператора «I» (значення має бути більше нижнього граничного значення і менше верхнього граничного значення). Виберемо значення відступу 30 пікселів для кожного краю екрана.

2.8 Обробка результатів

Отже, тепер ми маємо багато інформації про положення руки: кожен із п’яти пальців позначено точкою в масиві, номер пальця, якщо вважати ліворуч, відповідає номеру елемента масив. Чотири проміжки між пальцями також позначені іншим набором крапок. Крім того, маніпулюючи цією інформацією, можна досягти різних результатів. Наприклад, вдосконалимо код, щоб програма відображала на екрані кількість ненатиснутих пальців, а якщо пальці з’єднані, змінювала колір внутрішнього контуру на червоний. Інформація про кількість пальців вже є в програмі - це розмір масиву startFiltered. Давайте перетворимо його на тип даних String і відобразимо на екрані за допомогою методу Imgproc.putText.

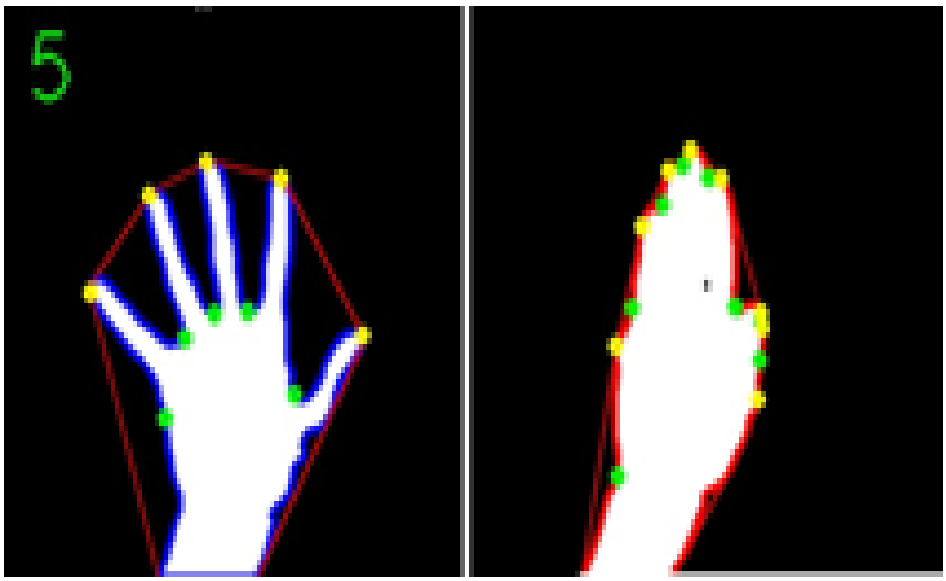


Рисунок 2.11 - Кінцевий варіант роботи програми

Метод обчислення кута також уже створено в процесі фільтрації, тому ми скористаємося ним, тільки цього разу ми підсумуємо всі кути в циклі та отримаємо їхнє середнє арифметичне. Середнє арифметичне тепер можна знову порівняти з порогом, і лише якщо його значення менше порогу, об’єкт класу Scalar, що відповідає за колір внутрішнього контуру, змінює свої властивості. Кінцевий результат показаний на рис. 2.11.

2.9 Висновки до другого розділу

Проведена розробка системи управління квадрокоптером на базі IMU та ультразвукового далекоміра HC -504. Для проведення розробки виконані наступні роботи: складання пристрою та код для отримання даних, перетворення даних, розробка програмного забезпечення для розпізнавання жестів руки з відеосигналу методами бібліотеки OpenCV.

Запропоновано для керування квадрокоптером використовувати жести рук. Для цього досліджено захоплення відеосигналу, обробка зображення, обробка результатів.

3 ФОРМУВАННЯ СИГНАЛІВ ТА ПЕРЕДАЧА ОТРИМАНИХ ДАНИХ

3.1 Передача даних через серійний порт

Після отримання даних виникає проблема їх передачі до диспетчера польотів. Дані з програми Java повинні бути передані на наземне кріплення на базі Arduino і далі надіслані на контролер польоту. У майбутньому, при реалізації проекту як додатку для смартфона, буде можливий бездротовий зв’язок за допомогою протоколу Bluetooth, але зараз більш зручним варіантом для налагодження є передача даних через послідовний порт.

Для використання послідовного порту ми будемо використовувати бібліотеку RXTX. Давайте додамо новий клас і назвемо його Serial.java. Після імпорту всіх необхідних елементів бібліотеки в тіло класу необхідно ініціалізувати об’єкти класів serialPort, input і output, а також оголосити константи: TIME_OUT=2000 (час очікування інформації), D ATA_RATE = 9600 (serial). швидкість передачі порту має відповідати швидкості, вказаній для Arduino), PORT_NAMES (масив імен портів для різних операційних систем). Клас також повинен мати декілька методів.

Перший метод, який називається ініціалізацією, використовується для підготовки портів та ініціалізації змінних і об'єктів. Спочатку він вибирає відповідне ім’я порту: у циклі, який проходить через усі елементи PORT_NAME, порівнює значення елемента масиву з ім’ям порту, отриманим від системи. Ця частина необов’язкова, якщо мультиплатформа не потрібна, для Windows достатньо встановити ім’я порту, наприклад, «COM 4». У методі потрібен метод portId. open відкриває послідовний порт і встановлює параметри DATA_RATE, DATABITS_8, STOPBITS_1 і PARITY_NONE.

Методи getInputStream і getOutputStream використовуються для відкриття вхідних і вихідних потоків. Вхідний потік не буде корисним, оскільки інформація буде передана лише стороні Arduino, але ви можете додати цей метод для подальшого вдосконалення. Також за допомогою методу addEventListener вмикається так зване прослуховування подій - тоді програма буде відповідати на вхідні сигнали.

Наступним кроком буде написання методу закриття послідовного порту. Давайте відкличемо це. Якщо послідовний порт існує, слухач подій має бути закритий за допомогою методу removeEventListener, а порт має бути закритий за допомогою методу close.

Для надсилання даних буде використано метод sendSingleByte. Він приймає один параметр - байт для передачі, назвемо це myByte. спосіб виведення. write Значення записується до послідовного порту.

Коли послідовний порт отримує дані, автоматично викликається метод serialEvent. Для цього в його параметри необхідно внести подію надходження даних SerialPortEvent. У цьому методі метод введення зчитує все значення. Читати, перетворювати з типу байт на тип і зберігати у значенні змінної. Ми також записуємо вихідні дані змінної на консоль, яку ми перевірили раніше, щоб побачити, чи вона знаходиться в діапазоні від 0 до 255.

Тепер ви можете створювати об'єкти цього класу і викликати їх методи. Давайте створимо серійний об’єкт класу Serial і запустимо його метод ініціалізації у функції start. У методі setClosed, який виконується, коли програма закрита, ми напишемо serial. закриття.

Тепер давайте закодуємо інформацію двох змінних, отриманих від камери, у байт. Моліться, цей байт міститиме лише кількість відфільтрованих контурів. Після стиснення байти будуть містити значення {1, 1, 1, 1, 1, 1, 1, 1} (127). Наприкінці функції, яка аналізує зображення, слід додати метод для

надсилання даних - serial. sendSingleByte і вкажіть отриманий байт у його аргументі.

Наступним кроком є оновлення коду Arduino, щоб він міг приймати дані, надіслані на послідовний порт. Давайте додамо змінну типу Byte і назвемо її getByte. На кожній ітерації циклу тепер ви можете додавати послідовну команду. Прочитайте, передумовою є серіалізація. придатний для використання

Відомо, що контролери ArduPilot сприймають сигнали зміни режиму польоту у вигляді сигналів з широтно-імпульсною модуляцією так само, як сигнали газу, повороту, кута та крену. Конкретний режим, який буде перемикатися цим сигналом, можна налаштувати в програмі MissionPlanner. На один канал можна призначити до 256 функцій перемикання режимів, але в більшості випадків один канал на пульті може перемикати два або три режими. У нашому випадку цей варіант оптимальний. Зробимо так, при стисканні пальців перший канал передає байти 11111111, а при розтягуванні - 0000000. Другий канал буде управлятися кількістю пальців - при стисненні пальців нехай буде значення 0, при розведенні трьох - 86, при чотирьох - 172, при чотирьох - 172.

Отриманий байт повинен бути «розкладений» назад на два сигнали з однаковими умовами: якщо отриманий байт дорівнює 127, то змінна режиму 1 типу int дорівнює 255, а режим 2 дорівнює нулю, інакше режим 1 дорівнює нулю і режим 2 Визначається вищевказаним методом.

3.2 Передача даних на бортовий контролер

Тепер ми передаємо дані на інший мікроконтролер, який буде встановлений на квадрокоптері. Для цього будуть використані два модулі nRF24L01. Це модулі радіозв'язку, що працюють на частоті 2,4 ГГц з

вбудованими бібліотеками для платформ Arduino і RaspberryPI. Модулі можуть передавати сигнали зі швидкістю до 2 Мбіт/с на відстані до 100 м. Модуль взаємодіє з контролером за допомогою інтерфейсу SPI.

Крім ніжок живлення, модуль має такі vidjlb:

– SCK (Serial ClocK) – тактування (синхронізація).

– MOSI / MI (Master Out Slave In) - вхід даних .

– MISO / MO (Master In Slave Out) - вихід даних .

– CE/SS - Вибір веденого на шині SPI із кількох пристроїв.

– SCN - вибір режиму прийому/передача, фактично той самий CE.

– IRQ - вихід переривання, найчастіше не використовується. Необхідний для негайної реакції мікроконтролера прийому нового пакета даних.

Бібліотека SPI повинна бути імпортована в код Arduino як на стороні прийому, так і на стороні передачі. h, nRF24L01.h і RF24.h, і навіть ініціалізувати кілька об'єктів і змінних і запустити деякі методи.

Спочатку давайте створимо радіооб’єкт класу RF 24, який приймає як параметр номер контакту Arduino, який використовується для передачі даних. Далі слід оголосити масив адрес типу байтів, який зберігає інформацію про імена так званих «каналів» (каналів, по яких передаються модулі).

У функції налаштування модуль потрібно активувати за допомогою . починати, а потім установити режим підтвердження отримання, якщо потрібно надіслати дані з плати на землю. Потім метод setRetries встановлює кількість спроб передати пакет і час між ними. Метод enableAckPayload дозволяє надсилати дані у відповідь на вхідний сигнал. setPayloadSize встановлює розмір пакету в байтах (6 байт у нашому випадку).

openReadingPipe включає для прослуховування «трубу» з попередньо оголошеним номером Adress . setChannel використовується для вибору каналу в шістнадцятковій системі числення (можна вибрати будь-який, але на деяких

частотах можуть бути шуми) setPALevel встановлює рівень потужності передавача. Аргументами можуть бути RF 24_ PA_MIN , RF 24_PA_LOW , RF 24_PA_HIGH або RF 24_PA_MAX. Метод setDataRate дозволяє вибрати максимальну швидкість. Ми обираємо 250 кбіт/с, тому що максимальна чутливість і діапазон спостерігаються на низьких швидкостях передачі. Метод powerUp сигналізує модулю про ввімкнення. Вам також потрібно написати метод startListening для одержувача та метод stopListening для відправника.

Код для надсилання та отримання такий же простий, як і у випадку з послідовним портом: надсилайте за допомогою методу запису радіооб'єкта, а приймайте за допомогою методу читання. Далі в приймачі кожне отримане значення може бути виведено на вихід, який підтримує широтно-імпульсну модуляцію, і відправлено безпосередньо на вхід контролера польоту.

У додатоку показано повний код програми розпізнавання жестів.

3.3 Висновки до третього розділу

В третьому розділі розглянуто формування сигналів та передача отриманих даних. Описані різні способи виконання поставленого завдання. Визначені напрямки побудови алгоритму. Запропонована програми передачі даних.

ВИСНОВКИ

Результатом цієї роботи став пристрій, здатний передавати сигнали на диспетчер польоту квадрокоптера на основі інформації, отриманої в результаті аналізу положення супротивника в просторі і характеру жестів. Пристрій є прототипом, і його реалізація потребує значного доопрацювання. У міру розвитку на кожному кроці будуть отримані нові знання, і кожен крок алгоритму, звичайно, слід оптимізувати на основі отриманого досвіду.

Перш за все, систему розпізнавання кольорових об’єктів слід додати до програми розпізнавання жестів, щоб підвищити здатність запобігати перешкодам, і в той же час покращити алгоритм фільтрації, щоб зробити керування оперативною пам’яттю більш розумним. У майбутньому програма повинна бути модернізована для розпізнавання нових, більш складних жестів і для оптимізації розпізнавання існуючих жестів.

Пристрій має реальні перспективи для фактичної розробки та використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRISK: Binary Robust Independent Elementary Features. //In Proceedings of the European Conference on Computer Vision (ECCV), 2010.
2. E. Mair, G. D. Hager, D. Burschka, M. Suppa, and G. Hirzinger. Adaptive and generic corner detection based on the accelerated segment test.// In Proceedings of the European Conference on Computer Vision (ECCV), 2010.
3. D. Jeon, M. B. Henry, Y. Kim, I. Lee, Z. Zhang, D. Blaauw, and D. Sylvester. 2014. An Energy Efficient Full-Frame Feature Extraction Accelerator With Shift-Latch FIFO in 28 nm CMOS. //IEEE Journal of Solid-State Circuits 49, 5 (May 2014), 1271–1284.
4. S. Chan, Y. Wong, and J. Daniel, Dense stereo correspondence based on recursive adaptive size multi-windowing //In Proc. Image and Vision Computing New Zealand (IVCNZ'03), volume 1, 256–260, 2003
5. B.D. Lucas, T. Kanade, "An Image Registration Technique with an Application to Stereo Vision", in Proceedings of Image Understanding Workshop, 1981, pp. 121-130.
6. P. Viola and M.J. Jones, «Rapid Object Detection using a Boosted Cascade of Simple Features», proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2001), 2001
7. P. Viola and M.J. Jones, «Robust real-time face detection», International Journal of Computer Vision, vol. 57, no. 2, 2004., pp.137–154
8. М. Шлезингер, В. Главач. Десять лекций по статистическому и структурному распознаванию // Киев, Наукова думка, 2004. 536 с.
9. Fujisada, H., M. Urai, and A. Iwasaki, 2011, Advanced methodology for ASTER DEM generation. IEEE Transactions on Geoscience and Remote Sensing, v. 49, no. 12, p. 5,080-5,091.

10. Light-scattering flow cytometry for identification and characterization of blood microparticles / Konokhova A.I., Yurkin M.A., Moskalensky A.E. et al// J. Biomed. Opt. –2012.– V. 17.– P. 057006.

11. Визильтер Ю. В. Структурная фильтрация цифровых изображений с использованием проективных морфологий // Вестник компьютерных и информационных технологий. 2008. № 3.С. 18–22.

12. Godbole S., Amin A. Mathematical morphology for edge overlap detection for medical images // Real-Time Imaging. 1995. № 1(3). P. 191–201.

13. B. Jähne, H. Scharr, and S. Körkel. Principles of filter design. In Handbook of Computer Vision and Applications. Academic Press, 1999.

14. J. Paumard, E. Aubourg. Adjusting astronomical images using a censored Hausdorff distance // In: Proc. Internat. Conf Image Process., on 26-29 October 3, 1997, 3, 232-235.

15. Bouma H., Dijk J. and van Eekeren A.W.M. Precise local blur estimation based on the first-order derivative // SPIE Defense, Security, and Sensing, 839904- 839904-8, 2012.

16. Deutsch B., Bajramovic F., Denzler J. A comparative evaluation of template and histogram based 2D tracking algorithms // In DAGM-Symposium, 2005. – P. 269–276.

17. Alhichiri H.S., Kamel. M. Multi-resolution image registration using multi-class Hausdorff fraction // Pattern Recognition Letters 23 (2002), pp. 279-286.

18. Alhichiri H.S., Kamel. M. Virtual circles: a new set of features for fast image registration // Pattern Recognition Letters 24 (2003), pp. 1181 -1190.

19. Yongsheng Gao, Maylor K.H. Leung. Line segment Hausdorff distance on face matching // Pattern Recognition 35 (2002), pp. 361-371.

20. Grabner H., Grabner M., Bischof H. Real-time tracking via on-line boosting. — 2006.

21. Sobel I., Feldman G. A 3x3 Isotropic Gradient Operator for Image Processing. — Never published but presented at a talk at the Stanford Artificial Project.

22. Kalal Z., Matas J., Mikolajczyk K. Online learning of robust object detectors during unstable tracking // In International Conference on Computer Vision. — 2009.

23. C++ STM32 Development Environment. — URL: http: //andybrown.me.uk/2015/03/22/stm32dev-windows/ (дата звернення: 14.05.2019).

24. Raspberry Pi compute module 3+. Datasheet. Release 1, january, 2019 https://www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DATA_CM3plus_1p0.pdf. - 20c.

25. ARM Cortex-M4 32bit MCU+FPU STM407F4xx datasheet. D022152 Rev.8 https://www.st.com/resource/en/datasheet/dm00037051.pdf

26. BNO055. Intelligent 9-axis absolute orientation sensor. Datasheet. Rev.1.2. November, 2014. https://cdn-shop.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf

27. Будіщев М. С. Електротехніка, електроніка та мікропроцесорна техніка : Підручник / М. С. Будіщев. – Львів : Афіша, 2001. – 424 с.

28. Електроніка та мікросхемотехніка: Навчальний посібник / За ред. проф. В.Ф. Яковлєва. – К.: Аграрна освіта, 2010. – 329 с.

29. Костін М. О. Теоретичні основи електротехніки [Текст]: підручник у 3 т. / М. О. Костін, О. Г. Шейкіна. – Дніпро: Видво ДНУЗТ, 2006. – Т. 1. – 336 с; 2007.- Т.2.- 276 с; 2011. – Т.3, Ч.1. – 224 с; 2012.– Т.3, Ч.2. – 352 с.

30. Автоматика та електропривод техніки реєстрації інформації [Електронний ресурс] : навч. посіб. / Г. Г. Власюк, В. М. Співак, К. О. Трапезон, В. Б. Швайчен-ко. - Київ : Освіта України, 2010. - 159 с. - Режим доступу: http://ela.kpi.ua/ handle/123456789/19129.

31. Бойко В. І. Мікрокомп'ютерна техніка / В. І. Бойко, А. Т. Нельга. - 2-ге вид. - Київ : Науково-методичний центр вищої освіти, 2008. - 254 с.

32. Експлуатація машин і обладнання: Навчальний посібник / Ружицький М.А., Рябець В.І., Кіяшко В.М. та ін. – К.: Аграрна освіта, 2010. – 617 с.

33. Колонтаєвський Ю. П. Електроніка і мікросхемотехніка : підручник / Ю. П. Колонтаєвський. - Київ : Каравела, 2006. - 384 с.

34. Коруд В.І., Електротехніка: Підручник / В.І. Коруд, О.Є. Гамола, С.М. Малинівський; За заг. ред. В.І. Коруда. – 3-є вид., переробл. і доп. – Львів: Магнолія Плюс, 2006. – 447 с.

35. Воробйова О. М. Технічні засоби автоматизації: навч. посіб. / О. М. Воробйо- ва, Ю. В. Флейта. - Одеса : ОНАЗ ім. О. С. Попова, 2018. - 208 с.

36. Експлуатація машин і обладнання: Навчальний посібник / Ружицький М.А., Рябець В.І., Кіяшко В.М. та ін. – К.: Аграрна освіта, 2010. – 617 с.

37. Загальна електротехніка з основами автоматики: Навчальний посібник / Т.В.Левченко. – К., 2010. – 358 с.

38. Електроніка і мікропроцесорна техніка / Сенько В.І., Лисенко В.П., Юрченко О.М., Лукін В.Є., Руденський А.А. — К. : «Агроосвіта», 2015. — 676 с.

39. Монтаж електрообладнання і систем керування / За заг. ред. проф. Яковлєва В.Ф. – К.: Аграрна освіта, 2009. – 348 с.

40. Механізація, електрифікація та автоматизація сільськогосподарського виробництва : підруч. у 2 т : Т 2 / А.В. Рудь, І.М. Бендера, Д.Г. Войтюк та ін. ; за ред. А.В. Рудя. – К. : Агроосвіта, 2012. – 434 с.; іл.

41. Костинюк Л.Д. Моделювання електроприводів/ Л.Д. Костинюк, В.І. Мороз, Я.С Паранчук.. - Львів: НУ “Львівська політехніка”, 2004. - 404 с.

Додаток А

Програмна реалізація

Main . java

```
package sample;
// імпортування JavaFX -класів
import javafx . application . Application ;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
// імпортування бібліотеки OpenCV
import org.opencv.core.Core;
public class Main extends Application {
@Override
public void start(Stage primaryStage) throws Exception{
// пов'язуємо з FXML- документом
FXMLLoader loader = новий FXMLLoader(getClass().getResource("sample.fxml")); // load the FXML ресурс
Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));
// заголовок вікна
primaryStage.setTitle("Gesture Detection 0.1");
// створюємо вікно з параметрами висоти та ширини
primaryStage.setScene( new Scene(root, 517, 441));
// Заборона зміни розмірів вікна
primaryStage.setResizable( false );
// Створюємо сцену і виводимо на екран
```

```
primaryStage.show();
}
public static void main(String[] args) {
// завантажуємо OpenCV
System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
// запускаємо JavaFX- код
launch(args);
}
}
```

Utils. java

```
package sample;
// імпортування засобу для буферизації зображень
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferByte;
// Імпортування JavaFX- класів
import javafx.application.Platform;
import javafx.beans.property.ObjectProperty;
import javafx.embed.swing.SwingFXUtils;
import javafx.scene.image.Image;
// імпортування бібліотеки OpenCV
import org.opencv.core.Mat;
public final class Utils
{
// Конвертує Mat- об'єкт з OpenCV в Image з JavaFX
public static Image mat2Image(Mat frame)
{
try
{
```

```
return SwingFXUtils.toFXImage(matToBufferedImage(frame), null );
}
catch (Exception e)
{
System.err.println("Cannot convert the Mat obejct: " + e);
return null ;
}
}
// метод для імпортування елементів у "нитку" JavaFX
// взято з відкритих джерел
public static <T> void onFXThread( final ObjectProperty<T> property, final T value)
{
Platform.runLater(() -> {
property.set(value);
});
}
// буферизація зображення
private static BufferedImage matToBufferedImage(Mat original)
{
BufferedImage image = null ;
int width = original.width(), height = original.height(), channels = original.channels();
byte [] sourcePixels = new byte [width * height * channels];
original.get(0, 0, sourcePixels);
if (original.channels() > 1)
{
```

```
image = новий BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR);
}
else
{
image = новий BufferedImage(width, height, BufferedImage.TYPE_BYTE_GRAY);
}
final byte [] targetPixels = ((DataBufferByte) image.getRaster().getDataBuffer()).getData();
System.arraycopy(sourcePixels, 0, targetPixels, 0, sourcePixels.length);
return image;
}
}
```

Sample.FXML

```
<?xml version="1.0" encoding="UTF-8" ?>
<?import javafx.scene.text.* ?>
<?import java.lang.* ?>
<?import javafx.geometry.* ?>
<?import javafx.scene.control.* ?>
<?import javafx.scene.image.* ?>
<?import javafx.scene.layout.* ?>
<?import javafx.geometry.Insets ?>
<?import javafx.scene.control.Button ?>
<?import javafx.scene.control.Slider ?>
<?import javafx.scene.image.ImageView ?>
<?import javafx.scene.layout.BorderPane ?>
<?import javafx.scene.layout.ColumnConstraints ?>
```

```
<?import javafx.scene.layout.GridPane ?>
<?import javafx.scene.layout.RowConstraints ?>
<GridPane alignment="center" hgap="10" maxHeight="600.0" maxWidth="726.0" minHeight="426.0" minWidth="517.0" prefHeight="441.0" prefWidth="517.0" http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.Controller" >
<columnConstraints>
<ColumnConstraints />
</columnConstraints>
<rowConstraints>
<RowConstraints />
</rowConstraints>
<children>
<BorderPane maxHeight="600.0" maxWidth="700.0" minHeight="438.0" minWidth="530.0" prefHeight="438.0" prefWidth="530.0" >
<bottom>
<Button fx:id="button" mnemonicParsing="false" onAction="#startCamera" text="Start Camera" BorderPane.alignment="CENTER" >
<BorderPane.margin>
<Insets bottom="15.0" />
</BorderPane.margin>
</Button>
</bottom>
<center>
<ImageView fx:id="currentFrame" fitHeight="325.0" fitWidth="431.0" pickOnBounds="true" preserveRatio="true" BorderPane.alignment="CENTER" >
<BorderPane.margin>
<Insets right="-300.0" />
```

```
</BorderPane.margin></ImageView>
</center>
<top>
<Slider fx:id="slider" max="300.0" maxWidth="400.0" minWidth="100.0" prefHeight="14.0" prefWidth="100.0" value="150.0" BorderPane.alignment=" TOP
<BorderPane.margin>
<Insets right="5.0" top="10.0" />
</BorderPane.margin></Slider>
</top>
<Left>
<Label alignment="TOP_LEFT" text="Threshold" BorderPane.alignment="TOP_LEFT" >
<font>
<Font size="17.0" />
</font>
<padding>
<Insets left="30.0" top="-20.0" />
</padding>
</Label>
</left>
<right>
<Slider fx:id="sliderBlur" max="20.0" min="2.0" prefHeight="14.0" prefWidth="402.0" value="5.0" BorderPane.alignment="CENTER" >
<padding>
<Insets right="5.0" top="-350.0" />
</padding>
</Slider>
</right>
```

```
</BorderPane>
<Label text="Blur" >
<GridPane.margin>
<Insets left="50.0" top="-365.0" />
</GridPane.margin>
<font>
<Font size="17.0" />
</font>
</Label>
</children>
</GridPane>
```

Controller.java

```
package sample;
// Імпортування JavaFX- класів
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Slider;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
// імпортування бібліотеки OpenCV
import org.opencv.core.*;
import org.opencv.imgproc.Imgproc;
import org.opencv.videoio.VideoCapture;
// імпортування утиліт Java
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
```

```
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
public class Controller {
@FXML
private Button button; // кнопка
@FXML
private Slider slider ; // слайдер вибору порога
@FXML
private Slider sliderBlur; // слайдер розмиття
@FXML
private ImageView currentFrame; //Об'єкт перегляду зображення
private ScheduledExecutorService timer; // Таймер для контролю відеострим
private VideoCapture capture = новий VideoCapture(); // об'єкт OpenCV, захоплюючий відео
private boolean cameraActive = false ; // Прапор , що змінює поведінка кнопки
private static int cameraId = 0; // id використовуваної камери
Serial serial = new Serial(); // Об'єкт для роботи із серійним портом
// подія натискання кнопки
@FXML
protected void startCamera(ActionEvent event)
{
if (! this .cameraActive) //якщо камера в даний момент вимкнена
{
// починаємо захоплення відео
this .capture.open(cameraId);
// відео доступно ?
if ( this .capture.isOpened())
```

```
{
this .cameraActive = true ; //прапорець піднімається
// захоплюємо 30 кадрів на секунду (33 мс)
Runnable frameGrabber = новий Runnable()
{
@Override
public void run()
{
// захоплюємо окремий кадр
Mat frame = grabFrame();
// тепер в зображення його
Image imageToShow = Utils.mat2Image(frame);
updateImageView(currentFrame, imageToShow);
}
};
// Налаштування таймера
this .timer = Executors.newSingleThreadScheduledExecutor();
this .timer.scheduleAtFixedRate(frameGrabber, 0, 33,
TimeUnit.MILLISECONDS);
// напис на кнопці змінюється
this .button.setText("Stop Camera");
serial.initialize();
}
else
{
// помилка
System.err.println("Impossible to open the camera connection...");
}
```

```
}
else
{
// вимикаємо камеру
this .cameraActive = false ;
// міняємо напис на кнопці
this .button.setText("Start Camera");
// зупиняємо таймер
this .stopAcquisition();
}
}
//функція, що захоплює кадр із відео, повертає змінну frame
private Mat grabFrame()
{
// створюємо об'єкт кадр
Mat frame = new Mat();
// оголошуємо змінну, яка відправлятиметься серійним портом
Byte sendByte = 0;
// якщо захоплення працює
if ( this .capture.isOpened())
{
// прочитати поточний кадр
this .capture.read(frame);
// якщо кадр не порожній, обробляємо його
if (!frame.empty())
{
// Переводимо у відтінки сірого
Imgproc.cvtColor(frame, frame, Imgproc.COLOR_BGR2GRAY);
```

```
// перекладаємо в чорне і біле
Imgproc.threshold(frame, frame, slider.getValue(), 5000, 1);
// згладжуємо ( позбавляємось від шумів )
Imgproc.medianBlur(frame, frame, (( int ) sliderBlur.getValue() * 2) - 1);
// оголошуємо масив контурів і очищаємо
LinkedList<MatOfPoint> contours = new LinkedList<>();
contours.clear();
// допоміжний масив
Mat hierarchy = new Mat();
// шукаємо контури
Imgproc.findContours(frame, contours, hierarchy,
Imgproc.RETR_EXTERNAL, Imgproc.CHAIN_APPROX_TC89_KCOS);
// Виділяємо самий великий контур
LinkedList<MatOfPoint> finalContours = new LinkedList<>();
finalContours.addFirst(contours.get(0));
for ( int j = 0; j < contours.size(); j++)
{
if (Imgproc.contourArea(contours.get(j)) >
Imgproc.contourArea(finalContours.getFirst()))
{
finalContours.removeFirst();
finalContours.addFirst(contours.get(j));
}
}
// Знаходимо обвідні контури
LinkedList<MatOfInt> convexHull = new LinkedList<>();
convexHull.add( new MatOfInt());
Imgproc.convexHull(finalContours.get(0), convexHull.get(0));
```

```
// конвертуємо MatOfInt в Point для малювання обведення
// цикл по всім контурам
List<Point[]> hullpoints = новий ArrayList<>();
for ( int j = 0; j < convexHull.size(); j++)
{
Point[] points = new Point[convexHull.get(j).rows()];
// цикл по всім опуклостям
for ( int k = 0; k < convexHull.get(j).rows(); k++)
{
int index2 = ( int ) convexHull.get(j).get(k, 0)[0];
points[k] = новий Point(finalContours.get(j).get(index2, 0)[0],
finalContours.get(j).get(index2, 0)[1]);
}
hullpoints.add(points);
}
// конвертуємо масив Point у MatOfPoint
List<MatOfPoint> hullmop = новий ArrayList<>();
for ( int j = 0; j < hullpoints.size(); j++)
{
MatOfPoint m = новий MatOfPoint();
m.fromArray(hullpoints.get(j));
hullmop.add(m);
}
// шукаємо дефекти контурів
List<Integer> cdList;
LinkedList<MatOfInt4> convDef = new LinkedList<>();
convDef.add( new MatOfInt4());
// функція шукає дефекти
```

```
Imgproc.convexityDefects(finalContours.get(0), convexHull.get(0),
convDef.get(0));
// залишаємо тільки перший
cdList = convDef.get(0).toList();
Point data[] = finalContours.get(0).toArray();
// Розбираємо дефекти на останні точки
Point defect[] = новий Point[20];
Point start[] = новий Point[20];
Point end [] = New Point [20];
int defectCounter = 0;
int defectFilteredCounter = 0;
// переносимо в масив точок
for ( int j = 0; j < cdList.size(); j = j + 4)
{
defect[defectCounter] = data[cdList.get(j + 2)];
end [defectCounter] = data[cdList.get(j + 1)];
start [defectCounter] = data[cdList.get(j)];
defectCounter++;
}
// масиви для відфільтрованих точок
Point defectFiltered[] = new Point[15];
Point startFiltered[] = новий Point[15];
// змінна для кута та її лічильник
double angle;
double angleCount = 0;
// фільтрація
for ( int i = 0; i < defectCounter; i++)
{
```

```
// координати , _ яким вважається кут
double dx1 = Math.abs (start[i].x - defect[i].x);
double dy1 = Math.abs (start[i].y - defect[i].y);
double dx2 = Math.abs (defect[i].x - end[i].x);
double dy2 = Math.abs (defect[i].y - end[i].y);
// Обчислення кута
angle = (dx1*dx2 + dy1*dy2)/Math.sqrt((dx1*dx1 + dy1*dy1)*(dx2*dx2 + dy2*dy2));
// обчислення відстані між відповідними точками end та start
double lenght = Math.sqrt (Math.pow((start[i].x - end[i].x), 2) + Math.pow((start[i].y - end[i].y), 2);
// умова з підібраними значеннями
if ((start[i].y < 450 || start[i].y > 50) && angle > 4.6 && lenght < 5.1)
{
// записуємо крапки в масив
defectFiltered [defectFilteredCounter] = defect [i];
startFiltered [defectFilteredCounter] = start [i];
// служить кількістю відфільтрованих контурів
defectFilteredCounter++;
angleCount = angleCount + angle;
}
}
// виводимо кількість стартів
String numberOfFingers = Integer.toString(defectFilteredCounter);
Point textPoint = новий Point (20, 80);
// Середнє арифметичне кутів
double angleThresh = angleCount/defectFilteredCounter;
// перекладаємо в колір
```

```
Imgproc.cvtColor(frame, frame, Imgproc.COLOR_GRAY2BGR);
// якщо середнє арифметичне кута більше певного значення, колір змінюється
if (angleThresh > 0.75 || defectFilteredCounter <= 5)
{
Imgproc.putText(frame, numberOfFingers, textPoint, 1, 6, новий Scalar(0, 255, 0), 4);
Imgproc.drawContours(frame, finalContours, -1, New Scalar(255, 0, 0), 5);
}
else
{
Imgproc.drawContours(frame, finalContours, -1, New Scalar(0, 0, 255), 5);
}
// обвідний контур
Imgproc.drawContours(frame, hullmop, -1, New Scalar(0, 0, 255), 2);
// точки старт і дефект позначаються колами різних кольорів
for ( int i = 0; i < defectFilteredCounter; i++)
{
Imgproc.circle(frame, defectFiltered [i], 5, новий Scalar(0, 255, 0), 5);
}
for ( int i = 0; i < defectFilteredCounter; i++)
{
Imgproc.circle(frame, startFiltered [i], 5, New Scalar(0, 255, 255), 5);
}
// робимо байт
if (angleThresh > 0.75 || defectFilteredCounter <= 5)
{
sendByte = ( byte ) 127;
```

```
}
if (angleThresh > 0.75)
{
sendByte = ( byte ) defectFilteredCounter;
}
}
}
serial.sendSingleByte(sendByte);
// Повертаємо оброблений кадр
return frame;
}
// Зупинити захоплення і відпустити всі ресурси
private void stopAcquisition()
{
if ( this .timer!= null && ! this .timer.isShutdown())
{
try
{
// зупинити таймер
this .timer.shutdown();
this .timer.awaitTermination(33, TimeUnit.MILLISECONDS);
}
catch (InterruptedException e)
{
System.err.println("Exception in stopping frame capture, trying to release the camera now... " + e);
}
}
```

```
if ( this .capture.isOpened())
{
// відпустити камеру
this .capture.release();
}
}
// оновлюємо {@link ImageView} в основний формат JavaFX
private void updateImageView(ImageView view, Image image)
{
Utils.onFXThread(view.imageProperty(), image);
}
// При закритті програми зупиняємо захоплення:
protected void setClosed()
{
this .stopAcquisition();
serial.close();
}
}
```

Serial.java

```
package sample;
// імпортування засобів введення-виведення
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;
import gnu.io.SerialPortEvent;
import gnu.io.SerialPortEventListener;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Enumeration;
```

```
public class Serial implements SerialPortEventListener
{
// створення об'єктів та констант
SerialPort serialPort;
private InputStream input;
private OutputStream output;
private static final int TIME_OUT = 2000;
private static final int DATA_RATE = 115200;
private static final String PORT_NAMES[] = {
"/dev/tty.usbserial-A9007UX1", // Mac OS X
"/dev/ttyUSB0", // Linux
"COM4", // Windows
};
// ініціалізація
public void initialize() {
CommPortIdentifier portId = null ;
Enumeration portEnum = CommPortIdentifier.getPortIdentifiers();
while (portEnum.hasMoreElements()) {
CommPortIdentifier currPortId = (CommPortIdentifier)
portEnum.nextElement();
for (String portName : PORT_NAMES) {
if (currPortId.getName().equals(portName)) {
portId = currPortId;
break ;
}
}
}
try {
```

```
// відкриваємо серійний порт
serialPort = (SerialPort) portId.open( this .getClass().getName(),TIME_OUT);
// встановлюємо параметри порту
serialPort.setSerialPortParams(DATA_RATE,
SerialPort.DATABITS_8,
SerialPort.STOPBITS_1,
SerialPort.PARITY_NONE);
// відкриваємо вхідні та вихідні потоки
input = serialPort.getInputStream();
output = serialPort.getOutputStream();
// додаємо слухачів подій
serialPort.addEventListener( this );
serialPort.notifyOnDataAvailable( true );
} catch (Exception e) {
System.err.println(e.toString());
}
}
// метод для закриття порту
public synchronized void close() {
if (serialPort != null ) {
serialPort.removeEventListener();
serialPort.close();
}
}
// метод для відправки одного байта
public void sendSingleByte( byte myByte) {
try {
output.write(myByte);
```

```
output.flush();
} catch (Exception e) {
System.err.println(e.toString());
}
}
// метод прийому даних (не використовується)
public synchronized void serialEvent (SerialPortEvent oEvent) {
if (oEvent.getEventType() == SerialPortEvent.DATA_AVAILABLE) {
try {
int myByte=input.read();
// Переклад з типу byte в int
int value = myByte & 0xff;
if (value>=0 && value<256){
System.out.println(value);
}
} catch (Exception e) {
System.err.println(e.toString());
}
}
}
}
```

Додаток 2

Arduino- передавач

```
#include <NewPing.h>
// оголошення пінів для далекоміра
# define TRIGGER _ PIN 7
# define ECHO _ PIN 8
// максимальна дистанція в сантиметрах, яку реагує далекомір
#define MAX_DISTANCE 200
// створюємо об'єкт далекоміра
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
// бібліотеки для радіозв'язку
#include <SPI.h> // бібліотека для роботи з шиною SPI
#include "nRF24L01.h" // бібліотека радіомодуля
#include "RF24.h" // ще бібліотека радіомодуля
// Створити об'єкт на пінах 9 і 10
RF24 radio(9, 10);
//можливі номери труб
byte address [][6] = {"1 Node ", "2 Node ", "3 Node ", "4 Node ", "5 Node ", "6 Node "};
// Ініціалізація функції фільтрації
float kalman ( float val);
// Функції обробки кнопок
void buttons();
// Оголошення змінних для фільтра Калмана
float varVolt = 21.47; // Середнє відхилення (шукаємо в Excel)
float varProcess = 0.02; // швидкість реакції зміну (підбирається вручну)
float Pc = 0.0;
```

```
float G = 0.0;
float P = 1.0;
float Xp = 0.0;
float Zp = 0.0;
float Xe = 0.0;
// Кінець оголошення змінних для фільтра Калмана
// Змінні для кнопки
int buttonPin = 6;
boolean button1S; // зберігаємо стану кнопок (S – State)
boolean button1F; // Прапорці кнопок (F – Flag)
boolean button1R; // Прапорці кнопок на відпускання (R - Release)
boolean button1P; // Прапорці кнопок на натискання (P - Press)
boolean button1H; // Прапорці кнопок на утримання (H - Hold)
boolean button1D; // Прапорці кнопок на подвійне натискання (D - Double)
boolean button1DP; // Прапорці кнопок на подвійне натискання і відпускання (D - Double Pressed)
#define double_timer 100 // час (мс), відведений друге натискання
#define hold 500 // час (мс), після якого кнопка вважається затиснутою
#define debounce 80 // (мс), антибрусок
unsigned long button1_timer; // Таймер останнього натискання кнопки
unsigned long button1_double; // Таймер подвійного натискання кнопки
// бібліотека до роботи I²C
#include <Wire.h>
// бібліотека до роботи з модулями IMU
#include <TroykaIMU.h>
// множник фільтра Магвіка
#define BETA 0.22
// створюємо об'єкт до роботи з акселерометром
```

```
Accelerometer accel;
// Створюємо об'єкт для роботи з гіроскопом
Gyroscope gyro;
// створюємо об'єкт до роботи з компасом
Compass compass;
// створюємо об'єкт для фільтра Магвіка
Madgwick filter;
// Змінні для даних з гіроскопа, акселерометра та компасу
float gx, gy, gz, ax, ay, az, mx, my, mz;
// одержувані кути орієнтації ( Ейлера )
float yaw, pitch, roll, throttle, yawCenter = 0;
// оброблені значення каналів
int sendYaw, sendPitch, sendThrottle, sendRoll;
// Змінна для зберігання частоти вибірок фільтра
float fps = 100;
// Пермінні для прийому із серійного порту
byte getByte;
int mode1, mode2;
//Прапор готовності до передачі
bool check = false ;
// калібрувальні значення компасу
// отримані в калібрувальній матриці з прикладу "compassCalibrateMatrixx"
const double compassCalibrationBias[3] = {
524.21,
3352.214,
-1402.236
};
const double compassCalibrationMatrix[3][3] = {
```

```
{1.757, 0.04, -0.028},
{0.008, 1.767, -0.016},
{-0.018, 0.077, 1.782}
};
void setup()
{
// Завдання режимів пінів
// пін для кнопки
pinMode(buttonPin, INPUT_PULLUP);
// відкриваємо послідовний порт
Serial.begin(115200);
Serial.println("Begin init...");
// ініціалізація акселерометра
accel.begin();
// ініціалізація гіроскопа
gyro.begin();
// ініціалізація компасу
compass.begin();
// ініціалізація радіомодуля
// активувати модуль
radio.begin();
// режим підтвердження прийому, 1 вкл 0 вимк
radio.setAutoAck(1);
// (Час між спробою достукатися, кількість спроб)
radio.setRetries(0, 15);
// дозволити відсилання даних у відповідь на вхідний сигнал
radio.enableAckPayload();
// Розмір пакета, в байтах
```

```
radio.setPayloadSize(32);
// відкриваємо канал передачі даних для " труби " 0
radio.openWritingPipe(address[0]);
// вибираємо канал
radio.setChannel(0x60);
// рівень потужності передавача
радіо . setPALevel ( RF 24_ PA _ MAX );
// Швидкість обміну
radio.setDataRate (RF24_250KBPS); .
// почати роботу
radio.powerUp();
// модуль не прослуховує ефір, а передає
radio.stopListening();
// калібрування компасу
compass.calibrateMatrix(compassCalibrationMatrix, compassCalibrationBias);
// Виводимо повідомлення про вдалу ініціалізацію
Serial.println("Initialization completed");
}
void loop()
{
// опитування кнопок
button1S = digitalRead(buttonPin);
//відпрацювання кнопок
buttons();
// Відпрацювання режимів кнопки
if (button1P)
{
// Перемикається режим готовності
```

```
check =! Check;
Serial.println("pressed");
// центрується нишпорення
yawCenter = yaw;
button1P = 0;
}
// канал газу отримує дані з далекоміра
throttle = sonar.ping();
// дані фільтруються
throttle = Kalman (throttle);
// запам'ятовуємо поточний час
unsigned long startMillis = millis();
// зчитуємо дані з акселерометра в одиницях G
accel.readGXYZ(&ax, &ay, &az);
// зчитуємо дані з гіроскопа в радіанах за секунду
gyro.readRadPerSecXYZ(&gx, &gy, &gz);
// зчитуємо дані з компасу в Гауссах
compass.readCalibrateGaussXYZ(&mx, &my, &mz);
// встановлюємо коефіцієнти фільтра
filter.setKoeff(fps, BETA);
// оновлюємо вхідні дані в фільтр
filter.update(gx, gy, gz, ax, ay, az, mx, my, mz);
// Отримання кутів yaw, pitch і roll з фільтра
yaw = filter.getYawDeg();
pitch = filter.getPitchDeg();
roll = filter.getRollDeg();
// перетворення даних
// нишпорення
```

```
sendYaw = -3.175 * (yaw - yawCenter);
if (sendYaw >= 127) sendYaw = 127;
if (sendYaw <= -127) sendYaw = -127;
// газ
sendThrottle = 0.159375 * throttle - 63.75;
if (throttle >= 2000) sendThrottle = 255;
if (throttle <= 400) sendThrottle = 0;
// тангаж
if (pitch >= -180 && pitch < -140) sendPitch = 3.175 * pitch + 571.5;
if (pitch > 140 && pitch <= 180) sendPitch = 3.175 * pitch - 571.5;
if (pitch >= -140 && pitch < 0) sendPitch = 127;
if (pitch <= 140 && pitch > 0) sendPitch = -127;
// крен
if (roll >= - 180 && roll < -140) sendRoll = -3.175 * roll - 571.5;
if (roll > 140 && roll <= 180) sendRoll = -3.175 * roll + 571.5;
if (roll <= 140 && roll > 0) sendRoll = 127;
if (roll >= -140 && roll < 0) sendRoll = -127;
// Виводимо отримані кути в serial-порт
if (check == false )
{
Serial.print("!");
Serial.print("yaw:");
Serial.print(yaw);
Serial.print("\t\t");
Serial.print("pitch:");
Serial.print(pitch);
Serial.print("\t\t");
Serial.print("roll:");
```

```
Serial.print(roll);
Serial.print("\t\t");
Serial.print("throttle:");
Serial.println(throttle);
}
else
{
Serial.print("yaw:");
Serial.print(sendYaw);
Serial.print("\t\t");
Serial.print("pitch:");
Serial.print(sendPitch);
Serial.print("\t\t");
Serial.print("roll:");
Serial.print(sendRoll);
Serial.print("\t\t");
Serial.print("throttle:");
Serial.println(sendThrottle);
}
// приймаємо дані із серійного порту
if (Serial.available())
{
getByte = (byte) Serial.read ();
}
// "Розбираємо" байт на відповідні величини
if (( int )getByte == 127) {
mode1 = 255;
mode2 = 0;
```

```
}
else {
mode1 = 0;
mode2 = ( int ) getByte * 86;
if (mode2> 255) {
mode2 = 255;
}
}
// обчислюємо витрачений час на обробку даних
unsigned long deltaMillis = millis() - startMillis;
// обчислюємо частоту обробки фільтра
fps = 1000/deltaMillis;
// При готовності надсилаємо дані на приймач
if (check == true )
{
radio.write(&sendThrottle, sizeof (sendThrottle));
radio.write(&sendYaw, sizeof (sendYaw));
radio.write(&sendPitch, sizeof (sendPitch));
radio.write(&sendRoll, sizeof (sendRoll));
radio.write(&mode1, sizeof (mode1));
radio.write(&mode2, sizeof (mode2));
}
delay(5);
}
// функція фільтрації
float kalman ( float val) {
Pc = P + varProcess;
G = Pc/(Pc + varVolt);
```

```
P = (1-G) * Pc;
Xp = Xe;
Zp = Xp;
// "фільтроване" значення
Xe = G * ( val - Zp ) + Xp ;
return (Xe);
}
void buttons() {
// натискання (з антибразгом)
if (button1S && !button1F && millis() - button1_timer > debounce) {
button1F = 1;
button1_timer = millis();
}
// якщо відпустили до hold, вважати відпущеною
if (!button1S && button1F && !button1R && !button1DP && millis() - button1_timer < hold) {
button1R = 1;
button1F = 0;
button1_double = millis();
}
// якщо відпустили та пройшло більше double_timer, вважати 1 натисканням
if (button1R && !button1DP && millis() - button1_double > double_timer) {
button1R = 0;
button1P = 1;
}
// якщо відпустили і пройшло менше double_timer і знову натиснуто, вважати що натиснуто 2 разів
```

```
if (button1F && !button1DP && button1R && millis() - button1_double < double_timer) {
button1F = 0;
button1R = 0;
button1DP = 1;
}
// якщо була натиснута 2 рази і відпущена, вважати що була натиснута 2 рази
if (button1DP && millis() - button1_timer < hold) {
button1DP = 0;
button1D = 1;
}
// Якщо утримується більш hold, то вважати утриманням
if (button1F && !button1D && !button1H && millis() - button1_timer > hold) {
button1H = 1;
}
// Якщо відпущена після hold, то вважати, що була утримана
if (!button1S && button1F && millis() - button1_timer > hold) {
button1F = 0;
button1H = 0;
}
}
```

Arduino- приймач

```
// бібліотеки для радіозв'язку
#include <SPI.h> // бібліотека для роботи з шиною SPI
#include "nRF24L01.h" // бібліотека радіомодуля
#include "RF24.h" // ще бібліотека радіомодуля
```

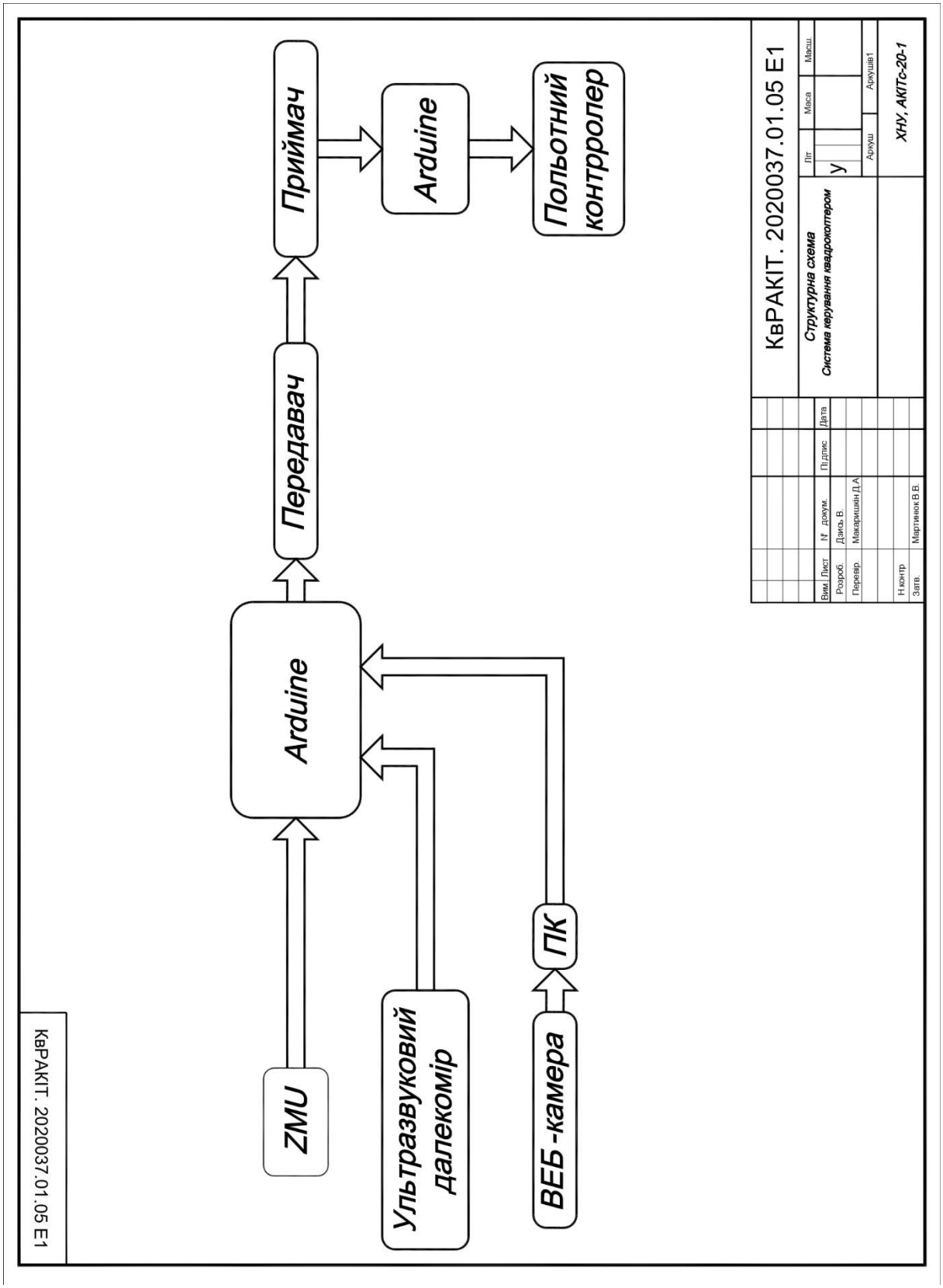
```
// Створити об'єкт на пінах 7 і 8
RF24 radio(7, 8);
//можливі номери труб
byte address [][6] = {"1 Node ", "2 Node ", "3 Node ", "4 Node ", "5 Node ", "6 Node "};
// ініціалізація виходів
#define THROTTLE_PIN 3
#define YAW_PIN 5
#define PITCH_PIN 6
#define ROLL_PIN 9
#define MODE1_PIN 10
#define MODE2_PIN 11
void setup
{
// відкриваємо послідовний порт
Serial.begin(115200);
// ініціалізація радіомодуля
// активувати модуль
radio.begin();
// режим підтвердження прийому, 1 вкл 0 вимк
radio.setAutoAck(1);
// час між спробою достукатися, кількість спроб
radio.setRetries(0, 15);
// дозволити відсилання даних у відповідь на вхідний сигнал
radio.enableAckPayload();
// Розмір пакета, в байтах
radio.setPayloadSize(32);
// відкриваємо канал передачі даних для " труби " 0
```

```
radio.openWritingPipe(address[0]);
// вибираємо канал
radio.setChannel(0x60);
// рівень потужності передавача
радіо . setPALevel ( RF 24_ PA _ MAX );
// Швидкість обміну
radio.setDataRate (RF24_250KBPS); .
// почати роботу
radio.powerUp();
// модуль не прослуховує ефір, а передає
radio.stopListening();
// калібрування компасу
compass.calibrateMatrix(compassCalibrationMatrix, compassCalibrationBias);
// Виводимо повідомлення про вдалу ініціалізацію
Serial.println("Initialization completed");
// приймальний модуль починає прослуховувати ефір
radio.startListening();
}
void loop() {
byte pipeNo, sendThrottle, sendYaw, sendPitch, sendRoll, mode1, mode2;
// слухаємо ефір з усіх "труб"
while ( radio.available(&pipeNo)){
{
radio.read(&sendThrottle, sizeof (sendThrottle));
radio.read(&sendYaw, sizeof (sendYaw));
radio.read(&sendPitch, sizeof (sendPitch));
radio.read(&sendRoll, sizeof (sendRoll));
radio.read(&mode1, sizeof (mode1));
```

```
radio.read(&mode2, sizeof (mode2));
}
// виводимо дані
AnalogWrite (3, sendThrottle);
AnalogWrite (5, sendYaw);
AnalogWrite (6, sendPitch);
analogWrite (9, sendRoll);
analogWrite (10, mode1);
analogWrite (11, mode2);
}
}
```

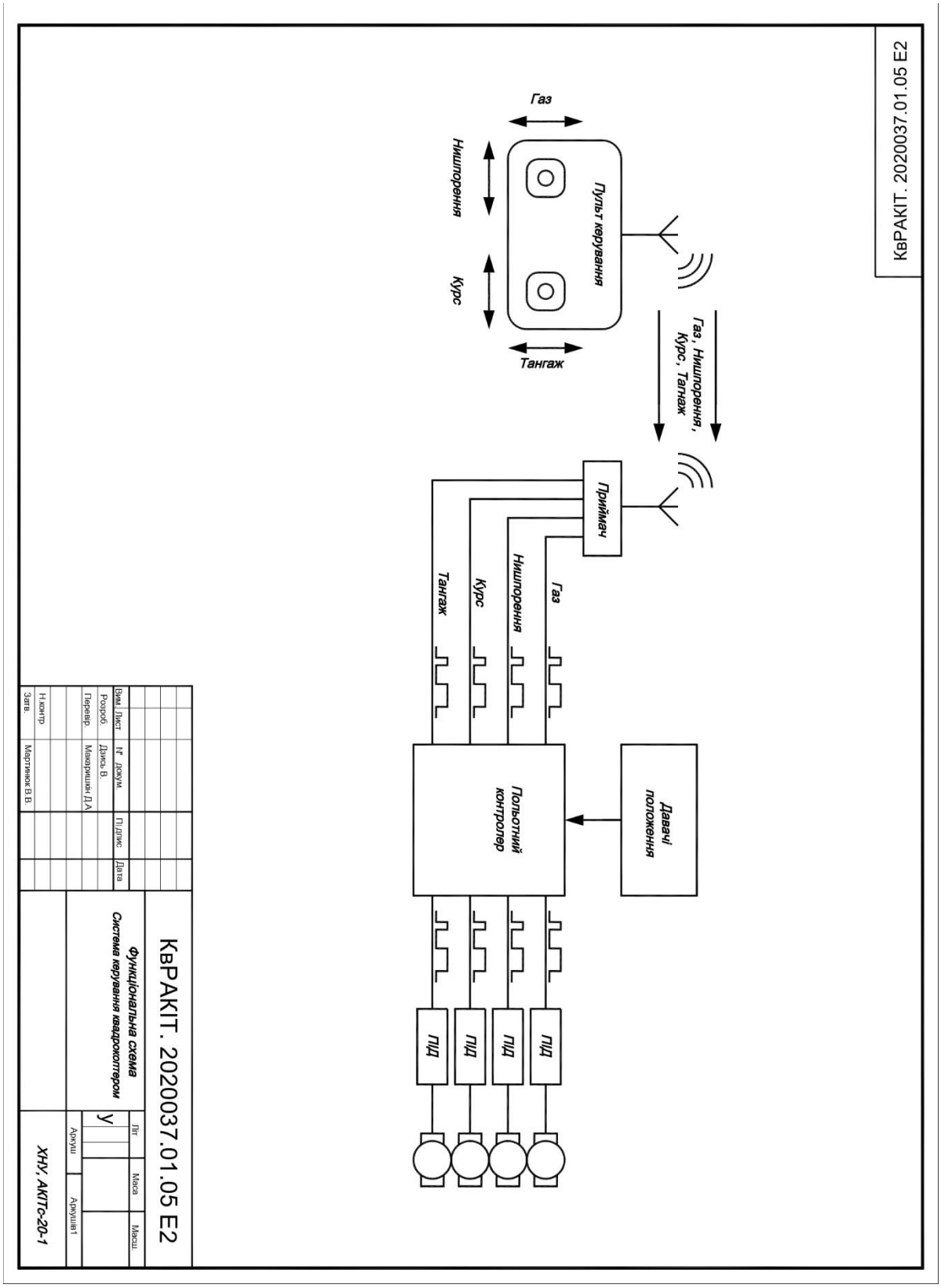
Додаток Б

Структурна схема



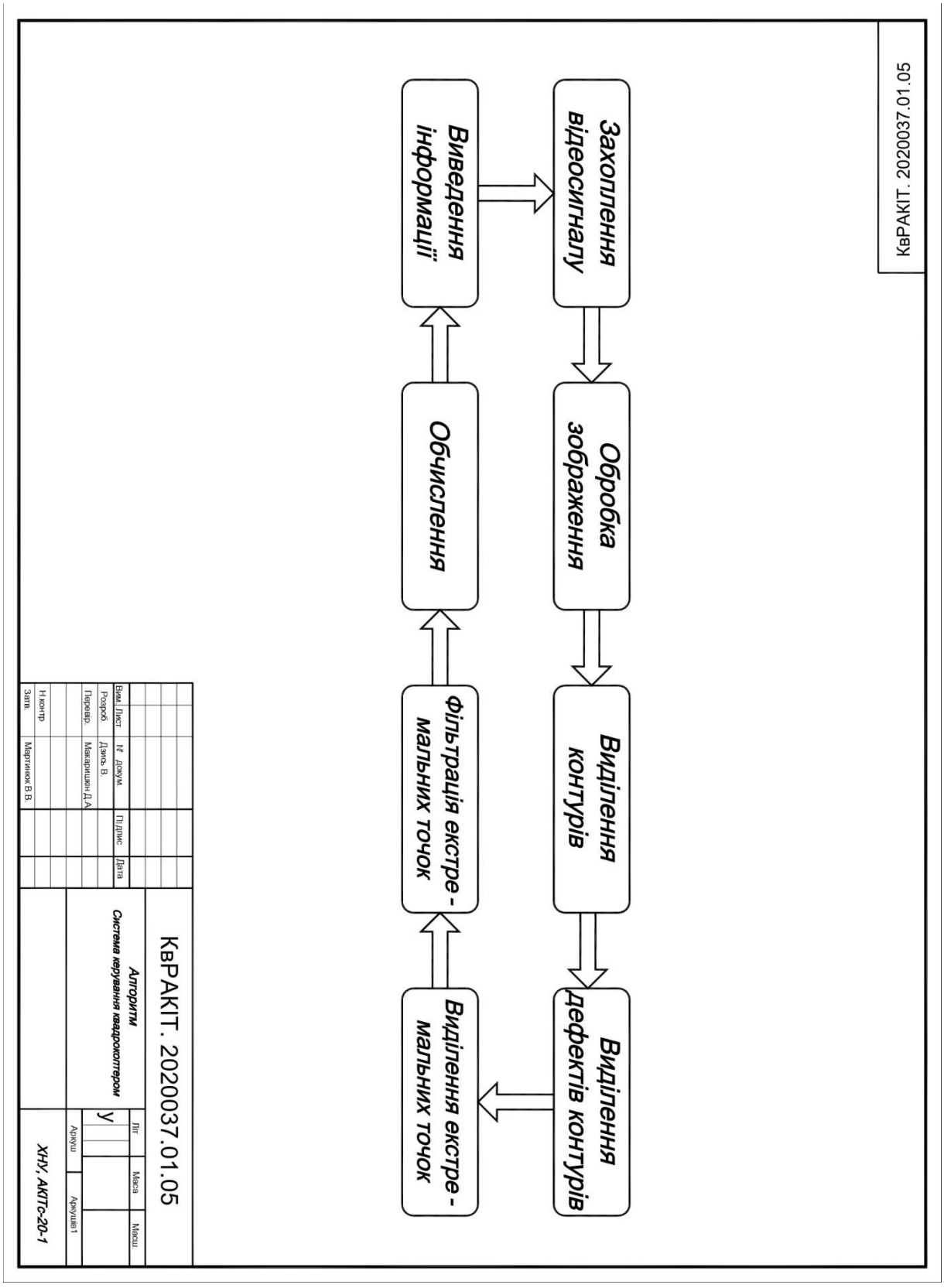
Додаток В

Функціональна схема



Додаток Г

Алгоритм



UNICHECK
by Turnitin

Ім'я користувача:
Кафедра АКІТіТК

ID перевірки:
1015687844

Дата перевірки:
24.06.2023 21:46:56 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
24.06.2023 21:51:48 EEST

ID користувача:
100005862

Назва документа: **Дзись**

Кількість сторінок: **59** Кількість слів: **11568** Кількість символів: **83652** Розмір файлу: **2.73 MB** ID файлу: **1015331807**

---

101 слово позначене як "вилучене" та не враховується у підрахунку слів

2.75%
Схожість

Найбільша схожість: **0.87%** з Інтернет-джерелом (https://arduino.ua/prod1134-poletnii-kontroller-ardupilot-apm-2-8-ko...

2.75% Джерела з Інтернету	105	Сторінка 61
0.08% Джерела з Бібліотеки	1	Сторінка 61

0% Цитат

Цитати	3	Сторінка 62
Посилання	1	Сторінка 62

0.01%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **8 слів** та **0%**)

Немає вилучених Інтернет-джерел

0.01% Вилученого тексту з Бібліотеки	15	Сторінка 62

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	5

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

	Документ		Сумарний збіг по Базі Даних	
ID: 118015 Назва: БКР Система керування квадрокоптером Додано в БД: 2023-06-24 Автора: Владислав ДЗИСЬ Керівники: Денис МАКАРИШКІН Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	76440	666	2768 (4%)	40 (6%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Дзись Владислав Сергійович

Тема: Система керування квадрокоптером

Спеціальність: 151 «Автоматизація та комп'ютерно-інтегровані технології»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 Кількість сторінок записки 60

1. Короткий зміст роботи та прийнятих рішень: Розроблено систему керування квадрокоптером

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі праведно дослідження безпілотних мультироторних літальних апаратів. Розглянуто базові принципи польоту квадрокоптера. Встановлено що для покращення роботи польотного контролера доцільне використання ПІД регуляторі. Наведено опис польотного контролера. У другому розділі проведена розробка системи управління квадрокоптером на базі IMU та ультразвукового далекоміра HC -504. Для проведення розробки виконані наступні роботи: складання пристрою та код для отримання даних, перетворення даних, розробка програмного забезпечення для розпізнавання жестів руки з відеосигналу методами бібліотеки OpenCV. Запропоновано для керування квадрокоптером використовувати жести рук. Для цього досліджено захоплення відеосигналу, обробка зображення, обробка результатів. В третьому розділі розглянуто формування сигналів та передача отриманих даних. Описані різні способи виконання поставленого завдання. Визначені напрямки побудови алгоритму. Запропонована програми передачі даних.

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: у роботі недостатньо уваги приділяється огляду існуючих технічних рішень

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. Інші зауваження: відсутні

9. Оцінка дипломної роботи: відмінно (4,75/А)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) к.т.н., доцент каф. кібербезпеки Тітова В.Ю.

"23" 06 2023 р. _____ (підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ АВТОМАТИЗАЦІЇ, КОМП'ЮТЕРНО-ІНТЕГРОВАНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Система керування квадрокоптером
Автор: Дзись Владислав Сергійович
Спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітня програма: Освітньо-професійна програма «Автоматизація та комп'ютерно-інтегровані технології»
Науковий керівник: Макаришкін Д.А., кандидат технічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій у переліку джерел посилання;

2) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

3) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 2,75% і адресується до 106 джерел, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 24.06.2023р.

Завідувач кафедри — Валерій МАРТИНЮК
Гарант освітньої програми — Юрій ФОРКУН
Керівник кваліфікаційної роботи — Денис МАКАРИШКІН

Завідувачу кафедри АКІТтаР
д-ру техн.наук, проф. Мартинюку В.В.

Дзись В.С.

ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи АКІТс-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність плагіату ознайомлений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

06.06.2023

дата

підпис