Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Кваліфікаційна робота магістра
з теми: **«МЕТОДИ ТА ЗАСОБИ МОНІТОРИНГУ ПОЗИЦІЇ ТА РУХУ ТІЛА В РЕАЛЬНОМУ ЧАСІ»**

Виконав здобувач вищої освіти групи KN1-M24
спеціальності 122 комп’ютерні науки
Василевич Олексій Андрійович
(прізвище та ім’я і по-батькові здобувача вищої освіти)

Керівник: Оптасюк С. В., канд. ф.-м. наук, доцент
(прізвище та ініціали, науковий ступінь, учене звання)

Рецензент: Поведа Р. А., канд. ф.-м. наук, доцент
(прізвище та ініціали, науковий ступінь, учене звання)

Кам’янець-Подільський – 2025 р.

ЗМІСТ

ВСТУП

Сучасні технології моніторингу позиції та руху тіла в реальному часі активно застосовуються у спорті, робототехніці, медицині та інших галузях. Висока точність визначення положення об’єкта в просторі дозволяє забезпечити високий рівень контролю за рухами, автоматизації процесів та оптимізації тренувань або реабілітації. Однак існуючі системи часто мають високу вартість, що іноді обмежує їх широке застосування, особливо у малобюджетних проєктах чи персональному використанні.

Актуальність дослідження полягає у створенні доступного пристрою для моніторингу руху, який поєднуватиме високу точність роботи з низькою вартістю. Використання сенсора MPU-6050, мікроконтролера ESP32 з OLED-дисплеєм дозволить зменшити кількість необхідних компонентів. Додаткова оптимізація алгоритмів обробки даних, забезпечать точне та швидке визначення положення тіла в просторі, зменшуючи вплив шумів.

Об’єктом дослідження є технології моніторингу позиції та руху тіла в реальному часі.

Предметом дослідження є методи та засоби реалізації пристрою для моніторингу позиції та руху тіла, включаючи сенсори, алгоритми обробки даних та програмне забезпечення.

Мета дослідження полягає у розробці пристрою, який здатний точно визначати положення тіла в просторі та зміну положення з часом, використовуючи мінімальну кількість компонентів, що забезпечить його доступність та функціональність.

Завдання дослідження:

1. Проаналізувати існуючі методи та технології моніторингу руху.

2. Розглянути компоненти, середовища розробки та алгоритми, які можуть бути застосовані у системі.

3. Розробити теоретичну модель пристрою, що враховує обрані компоненти та алгоритми.

4. Реалізувати пристрій на основі інерційного сенсора MPU-6050, плати ESP32.

5. Провести тестування пристрою та оцінити його точність і стабільність.

Наукова новизна роботи полягає у створенні недорогого та доступного пристрою для моніторингу руху, що відрізняється від існуючих аналогів меншою кількістю компонентів та використанням оптимізованих алгоритмів для визначення позиції тіла. Такий підхід дозволить знизити вартість розробки, при тому, зберігаючи високу точність і ефективність системи.

РОЗДІЛ 1. АНАЛІЗ ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1 Сучасні методи визначення позиції тіла в просторі

Сучасні технології моніторингу руху змінили багато галузей, від спорту до робототехніки. Вони дозволяють точно відстежувати рухи, оцінювати фізичний стан і коригувати дії в реальному часі. Завдяки новим сенсорам і алгоритмам, такі системи стають все точнішими, доступнішими та забезпечують точність зворотного зв’язку.

Основними методами та засобами, які вирішують проблему моніторингу позиції тіла та його руху в реальному часі можна вважати наступні: інерційні сенсори (IMU), ультразвукові та радіочастотні технології, оптичні системи, сучасні технології на основі штучного інтелекту.

У подальшій роботі буде детальніше розглянуто кожну з цих технологій.

Інерційні сенсори (IMU) є ключовими інструментами для моніторингу руху та положення тіла в реальному часі. Вони представляють собою пристрої, що поєднують кілька сенсорів, таких як акселерометри, гіроскопи і, у деяких випадках, магнітометри. Ці компоненти дозволяють вимірювати прискорення, кутову швидкість і орієнтацію об'єкта в просторі, забезпечуючи точне відстеження тривимірного руху. Завдяки своїй універсальності IMU знаходять застосування в спортивній науці, біомеханіці, робототехніці та індустрії розваг, зокрема для аналізу рухів спортсменів, керування дронами, роботами та створення віртуальної реальності.

Принцип роботи IMU заснований на вимірюванні змін руху та орієнтації через вбудовані сенсори. Акселерометри фіксують лінійне прискорення вздовж трьох осей, тоді як гіроскопи визначають кутову швидкість обертання. У випадках, коли пристрій оснащений магнітометрами, ці сенсори додають можливість калібрувати орієнтацію відносно магнітного поля Землі, що забезпечує додаткову точність. Завдяки поєднанню цих даних IMU здатні обчислювати положення та орієнтацію навіть без зовнішніх систем відстеження, таких як камери чи GPS.

Основною перевагою сенсорних систем IMU є їхня відносна низька вартість та набагато менша вага, завдяки чому вони стають ідеальним рішенням для використання у невеликий мобільних та портативних приладах.

Основним недоліком цієї технології є накопичення помилки при тривалому використанні без постійного калібрування. Це пов’язано з тенденцією гіроскопів до зсуву показників. Для того, щоб мінімізувати дану проблему використовуються різноманітні алгоритми та спеціальні фільтри, які допомагають коригувати дані та підвищують загальну точність вимірювань [1-2].

Ультразвукові та радіочастотні технології використовуються для визначення просторового положення та моніторингу руху об'єктів за допомогою звукових або електромагнітних хвиль. Ультразвукові системи функціонують на основі принципу ехолокації: сенсори випромінюють ультразвукові хвилі, які відбиваються від об'єкта і повертаються до приймача. Час повернення хвилі дозволяє точно обчислити відстань до об'єкта. Такі системи широко застосовуються у медичних цілях, наприклад, для контролю положення пацієнтів під час операцій чи реабілітації.

Радіочастотні технології, навпаки, базуються на використанні радіохвиль для моніторингу руху, часто реалізуючи технології RFID (Radio Frequency Identification) або UWB (Ultra-Wideband). Їх основною перевагою є здатність працювати у складних умовах, таких як наявність перешкод чи відсутність прямої видимості. Ці системи забезпечують безконтактний моніторинг руху в реальному часі, що робить їх корисними у спортивних і медичних застосуваннях, навіть у динамічних середовищах.

Основною перевагою цих технологій є їх висока стійкість до зовнішніх перешкод, а також здатність працювати на великій відстані без втрати точності. Вони менш чутливі до умов освітлення та можуть працювати у складних середовищах, де інші технології, як-от оптичні системи, можуть мати обмеження. Однак ультразвукові системи можуть втрачати точність через ефект відбиття хвиль від різних поверхонь, тоді як радіочастотні технології можуть потребувати

складного налаштування і калібрування для забезпечення максимальної точності. Незважаючи на ці виклики, вони залишаються важливими інструментами для застосувань, що потребують надійного моніторингу у режимі реального часу [3].

Оптичні системи моніторингу позиції та руху тіла є високоточними технологіями, що використовують камери та інфрачервоні датчики для тривимірного відстеження рухів. Принцип їх роботи базується на захопленні зображень з високою частотою кадрів, після чого спеціальні алгоритми аналізують положення маркерів або окремих точок на тілі людини. Ці системи, такі як Vicon, OptiTrack та Kinect, широко використовуються у спортивній науці, реабілітаційній медицині, а також у розважальній індустрії для створення персонажів у відеоіграх та анімації.

Однією з головних переваг оптичних систем є їх висока точність і можливість отримання детальних даних про рухи в реальному часі. Вони дозволяють відстежувати навіть найменші зміни в положенні тіла, що робить їх незамінними для наукових досліджень та аналізу складних рухів, таких як спортивні тренування або кінезіологічні дослідження. Крім того, ці системи можуть відстежувати декілька об'єктів одночасно, що відкриває можливості для аналізу командних видів спорту або комплексних рухових завдань.

Проте оптичні системи мають і певні обмеження. Вони вимагають спеціально обладнаного середовища з кількома камерами, що може обмежувати їх використання на відкритому повітрі або в нестандартних умовах. Також вони можуть бути чутливими до освітлення та потребують регулярної калібрування для підтримки точності. Незважаючи на ці недоліки, оптичні системи залишаються одними з найпопулярніших рішень для високоточних додатків завдяки своїй здатності надавати деталізовану інформацію про динаміку руху.

Технології на основі штучного інтелекту (ШІ) відіграють все більшу роль у моніторингу позиції та руху тіла в реальному часі. Використовуючи машинне навчання, комп'ютерний зір та алгоритми глибокого навчання, ці системи здатні

аналізувати рухи з високою точністю без необхідності в додаткових сенсорах або маркерах. Завдяки камерам і потужним алгоритмам обробки зображень, такі системи можуть розпізнавати складні рухові патерни, оцінювати позу тіла, ідентифікувати помилки у виконанні вправ та навіть прогнозувати ризик травм.

Однією з основних переваг технологій на базі ШІ є їх здатність працювати у режимі реального часу, що дозволяє отримувати оперативний зворотний зв'язок для користувача. Це має важливе значення у спортивних тренуваннях, реабілітації та навіть у системах розумного будинку, де потрібне швидке реагування на рухи. Такі системи можуть адаптуватися до індивідуальних особливостей користувача, аналізуючи дані для побудови персоналізованих тренувальних програм або оптимізації терапії. Крім того, ШІ-технології можуть працювати безпосередньо з відеопотоком із звичайних камер, що робить їх доступними для широкого кола користувачів без необхідності у спеціалізованому обладнанні.

Однак, незважаючи на всі переваги, ці технології мають і свої виклики. Висока точність аналізу рухів залежить від якості та кількості навчальних даних, що використовуються для навчання моделей. Недостатня кількість даних або їхня однорідність можуть призвести до помилок у розпізнаванні рухів або позицій тіла. Крім того, використання ШІ вимагає значних обчислювальних ресурсів, особливо якщо мова йде про обробку відеопотоку у високій роздільній здатності. Тим не менш, розвиток хмарних технологій та спеціалізованих чипів для ШІ дозволяє розширювати можливості цих систем, роблячи їх більш доступними та ефективними у різних сферах застосування.

Завдяки своїй універсальності та здатності адаптуватися до різних завдань, ШІ-технології мають потенціал стати однією з основних платформ для моніторингу руху в реальному часі у майбутньому. Вони вже знаходять застосування у таких сферах, як спорт, охорона здоров'я, виробництво та навіть безпека, забезпечуючи точний аналіз і моніторинг без необхідності у фізичних сенсорах або маркерах.

Аналіз сучасних методів визначення позиції та руху тіла в просторі показав, що інерційні сенсори (IMU) мають значні переваги у порівнянні з іншими технологіями. Завдяки своїй доступній вартості, компактності та малій вазі, вони ідеально підходять для портативних і мобільних застосувань. Ці характеристики роблять IMU оптимальним вибором для розробки пристрою, спрямованого на забезпечення доступного і точного моніторингу руху тіла. Незважаючи на різноманітні обмеження, IMU залишається одним із найефективніших рішень для моніторингу руху в реальному часі завдяки своїй універсальності та широкому спектру застосувань.

1.2 Огляд сенсорів для відстеження положення тіла в просторі

MPU-6050 GY-521 – компактний і легкий модуль 3х осьового акселерометра і 3х осьового гіроскопа, керований за протоколом I2C (TWI). Цей датчик чудово підходить для визначення положення в просторі.

MPU6050 - це інтегрована мікросхема, яка містить у собі:

1. 3-осьовий MEMS гіроскоп.
2. 3-осьовий MEMS акселерометр.
3. Цифровий процесор руху (DMP).
4. 16-бітний АЦП для кожного каналу.

Цей сенсор може вимірювати кутову швидкість і прискорення з високою точністю. Діапазони вимірювання можна налаштувати.

MPU-6050 GY-521 зображений на рисунку 1.1

Рис. 1.1 — MPU-6050 GY-521

MPU6050 GY-521 — це потужний інструмент для вимірювання руху, який легко інтегрується з Arduino [4].

Мікроконтролери — це невеликі комп'ютери на одному чипі, які містять процесорне ядро, пам'ять (ПЗП та ОЗП), а також периферійні пристрої вводу та виводу. Вони призначені для керування конкретними функціями в електронних пристроях, на відміну від потужних комп'ютерів загального призначення.

Серія ESP32 від Espressif Systems — це популярне сімейство мікроконтролерів-на-кристалі (SoC), яке здобуло велику популярність у сфері Інтернету речей (IoT) завдяки інтеграції Wi-Fi та Bluetooth (зокрема Bluetooth Low Energy - BLE) у поєднанні з низьким енергоспоживанням та доступною ціною. Ці чипи дозволяють створювати пристрої, які можуть легко підключатися до мережі та взаємодіяти з іншими пристроями.

Таблиця 1.1 — порівняльна таблиця основних чипів ESP32

Характеристика	ESP32	ESP32-S2	ESP32-C3	ESP32-S3
Архітектура CPU	Tensilica Xtensa LX6 (32-біт)	Tensilica Xtensa LX6 (32-біт)	Tensilica Xtensa LX6 (32-біт)	Tensilica Xtensa LX7 (32-біт)
Кількість ядер	Двоядерний	Одноядерний	Одноядерний	Двоядерний
Макс. частота	240 МГц	240 МГц	240 МГц	240 МГц
Wi-Fi	802.11 b/g/n (2.4 ГГц)	802.11 b/g/n (2.4 ГГц)	802.11 b/g/n (2.4 ГГц)	802.11 b/g/n (2.4 ГГц)
Bluetooth	BT Classic + BLE 4.2	Немає (лише Wi-Fi)	BLE 5.0	BLE 5.0
SRAM	520 КБ	320 КБ	400 КБ	512 КБ
Ключовий фокус	Загальне використання, продуктивність, BT Classic	Енергоефективний Wi-Fi, безпека	Низька вартість, безпека, BLE 5.0	AI, мультимедіа, потужність, широкі I/O

				Экспортировать в Таблицы

Вибір ESP32-C3 часто зумовлений низкою факторів, які роблять його ідеальним для більшості сучасних проетів, особливо тих, що вимагають балансу між вартістю, енергоспоживанням та функціональністю.

ESP32-C3 з — недорогий мікроконтролер від компанії Espressif з підтримкою бездротової Wi-Fi мережі 2.4 ГГц і Bluetooth Low Energy (Bluetooth 5 (LE)). У мікроконтролері використано високопродуктивне RISC V ядро. Процесор контролера розроблений для високопродуктивних систем з малим енергоспоживанням і низькою вартістю. На платі модуля встановлено OLED I2C дисплей на драйвері SSD1306 розміром 0.42 дюйма і роздільною здатністю 72X40 пікселів.

ESP32-C3 об'єднує багатий набір периферійних пристроїв, таких як UART, I2C, I2S, периферійних пристроїв дистанційного керування, контролера LED PWM, контролера DMA, контролера TWAI, контролера USB Serial/JTAG, датчика температури та АЦП. Він також містить інтерфейси SPI, Dual SPI і Quad SPI.

ESP32-C3 зображено на рисунку 1.2

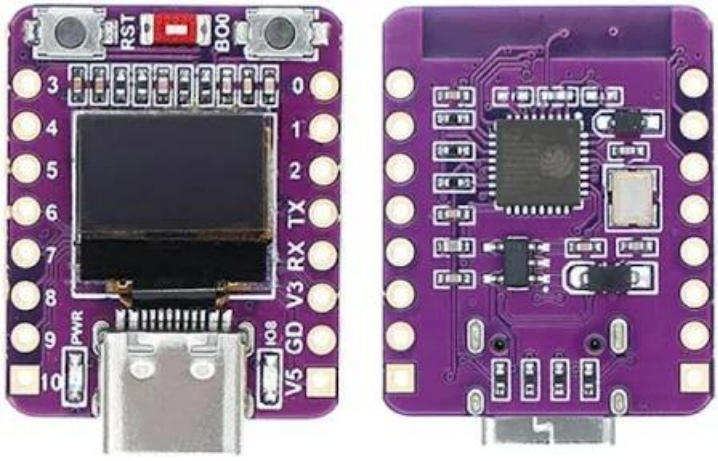


Рис 1.2 — ESP32-C3

ESP32-C3 позиціонується як пряма, але більш функціональна та безпечна альтернатива старому ESP8266, а також як економічно ефективна альтернатива

оригінальному ESP32, коли не потрібна висока продуктивність двоядерного процесора або підтримка BT Classic. [5].

1.3 Огляд середовищ розробки для мікроконтролерів

Середовища розробки (IDE) відіграють ключову роль у програмуванні мікроконтролерів, забезпечуючи інженерів і розробників зручними інструментами для створення, тестування та впровадження програмного забезпечення. Вони об'єднують у собі текстовий редактор, компілятор, засоби налагодження та завантаження прошивки в мікроконтролер, що значно спрощує процес розробки.

Роль IDE полягає у прискоренні роботи над проєктами, надаючи інтуїтивний інтерфейс для написання коду, інструменти для швидкого виявлення та усунення помилок, а також засоби для перевірки функціональності мікроконтролерів у реальному часі. Використання сучасних середовищ розробки дає змогу підвищити продуктивність, зменшити кількість помилок і забезпечити зручність роботи як для новачків, так і для досвідчених розробників.

Основними середовищами розробки для програмування мікроконтролерів є: Arduino IDE та PlatformIO.

PlatformIO — це потужне середовище розробки, яке пропонує ширші можливості порівняно з Arduino IDE. Воно інтегрується з Visual Studio Code і забезпечує підтримку багатьох платформ і мікроконтролерів. PlatformIO надає інструменти для професійного налагодження, таких як, управління залежностями та автоматизоване тестування.

Перевагами PlatformIO є багатий функціонал, масштабованість і підтримка професійних інструментів, що робить його ідеальним вибором для складних проєктів. Однак недоліками є більш високий поріг входу для новачків і триваліший процес налаштування середовища [6].

Arduino IDE є одним із найпопулярніших середовищ розробки, орієнтованих на простоту та доступність. Воно відоме своїм інтуїтивно зрозумілим інтерфейсом, який підходить як для початківців, так і для досвідчених розробників. Arduino IDE підтримує велику кількість мікроконтролерів, таких як Arduino, ESP32, STM32, і пропонує доступ до широкого вибору бібліотек для сенсорів, дисплеїв та інших периферійних пристроїв. Готові приклади значно спрощують процес навчання та розробки.

Основними перевагами Arduino IDE є його простота, швидкість налаштування та використання, а також активна спільнота розробників, яка постійно ділиться новими матеріалами. Проте серед недоліків можна виділити обмежений функціонал редактора коду (відсутність сучасних інструментів автодоповнення та рефакторингу), що може ускладнити роботу над великими проєктами.

Саме наявність вбудованої підтримки мікроконтролера ESP32 через менеджер плат та велика кількість вбудованих бібліотек стала вирішальним фактором для вибору Arduino IDE як основного середовища для розробки.

Вигляд основоного вікна середовища розробки Arduino IDE зображено на рисунку 1.3

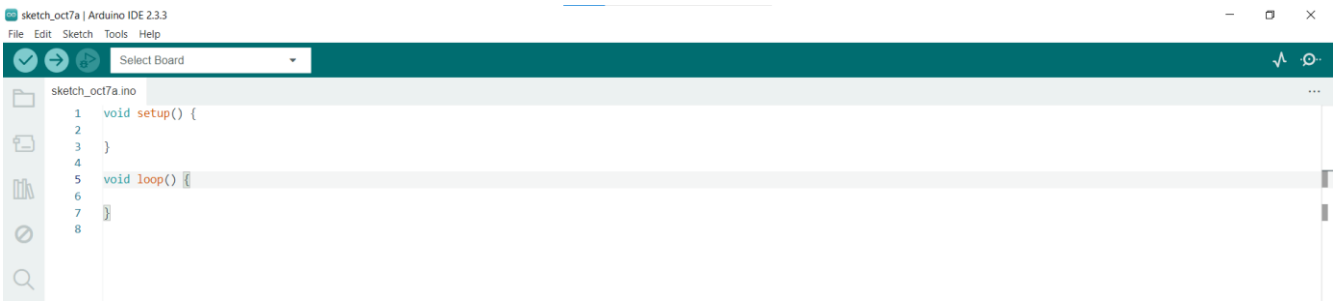


Рис. 1.3 — вікно для розробки Arduino IDE

У Arduino IDE при створенні нового проєкту автоматично створюються дві основні функції: setup() і loop():

1. setup() — виконується один раз після запуску або перезавантаження мікроконтролера. У цій функції зазвичай здійснюється початкове налаштування,

наприклад, ініціалізація портів вводу/виводу, встановлення зв’язку з периферійними пристроями або налаштування серійного зв’язку.

2. loop() — виконується постійно у циклі після завершення функції setup(). У цій функції реалізується основна логіка роботи програми, така як зчитування даних із сенсорів, обробка інформації та керування пристроями.

Ці функції забезпечують зручну структуру для написання програм під мікроконтролери, розділяючи процеси ініціалізації та виконання [7].

1.4 Аналіз алгоритмів обробки даних з сенсорів

Алгоритми обробки даних відіграють ключову роль у точному визначенні позиції та орієнтації тіла в просторі. Вони забезпечують можливість перетворювати "сирі" дані, отримані з сенсорів, у надійну та точну інформацію, що необхідна для навігації, робототехніки, медичних пристроїв, віртуальної реальності та інших застосувань. Застосування таких алгоритмів дозволяє значно підвищити точність і стабільність роботи систем, що працюють з просторовими координатами.

Однак, дані, які надходять із сенсорів, таких як акселерометри та гіроскопи, часто супроводжуються значними похибками. Це може бути викликано різними факторами, зокрема шумом, дрейфом гіроскопа, неточностями в калібруванні та затримками передачі сигналу. Такі проблеми можуть призводити до накопичення помилок і погіршення точності вимірювань, що є неприйнятним для більшості практичних застосувань.

Для вирішення цих проблем необхідно застосовувати алгоритми, здатні фільтрувати шуми, компенсувати дрейф та об’єднувати дані з різних сенсорів. Використання таких алгоритмів, як фільтр Калмана, комплементарний фільтр і кватерніони, дозволяє досягати високої точності при мінімальних обчислювальних витратах. Саме ці алгоритми забезпечують стабільність і надійність систем визначення позиції та орієнтації тіла, роблячи їх придатними для роботи в реальному часі.

Одним із найпоширеніших та найефективніших алгоритмів для оцінки стану динамічних систем у реальному часі є Фільтр Калмана. Його основа — це математична модель, яка комбінує попередні оцінки з новими вимірюваннями, враховуючи статистичні властивості шуму. Такий підхід дозволяє не лише мінімізувати вплив шуму на результати, а й прогнозувати майбутній стан системи.

Однією з головних переваг фільтра Калмана є його здатність адаптивно оцінювати стан навіть у випадках, коли доступні дані є неповними чи містять похибки. Завдяки врахуванню як систематичних, так і випадкових похибок, алгоритм забезпечує високу точність та стабільність. Це робить його ідеальним для роботи з даними з акселерометра та гіроскопа, які часто страждають від шуму, дрейфу та інших спотворень.

Наприклад, акселерометр добре визначає напрямок сили тяжіння, але його дані зазвичай зашумлені під час руху. Гіроскоп, своєю чергою, забезпечує стабільну оцінку кутової швидкості, але має проблему з накопиченням дрейфу. Фільтр Калмана дозволяє об’єднувати ці дані таким чином, щоб компенсувати недоліки кожного з сенсорів, отримуючи точну інформацію про положення та орієнтацію [8-9].

Одним із найпростіших алгоритмів для обробки даних з акселерометра та гіроскопа є Комплементарний фільтр. Його принцип роботи базується на комбінуванні даних з цих двох сенсорів, щоб компенсувати їхні недоліки і отримати стабільні та точні результати. Акселерометр надає дані про напрямок сили тяжіння, але його вимірювання зазнають впливу шуму під час руху. Гіроскоп забезпечує плавні зміни орієнтації, але має проблему з накопиченням дрейфу. Комплементарний фільтр комбінує ці дані, забезпечуючи точну оцінку орієнтації.

Принцип роботи комплементарного фільтра полягає в об’єднанні довготривалих стабільних даних акселерометра із швидкими, але короткотривалими точними вимірюваннями гіроскопа. Це досягається шляхом застосування зваженого сумування даних: низькочастотна інформація

акселерометра доповнюється високочастотною інформацією гіроскопа. У формі простого рівняння це виглядає як:

$$\text{Кут} = \alpha * (\text{Кутгіроскопа}) + (1 - \alpha) * (\text{Кутакселерометра})$$

α — ваговий коефіцієнт, що налаштовується відповідно до характеристик сенсорів.

Головною перевагою комплементарного фільтра є його простота реалізації. Він вимагає значно менших обчислювальних ресурсів порівняно з фільтром Калмана, що робить його ідеальним вибором для систем із обмеженими можливостями, таких як мікроконтролери Arduino. Завдяки цьому фільтр може бути використаний навіть у реальному часі, не створюючи значного навантаження на процесор.

Однак точність комплементарного фільтра може поступатися складнішим алгоритмам, таким як фільтр Калмана, особливо у випадках сильного шуму або складних динамічних рухів. Незважаючи на це, його простота та ефективність роблять його популярним вибором для багатьох портативних застосувань.

Для нашого проєкту використання комплементарного фільтра дозволить зменшити обчислювальні вимоги і забезпечити достатню точність для задачі визначення позиції та орієнтації тіла в просторі. Це особливо важливо, оскільки ми використовуємо малопотужний мікроконтролер ESP32 і прагнемо зберегти економічність системи.

Застосування алгоритмів обробки даних, таких як фільтр Калмана або комплементарний фільтр, є ключовим елементом у забезпеченні точності та надійності роботи системи моніторингу позиції та орієнтації тіла в просторі. Кожен із цих алгоритмів має свої переваги: фільтр Калмана дозволяє ефективно зменшувати шум і прогнозувати положення, тоді як комплементарний фільтр пропонує просте й ресурсозберігаюче рішення для інтеграції даних з акселерометра та гіроскопа [10].

У даному проекті можливе поєднання цих алгоритмів для досягнення оптимального результату. Наприклад, у сценаріях, де необхідна висока точність при обмежених ресурсах, використання комплементарного фільтра може забезпечити базовий рівень стабільності, тоді як у складніших умовах фільтр Калмана може застосовуватися для обробки критично важливих даних.

Очікувані результати включають стабільне й точне визначення позиції тіла в просторі навіть у динамічних умовах. Це дозволить системі працювати ефективно, забезпечуючи реальний час обробки даних із сенсорів при мінімальних апаратних витратах. Таким чином, запропоновані алгоритми зроблять нашу розробку конкурентоспроможною завдяки поєднанню економічності, портативності та функціональності.

1.5 Аналіз бездротових технологій передачі даних

Бездротові технології — це системи передачі інформації на відстань між двома або більше об'єктами за допомогою електромагнітного випромінювання (радіохвиль, інфрачервоного випромінювання тощо) без використання фізичних кабелів.

Основні бездротові технології, які є критично важливими для більшості сучасних пристроїв — це Wi-Fi та Bluetooth.

Wi-Fi — це технологія бездротової локальної мережі (WLAN), заснована на стандартах IEEE 802.11, яка дозволяє пристроям підключатися до Інтернету або створювати локальні мережі через радіохвилі, працюючи переважно у діапазонах 2.4 ГГц та 5 ГГц. Вона забезпечує високу швидкість передачі даних та великий радіус дії, завдяки чому ідеально підходить для додатків, що вимагають потокового передавання великих обсягів інформації (наприклад, відео, оновлення прошивки) та постійного зв'язку з централізованим маршрутизатором, але при цьому характеризується відносно високим енергоспоживанням, що вимагає постійного живлення пристрою.

Bluetooth — це технологія бездротової персональної мережі (WPAN), яка використовує діапазон 2.4 ГГц для створення короткодіючого зв'язку між двома або кількома пристроями. Сучасна версія Bluetooth Low Energy (BLE) особливо цінується за надзвичайно низьке енергоспоживання, що є ключовим фактором для портативних пристроїв з батарейним живленням, таких як фітнес-трекери, бездротові навушники та сенсори. Хоча Bluetooth пропонує нижчу швидкість передачі даних і менший радіус дії порівняно з Wi-Fi, він ідеально підходить для ефективного обміну невеликими періодичними пакетами інформації, встановлюючи пряме з'єднання між пристроєм та смартфоном без потреби у проміжній точці доступу [11-13].

Таблиця 1.2 — порівняння Wi-Fi та Bluetooth

Характеристика	Wi-Fi	Bluetooth
Призначення	Підключення до локальної мережі/Інтернету, передача великих файлів.	Персональний зв'язок між пристроями, заміна кабелів.
Енергоспоживання	Високе (зазвичай вимагає постійного живлення)	Дуже низьке (ідеально для батарейного живлення)
Швидкість даних	Висока (до 150 Мбіт/с і вище)	Низька (до 1-2 Мбіт/с для BLE)
Радіус дії (номінальний)	Середній/Великий (20-100 м)	Короткий (10-30 м)
Мережева топологія	Зірка (усі підключені до центрального роутера)	Пікомережа (Master-Slave, об'єднання 2-8 пристроїв)
Складність налаштування	Вища (потрібен роутер, налаштування мережі)	Низька (просте сполучення пристроїв)

Для роботи над проектом, був обраний Bluetooth, оскільки ключовою вимогою для такого портативного та інтегрованого у тіло пристрою є максимальна автономність і час роботи від батареї, що забезпечується надзвичайно низьким енергоспоживанням. Функціональність пристрою вимагає лише періодичної передачі невеликих пакетів даних із сенсорів на мобільний пристрій користувача,

для чого пропускна здатність Wi-Fi є надмірною. Використання Bluetooth дозволяє встановити пряме, енергоефективне з'єднання "пристрій-смартфон", уникаючи складнощів та високих енерговитрат, пов'язаних із підключенням до зовнішньої мережі Wi-Fi.

Висновки до 1 розділу

У першому розділі було проведено комплексний аналіз сучасних методів та технологій для визначення позиції й орієнтації тіла в просторі. Розглянуто переваги й недоліки інерційних сенсорів, ультразвукових і радіочастотних технологій, оптичних систем та підходів, заснованих на штучному інтелекті. На основі аналізу було обґрунтовано вибір інерційних сенсорів (IMU) для реалізації проєкту через їхню доступність, портативність і високу ефективність у мобільних системах.

Також було досліджено апаратне забезпечення, включаючи акселерометри, гіроскопи, мікроконтролери ESP32, які забезпечують функціональність розроблюваної системи. У рамках огляду середовищ розробки Arduino IDE та PlatformIO проаналізовано їхні можливості для швидкого та ефективного створення прошивок для мікроконтролерів, що дозволяє обрати найбільш оптимальне рішення для реалізації проєкту.

Окрім цього, було розглянуто основні алгоритми обробки даних із сенсорів, включаючи фільтр Калмана та комплементарний фільтр. Проведений аналіз підтвердив важливість використання алгоритмів для зменшення шумів, дрейфу й похибок у даних із сенсорів, що критично важливо для забезпечення точності та надійності роботи системи.

Таким чином, проведений у розділі аналіз створює міцну основу для переходу до теоретичної та практичної реалізації проєкту в наступних розділах.

РОЗДІЛ 2. ТЕОРЕТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ МОНІТОРИНГУ ПОЗИЦІЇ ТІЛА В ПРОСТОРІ

2.1 Розробка функціональної структури системи

Функціонально-структурна схема — це графічне зображення, яке об'єднує два типи схем: структурну та функціональну, показуючи основні функціональні частини виробу, їх взаємозв'язки та процеси, що в них відбуваються, а також загальний принцип роботи виробу. Така схема дозволяє зрозуміти структуру об'єкта та процеси, які забезпечують його функціонування, що корисно для вивчення, налагодження та ремонту.

Розробка функціональної структурної схеми є фундаментальним етапом у проєктуванні будь-якої технічної системи, оскільки вона дозволяє візуалізувати архітектуру проєкту на високому рівні абстракції. Ця схема відображає основні функціональні блоки системи, їхнє призначення та взаємозв’язки між ними, що дає змогу зрозуміти загальний принцип роботи, не заглиблюючись у специфічні технічні деталі апаратної чи програмної реалізації. Вона чітко ілюструє потік даних — від моменту їхнього збору до фінального представлення користувачу. У контексті даної дипломної роботи, система моніторингу позиції та руху тіла в реальному часі побудована на взаємодії трьох ключових функціональних блоків: сенсорного модуля, відповідального за збір первинних даних, обчислювального модуля, що виконує їхню обробку, та модуля візуалізації, призначеного для відображення результатів.

Функціонально-структурну схему підключення ESP32-C3 до MPU-6050 зображено на рисунку 2.1

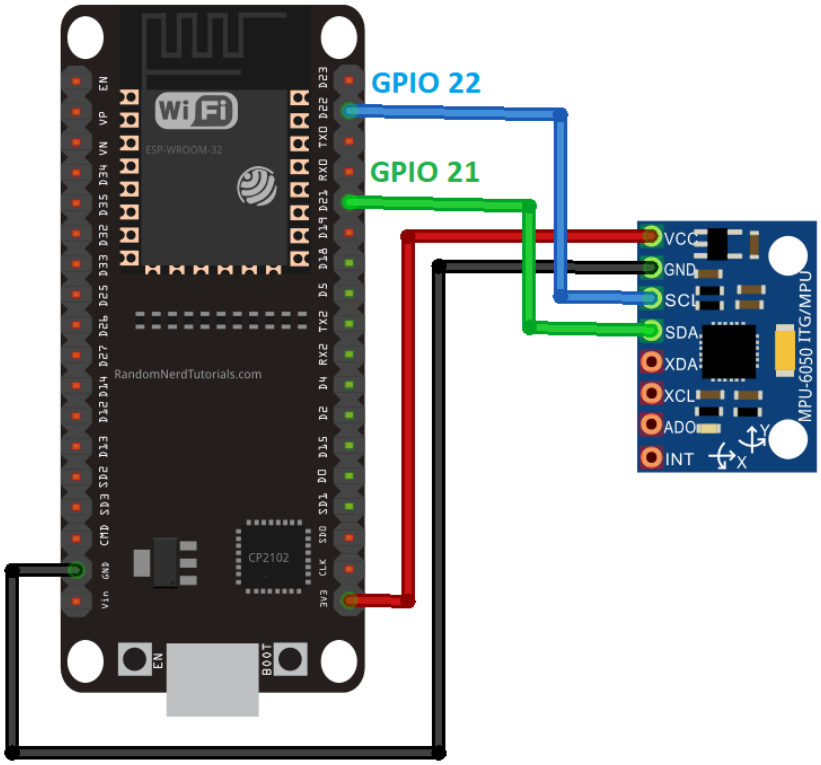


Рис 2.1 — функціонально-структурна схема

Це підключення реалізується за допомогою інтерфейсу I2C (Inter-Integrated Circuit), що вимагає лише двох ліній даних для зв'язку між мікроконтролером (ESP32) і датчиком (MPU-6050), крім ліній живлення [14-15].

З'єднання складається з чотирьох основних ліній: живлення, заземлення та двох ліній I2C.

Таблиця 2.1 — деталі підключення

Пін MPU-6050	Пін ESP32	Призначення
VCC (Червоний провід)	3V3 (Пін живлення 3.3 В)	Живлення модуля. MPU-6050 працює з логічними рівнями 3.3 В, які забезпечує ESP32.
GND (Чорний провід)	GND (Пін заземлення)	Заземлення (спільний нульовий потенціал).

SCL (Синій провід)	GPIO 22	Лінія Тактування (Serial Clock Line) для I2C. Керує синхронізацією передачі даних.
SDA (Зелений провід)	GPIO 21	Лінія Даних (Serial Data Line) для I2C. По ній передаються виміряні дані.

Ця схема забезпечує надійний цифровий зв'язок для отримання даних інерціального вимірювального блоку (IMU) MPU-6050 мікроконтролером ESP32.

2.2 Принцип роботи акселерометра та гіроскопа MPU-6050

Для ефективного моніторингу позиції та руху тіла в просторі ключовим елементом системи є інерційний вимірювальний блок (IMU). . Саме MPU-6050 було обрано як основу сенсорного модуля завдяки його високій функціональності та інтегрованому дизайну. Цей компактний чіп містить в собі два основних сенсори: 3-осьовий акселерометр та 3-осьовий гіроскоп, що працюють на одному кристалі. Така інтеграція є критично важливою, оскільки вона забезпечує синхронне зчитування даних і мінімізує похибки, спричинені фізичним розташуванням сенсорів. Дані, отримані з цих двох компонентів, є основою для подальших обчислень, що дозволяють визначити орієнтацію пристрою [16-17].

Принцип роботи кожного з цих сенсорів ґрунтується на різних фізичних явищах. Акселерометр вимірює прискорення, що дозволяє визначити кут нахилу пристрою відносно площини горизонту, використовуючи силу тяжіння як опорний вектор. Водночас, гіроскоп вимірює кутову швидкість, що дає можливість відстежувати динамічні обертальні рухи. Хоча кожен сенсор окремо має свої недоліки — акселерометр чутливий до вібрацій, а гіроскоп схильний до накопичення помилок (дрейфу) з часом — їхня комбінація дозволяє компенсувати ці недоліки.

Саме тому, для отримання точних і стабільних даних про орієнтацію, в системі використовується метод сенсорного ф’южну. Це означає, що дані з обох сенсорів об'єднуються та обробляються за допомогою спеціальних алгоритмів, наприклад, фільтра Кальмана, що дозволяє досягти високої точності. У наступних розділах буде детально розглянуто, як працює кожен із сенсорів, а також як їхні дані використовуються для обчислення кінцевих параметрів руху та позиції.

Акселерометр — це сенсор, який вимірює прискорення. Його робота базується на законі інерції: тіло зберігає свій стан спокою або рівномірного прямолінійного руху, доки на нього не діє зовнішня сила. В основі сучасних акселерометрів, зокрема тих, що використовуються в MPU-6050, лежить технологія мікроелектромеханічних систем (MEMS). Це крихітні механічні структури, що виробляються за допомогою технологій мікрообробки.

Всередині MEMS-акселерометра розташована інерційна маса, яка вільно рухається вздовж однієї або кількох осей. . Ця маса закріплена на гнучких елементах, що діють як пружини. Коли пристрій починає прискорюватися, інерційна маса, відповідно до закону інерції, прагне зберегти свій початковий стан, що призводить до її зміщення відносно корпусу сенсора. Це зміщення вимірюється за допомогою ємнісних сенсорів, які реєструють зміну ємності між рухомою масою та нерухомими електродами. Ця зміна ємності прямо пропорційна прикладеному прискоренню, що дозволяє перевести її в цифрове значення.

Крім динамічних прискорень, акселерометр також здатний вимірювати статичне прискорення сили тяжіння. Ця властивість є критично важливою для визначення орієнтації. Сила тяжіння ($9.81 м/с^2$) діє завжди вертикально вниз. Коли пристрій перебуває в стані спокою, акселерометр "відчуває" цю силу. Розкладаючи вектор сили тяжіння на три осі (X, Y, Z), можна визначити кут нахилу пристрою відносно горизонтальної площини. Наприклад, якщо пристрій лежить рівно, акселерометр показуватиме прискорення, близьке до 1g по осі Z, і нульові значення по осях X та Y. При нахилі, сила тяжіння буде розподілятися між осями, що дозволяє

обчислити кути тангажу (Pitch) та крену (Roll) за допомогою тригонометричних функцій. Таким чином, акселерометр слугує надійним орієнтиром для визначення статичної позиції пристрою.

На відміну від акселерометра, що вимірює прискорення, гіроскоп призначений для вимірювання кутової швидкості — швидкості обертання пристрою навколо своїх осей. У MPU-6050, як і в більшості сучасних сенсорів, використовується технологія мікроелектромеханічних систем (MEMS), що працює на основі ефекту Коріоліса. Цей ефект проявляється як інерційна сила, що діє на об'єкт, який рухається в системі відліку, що обертається. У випадку MEMS-гіроскопа, вібруючий елемент, що постійно коливається з певною частотою, є "об'єктом", а обертання сенсора створює "систему відліку, що обертається".

Коли пристрій, а отже і вібруючий елемент, починає обертатися, на нього діє сила Коріоліса, яка змушує його коливатися перпендикулярно до початкової вібрації та осі обертання. Амплітуда цього додаткового коливання прямо пропорційна кутовій швидкості. Спеціальні ємнісні сенсори вимірюють це зміщення і перетворюють його на електричний сигнал, який потім оцифровується. Таким чином, гіроскоп надає точні дані про швидкість, з якою обертається пристрій у кожний момент часу.

Для отримання кутів орієнтації, таких як тангаж (Pitch), крен (Roll) та рискання (Yaw), необхідно інтегрувати (сумувати) значення кутової швидкості, виміряні гіроскопом, за певний час. Це дозволяє визначити зміну орієнтації відносно початкового положення. Наприклад, якщо пристрій обертається з кутовою швидкістю 200градусів/с протягом однієї секунди, то його орієнтація зміниться на 200градусів.

Проте, цей метод має суттєвий недолік — дрейф (drift). Навіть найменша похибка в початковому вимірюванні або шум у даних поступово накопичується під час інтегрування, що призводить до значної неточності з часом. Цей дрейф робить гіроскоп ненадійним для тривалого вимірювання статичних кутів, але він ідеально

підходить для фіксації швидких та динамічних рухів, де акселерометр може бути неточним через вібрації. Компенсація цього недоліку є однією з ключових задач, що вирішується за допомогою програмних алгоритмів сенсорного ф’южну.

Взаємодоповнення сенсорів, або сенсорний ф’южн, є ключовим принципом для отримання точних і надійних даних про орієнтацію пристрою. Це програмна техніка, яка об'єднує дані з різних сенсорів, щоб компенсувати їхні індивідуальні недоліки.

1) Проблема акселерометра: Хоча акселерометр надійно вимірює статичні кути нахилу відносно сили тяжіння, він дуже чутливий до динамічних прискорень та вібрацій. Будь-який різкий рух, такий як крок або поштовх, може створити "шум" у даних, що призводить до тимчасових спотворень вимірюваних кутів.

2) Проблема гіроскопа: Гіроскоп чудово відстежує швидкі обертальні рухи, але його головний недолік — дрейф. Навіть мінімальні помилки вимірювання з часом накопичуються під час інтегрування, що призводить до поступового відхилення від істинного значення кута.

Щоб вирішити ці проблеми, дані з акселерометра (який дає точну інформацію про довготривалу орієнтацію) та гіроскопа (який надає точні дані про короткочасні зміни) комбінуються. Для цього використовуються спеціальні алгоритми, такі як фільтр Кальмана або фільтр комплементарності. Ці фільтри дозволяють "злити" дані, використовуючи гіроскоп для відстеження швидких змін і "виправляючи" його дрейф за допомогою більш стабільних, але повільніших даних від акселерометра. В результаті система отримує точні, стабільні та швидкі дані про орієнтацію, що є критично важливим для вашої системи моніторингу руху тіла [18-20].

2.3 Вибір алгоритмів для обробки даних з сенсорів

Для того щоб отримати точні та стабільні дані про орієнтацію пристрою, недостатньо просто зчитувати показники з сенсорів. Як було зазначено раніше, акселерометр та гіроскоп мають свої унікальні недоліки: акселерометр чутливий до вібрацій, а гіроскоп схильний до дрейфу. Вирішенням цієї проблеми є застосування

алгоритмів сенсорного ф’южну, які комбінують дані з обох сенсорів для отримання більш надійного результату. Серед багатьох існуючих методів найбільш поширеними є фільтр Калмана та компліментарний фільтр. Вибір між ними залежить від вимог до точності, складності реалізації та доступних обчислювальних ресурсів мікроконтролера.

Фільтр Калмана є потужним ітеративним алгоритмом, який використовується для оцінки стану динамічної системи на основі ряду неповних або зашумлених вимірювань. Його основна ідея полягає в тому, що він прогнозує майбутній стан системи (на основі попередніх даних) та коригує цей прогноз, використовуючи нові вимірювання. Це робить його надзвичайно ефективним для усунення випадкового шуму та отримання високоточної оцінки. . Однак, його реалізація вимагає значних обчислювальних ресурсів, оскільки алгоритм оперує матрицями та виконує складні математичні операції. Це може бути проблематично для мікроконтролерів з обмеженою потужністю, таких як ESP32.

Комплементарний фільтр — це простіша та менш вимоглива до ресурсів альтернатива. Його назва походить від принципу "доповнення" даних. Він працює як два окремі фільтри, що діють одночасно:

1) Фільтр низьких частот (Low-pass filter) застосовується до даних акселерометра, щоб усунути високочастотні шуми та вібрації, залишаючи лише повільні зміни, спричинені нахилом;

2) Фільтр високих частот (High-pass filter) застосовується до даних гіроскопа, щоб прибрати повільний дрейф і зберегти лише швидкі обертальні рухи [16].

Потім результати з обох фільтрів об'єднуються з використанням коефіцієнта α, що визначає, наскільки "довіряти" кожному сенсору. Проста математична формула і мінімальні обчислювальні вимоги роблять комплементарний фільтр ідеальним вибором для мікроконтролерних систем. Хоча він менш точний, ніж фільтр Кальмана, його простота та швидкість роботи є ключовими перевагами для багатьох практичних застосувань.

Вибір комплементарного фільтра для проєкту моніторингу позиції та руху тіла є оптимальним рішенням, що ґрунтується на балансі між точністю, простотою реалізації та обчислювальною ефективністю. На відміну від складніших алгоритмів, таких як фільтр Кальмана, який вимагає значних обчислювальних ресурсів і може бути надмірним для мікроконтролера ESP32, комплементарний фільтр є ідеальною альтернативою. Він забезпечує достатню точність для більшості практичних застосувань моніторингу руху і, що найважливіше, може бути легко реалізований на мікроконтролері з обмеженою потужністю. Його проста математична модель мінімізує ризик помилок у коді та забезпечує високу швидкість обробки даних, що є критичним для моніторингу в реальному часі.

Комплементарний фільтр працює за принципом об'єднання даних з двох сенсорів, що мають взаємодоповнюючі характеристики. Схема, що представлена, чітко відображає цей процес зображена на рисунку 2.2.

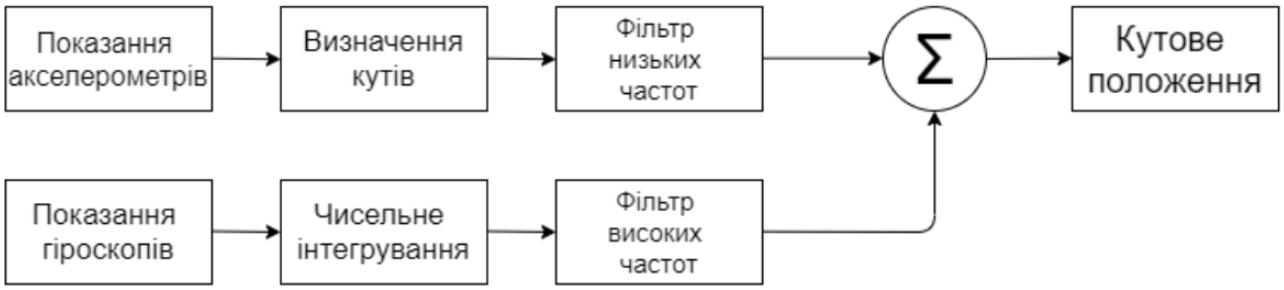


Рис 2.3 — схема роботи комплементарного фільтру

Верхня гілка (Акселерометр):

1) Показання акселерометрів отримують вихідні дані з сенсора;

2) Далі блок "Визначення кутів" перетворює ці показання (прискорення) у кути нахилу відносно сили тяжіння. Ці дані є точними для статичних положень, але містять шум, спричинений динамічними рухами;

3) Блок "Фільтр низьких частот" застосовується для усунення високочастотних шумів (вібрацій), що дозволяє отримати стабільну оцінку орієнтації в довготривалій перспективі.

Нижня гілка (Гіроскоп):

1) Показання гіроскопів надають дані про кутову швидкість;

2) Блок "Чисельне інтегрування" інтегрує ці значення, перетворюючи кутову швидкість у кут. Цей процес є точним для короткочасних змін, але з часом призводить до накопичення помилок (дрейфу);

3) Блок "Фільтр високих частот" усуває повільний дрейф, що накопичується під час інтегрування [21].

Обидві гілки алгоритму працюють паралельно, а їхні результати об'єднуються у блоці "∑" для отримання кінцевого "Кутового положення". Математично цей процес описується формулою:

$angle_{final} = (1 - \alpha) * (angle_{gyrocurrent} + \omega * \Delta t) + \alpha * angle_{accel}$

$angle_{final}$ – кінцеве положення кута;

$angle_{accel}$ – кут, отриманий з акселерометра;

$angle_{gyrocurrent} + \omega * \Delta t$ – новий кут отриманий з гіроскопа шляхом додавання зміненого кута, де ω — кутова швидкіть, Δt — інтервал часу:

α – коефіцієнт кута (значення від 0 до 1), що визначає співвідношення між даними гіроскопа і акселерометра. Чим ближче α до 1, тим більше фільтр "довіряє" акселерометру, і навпаки [22].

Описана модель дозволяє доповнювати переваги одного сенсора перевагами іншого, що призводить до стабільної, швидкої та точної оцінки орієнтації.

2.4 Моделювання взаємодії компонентів системи

Успішна робота системи моніторингу руху в реальному часі залежить не лише від вибору якісних компонентів, але й від ефективного моделювання їхньої взаємодії. Цей розділ присвячений детальному розгляду процесів обміну даними між ключовими апаратними та програмними елементами системи. Моделювання дозволяє візуалізувати потік інформації та логіку її обробки, що є основою для подальшої програмної реалізації.

Взаємодія між мікроконтролером ESP32 (обчислювальним модулем) та сенсором MPU-6050 (сенсорним модулем) є першим і найважливішим етапом у ланцюгу обробки даних. Цей зв'язок реалізується за допомогою цифрового послідовного інтерфейсу I2C (Inter-Integrated Circuit).

Процес взаємодії відбувається в такій у наступні чотири етапи:

1) Спочатку ESP32 відправляє на I2C-шину унікальну 7-бітну адресу MPU-6050 (0x68 або 0x69) та біт запису, щоб встановити зв'язок. Після цього відбувається налаштування сенсора: ESP32 надсилає команди, щоб вивести MPU-6050 з режиму сну, встановити чутливість (шкалу) акселерометра та гіроскопа, а також частоту дискретизації;

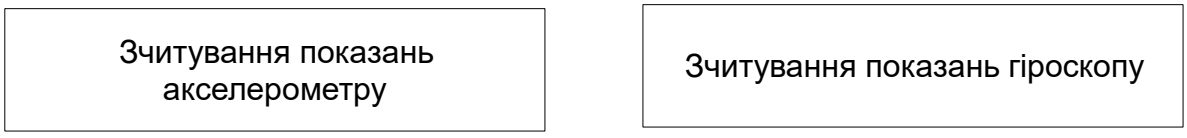
2) Коли сенсор налаштовано, ESP32 регулярно (наприклад, кожні 10 мс) надсилає запит на зчитування даних. Він вказує, з якого регістру MPU-6050 потрібно зчитати інформацію (наприклад, регістри даних акселерометра та гіроскопа);

3) Після отримання запиту MPU-6050 надсилає пакет даних, що містить сирі показники з трьох осей акселерометра (Ax, Ay, Az) та трьох осей гіроскопа (Gx, Gy, Gz). Кожне значення є 16-бітним цілим числом;

4) Отримавши сирі дані, ESP32 конвертує їх у фізичні одиниці (наприклад, м/с^2 для прискорення та градуси/с для кутової швидкості) з урахуванням встановленої шкали чутливості. На цьому етапі також може відбуватися первинна фільтрація або калібрування.

Взаємодія між акселерометром і гіроскопом відбувається не на фізичному, а на програмному рівні — в рамках алгоритму сенсорного ф’южну. Це моделюється як логічний процес, в якому дані з обох сенсорів комбінуються для отримання більш точного результату, ніж той, що може надати кожен сенсор окремо.

Взаємодію між акселерометром та гіроскопом зображено на рисунку 2.3



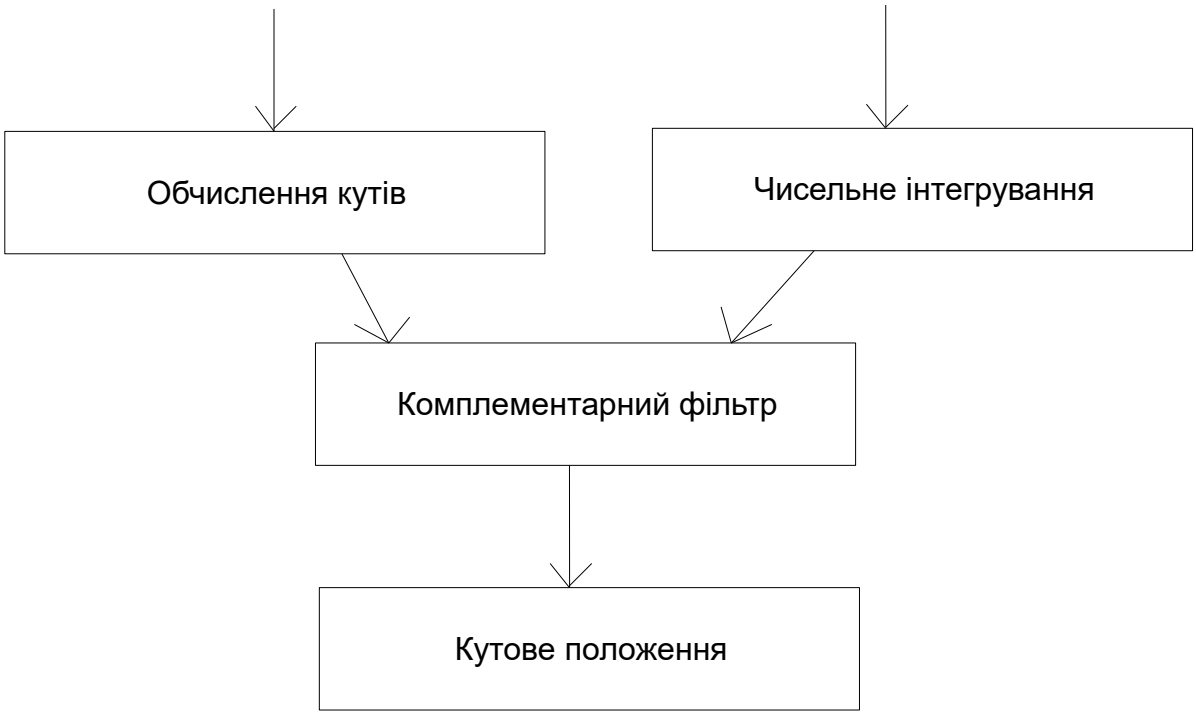


Рис 2.3 — взаємодія між акселерометром та гіроскопом

Ця схема ілюструє, що дані з кожного сенсора проходять свій власний етап обробки. Показання акселерометра перетворюються на кути нахилу (тангаж і крен) через тригонометричні функції, а показання гіроскопа (кутова швидкість) інтегруються з часом, щоб отримати кутові зміни.

Потім дані зливаються відбувається взаємодія за допомогою комплементарного фільтра, що є математичним алгоритмом. Він "доповнює" один тип даних іншим. Швидкі та точні дані гіроскопа використовуються для відстеження миттєвих змін орієнтації, тоді як повільні та стабільні дані акселерометра служать для корекції довготривалого дрейфу гіроскопа.

Результатом є точний, стабільний і швидкий показник кутового положення, який використовується для подальшої візуалізації.

2.5 Архітектура програми та організація обробки даних

Робота з інерціальним вимірювальним блоком (IMU) MPU-6050 вимагає чіткого структурування програмного коду та послідовного застосування алгоритмів для перетворення сирих даних на значущі показники орієнтації.

Архітектура програми побудована за стандартною моделлю Arduino, яка забезпечує фази ініціалізації та безперервної роботи, а також використовує модульний підхід для калібрування.

Перша фаза це фаза ініціалізації (функція setup()). Ця фаза виконується один раз при запуску контролера і відповідає за налаштування системного середовища та датчика. Спочатку відбувається встановлення послідовного зв'язку (Serial.begin) для налагодження та виведення результатів. Потім починається ініціалізація шини I2C (Wire.begin), яка є комунікаційним протоколом для обміну даними між ESP32 та MPU-6050. Для налаштування MPU-6050 використовується надсилання команд через I2C для виведення датчика з режиму сну (запис 0x00 у регістр 0x6B) та, за потреби, налаштування діапазону чутливості акселерометра та гіроскопа. Також на фазі ініціалізації проходить калібрування: виклик функції calculate_IMU_error() для визначення та корекції систематичних похибок (зміщення, або bias) акселерометра та гіроскопа.

Друга, основна фаза — це фаза основного циклу (функція loop()). Вона виконується безперервно і слугує ядром програми, реалізуючи алгоритм обробки даних, забезпечує послідовне виконання зчитування, розрахунку кутів та фільтрації. Також у ній Використовується логіка на основі часу (currentTime - lastPrintTime >= printInterval) для обмеження частоти виведення даних у послідовний порт, запобігаючи переповненню та забезпечуючи читабельність.

Модуль калібрування (функція calculate_IMY_error()) — цей окремий програмний модуль ізолює процес визначення систематичних похибок сенсора. Він виконує множинне зчитування сирих даних, коли датчик перебуває у статичному положенні, обчислює середнє значення та зберігає його як значення похибки для подальшої корекції.

Обробка даних включає кілька послідовних кроків, необхідних для отримання точних кутів орієнтації (крену - Roll, тангажу - Pitch, та рискання - Yaw).

Зчитування та Нормалізація Сирих Даних

1) I2C Зчитування здійснюється запит до відповідних регістрів MPU-6050 (0x3B для акселерометра, 0x43 для гіроскопа) для отримання 16-бітних сирих значень (X,Y,Z) для обох сенсорів;

2) Сирі цілочисельні значення перетворюються на фізичні одиниці шляхом ділення на коефіцієнти чутливості (наприклад, 16384.0 для акселерометра ±2g, 131.0 для гіроскопа ±250deg/s);

3) До отриманих значень гіроскопа застосовується поправка на систематичну похибку, розраховану під час калібрування [23].

Акселерометр використовується для визначення кутів крену та тангажу, ґрунтуючись на вимірюванні сили тяжіння:

Формула для розрахунку, яка використовує тригонометричні функції (arctan) зображена на рисунку 2.4

$$\text{accAngleX (Roll)} = \left(\arctan\left(\frac{\text{AccY}}{\sqrt{\text{AccX}^2 + \text{AccZ}^2}} \right) \cdot \frac{180}{\pi} \right) - \text{Похибка}$$

Рис 2.4 — формула для розрахунку кутів крену

Ці кути є точними у статиці, але нестабільні при лінійному прискоренні або вібраціях.

Гіроскоп вимірює кутову швидкість (швидкість обертання) і використовується для відстеження руху. Розрахунок інтервалу визначає час, що минув між послідовними циклами (elapsedTime = (currentTime - previousTime) / 1000.0).

Кутова швидкість інтегрується (множиться на час і додається до попереднього кута) для отримання кута зміщення: gyroAngleX=gyroAngleX+GyroX□elapsedTime

Кути, отримані гіроскопом, точні у динаміці, але накопичують похибку (дрейф) з часом.

Для поєднання переваг обох сенсорів використовується Комплементарний Фільтр (Complementary Filter). Цей алгоритм є ключовим у визначенні орієнтації.

Фільтр використовує дані акселерометра (точні на низьких частотах/у статиці) та гіроскопа (точні на високих частотах/у динаміці) з різною вагою [24-25].

Формула використання комплементарного фільтру зображена на рисунку 2.5

$$\text{Roll} = \mathbf{0.96} \cdot (\text{Roll(стар)} + \text{GyroX} \cdot \text{elapsedTime}) + \mathbf{0.04} \cdot \text{accAngleX}$$

Рис 2.5 — формула комплементарного фільтра

Перша складова формули — це дані гіроскопа (висока вага) 0.96□(Roll(стар) + GyroX□elapsedTime)

Ця частина використовує гіроскоп (GyroX) для визначення зміни кута:

1) Roll(стар) (або Pitch(стар)) — це кут, розрахований на попередньому кроці циклу;

2) GyroX * elapsedTime — це зміна кута за минулий проміжок часу. Гіроскоп вимірює кутову швидкість (град/сек), а множення на час (сек) дає зміну кута (град);

3) Множник 0.96 — це висока вага для гіроскопа. Вона означає, що 96% нового кута формується на основі вимірювань гіроскопа. Це гарантує швидку реакцію на рухи.

Друга складова формули — це дані акселерометра (низька вага) 0.04□accAngleX

Цю частину використовує акселерометр (accAngleX) для корекції:

1) accAngleX (або accAngleY) — це кут, розрахований на основі сили тяжіння (g) (Це статичний кут, який є точним у стані спокою (низькі частоти) і не має дрейфу);

2) Множник 0.04 — це низька вага для акселерометра. Вона означає, що лише 4% нового кута формується на основі акселерометра.

Комплементарний фільтр використовує гіроскоп для відстеження руху (швидка реакція) та акселерометр для повільної корекції дрейфу гіроскопа, фактично використовуючи його як "еталон горизонту" для низьких частот. Сума ваг завжди дорівнює 1.0 (0.96+0.04=1) [26].

2.6 Система передачі даних

Для бездротової передачі даних про орієнтацію, отриманих від сенсора MPU-6050, мікроконтролер ESP32 використовуватиме вбудований модуль Bluetooth Classic.

Bluetooth — це стандарт бездротової технології з коротким радіусом дії, призначений для обміну даними між фіксованими та мобільними пристроями.

Bluetooth працює у неліцензійному промисловому, науковому та медичному (ISM) радіодіапазоні 2.4 ГГц.

Пристрої формують невелику приватну мережу, відому як піконет (piconet), де один пристрій є Майстром (Master), а інші — Слейвами (Slaves).

Для роботи між двома пристроями необхідно провести процес створення пар (Pairing)

Створення пару (Pairing) – це процес, під час якого два Bluetooth-пристрої встановлюють криптографічно захищене з'єднання шляхом обміну та зберігання спільного секретного ключа, відомого як ключ зв'язку (link key). Це дозволяє їм аутентифікувати одне одного і автоматично відновлювати зашифрований зв'язок у майбутньому без додаткового втручання користувача, цей процес створюється за наступні 4 етапи:

1) Пристрій-Майстер (наприклад, смартфон або ПК) ініціює запит на з'єднання з ESP32:

2) Пристрої обмінюються інформацією та використовують спільний PIN-код або механізм Simple Secure Pairing (SSP) для автентифікації один одного;

3) Після успішної аутентифікації генерується ключ зв'язку, який зберігається в пам'яті обох пристроїв. Це дозволяє їм автоматично відновлювати з'єднання при наступних сеансах без повторного введення PIN-коду;

4) Для передачі даних (Roll, Pitch, Yaw) використовується профіль Serial Port Profile (SPP), який емулює стандартний послідовний порт (UART) по бездротовому каналу.

Після процесу створення пари відбувається процес встановлення захищеного з’єднання, на якому пристрій-майстер надсилає запит на з’єднання, пристрої обмінюються публічними ключами та підтверджують свою ідентичність і на основі обміненої інформації вже генерується довгостроковий ключ зв’язку (Link Key), який зберігається в енергозалежній пам’яті обох пристроїв. З цього моменту пристрої стають спареними (paired).

Оскільки ESP32 зазвичай не має дисплея чи клавіатури, він часто використовує модель Just Works або Passkey Entry (з фіксованим PIN-кодом), визначену в налаштуваннях прошивки. Пристрій-клієнт (наприклад, додаток на смартфоні) шукає ESP32 за його ім'ям (ESP32_MPU6050) і ініціює процедуру Pairing [27-29].

Для спрощення роботи з Bluetooth Classic на ESP32 у проекті використовується бібліотека BluetoothSerial.h (з набору ESP32 Arduino Core).

Ця бібліотека надає програмний інтерфейс, який дозволяє ставитися до Bluetooth-з'єднання як до звичайного послідовного порту (Serial Port). Це спрощує розробку, оскільки для відправки даних можна використовувати знайомі функції, такі як print() та println().

Основні функції бібліотеки BluetoothSerial.h подано у таблиці 2.2

Функція	Призначення
SerialBT.begin(name)	Ініціалізує Bluetooth-модуль, робить пристрій видимим для пошуку та встановлює його ім'я
SerialBT.print()	Надсилає дані (у нашому випадку, кути) через Bluetooth.
SerialBT.available()	Перевіряє, чи є дані, що надійшли від Майстра.
SerialBT.read()	Зчитує дані, отримані через Bluetooth.

Для оголошення об’єкту BluetoothSerial створюється екземпляр класу BluetoothSerial з назвою SerialBT.

SerialBT є об'єктом, через який програма буде викликати всі функції для керування та обміну даними по Bluetooth (наприклад, SerialBT.begin(), SerialBT.print()).

Після цього оголошується константна змінна, яка зберігає ім’я Bluetooth-пристрою (const char* bt_device_name = "ESP32_MPU6050";). При пошуку доступних пристроїв на смартфоні чи комп'ютері користувач побачить саме це ім'я, що дозволяє легко ідентифікувати ESP32.

Для ініціалізації BluetoothSerial викликається функція SerialBT.begin, вона запускає апаратний Bluetooth-модуль ESP32. Передане ім’я (у нашому випадку: ESP32_MPU6050) встановлюється як видиме ім’я пристрою [30-31].

Після виклику функції SerialBT.begin ESP32 починає транслювати свою присутність і готовий приймати запити на з’єднання від інших пристроїв.

Висновки до 2 розділу

У розділі 2 було проведено детальне проектування апаратної та програмної складових системи визначення орієнтації на базі мікроконтролера ESP32 та інерціального вимірювального блоку MPU-6050.

Розроблено функціональну та структурну схеми підключення. Вони чітко визначили апаратну взаємодію: ESP32-C3 виступає як керуючий мікроконтролер, а MPU-6050 — як датчик. Зв'язок між ними реалізується через швидкий, двонаправлений інтерфейс I2C, використовуючи лише дві лінії (SDA та SCL) для обміну даними.

Описано принцип роботи сенсорів MPU-6050. Акселерометр вимірює лінійне прискорення по трьох осях, що використовується для розрахунку статичних кутів (крену та тангажу) на основі сили тяжіння. Гіроскоп вимірює кутову швидкість по трьох осях, що дозволяє відстежувати динамічні зміни кутів. Обидва сенсори надають сирі 16-бітні дані, які потребують подальшої обробки. Для подолання

недоліків окремих сенсорів (дрейф гіроскопа та чутливість акселерометра до шуму) обрано Комплементарний Фільтр. Цей алгоритм поєднує дані: інтегрованому гіроскопу надається висока вага (для відстеження швидкого руху), а статичному куту від акселерометра — низька вага (для повільної корекції накопиченого дрейфу гіроскопа). Це забезпечує стабільні та точні значення кутів Roll та Pitch. Системна взаємодія компонентів побудована на моделі, де ESP32-C3 ініціює та керує всіма операціями. Циклічно виконуються:

1) ESP32 запитує сирі дані у MPU-6050;

2) Застосовується масштабування, корекція похибок та комплементарна фільтрація;

3) Фінальні кути орієнтації передаються на зовнішній пристрій через Bluetooth.

Архітектура програми реалізована за структурою Arduino: фаза ініціалізації (setup()) для налаштування I2C, сенсора та калібрування, та фаза основного циклу (loop()) для безперервного зчитування, обробки та виведення. Організація обробки даних включає сувору послідовність кроків:

1) Зчитування;

2) Масштабування;

3) Корекція похибок;

4) Обчислення статичних та динамічних кутів;

5) Застосування комплементарного фільтра.

Для бездротової передачі результатів обрано вбудований у ESP32 модуль Bluetooth Classic з використанням профілю Serial Port Profile (SPP). Це дозволяє емулювати послідовний порт. Використання бібліотеки BluetoothSerial.h значно спрощує процес. Описано процедуру створення пару (Pairing), яка передбачає обмін ключами та аутентифікацію для встановлення захищеного з'єднання між ESP32 та зовнішнім пристроєм.

Комплексна робота, проведена у другому розділі, дозволила повністю спроектувати та теоретично обґрунтувати функціональну основу системи. Було визначено оптимальні апаратні компоненти (ESP32-C3 та MPU-6050), обрано надійний алгоритм обробки даних (Комплементарний Фільтр) для забезпечення точності, і розроблено архітектуру програми з інтеграцією бездротового зв'язку (Bluetooth). Усі ці кроки закладають міцну основу для подальшої практичної реалізації та тестування системи.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ МОНІТОРИНГУ ПОЗИЦІЇ ТІЛА В ПРОСТОРІ

3.1 Підготовка обладнання

Розробка та реалізація системи моніторингу позиції та руху тіла вимагає ретельної підготовки апаратного забезпечення. На цьому етапі буде підготовлено всі необхідні компоненти, що забезпечують збір, обробку та передачу даних.

Етап підготовки обладнання включає ідентифікацію, перевірку та фізичне з'єднання основних компонентів системи: плати мікроконтролера ESP32 та інерціального вимірювального блоку MPU-6050.

Список усіх апаратних компонентів, що використовуються для створення прототипу:

1. Плата Мікроконтролера ESP32 (з модулем ESP-WROOM-32) на базі мікросхеми ESP32-D0WDQ6. Плата оснащена розширювальною платою, яка полегшує доступ до всіх виводів GPIO. Призначена для забезпечення керування, виконання алгоритму Комплементарного фільтра та бездротової передачі даних через Bluetooth;

2. Інерційний сенсор MPU-6050, шестиосьовий IMU, що містить 3-осьовий акселерометр та 3-осьовий гіроскоп. Призначений для вимірювання прискорення та кутової швидкості для визначення орієнтації об'єкта;

3. З’єднувальні дроти, чотири провідники залежно від конфігурації пінів MPU-6050 для встановлення інтерфейсу I2C та живлення;

4. Кабель для підключення з роз’ємами USB 2.0 (для підключення до хост пристрою — комп’ютера або зарядного пристрою) та USB mini (для підключення безпосередньо до плати ESP32). Призначений для підключення плати до джерела живлення та передачі даних. Кабель виконує дві критично важливі функції: забезпечує необхідну напругу 5 В для роботи плати ESP32 та створює віртуальний послідовний порт через вбудований USB-UART мікросхему що дозволяє завантажувати програмний код на ESP32 та отримувати налагоджувальну інформацію.

Фотографія плати ESP32 зображена на рисунку 3.1

Рис 3.1 — ESP32

Рис 3.2 — MPU-6050

Фотографія MPU-6050 з підключеними проводами зображена на рисунку 3.2

Фізичне з'єднання компонентів було виконано згідно зі структурною схемою, розробленою у другому розділі, використовуючи інтерфейс I2C, який вимагає чотирьох основних підключень: VCC, GND, SCL, SDA.

Інформацію про з’єднання пінів з плати та інерційного сенсора подано у таблиці 3.1

Пін MPU-6050	Функція	Пін ESP32 (Типове I2C)
VCC	Живлення (+3.3V)	3V3
GND	Заземлення	GND
SCL	Лінія Тактування I2C	GPIO 22
SDA	Лінія Даних I2C	GPIO 21

Етапи підключення:

1) Живлення — пін VCC датчика MPU-6050 підключений до піна 3V3 на платі ESP32. Це забезпечує необхідне для роботи датчика стабілізоване живлення напругою 3.3 В;

2) Заземлення — пін GND датчика підключений до спільного піна GND на платі ESP32, встановлюючи спільний нульовий потенціал;

3) Лінія Тактування (SCL) — лінія SCL підключена до виводу GPIO 22 на ESP32, цей вивід слугує для синхронізації передачі даних;

4) Лінія Даних (SDA) — лінія SDA підключена до виводу GPIO 21 на ESP32, по цій лінії відбувається безпосередній обмін даними вимірювань між мікроконтролером і сенсором [32-34].

На платі ESP32, виводи GPIO 21 та 22 зручно розташовані на розширювачі та були обрані як типові піни для апаратного I2C 0.

У результаті фізичної реалізації було отримано компактний апаратний вузол, готовий до програмування. Налагоджений I2C-зв'язок дозволив мікроконтролеру ESP32 зчитувати 16-бітні сирі дані з регістрів MPU-6050.

3.2 Програмування системи в Arduino IDE

Програмна реалізація системи розпочинається з налаштування середовища розробки Arduino IDE, підключення усіх необхідних для роботи бібліотек та написання базового коду, що забезпечує зв'язок між мікроконтролером ESP32 та датчиком MPU-6050. На даному етапі необхідно реалізувати: оголошення

глобальних змінних, ініціалізацію послідовного зв'язку та шини I2C, а також функціонал для зчитування сирих даних.

На початку коду оголошуються всі змінні, які використовуються в програмі, забезпечуючи їх доступність у функціях setup() та loop(). Всі змінні для вимірювань оголошені як float для підтримки точності обчислень з плаваючою комою.

Перелік усіх глобальних змін та їх призначення подано у таблиці 3.2

Група Змінних	Призначення	Приклади Змінних
I2C та Регістри	Адреса датчика MPU-6050.	const int MPU = 0x68;
Сирі та Масштабовані Дані	Значення прискорення та кутової швидкості по 3-х осях.	AccX, AccY, GyroZ;
Розрахункові Кути	Проміжні кути від акселерометра (accAngle) та гіроскопа (gyroAngle).	accAngleX, gyroAngleY;
Фінальні Кути	Кути, отримані після фільтрації (крену, тангажу, рискання).	roll, pitch, yaw;
Помилки та Калібрування	Значення систематичної похибки, отримані при калібруванні.	AccErrorX, GyroErrorZ;
Управління Часом	Змінні для вимірювання інтервалу між циклами (elapsedTime) та контролю виведення.	currentTime, previousTime, printInterval;

Функція setup() виконується лише один раз після завантаження програми і відповідає за підготовку комунікаційних інтерфейсів та конфігурацію датчика MPU-6050. Елементи ініціалізації у функції setup():

1) Послідовний зв'язок — Serial.begin(9600), ініціалізує комунікацію з комп'ютером через USB на швидкості 9600 бод для налагодження;

2) I2C ініціалізація — Wire.begin(), активує апаратний інтерфейс I2C на пінах GPIO 21 (SDA) та GPIO 22 (SCL), як було визначено апаратною схемою;

3) Конфігурація для MPU-6050 — Wire.beginTransmission(MPU), функція використовується для звернення до датчика. Запис 0x00 у регістр 0x6B (PWR_MGMT_1) є критичним кроком, оскільки він виводить датчик зі сплячого режиму, роблячи його доступним для зчитування даних [35].

Функція loop() (функція основного циклу) виконується постійно і містить логіку для безперервного зчитування даних з акселерометра та гіроскопа. На цьому етапі вона містить лише операції зчитування та початкового масштабування.

Акселерометр зчитується першим, оскільки він надає статичні кути, необхідні для корекції дрейфу гіроскопа.

Порядок операцій при зчитувані даних з гіроскопа:

1) Wire.write(0x3B) — запис у регістр 0x3B, це старший байт вимірювання прискорення по осі X (ACCEL_XOUT_H). Оскільки дані для X, Y, Z розміщені послідовно, запит на 6 байтів отримує всі три виміри;

2) Оператор (Wire.read() << 8 | Wire.read()) — цей оператор об'єднує два послідовно зчитаних байти (старший байт << 8 та молодший байт) у єдине 16-бітне ціле число;

3) AccX = (Wire.read() << 8 | Wire.read()) / 16384.0 (аналогічно для AccX, AccY, AccZ) – масштабування 16384, отримане 16-бітне число ділиться на коефіцієнт чутливості 16384.0 (для діапазону +-2), перетворюючи його на фізичну одиницю прискорення [36].

Після зчитування даних з акселерометра відбувається зчитування сирих даних з гіроскопа.

Порядок операцій при зчитуванні даних з гіроскопа:

1) Розрахунок часу виконання у секундах (elapsedTime = (currentTime - previousTime) / 1000.0;) – перед зчитуванням гіроскопа обчислюється elapsedTime (час, що минув) у секундах, необхідний для точного інтегрування кутової швидкості;

2) Wire.write(0x43) — запис у регістр 0x43, зчитування починається з регістру GYRO_XOUT_H;

3) GyroX = (Wire.read() << 8 | Wire.read()) / 131.0 (аналогічно для GyroX, GyroY, GyroZ) — масштабування 131.0 (відбувається за тим самим принципом, що і у акселерометра), сирі дані гіроскопа діляться на коефіцієнт 131.0$(для типового діапазону +-250=), перетворюючи їх на кутову швидкість в градусах за секунду [37].

Цей базовий код закладає функціональний фундамент, забезпечуючи надійний обмін даними з MPU-6050, що є необхідною умовою для подальшої реалізації складних алгоритмів фільтрації та бездротової передачі.

3.3 Реалізація алгоритмів обробки даних з MPU-6050

Ключовим етапом програмної реалізації є перетворення сирих, зашумлених вимірювань MPU-6050 на стабільні та точні кути орієнтації. Цей процес охоплює початкову ініціалізацію, корекцію систематичних похибок та застосування комплементарного фільтра.

Для коректної роботи алгоритму фільтрації необхідно задати початкові значення кутів крену (roll) та тангажу (pitch). Це виконується у функції setup() шляхом одноразового зчитування даних акселерометра за наступні три етапи роботи (розрахунок, корекція, ініціалізація):

1) Для розрахунку використовуються тригонометричні формули, arctan для визначення кутів, виходячи з орієнтації вектора сили тяжіння (g);

2) Для корекції застосовується віднімання похибок AccErrorX та AccErrorY, отриманих при калібруванні, для встановлення точного нуля у початковому статичному положенні;

3) Для ініціалізації кут рискання (yaw) встановлюється на 0.0, оскільки акселерометр не може його виміряти. Всі подальші зміни yaw будуть базуватися виключно на інтегруванні гіроскопа.

Основна логіка обробки даних реалізується в нескінченному циклі loop(), який включає чотири ключові етапи: корекцію гіроскопа, комплементарну фільтрацію та обмеження кута рискання:

1) Корекція похибок гіроскопа відбувається після зчитування сирих даних гіроскопа та їх масштабування до deg/s, застосовується корекція зміщення, розрахована раніше.

Код роботи корекції помилок у програмі:

```
// Correct the outputs with the calculated error values
GyroX = GyroX - GyroErrorX;
GyroY = GyroY - GyroErrorY;
GyroZ = GyroZ - GyroErrorZ;
```

Від кожного виміряного значення кутової швидкості віднімається його середня статична похибка (GyroErrorX/Y/Z). Це гарантує, що у стані спокою датчик видає нульові значення, запобігаючи миттєвому дрейфу;

2) Комплементарний фільтр — це основний алгоритм, що комбінує дані гіроскопа та акселерометра для отримання стабільних кутів roll та pitch.

Код роботи комплементарного фільтра у програмі:

```
// Complementary filter - combine acceleromter and gyro angle values
roll = 0.96 * (roll + GyroX * elapsedTime) + 0.04 * accAngleX;
pitch = 0.96 * (pitch + GyroY * elapsedTime) + 0.04 * accAngleY;
```

Висока вага 0.96 надається інтегрованому гіроскопу: (roll + GyroX * elapsedTime). Це забезпечує швидку реакцію на динамічні рухи, оскільки гіроскоп реагує миттєво.

Низька вага 0.04 надається куту accAngleX. Ця частина відповідає за повільну корекцію дрейфу гіроскопа, використовуючи акселерометр як надійний, недрейфуючий орієнтир горизонту.;

3) Інтегрування кута рискання (yaw) розраховується виключно шляхом інтегрування (накопичення) кутової швидкості гіроскопа навколо вертикальної осі Z.Код роботи інтегрування кута рискання у програмі:

// Інтегрування для yaw

yaw = yaw + GyroZ * elapsedTime

Оскільки для осі Z немає корекції від акселерометра, цей кут є сумою всіх попередніх зміщень;

4) Обмеження кута рискання, оскільки інтегрування Yaw може призвести до необмеженого накопичення значень (наприклад, 360, 720 градусів), застосовується операція обмеження, що підтримує кут у діапазоні від -180 до +180 градусів [38].

Код роботи обмеження кута рискання у програмі:

// Обмежуємо yaw від -180 до 180

if (yaw > 180) yaw -= 360;

if (yaw < -180) yaw += 360;

Блок-схему основного алгоритму обробки даних з MPU-6050 зображено на рисунку 3.3

Рис 3.3 — блок-схема обробки даних з MPU-6050

Виходячи з блок-схеми можна зробити висновок, що дані обробляються лінійно.

3.4 Реалізація передачі даних за допомогою безпровідної технології Bluetooth

Етап реалізації бездротової передачі даних є критично важливим для забезпечення мобільності та зручності використання системи моніторингу орієнтації. Для цього було використано вбудований Bluetooth-модуль

мікроконтролера ESP32 та профіль Serial Port Profile (SPP), який емулює стандартний послідовний порт.

Для роботи з Bluetooth Classic в Arduino IDE необхідно підключити спеціалізовану бібліотеку BluetoothSerial.h та ініціалізувати об'єкт зв'язку.

Бібліотека BluetoothSerial.h надає всі необхідні функції для управління Bluetooth-модулем ESP32, дозволяючи використовувати звичні методи print() та println() для бездротової передачі.

Запуск зв’язку відбувається у функції setup(), у якій ініціалізується апаратний Bluetooth-модуль, що і робить його видимим та готовим до прийому запитів на з'єднання.

Повідомлення про підтвердження успішної ініціалізації та ім’я користувача виводяться через Serial.print.

Передача обчислених кутів roll, pitch та yaw здійснюється всередині функції loop(). Для ефективності та запобігання переповненню послідовного порту, передача даних обмежена інтервалом в 1 секунду (printInterval = 1000).

Передача даних у функції loop() відбувається за наступних чотири етапи:

1) Усі три кути об'єднуються в єдиний рядок формату "Roll/Pitch/Yaw" (наприклад, "1.54/88.30/12.01"). Використання роздільника (слеш /) та фіксованої точності (String(..., 2)) дозволяє зовнішньому додатку легко розпарсити (розділити) рядок на окремі числові значення;

2) Serial.println(output_data); продовжує виводити дані через USB для паралельного контролю;

3) Відбувається умовна Bluetooth-передача — if (SerialBT.hasClient()). Вона перевіряє, чи активний Bluetooth-модуль має встановлене з'єднання із зовнішнім пристроєм. Це запобігає спробам відправки даних у неіснуючий канал, що значно підвищує стабільність роботи програми;

4) Безпосередня передача відбувається за допомогою функції SerialBT.println(output_data), яка і надсилає сформований рядок на спарений пристрій.

Для первинного тестування та моніторингу даних можна використовувати функціонал послідовного порту комп'ютера, який емулюється Bluetooth SPP та подивитися на дані через консоль.

Процедура підключення пристрою до комп’ютера та моніторинг даних через консоль відбувається у наступні 4 етапи:

1) На комп'ютері (Windows, macOS, Linux) необхідно активувати Bluetooth і провести пошук доступних пристроїв. У списку буде знайдено пристрій з іменем ESP32_MPU6050;

2) Ініціювання створеної пари, зазвичай, ESP32 використовує режим Just Works або запитує стандартний PIN-код (наприклад, 0000, 1234, або може зовсім його не запросити);

3) Після успішного спарювання відбувається визначення віртуального порту, операційна система створює віртуальний COM-порт (наприклад, COM7 або /dev/tty.ESP32) для Bluetooth-з'єднання;

4) Після цього необхідно відкрити будь-який термінальний емулятор (наприклад, PuTTY, RealTerm, або вбудований в ОС інструмент), вказати номер цього Віртуального COM-порту та встановити швидкість 9600 бод (ця швидкість не має значення для Bluetooth SPP, але її потрібно вказати для коректної роботи терміналу) [39-40].

Після встановлення Bluetooth-з'єднання та відкриття віртуального порту, консоль буде відображати потік даних з інтервалом в 1 секунду.

Результат виведення передачі даних через Bluetooth зображено на рисунку 3.4

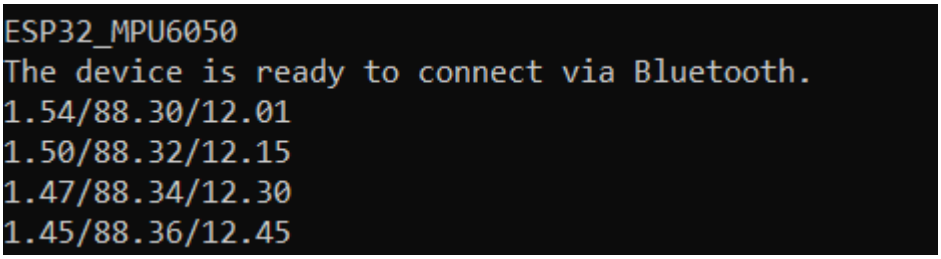


Рис 3.4 — результат виведення передачі даних через Bluetooth

Кожен рядок містить поточні значення Roll, Pitch, Yaw з точністю до двох знаків після коми, розділені символом /. Це підтверджує, що ініціалізація та передача даних через Bluetooth працюють коректно.

3.5 Тестування системи та калібрування помилок

Перед початком експлуатації системи, критично важливо провести тестування та калібрування MPU-6050. Це необхідно для визначення та усунення систематичних похибок, які виникають через виробничі допуски сенсорів. Якщо ці похибки не будуть скориговані, вони призведуть до постійного зміщення кутів та швидкого дрейфу (yaw).

Калібрування системи є критично важливим етапом роботи для отримання точних значень.

Перед калібруванням необхідно провести тестування системи та проаналізувати дані, які виводяться у стані спокою.

Для цього необхідно запустити усю систему, як було показано у минулих кроках.

За умовами, що датчик MPU-6050 повинен бути розміщений на абсолютно рівній горизонтальній поверхні та перебувати у повному спокої протягом усього процесу калібрування. Значення акселерометра у такому статичному положенні ідеальні кути крену та тангажу мають дорівнювати 0 градусів. Значення гіроскопа у аналогічному статичному положенні ідеальні кути крену та тангажу мають також дорівнювати 0 градусів.

Приклад виводу даних на консоль (roll / pitch / yaw) без корекції зображено на рисунку 3.5

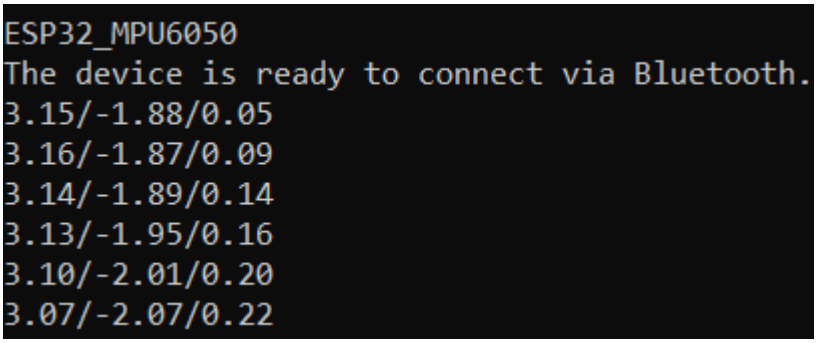


Рис 3.5 — виведення даних до корекції

Статичні кути не дорівнюють 0.00, що саме і вказує на наявність похибки, також з часом похибка накопичується, що призводить до отримання все більш неправильних даних з більшим часом роботи системи.

Калібрування проводиться у функції calculate_IMU_error() та базується на принципі вимірювання середнього зміщення у статичному положенні.

Калібрувальні похибки застосовуються у двох ключових місцях коду:

1) В setup(): для встановлення коректного початкового кута roll та pitch системи;

2) В loop() (кожен цикл): для корекції поточних кутів акселерометра та кутової швидкості гіроскопа перед застосуванням комплементарного фільтра [41-42].

Блок-схема роботи функції calculate_IMU_error() зображена на рисунку 3.6

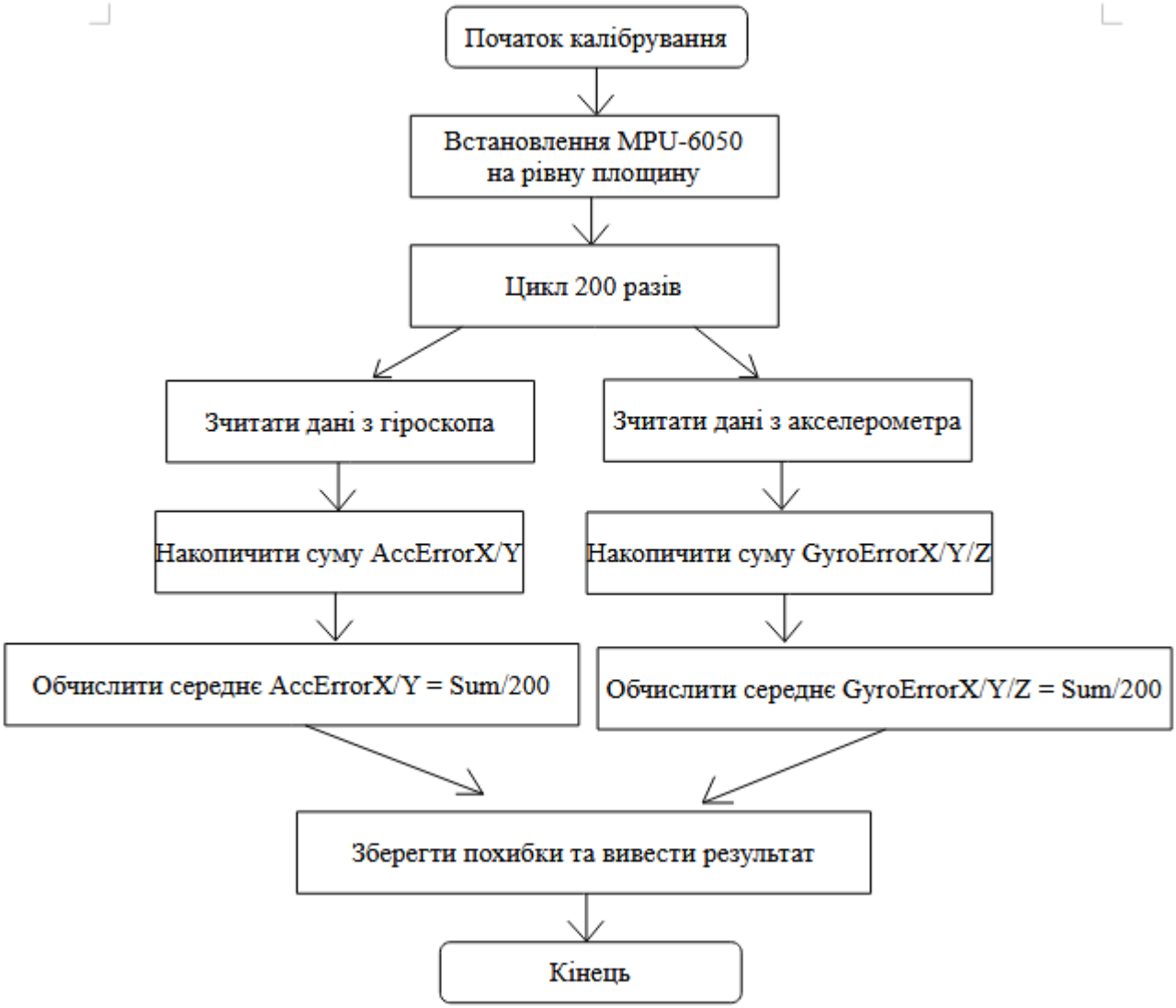


Рис 3.6 — блок-схема функції calculate_IMU_error()

Сенсор зчитує 200 вимірювань, розраховує початкові кути, і сумує їх. Отримана середня різниця від 0 градусів і стають похибкою акселерометра та гіроскопа.

Накопичуються фактичні кути (Roll та Pitch), отримані за допомогою функції arctan, коли датчик перебуває в нульовому положенні. Це дозволяє визначити зміщення від ідеального 0 градусів.

Накопичуються масштабовані, але некориговані значення кутової швидкості (deg/s). Ці значення є ненульовим зміщенням, яке має бути відняте в процесі роботи.

Отримані змінні AccError та GyroError зберігаються і застосовуються в основному циклі (loop()) для корекції всіх майбутніх вимірювань [43].

Приклад виведення результатів у консоль після калібрування зображено на рисунку 3.7

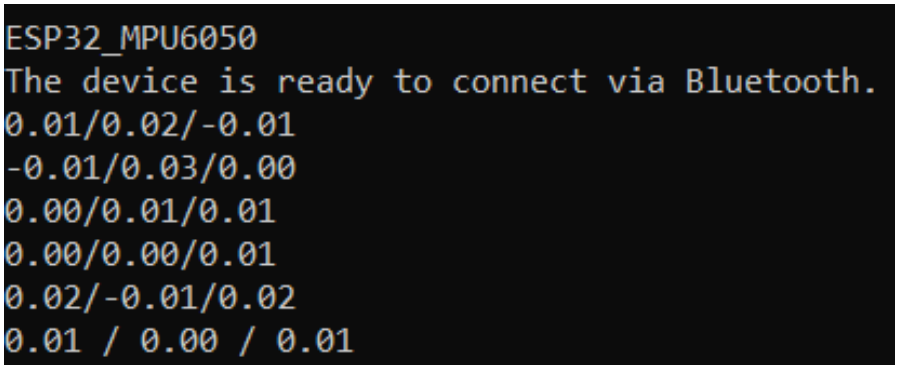


Рис 3.7 — виведення даних після калібрування

Усі три кути максимально наближені до 0.00, що підтверджує успішне усунення систематичних похибок. Це критично важливо для забезпечення довгострокової стабільності даних.

Висновки до 3 розділу

У розділі 3 було проведено повну практичну реалізацію системи моніторингу тіла в просторі, що перетворило теоретичні схеми та алгоритми, розроблені у другому розділі, на функціональний апаратно-програмний комплекс. Було успішно здійснено збірку обладнання, розроблено програмне забезпечення в середовищі Arduino IDE, інтегровано алгоритми обробки даних та налаштовано бездротовий зв'язок.

Етап підготовки розпочався з ідентифікації та перевірки всіх компонентів, включаючи мікроконтролер ESP32 (з розширювальною платою), датчик MPU-6050 та необхідні з'єднувальні провідники. Апаратна збірка була виконана шляхом встановлення зв'язку між ESP32 та MPU-6050 через інтерфейс I2C. Зокрема, було правильно підключено лінії живлення (3V3 та GND) та комунікаційні лінії SDA (GPIO 21) і SCL (GPIO 22). Фізичне з'єднання забезпечило надійний канал для

обміну сирими даними вимірювань, створюючи мінімальний, але функціональний апаратний вузол.

Програмна реалізація заклала основу для керування системою. У функції setup() було проведено ініціалізацію послідовного зв'язку, шини I2C та, що критично важливо, виведено MPU-6050 зі сплячого режиму, що зробило його доступним для запитів. Основний цикл loop() був запрограмований для безперервного зчитування сирих 16-бітних даних з акселерометра та гіроскопа, включаючи їхнє масштабування до фізичних одиниць (g та deg/s).

Ключовим досягненням стала реалізація комплементарного фільтра. Цей алгоритм вирішив проблему дрейфу, характерну для інерціальних датчиків. Шляхом надання 96% ваги гіроскопу (для динаміки) та 4% ваги акселерометру (для корекції горизонту) було досягнуто значної стабілізації кутів roll та pitch. Кут yaw був реалізований шляхом інтегрування гіроскопа з подальшим обмеженням діапазону від -180 градусів до +180 градусів.

Бездротова передача даних була реалізована за допомогою бібліотеки BluetoothSerial.h, яка дозволила перетворити Bluetooth-модуль ESP32 на емулятор послідовного порту (SPP). У setup() було ініціалізовано об'єкт SerialBT та встановлено ім'я пристрою (ESP32_MPU6050). У циклі loop(), обчислені кути форматувалися в єдиний рядок "Roll/Pitch/Yaw" і надсилалися на спарений клієнт, забезпечуючи моніторинг у реальному часі з інтервалом в 1 секунду.

Завершальним етапом було тестування та калібрування. Була розроблена функція calculate_IMU_error(), яка автоматично визначає систематичні похибки акселерометра та гіроскопа шляхом усереднення 200 статичних вимірювань. Вивід даних на консоль підтвердив, що до калібрування у стані спокою спостерігалися значні зміщення, тоді як після застосування розрахованих похибок всі кути були ефективно приведені до 0.00 +- похибка шуму. Успішне калібрування гарантує високу точність та мінімізує накопичення помилок у фінальних даних орієнтації.

Таким чином, третій розділ успішно довів можливість практичної реалізації системи моніторингу. Збірка обладнання, інтеграція Комплементарного фільтра та налаштування надійного Bluetooth-зв'язку створили повноцінний, автономний пристрій, здатний перетворювати сирі дані сенсорів на стабільну інформацію про орієнтацію, готову до подальшого використання у зовнішніх моніторингових додатках.

ВИСНОВОК

Проведена робота була спрямована на розробку та практичну реалізацію високоточного, економічно ефективного прототипу системи для моніторингу орієнтації тіла (або окремих його сегментів) у тривимірному просторі. Поставлені цілі були повністю досягнуті шляхом поетапного виконання завдань, починаючи з глибокого теоретичного аналізу та закінчуючи детальною програмною та апаратною реалізацією.

На початковому етапі було проведено комплексний огляд та аналіз предметної області. Розглянуто сучасні підходи та технології визначення орієнтації, включаючи оптичні, магнітні та інерціальні системи. Було підтверджено, що системи на базі інерціальних вимірювальних блоків (IMU) є найбільш оптимальним рішенням для мобільного моніторингу завдяки їхній компактності, автономності та відсутності необхідності у зовнішній інфраструктурі.

Серед мікроконтролерів було обрано ESP32. Вибір обґрунтовано оптимальним співвідношенням обчислювальної потужності, низького енергоспоживання та, найважливіше, інтегрованим модулем Bluetooth Classic, що є необхідним для бездротової передачі даних.

Після аналізу методів злиття даних (зокрема, порівняння з фільтром Калмана) було обрано комплементарний фільтр. Це рішення забезпечило необхідний баланс між точністю та обчислювальною ефективністю, дозволяючи виконувати всі складні математичні операції без перевантаження ресурсів мікроконтролера.

Цей теоретичний фундамент дозволив перейти до проектування системи, базуючись на найкращих доступних рішеннях, що відповідають вимогам проекту.

На етапі теоретичної реалізації було розроблено загальну архітектуру проекту:

1) Розроблено схеми, що чітко визначають апаратну взаємодію: мікроконтролер ESP32 виступає як керуючий зчитуванням даних з датчика MPU-6050 через інтерфейс I2C. Це забезпечило високу швидкість та надійність обміну даними;

2) Детально описано функціональність акселерометра (визначення статичних кутів через вектор сили тяжіння) та гіроскопа (вимірювання динамічної кутової швидкості);

3) Обґрунтовано принцип злиття даних, де гіроскопу надається висока вага (для відстеження швидких змін), а акселерометру — низька вага (для повільної корекції дрейфу). Це гарантувало, що фінальні кути roll та pitch будуть вільні від швидкого шуму та довгострокового дрейфу.

Практичний розділ роботи підтвердив життєздатність розробленої архітектури:

1) Успішно підготовлено та з'єднано всі компоненти, включаючи ESP32, MPU-6050 та налагоджувальний кабель. Апаратна взаємодія по I2C була встановлена на пінах GPIO 21/22;

2) Реалізовано код для низькорівневої комунікації з MPU-6050 (зчитування та масштабування 16-бітних даних);

3) Комплементарний Фільтр був реалізований із ваговими коефіцієнтами, які забезпечили ефективне злиття даних, та інтегрування кута yaw (рискання);

4) Інтеграція технології Bluetooth SPP дозволила створити бездротовий канал передачі даних. Обчислені кути roll, pitch та yaw були форматовані в єдиний рядок та передавалися на зовнішній клієнт, що підтвердило можливість віддаленого моніторингу;

5) Проведено критично важливий етап тестування та калібрування. Спеціально розроблена функція calculate_IMU_error() автоматично виявила та розрахувала систематичні похибки акселерометра та гіроскопа. Аналіз консольного виводу до та після калібрування довів, що цей крок успішно усунув початкове зміщення, забезпечивши стабільність показань.

У результаті проведеної роботи було створено повністю працюючий прототип системи моніторингу орієнтації тіла в просторі, який успішно та надійно зчитує дані з MPU-6050 та забезпечує бездротову передачу даних через Bluetooth.

Подальша робота може бути зосереджена на розробці спеціалізованого мобільного додатку для візуалізації отриманих даних та на інтеграції додаткових фільтрів (наприклад, фільтра Калмана) для порівняння продуктивності, якщо дозволяє обчислювальна потужність.

Загальний висновок полягає в тому, що поставлене завдання створення функціональної та надійної системи моніторингу позиції та руху тіла в реальному часі було успішно виконано, що підтверджує готовність розробленого рішення до подальшого впровадження.

Перелік використаних джерел:

1. Інерційний вимірювальний пристрій (ІВП). URL: https://www.wikidata.uk-ua.nina.az/Інерційний_вимірювальний_пристрій.html (дата звернення: 11.11.2024)

2. Stancic I., Grujic Supuk T., Panjkota A. (2013). Design, development and evaluation of optical motion-tracking system based on active white light markers. IET Science, Measurement & Technology, 7(4), 206-214.

3. Основи радіолокації, активна радіолокація URL: https://www.radartutorial.eu/02.basics/rp04.uk.html (дата звернення 12.11.2024)

4. MPU-6050 GY-521 datasheet URL: https://arduino.ua/docs/PS-MPU-6000A.pdf (дата звернення 07.01.2025).

5. ESP32-C3 documentation URL: https://github.com/01Space/ESP32-C3-0.42LCD (дата звернення 07.01.2025).

6. PlatformIO documentation URL: https://github.com/01Space/ESP32-C3-0.42LCD (дата звернення 10.01.2025).

7. Arduino IDE documentation URL: https://docs.arduino.cc/ (дата звернення 10.01.2025).

8. Stochastic Models, Estimation, and Control, vol. 1, Peter S. Maybeck URL: https://www.cs.unc.edu/~welch/kalman/media/pdf/maybeck_ch1.pdf (дата звернення 11.01.2025).

9. A New Approach to Linear Filtering and Prediction Problems, Рудольф Калман, 1960 рік URL: https://www.cs.unc.edu/~welch/kalman/kalmanPaper.html (дата звернення 11.01.2025).

10. Примушко А.,М., Рижков Л.М. Дослідження комплементарного фільтра на МЕМС-вимірювачах//Інформаційні системи, механіка та керування. 2019. Випуск 20. С.47-53.

11. Abedi, Ali, Omid Abari, and Tim Brecht. "Wi-le: Can wifi replace bluetooth?." *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*. 2019.

12. Friedman, Roy, Alex Kogan, and Yevgeny Krivolapov. "On power and throughput tradeoffs of wifi and bluetooth in smartphones." *IEEE Transactions on Mobile Computing* 12.7 (2012): 1363-1376.

13. Danbatta, Salim Jibrin, and Asaf Varol. "Comparison of Zigbee, Z-Wave, Wi-Fi, and bluetooth wireless technologies used in home automation." *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE, 2019.

14. ESP32 with MPU-6050 Accelerometer, Gyroscope and Temperature Sensor (Arduino) https://randomnerdtutorials.com/esp32-mpu-6050-accelerometer-gyroscope-arduino/ (дата звернення 06.10.2025).

15. How to Connect MPU6050 to ESP32: Physical Setup and Code https://www.youtube.com/watch?v=H9e1Up7xHjc (дата звернення 06.10.2025).

16. Nasution, T. I., and P. F. A. Azis. "MPU-6050 Wheeled Robot Controlled Hand Gesture Using L298N Driver Based on Arduino." *Journal of Physics: Conference Series*. Vol. 2421. No. 1. IOP Publishing, 2023.

17. Rafiq, Arif Ainur, Wahid Nur Rohman, and Sugeng Dwi Riyanto. "Development of a simple and low-cost smartphone gimbal with MPU-6050 sensor." *Journal of Robotics and Control (JRC)* 1.4 (2020): 136-140.

18. Pokydko, Maksym, Ostap Oliinyk, and Vadym Tymchenko. "MEMS Gyroscope Based on MPU-6050 Sensor and ATmega328 Microcontroller." *2024 IEEE 7th International Conference on Smart Technologies in Power Engineering and Electronics (STEE)*. IEEE, 2024.

19. Ю. В. Герман, “Двоколісний автобалансуючий робот з верхньою маятниковістю”. XI Наук.-практ. конф. студ. та асп. «Погляд у майбутнє приладобудування», 2018, С. 30–33.

20. S. Madgwick, “An efficient orientation filter for inertial and inertial / magnetic sensor arrays,” 2010.

21. Г. Ю. Строкач, О. М. Сапегін, “Комплементарний фільтр для мікромеханічного інклінометру”. Проблеми та перспективи реалізайії та впровадження міждисциплінарних наукових досягнень (Т.1), 2020, С. 58–60.

22. Примушко А.,М., Рижков Л.М. Дослідження комплементарного фільтра на МЕМС-вимірювачах//Інформаційні системи, механіка та керування. 2019. Випуск 20. С.47-53.

23. Миколайчук, Володимир Ігорович. "Система візуалізації кутів повороту об'єкта." (2025).

24. Строкач, Григорій Юрійович, and Олександр Миколайович Сапегін. "Комплементарний фільтр для мікромеханічного інклінометру." *Проблеми та перспективи реалізайії та впровадження міждисциплінарних наукових досягнень* 1 (2020): 58-60.

25. Derkachenko, A. O., and O. V. Polyvoda. "Використання комплементарного фільтра для ідентифікації параметрів руху елементів сферичного паралельного механізма." *Problems of Informatization and Management* 1.81.

26. Sultan, Juwita Mohd, et al. "Analysis of inertial measurement accuracy using complementary filter for MPU6050 sensor." *Jurnal Kejuruteraan* 34.5 (2022): 959-964.

27. Von Tschirschnitz M. et al. Method confusion attack on bluetooth pairing //2021 IEEE symposium on security and privacy (SP). – IEEE, 2021. – С. 1332-1347.

28. Claverie T., Esteves J. L. Bluemirror: Reflections on bluetooth pairing and provisioning protocols //2021 IEEE Security and Privacy Workshops (SPW). – IEEE, 2021. – С. 339-351.

29. Singelée D., Preneel B. Improved pairing protocol for Bluetooth //International Conference on Ad-Hoc Networks and Wireless. – Berlin, Heidelberg : Springer Berlin Heidelberg, 2006. – С. 252-265.

30. Didi Z., El Azami I. Monitoring photovoltaic panels using the ESP32 microcontroller via low-power Bluetooth communication //2022 International Conference on Intelligent Systems and Computer Vision (ISCV). – IEEE, 2022. – С. 1-6.

31. Technical documentation and repository library BluetoothSerial.h Arduino Docs https://docs.arduino.cc/libraries/bluetoothserial/#Compatibility (дата звернення 08.09.2025)

32. Artanto D. et al. Combining Mindwave, MPU6050, internet of things for reliable safe monitored wheelchair control system //Indonesian Journal of Electrical Engineering and Computer Science. – 2023. – Т. 32. – №. 2. – С. 742-751.

33. Tahier A. R. H. et al. Design of a Covid-19 Patient Respiration Monitoring System Using an ESP 32 Microcontroller and an MPU 6050 Sensor for the Internet of Medical Things (IoMT) System.

34. Mituletu I. C., Muresan V. Wireless Communication System with High Data Flow using an ESP32-Based Interface //2024 5th International Conference on

Communications, Information, Electronic and Energy Systems (CIEES). – IEEE, 2024. – С. 1-6.

35. GRZYBOWSKÁ B. M. Human Mo-cap System Based on Inertial Measurement Units.

36. Hsieh C. T. Development of Proportional-Integral-Derivative Based Self-Balancing Robot Using ESP32 for STEM Education //Engineering Proceedings. – 2025. – Т. 92. – №. 1. – С. 24.

37. Hsieh C. T. Development of Proportional-Integral-Derivative Based Self-Balancing Robot Using ESP32 for STEM Education //Engineering Proceedings. – 2025. – Т. 92. – №. 1. – С. 24.

38. Новацький А. О., Коломійцев П. Є., Сапсай П. О. Комплементарний фільтр для квадрокоптера з компенсацією температурного дрейфу нуля датчика кутової швидкості //Молодий вчений. – 2014. – №. 5 (1). – С. 15-18.

39. Bluetooth LE AT Examples https://docs.espressif.com/projects/esp-at/en/latest/esp32c3/AT_Command_Examples/bluetooth_le_at_examples.html (дата звернення 22.10.2025).

40. ESP32 Bluetooth Classic - ESP32 Beginner's Guide https://youtu.be/EWxM8Ixnrqo?si=D16zO5Miq-LD_sp_ (дата звернення 22.10.2025).

41. Unsal D., Demirbas K. Estimation of deterministic and stochastic IMU error parameters //Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium. – IEEE, 2012. – С. 862-868.

42. Qing Y., Wang L., Zheng Y. Installation Error Calibration Method for Redundant MEMS-IMU MWD //Micromachines. – 2025. – Т. 16. – №. 4. – С. 391.

43. Yi Y. et al. A Plug-and-Play Learning-based IMU Bias Factor for Robust Visual-Inertial Odometry //arXiv preprint arXiv:2503.12527. – 2025.

ДОДАТКИ

Загальний код роботи програми

```
#include <Wire.h>
```

```
#include <BluetoothSerial.h> // Бібліотека для Bluetooth
#include <math.h> // Для математичних функцій, якщо ще не включено

// Перевіряємо, чи визначено PI, якщо ні, визначаємо його
#ifndef PI
#define PI 3.14159265358979323846
#endif

const int MPU = 0x68; // MPU6050 I2C address
float AccX, AccY, AccZ;
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ;
float roll, pitch, yaw;
float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ;
float elapsedTime, currentTime, previousTime;
int c = 0;

// Змінна для зберігання часу останнього виведення
unsigned long lastPrintTime = 0;
// Інтервал між виведеннями (у мілісекундах) - 1000 мс = 1 секунда
const long printInterval = 1000;

// Об'єкт для Bluetooth Serial ***
BluetoothSerial SerialBT;
// Ім'я Bluetooth пристрою
const char* bt_device_name = "ESP32_MPU6050";
// ******************************************


void calculate_IMU_error() {
  // код для калібрування помилок
  Serial.println("Starting IMU calibration...");
  // Read accelerometer values 200 times
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x3B);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 6, true);
    AccX = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    AccY = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    // Sum all readings
    AccErrorX = AccErrorX + ((atan((AccY) / sqrt(pow((AccX), 2) + pow((AccZ), 2))) * 180 / PI));
    AccErrorY = AccErrorY + ((atan(-1 * (AccX) / sqrt(pow((AccY), 2) + pow((AccZ), 2))) * 180 / PI));
    c++;
  }
  //Divide the sum by 200 to get the error value
```

```
  AccErrorX = AccErrorX / 200;
  AccErrorY = AccErrorY / 200;
  c = 0;
  // Read gyro values 200 times
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x43);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 6, true);
    GyroX = Wire.read() << 8 | Wire.read();
    GyroY = Wire.read() << 8 | Wire.read();
    GyroZ = Wire.read() << 8 | Wire.read();
    // Sum all readings
    GyroErrorX = GyroErrorX + (GyroX / 131.0);
    GyroErrorY = GyroErrorY + (GyroY / 131.0);
    GyroErrorZ = GyroErrorZ + (GyroZ / 131.0);
    c++;
  }
  //Divide the sum by 200 to get the error value
  GyroErrorX = GyroErrorX / 200;
  GyroErrorY = GyroErrorY / 200;
  GyroErrorZ = GyroErrorZ / 200;
  // Print the error values on the Serial Monitor
  Serial.print("AccErrorX: "); Serial.println(AccErrorX);
  Serial.print("AccErrorY: "); Serial.println(AccErrorY);
  Serial.print("GyroErrorX: "); Serial.println(GyroErrorX);
  Serial.print("GyroErrorY: "); Serial.println(GyroErrorY);
  Serial.print("GyroErrorZ: "); Serial.println(GyroErrorZ);
  Serial.println("Calibration complete.");
}


void setup() {
  Serial.begin(9600);
  Wire.begin(); // Initialize comunication
  Wire.beginTransmission(MPU); // Start communication with MPU6050 // MPU=0x68
  Wire.write(0x6B); // Talk to the register 6B
  Wire.write(0x00); // Make reset - place a 0 into the 6B register
  Wire.endTransmission(true); //end the transmission

  // Ініціалізація Bluetooth Serial
  SerialBT.begin(bt_device_name); // Запускаємо Bluetooth з вказаним ім'ям
  Serial.print("Bluetooth device name: ");
  Serial.println(bt_device_name);
  Serial.println("The device is ready to connect via Bluetooth.");
  // ********************************************
```

```
  calculate_IMU_error();
  // Оновлення початкових значень roll та pitch після калібрування
  Wire.beginTransmission(MPU);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true);
  AccX = (Wire.read() << 8 | Wire.read()) / 16384.0;
  AccY = (Wire.read() << 8 | Wire.read()) / 16384.0;
  AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0;
  roll = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) - AccErrorX;
  pitch = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI) - AccErrorY;
  yaw = 0.0; // Yaw завжди починається з 0

  delay(20);
  previousTime = millis();
}

void loop() {
  // === Read acceleromter data === //
  Wire.beginTransmission(MPU);
  Wire.write(0x3B); // Start with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 6 registers total, each axis value is stored in 2 registers
  //For a range of +-2g, we need to divide the raw values by 16384, according to the datasheet
  AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // X-axis value
  AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // Y-axis value
  AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // Z-axis value

  // Calculating Roll and Pitch from the accelerometer data
  // Застосовуємо калібровані помилки
  accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) - AccErrorX;
  accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI) - AccErrorY;

  // === Read gyroscope data === //
  currentTime = millis(); // Current time actual time read
  elapsedTime = (currentTime - previousTime) / 1000.0; // Divide by 1000.0 to get seconds
  previousTime = currentTime; // Оновлюємо час для наступного циклу

  Wire.beginTransmission(MPU);
  Wire.write(0x43); // Gyro data first register address 0x43
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 6 registers total
  GyroX = (Wire.read() << 8 | Wire.read()) / 131.0; // deg/s
  GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
  GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;

  // Correct the outputs with the calculated error values
```

```
GyroX = GyroX - GyroErrorX;
GyroY = GyroY - GyroErrorY;
GyroZ = GyroZ - GyroErrorZ;

// Complementary filter - combine acceleromter and gyro angle values
// Оновлення roll та pitch за допомогою комплементарного фільтра
roll = 0.96 * (roll + GyroX * elapsedTime) + 0.04 * accAngleX;
pitch = 0.96 * (pitch + GyroY * elapsedTime) + 0.04 * accAngleY;
// Інтегрування для Yaw
yaw = yaw + GyroZ * elapsedTime;

// Обмежуємо yaw від -180 до 180, щоб не накопичувалися великі значення
if (yaw > 180) yaw -= 360;
if (yaw < -180) yaw += 360;

// Print the values on the serial monitor AND send via Bluetooth
// check if the specified time has passed since the last output
if (currentTime - lastPrintTime >= printInterval) {
  // Форматуємо рядок для відправки
  String output_data = String(roll, 2) + "/" + String(pitch, 2) + "/" + String(yaw, 2); // 2 знаки після коми

  // Виведення на Serial
  Serial.println(output_data);

  //  Передача даних через Bluetooth
  if (SerialBT.hasClient()) {
    SerialBT.println(output_data);
  }
  // ****************************************

  // Updating the last output time
  lastPrintTime = currentTime;
}
```