Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

КОМПЛЕКСНА БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Кіберфізичний комплекс збору даних з можливістю обходу перешкод. Частина 2. Програмне забезпечення»

08-54. КБКР.039.00.000 ПЗ

Виконав : студент 4 курсу, групи 1СП-21б
спеціальності 123 — Комп'ютерна інженерія
освітня програма — «Системне програмуванн»
Данилюк І.В.

Керівник: к.т. н., доц. кафедри ОТ
Богомолов С.В.

Рецензент: к.фіз.-мат. н., доц. кафедри МБіС
Шиян А. А.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
13 06 2025 року

Вінниця ВНТУ — 2025 рік

6 Консультанти розділів роботи приведено в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Дата		Підпис
		видав	прийняв	
1-3	Богомолов С.В., к.т.н., доцент каф. ОТ	21.03.25	13.05.25	
Нормоконтроль	ас. каф. ОТ Швець С. І.	14.05.25	30.05.25	

7 Дата видачі завдання 21.03.2025 р.

8 Календарний план виконання БКР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної роботи	Термін виконання	Підпис
1	Аналіз завдання	21.03.25 — 30.05.25	
2	Вибір платформи та середовища розробки	21.03.25 — 30.05.25	
3	Архітектура програмного забезпечення	21.03.25 — 30.05.25	
4	Алгоритми збору даних та обходу перешкод	31.03.25—13.04.25	
6	Програмування контролера	14.04.25—27.04.25	
7	Інтеграція сенсорів та обробка сигналів	14.04.25—27.04.25	
9	Випробування системи в реальних умовах	14.04.25—27.04.25	
10	Оптимізація коду та підвищення ефективності	14.04.25—27.04.25	
11	Можливості розширення функціоналу	28.04.25—11.05.25	
12	Аналіз результатів та висновки	11.05.25-12.05.25	
13	Перевірка якості виконання бакалаврської кваліфікаційної роботи та усунення недоліків	13.05.25	

Студент Ілля ДАНИЛЮК

Керівник роботи к.т.н., доц. каф. ОТ Сергій БОГОМОЛОВ

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань — 12 — Інформаційні технології
Спеціальність — 123 — Комп'ютерна інженерія
Освітньо-професійна програма — Системне програмування

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф._____Азаров О.Д.
«21» березня 2025 року

З А В Д А Н Н Я

НА КОМПЛЕКСНУ БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

СТУДЕНТУ

Данилюку Іллі Васильовичу

1 Кіберфізичний комплекс збору даних з можливістю обходу перешкод. Частина 2. Програмне забезпечення, керівник роботи Богомолов Сергій Віталійович, к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу від «20» березня 2025 року №97.

2 Термін подання студентом роботи 13.05.2025

3 Вихідні дані: до роботи технічна документація на модуль Keyestudio v4.0, технічна документація із прикладами підключення та налаштування основних датчиків та модулів, базові схеми підключення усіх компонентів пристрою, середовище розробки — Arduino IDE.

4 Зміст розрахунково-пояснювальної записки: вступ; розробка програмного забезпечення; реалізація та тестування; оптимізація та перспективи розвитку; висновки.

5 Перелік ілюстративного матеріалу: структурна схема алгоритму роботи пристрою; структурна схема алгоритму обходу перешкод.

АНОТАЦІЯ

УДК 004.77:004.738.5

Данилюк І.В. Кіберфізичний комплекс збору даних з можливістю обходу перешкод. Бакалаврська кваліфікаційна дипломна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Системне програмування». Вінниця: ВНТУ, 2025. 77 с.

На укр.мові. Бібліогр.: 25 назв, рис.: 9, табл.: 8.

У комплексній кваліфікаційній роботі розроблено програмне забезпечення для кіберфізичного комплексу збору даних з можливістю обходу перешкод.

В роботі спроєктовано структуру програмної частини кіберфізичного комплексу збору даних з можливістю обходу перешкод. Розроблене програмне забезпечення керування датчиками, що роблять можливим процес обходу перешкод та збору даних. Серед таких ультразвуковий датчик відстані та датчик кольору.

У роботі описані архітектуру програмного забезпечення та розроблені алгоритми. Також наведені теоретичні про особливості програмування мікроконтролерів та модулів, а також наведено програмну реалізацію використання датчиків та алгоритмів.

Ключові слова: кіберфізичний комплекс, програмне забезпечення, Arduino, датчик, модуль, метод.

ABSTRACT

Daniliuk I.V. Cyber-physical complex of data collection with the possibility of obstacle avoidance. Bachelor's degree thesis in speciality 123 ‘Computer Engineering’, educational programme ‘System Programming’. Vinnytsia: VNTU, 2025. 77 p.

In Ukrainian. Bibliogr.: 25 titles, figures: 9, tables: 8.

In a complex bachelor's qualification work, software for a cyber-physical data collection complex with the ability to bypass obstacles is developed.

The structure of the software part of the cyber-physical data acquisition complex with the ability to bypass obstacles is designed. The software for controlling the sensors that make the process of obstacle avoidance and data collection possible has been developed. These include an ultrasonic distance sensor and a colour sensor.

The paper describes the software architecture and developed algorithms. The paper also presents theoretical information about the features of programming microcontrollers and modules, as well as the software implementation of the use of sensors and algorithms.

Keywords: cyber-physical complex, software, Arduino, sensor, module, method.

ЗМІСТ

ВСТУП

Кіберфізичні системи є сучасним технологічним рішенням, яке поєднує фізичні процеси з цифровими обчисленнями та взаємодію з допомогою мережевих технологій. Вони мають застосування в різних сферах промисловості, транспорту, сільського господарства та дослідження навколишнього середовища. Актуальності набувають системи збору даних, які здатні адаптуватись до зовнішніх динамічних умов, зокрема зі здатністю обходу перешкод, що дозволить забезпечити автономне керування системи та безперервність передачі інформації.

Актуальність роботи полягає в необхідності розробки програмного забезпечення для кіберфізичної системи збору даних з можливістю обходу перешкод, яке б забезпечувало ефективну та безперервну обробку отриманої інформації та мало зручний інтерфейс для керування пристроєм.

Метою роботи є розробка програмного забезпечення для кіберфізичної системи на базі платформи Arduino, яка виконує автоматичне визначення та обхід перешкоди у фізичному середовищі й виконує збір обробку та аналіз даних з датчиків.

Для досягнення мети необхідно виконати такі **завдання**:

— проаналізувати існуючі платформи та середовища розробки та обґрунтувати їх вибір;

— розробити архітектуру програмного забезпечення;

— розробити алгоритми збору даних та обходу перешкод;

— виконати програмування контролера;

— інтегрувати необхідні датчики та обробити отримані сигнали;

— виконати випробовування системи в реальних умовах;

— оптимізувати програмне забезпечення для підвищення ефективності;

— проаналізувати можливості для подальшого розширення функціоналу системи;

— проаналізувати результати розробки.

Об’єктом дослідження є розробка процесу збору та обробки даних у кіберфізичних системах з урахуванням змін у фізичному середовищі.

Предметом дослідження є програмне забезпечення для керування кіберфізичного комплексу з перешкоди.

Методами дослідження є аналіз існуючих алгоритмів маршрутизації, обходу перешкод з компонентним підходом до розробки програмного забезпечення та виконання реального тестування.

Апробація результатів бакалаврської роботи: зроблено доповідь на LIV науково-технічну конференцію підрозділів ВНТУ [1].

Публікації за темою роботи: Кіберфізичний комплекс збору даних з можливістю обходу перешкод / Богомолов С.В., Заїченко І.Г., Данилюк І.В. // Матеріали LIV наукової-технічної конференції підрозділів ВНТУ, 2025 р. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24075/19939

1 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Вибір платформи та середовища розробки

Інтегроване середовище розробки (ICP, англ. integrated development environment або англ. IDE) — це програмне рішення для розробки програмного забезпечення. Як правило, ICP є редакторами початкового коду, в якому пишеться програмний код з підсвіткою та перевіркою синтаксиса й автодоповненням, та місять інструменти для автоматичного складання та відлагодження програм.

Середовища розробки можуть містити компілятори та інтерпретатори. Компілятор є перекладачем всього коду з мови програмування високого рівня в машинний код, який мікроконтролер можу опрацювати. Інтерпретатор в свою чергу виконує по-командне перетворення у машинний код, він аналізує та відразу виконує програму, що дозволяє швидше обробляти програму, але при цьому інтерпретатор виявляє помилки в програмі тільки після спроби виконання програми або певного рядка в якому є помилка [2]. Деякі з них не містять жодного з них. Більшість сучасних ICP мають інспектори класів, схему ієрархії класів або інспектор об’єктів, для спрощення розробки з використанням об’єктно-орієнтованого програмування.

Основним завдання ICP є підвищення продуктивності інженера під час розробки, надавши корисні та зручні йому інструменти розробки та дають можливість налагоджувати, компілювати, модифікувати, розгортати програмне забезпечення.

Деякі середовища розробки призначені для використання певних або декількох споріднених мов програмування. Також існують більш універсальні, багатомовні ICP. Більшість сучасних середовищ мають графічний інтерфейс користувача, який дозволяє розробнику ефективніше та швидше вивчити або ж пристосуватись до певного середовища розробки.

Програми для апаратно-обчислювальної платформи Arduino розробляються з використанням мов програмування C та C++, які, згідно рейтингу IEEE Spectrum, в 2023 році входили до категорії найпопулярніших мов програмування. Для розробки на цих мовах програмування існує безліч зручних ICP, такі як: Visual

Studio, Qt Creator, CLion, Eclipse. Проте, вони не мають прямої можливості для взаємодії з платформою Arduino.

Існують ІСР, які призначені для взаємодії з апаратно-обчислювальними платформами (Arduino IDE, PlatformIO, Atmel Studio) [3]. Офіційним середовищем розробки для платформ Arduino є Arduino IDE. Воно підтримує всі офіційні плати має простий текстовий редактор та монітор портів платформ. Підтримується на таких платформах як Linux, Windows, macOS. На офіційному сайті розташована інструкція для користування даним середовищем, тому розробнику не доводиться витрачати багато часу для освоєння в ньому.

PlatformIO являється додатковим модулем для Visual Studio, яке надає можливість працювати з платформами Arduino. Також дане ІСР може мати вигляд окремого інтерфейсу, воно підтримує платформи: Arduino, ESP8266, STM32. Має систему керування бібліотек та інтеграцію з Git.

Atmel Studio є середовищем для розробки та відлагодження програм для мікроконтролерів AVR та AVR32 на операційній системі Windows. Для роботи з Arduino потребує додаткових налаштувань.

Опираючись на основну мету роботи, для найбільш ефективної розробки програми для кіберфізичного пристрою, як основне інтегроване середовище розробки для платформи Arduino обрано Arduino IDE, як найзручніше та найбільш універсальне середовище для програмування та відлагодження програм для даної платформи.

1.2 Архітектура програмного забезпечення

Архітектура програмного забезпечення є системною структурою, яка визначає основні компоненти програми, їх взаємодію між собою, та організацію програмного коду за необхідними принципами. При роботі з Arduino, архітектура указує на те, як апаратні модулі взаємодіють з програмними блоками, яким способом організовано код та які рівні програмної абстракції при цьому використовуються.

Особливостями архітектури для створення програми для Arduino є те що мікроконтролер має обмежені ресурси, що вказує на те що програма повинна бути оптимізованою для коректної роботи плати. Програми для Arduino будуються навколо двох основних функцій setup() і loop(), де функція loop() є нескінченним циклом, а setup() — це функція для ініціалізації платформи в якій можна задавати початкові значення для модулів, які підключені[4]. Потрібно враховувати те, що необхідно працювати з апаратними засобами напряму, для цього часто використовуються драйвери та бібліотеки для датчиків. Також в Arduino обмежена багатозадачність, обумовлено це тим, що Arduino-плати працюють без операційних систем, а самі програми виконуються в одному потоці, тобто по черзі, від початку програми до її кінця. Arduino побудована на базі AVR мікроконтролерів, в них переважно знаходиться одне ядро та немає спеціальних таймерів переривань для керування потоками як у сучасних процесорах.

Рисунок 1.1 — Базова архітектура програми для Arduino

При створенні програми для Arduino необхідно розділити проєкт на певні рівні. Це спрощує процес створення, підтримки, масштабування та відлагодження проєкту.

Як правило в структурі архітектури програм для Arduino визначають основні чотири рівня:

— прикладний рівень — найвищий рівень архітектури на якому реалізується головна логіка поведінки пристрою;

— інтерфейсний рівень відповідає за взаємодію користувачем або іншими пристроями, комунікація може виконуватись за допомогою дисплеїв, кнопок, через Bluetooth або Wi-Fi;

— функціональний рівень — це основа програми, функціональні модулі, які виконують певні задачі, як зчитування даних із датчика, керування реле або світлодіодом, обробка сигналів з кнопка;

— апаратний рівень, цей рівень взаємодіє безпосередньо з апаратно частиною Arduino: виходами, таймерами, інтерфейсами(SPI, I2C), перериваннями.

Існує декілька підходів до побудови архітектури побудування програм для системних плат Arduino:

— процедурний підхід — цей підхід указує на те, що вся програма знаходиться в одному файлі;

— модульний підхід, коли кожен функціонал програми є окремим модулем;

— об’єктно-орієнтований підхід — кожен пристрій або функціонал є окремим об’єктом програми.

В процедурному підході сама програма є послідовністю команд, які виконуються почергово, по тому як вони розташовані в програмі. Даний підхід є непрактичним, та зазвичай застосовується в простих програмах, які не мають обширного функціоналу та містять строгу послідовність дій. Даний архітектура не підходить для реалізації кіберфізичного пристрою, тому що він містить багато модулів, які часто між собою взаємодіють. Також потрібно врахувати дуже багато аспектів при обробці отриманих даних, які при даному підході буде проблематично реалізувати.

Модульний підхід є найкращим вибором для реалізації роботи, тому що функціонал кожного модуля пристрою можна виділити в окремий файл, що дозволить ефективно та просто їх використовувати. При цьому зміни в одному модулі не будуть впливати на інші. Також це зручно при тестуванні пристрою, тому що тестування можна проводити тільки для певних частин програми. Даний підхід покращує читабельність та будує логічне представлення програми, де кожен модуль виконує певну функцію. Новий функціонал можна додавати як окремий модуль, що сприяє швидкому розвитку проекту.

При об’єктно-орієнтованому підході використовуються класи, тобто кожен функціонал модуля буде описаний в об’єкті [5]. Цей підхід добре підходить для бібліотек, але через обмежені ресурси пам’яті в Arduino, використання об’єктів створює додаткові витрати пам’яті, що може привести до переповнення пам’яті або певних програмних збоїв.

Основними задачами кіберфізичної системи є безперервний рух, знаходження перешкоди, визначення кольору перешкоди та обхід перешкоди за визначеним алгоритмом. Під поставлені задачі потрібно розробити відповідні програмні модулі, які будуть їх вирішувати за будь-яких умов навколишнього середовища.

Розробка програмних модулів для Arduino виконується за допомогою тієї ж мови програмування на якій пишеться код для програм цієї плати, тобто на мові C++. Модулі мають файли з розширеннями .cpp — файл в якому описана реалізації функції та .h — це файли заголовків, де оголошуються функції [6].

Функція — це частина коду, яка виконує певні дії та може бути викликана(повторитись) в будь-якому місці програми. Функція має свою назву, може приймати значення, та повертати результат її виконання [7]. В залежності від того, якого типу даних функція, вона може повертати різні значення. Тип даних — характеристика, яка зазначає допустимі значення, формат, розмір викростання пам’яті та операції, які можна виконати над цими даними [8].

Існує декілька основних типів даних в мові C++:

— integer — це цілочисельний тип даних, розміром в 2 або 4 байти, діапазон чисел, який може містити даний тип даних від — 32768 до 32767;

— float — тип даних, який повертає числа з плаваючою комою, розміром в 4 байти, діапазон чисел, який може містити даний тип даних від 1.2e-38 до 3.4e38;

— double — тип, даних більш точних дробових чисел, розміром в 8 байти, діапазон чисел, який може містити даний тип даних від від 2.2e-308 до 1.8e308;

— bool — логічний тип даних, розміром в 1 байт, може містити одне з двох значень істина або брехня, які в машинному коді представлені нулем або одиницею;

— char — символьний тип, розміром 1 байт, який містить число від 0 до 255, де число є представленням певного символу в коді ASCII.

Функції також можуть повертати ніякого значення, а просто виконувати певні дії, такі функції мають тип void.

Основний цикл роботи буде результатом розробленої архітектури. Це серце програми, яка безперервно отримує дані з модулів, керує пристроями та виконує відповідні дії, щодо зовнішніх подій [9].

Основний цикл роботи пристрою складається з таких дій:

— ініціалізація плати Arduino та всіх датчиків;

— зчитування отриманих значень з датчиків;

— аналіз отриманих даних та виконання необхідних з ними дій, як нормалізація значень, забезпечення зберігання з майбутньою можливістю вміння оперувати ними;

— виконання розрахунків та отримання кінцевих даних для виконання дій в залежності від отриманої інформації з датчиків;

— вивід інформації про виконання;

— передача даних та виконання відданих команд апаратними засобами.

1.3 Алгоритми збору даних та обходу перешкод

Алгоритм — це чітка послідовність дій, які виконуються для досягнення певної мети, реагуючи на зовнішні чинники. У випадку реалізації кіберфізичного

пристрою для збору даних — це алгоритм обробки показників отриманих з датчиків, а при обході перешкод — це алгоритм прийняття рішень базуючись на зібраних даних, для того щоб уникнути зіткнення з перешкодою або знаходження альтернативного маршруту.

Структура алгоритму відіграє важливу роль для досягнення поставленої цілі, щодо поведінки пристрою та його дій щодо навколишнього середовища [10].

Основними компонентами структури алгоритму є:

— розгалуження;

— цикли;

— перемикачі.

В мові C++ ці компоненти можна реалізувати за допомогою операторів. Оператор — це конструкція, яка виконує певну дію з даними, які вона отримала [11].

Оператори поділяються на:

— оператори циклів;

— арифметичні;

— оператори порівняння;

— логічні;

— побітові;

— оператори розгалуження.

Перелік всіх операторів подано в таблицях.

Таблиця 1.1 — Арифметичні оператори

Назва оператора	Синтаксис	Дії які виконує оператор
Присвоєння	a=b	Копіює значення однієї змінної в іншу.
Додавання	a+b	Додає значення двох змінних.
Віднімання	a-b	Знаходить різницю між значеннями двох змінних.
Множення	a*b	Множить значення між собою двох змінних.
Ділення	a/b	Знаходить часту значень двох змінних.
Остача від ділення	a%b	Знаходить остачу від ділення двох значень.
Інкремент	a++	Додає одиницю до значення змінної.
Декремент	a--	Віднімає одиницю до значення змінної.

Таблиця 1.2 — Оператори порівняння

Назва оператора	Синтаксис	Дії які виконує оператор
Дорівнює	a == b	Перевіряє чи два значення однакові.
Не дорівнює	a != b	Перевіряє чи два значення різні.
Більше	a > b	Перевіряє чи перше значення більше за друге.
Менше	a < b	Перевіряє чи перше значення менше за друге.
Більше або дорівнює	a >= b	Перевіряє чи перше значення більше або дорівнює другому.
Менше або дорівнює	a <= b	Перевіряє чи перше значення менше або дорівнює другому.

Таблиця 1.3 — Логічні оператори

Назва оператора	Синтаксис	Дії які виконує оператор
Логічне І	a&&b	Повертає значення істиності, якщо виконуються обидві умови.
Логічне Або	a||b	Повертає значення істиності, якщо хоча б одна умова виконується.
Логічне заперечення	!a	Значення стає протилежним початковому.

Таблиця 1.4 — Побітові оператори

Назва оператора	Синтаксис	Дії які виконує оператор
Побітове І	a&b	Кожен біт в a І кожен біт в b.
Побітове Або	a|b	Кожен біт в a АБО кожен біт в b.
Побітове заперечення	~a	Всі біти в a змінюються на протилежні.
Побітове виключне Або	a ^ b	Кожен біт в a XOR кожен біт в b.
Побітовий зсув вліво	a << b	Всі біти в a зміщуються вліво на b біт.
Побітовий зсув вправо	a >> b	Всі біти в a зміщуються вправо на b біт.

Для розробки алгоритмів збору даних та обходу перешкод використовуються відповідні оператори. Вони дозволять програмно реалізувати поведінку робота та його реакцію на навколишнє середовище.

Початковим етапом в алгоритмі роботи пристрою є збір даних. Система повинна знати про її оточення перед тим як виконувати перші дії. Збором даних займаються ультразвуковий та визначення кольору датчики.

Таблиця 1.5 — Оператори циклів

Оператор циклу з післяумовою	do{}while(умова)	Виконується повторення хоча б один раз, поки умова не задовільниться.
Оператор циклу з параметром	for(параметри){}	Виконується повторення з відомою кількістю заздалегіть.
Оператор циклу з передумовою	while(умова)	Виконується повторення, поки умова не задовільниться.

Таблиця 1.6 — Оператори розгалужень

Назва оператора	Синтаксис	Дії які виконує оператор
Умовне розгалуження	if(умова){} else if(умова){} else{}	Виконується дія, коли умова задовільняється, якщо ні — тоді виконуються дії за замовчуванням.

Перемикач	switch(значення або вираз){
	case умова:
	дії при задоволенні умови
	}
		Виконується відповідна дія, в залежності від умови та значення чи виразу, яке перевіряється.

Датчик — пристрій, який займається збором фізичних величин та передає відповідні електричні сигнали, такі сигнали можуть бути аналоговим чи цифровими. Для того щоб оперувати отриманими сигналами з датчика, необхідно запрограмувати мікроконтролер, який призначений для керування цими сенсорами, обробки даних, прийняття відповідних рішень та взаємодії між різними пристроями.

Етапи програмування сенсорів:

— ініціалізація — це процес підключення бібліотек в програму, налаштування виходів мікроконтролеру;

— зчитування даних, здійснюється у безперервному циклі або ж за певних умов;

— попередня обробка даних для майбутньої можливості оперувати ними.

Платформа Arduino після запуску не має інформацію про пристрої, які є під'єднані до самої плати, тому ініціалізація являється першим та обов'язковим етапом. В цей час готує пристрій до роботи, налаштовує піни для зчитування або запису даних та перевіряє з'єднання з під'єднаними модулями.

В цей час виконується:

— підключення бібліотеки для модуля;

— визначення типу та адреси модуля;

— включення датчика та підготування його до роботи;

— перевірка коректності з'єднання;

— визначення параметрів для зчитування(налаштування чутливості або частоти).

Переважно датчики мають свої бібліотеки, які включаються в програму в коді на допомогою команди «#include», після команди вказується назва бібліотеки датчика, що дозволяє створити екземпляр класу, який взаємодіє з пристроєм.

В функції setup() ініціалізується датчик. Якщо датчик має своє бібліотеку, тобто можна створити об'єкт класу датчика, тоді його можна активувати методом begin(). Дана функція є стандартним методом ініціалізації для налаштування інтерфейсу зв'язку з модулем та виконує первинне налаштування пристрою. Також перевіряє доступність модуля, оскільки дана функція є типу bool() вона повертає значення істинності, коли датчик має зв'язок з платформою Arduino чи неістинності(true або false), коли зв'язок відсутній. Ця функція може приймати цілочисельне значення, яке вказує на швидкість передачі даних, тобто кількість бітів, яка може передаватись в секунду. Зазвичай це значення рівне 9600.

Приклад виклику даної функції та обробка значення, що повертається показано в лістингу 1.1.

Лістинг 1.1 — Ініціалізація датчика TCS34725

```
void setup() {
   Serial.begin(9600);
   if (tcs.begin()) {
      Serial.println("TCS34725 знайдено.");
   }
}
```

В прикладі, ініціалізується спочатку Serial-монітор, з допомогою якого, інші пристрої, що під’єднано напряму до платформи Arduino, можуть отримувати повідомлення через UART-порт. Serial — це об’єкт класу Serial-монітору, begin() — це метод який ініціалізує даний монітор, також в дужках в цю функцію передається значення 9600, яке визначає швидкість передачі даних.

Після ініціалізації Serial-монітору, йде перевірка з’єднання з датчиком TCS34725, де tcs — це об’єкт класу цього датчика, а begin() — метод для ініціалізації, який поверне значення true або false. При отриманні значення true для оператору if, виконається вивід інформації в Serial-монітор про успішне під’єднання даного датчику. Деякі датчики можуть вимагати додаткових налаштувань. Такі налаштування переважно виконуються в коді відповідними функціями, які знаходяться в бібліотеці сенсора. Налаштування можуть виконуватись відповідно до вимог використання пристрою, функціонал переважно досить обширний. Можна налаштовувати час зчитування даних(для отримання більш точних значень) або підсилення сигналу датчика. Це все безпосередньою буде впливати на точності та якості отриманих даних. Після проведення ініціалізації та первинного налаштування сенсора в систему, можна отримувати з його допомогою дані про навколишнє середовище.

Отриманні дані можуть бути занадто неточними, нестабільними для того щоб виконувати майбутні дії з ними, саме тому часто використовується усереднення кількох отриманих значень.

Усереднення значень — метод обробки отримання даних, суть методу в тому, що вимірювані певного значення відбувається декілька разів і обчислюється середнє значення. Цей метод зменшує вплив випадкових шумів або похибок при

вимірюваннях з допомогою датчиків. Також підвищується точність вимірювання для аналогових або нестабільних цифрових датчиків.

Головними недоліками даного способу є:

— затримка у відповіді — зчитування декількох значень займає набагато більше часу;

— використання додаткової пам’яті для зберігання декількох значень;

— сповільнює час реакції на зовнішні фактори, особливо помітно, коли потребується миттєва реакція.

Також часто буває, що датчик отримує аномально великі чи малі значення, які не будуть враховуватись при обрахунках. Цей метод для фільтрації значень, дозволяє позбутись помилкових вимірювань, які дуже сильно відрізняються від певної вибірки значень. Такі значення можуть з’являтись внаслідок електричних перешкод, помилок датчика, тимчасових збоїв. Основна мета — позбутись некоректних значень при вимірюванні перед майбутніми розрахунками [13].

Існує декілька способів відсіювання аномальних значень:

— просте обмеження — це коли задається діапазон допустимих значень основуючись на попередніх замірах за нормальних умов;

— видалення мінімального та максимального значення — з декількох вимірювань береться найбільший на найменший результат вимірювань, після чого вони видаляються з масиву значень;

— медіанне фільтрування — отримані дані сортуються, після чого береться середнє за значеннями, а не за арифметикою;

— статистичне відсіювання — значення, які є більшими чи меншими на певну різницю будуть вважатись аномальними.

Недоліками відсіювання аномалій являються:

— ймовірність неправильних відсіювання значень, деколи аномальні значення, можуть виявитись досить звичайними, але через задані обмеження вони можуть неправильно визначитись програмою, тому для цього необхідно виконати досить об’ємні розрахунки, що не є можливим для платформи Arduino через обмеженість в ресурсах плати;

— необхідність в додатковій пам’яті (особливо коли виконується сортування при медіанному фільтруванні)

Для отримання більш точних необхідних даних, можна застосувати такий математичні прийоми, як інтерполяція. Інтерполяція — це спосіб для вирахування проміжних значень отриманих величин за наявним набором відомих значень [14].

До переваг інтерполяції можна віднести:

— злагодження шумів — зменшує вплив коливань або неточностей при вимірюваннях;

— заповнення пропусків — при тимчасових збоях датчика, інтерполяціє заповнює прогалину значеннями, тобто додає стабільності до роби програми;

— можливість налаштування ступеню згладжування значень;

— зменшення навантаження — інтерполяція не вимагає багато оперативної пам’яті або обчислювальних ресурсів.

Провівши детальний аналіз способів обробки даних для датчиків, створено порівняльну таблицю, для того щоб визначити який метод буде використано (таблиця 1.7).

Для розробленого пристрою основними критеріями оцінки є:

— швидкість реакції;

— стабільність зчитування даних;

— кількість похибок та аномалій при вимірах.

Для отримання якісних даних необхідно уникати занадто сильних змін порівняно з первинними. Саме тому оптимальним варіантом буде використання декількох способів обробки даних, а саме використати інтерполяцію з усередненням. Ці методи задовільнять вимоги, щодо якості оброблених даних та не будуть вимагати занадто багато ресурсів для цього.

Отримавши оброблені дані з датчиків, пристрій повинен їх ефективно використати.

Для цього можна використати такі алгоритми для обходу перешкоди:

— реактивний алгоритм;

— повне сканування середовища;

— алгоритм проходу;

— розроблення карта ходів;

Реактивний метод є найпростішим, найпоширенішим та більший ненадійним серед вище перелічених. Реалізується він через прості умовні оператори if(), else(). Оператори будуть перевіряти відстань до перешкоди, та якщо вона задовільна, тоді пристрій рухається в цю ж сторону, в протилежному випадку — виконується маневрування. Такий метод є легким в реалізації, швидкий в процесі руху та чудово працює зі статичними перешкодами. Проте не зможе стабільно та ефективно реалізовувати себе в складних середовищах [15].

При повному скануванні пристрій буде рухатись крок за кроком, робити сканування максимально можливої площі навколо нього, аналізувати де найбільша відстань до перешкоди і від цього робити відповідні маневри. Метод дуже стабільний, робот буде краще орієнтуватись, що буде запобігати зіткненням з перешкодами, але при цьому це буде займати набагато більше часу на отримання та аналіз даних. При цьому, якщо відбудеться різка зміна середовища, пристрій не зможе на неї відреагувати через те що він вже прийняв рішення про маневр до цього.

Таблиця 1.7 — Таблиця оцінювання критерій методів обробки даних

Критерій	Усереднення значень	Відсіювання аномальних значень	Інтерполяція
Точність	Висока для аналогових або нестабільних цифрових елементів	Покращується за рахунок виключення помилкових значень	Вища стабільність, але не впливає на апаратну точність датчика
Згладжування шумів	Так	Так(особливо при сплесках)	Так(плавний перехід між значеннями)
Часова затримка	Так , є необхідність використання додаткового часу для зчитування значень	Так, але залежить від способу, який використовується	Низька швидкість обчислень
Ресурсозатратність (ОЗУ/ЦП)	Середня — необхідність в зберіганні декількох замірів	Висока — сортування та обчислення займають занадто багато пам’яті	Низька — прості обчислення без сильного навантаження пам’яті
Обробка критичних значень	Ні	Так	Частково, тому що не розрізняє нормальні значення від аномальних
Швидкість реакції на різкі зміни	Повільна	Середня(залежить від методу фільтрації)	Швидка
Швидкість реакції на різкі зміни	Повільна	Середня(залежить від методу фільтрації)	Швидка

Алгоритм проходів базується на постійних вимірювань відстані до бокової стіни. Наприклад, задається певна допустима дистанція до якоїсь перешкоди, якої притримується пристрій. Якщо дана відстань зменшується, виконується корекція напрямку руху. Цей алгоритм підходить для вузьких лабіринтних умов, проте він повністю безкорисний на відкритому просторі. Також є імовірність, що робот може

зациклитись тільки на одній перешкоді, що не дасть йому можливості знайти інші варіанти обходу перешкоди.

Створення карти ходів є найскладнішим, найефективнішим та дуже ресурсозатратним для реалізації. Він дозволяє створити карту перешкод та легко адаптуватись при змінах в середовищі. Пристрій знаходить перешкоду, при цьому запам’ятовуючи де вона розташована, для того щоб в майбутньому уникати її заздалегідь. При розробленій карті не виникають проблеми з ухиленням від перешкод та будується оптимальний маршрут, що впливає на швидкість виконання ухилу. Проте дайни спосіб вимагає неймовірно високої вимоги до кількості пам’яті.

Можна використовувати комбінацію певних методів, які будуть, як компенсувати одне-одного за недоліками так і доповнювати себе перевагами. Для визначення, який метод буде використовуватись, складемо таблицю порівняльних характеристик в таблиці 1.8. У ній буде враховано такі параметри, як точність обходу перешкод, адаптивність до змін середовища, обчислювальна складність, потреба в пам’яті, швидкість реакції та ефективність побудови маршруту. Це дозволить об’єктивно оцінити кожен із підходів, а також обґрунтувати доцільність використання комбінованої стратегії. На основі результатів аналізу буде обрано оптимальний варіант або їх поєднання, що забезпечить баланс між якістю навігації та апаратними можливостями пристрою.

Таблиця 1.8 — Порівняльна характеристика алгоритмів обходу перешкод

Алгоритм	Складність реалізації	Потреби в ресурсах	Точність та адаптивність
Реактивний алгоритм	Низька	Низька	Низька
Алгоритм проходу	Середня	Середня	Висока
Розроблення карта ходів	Висока	Середня	Висока

Опираючись на таблицю порівняльних характеристик алгоритмів обходу перешкод, створює алгоритм, який буде задовільнять потреби щодо основних їх характеристик. Алгоритм виглядає наступним чином (рисунок В.2).

2 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

2.1 Програмування контролера

Даний розділ присвячено детальному аналізу коду програми для кіберфізичного пристрою на базі платформи Arduino. Програма включає код функцій керування рухом пристрою в ручному та автономному, з можливістю обходу перешкод, режимах керування.

Для руху робота використовується:

— два DC-мотори;

— Bluetooth-модуль для керування;

— ультразвуковий датчик;

За переміщення пристрою відповідають мотори, підключені через драйвер, який керує напрямком через цифрові виходи платформи Arduino, та швидкістю, через аналогові виходи. Визначення цифрових та аналогових виходів показано в лістигну 2.1.

Лістинг 2.1 — Визначення виходів, що керують моторами

```
#define ML_Ctrl 4
#define ML_PWM 6
#define MR_Ctrl 2
#define MR_PWM 5
```

Де ML_Ctrl — четвертий цифровий вихід, що визначає лівий мотор, MR_Ctrl — другий цифровий вихід, що визначає правий мотор, ML_PWM — шостий аналоговий вихід, що визначає швидкість лівого мотору, MR _PWM — п’ятий аналоговий вихід, що визначає швидкість лівого мотору.

Для визначення початкової швидкості двигунів додано 2 змінні типу byte, з значеннями (лістинг 2.2).

Лістинг 2.2 — Оголошення змінних з початковими значеннями швидкості двигунів

```
byte speeds_L = 200;
byte speeds_R = 200;
```

Для зміни швидкості двигунів через Bluetooth-модуль необхідно оголосити дві додаткові змінні рядкового типу, щоб в майбутньому перетворити їх в змінну цілочисельного типу даних (лістинг 2.3).

Лістинг 2.3 — Оголошення змінних для прийому даних про швидкість двигунів з Bluetooth-модуля

```
String speeds_l,
String speeds_r;
```

Для того, щоб керувати моторами, відповідні виходи, потрібно перевести в режим виводу інформації. Оскільки дану дію потрібно виконати один раз, тому вона виконується в функції setup() (рисунок 2.4).

Лістинг 2.4 — Переведення виходів керування моторами в режим виводу інформації

```
void setup() {
  pinMode(ML_Ctrl, OUTPUT);
  pinMode(ML_PWM, OUTPUT);
  pinMode(MR_Ctrl, OUTPUT);
  pinMode(MR_PWM, OUTPUT);
```

Щоб кожна дія пристрою могла повторюватись безліч раз, кожен його рух(їзда вперед, назад, ліворуч, праворуч і зупинка) описано в окремих функціях. В лістингу 2.5, показано опис функції руху вперед.

Лістинг 2.5 — Опис функції руху вперед

```
void Car_front() {
  digitalWrite(MR_Ctrl, HIGH);
  analogWrite(MR_PWM, 255 - speeds_R);
  digitalWrite(ML_Ctrl, HIGH);
  analogWrite(ML_PWM, 255 - speeds_L);
  matrix_display(front);
}
```

Дана функція відповідає за рух пристрою вперед. Відбувається це за рахунок ввімкнення двох моторів, а саме подачі сигналу на відповідні виходи платформи Arduino, які були описані раніше, та задання швидкості руху на виходи, які за це відповідають.

Для того щоб подати цифровий сигнал на вихід плати використовується функція digitalWrite, яка приймає два значення:

— вихід на який подається сигнал;

— напруга яка подається на вихід.

Напруга, яка подається, в даній функції визначається двома значеннями HIGH або LOW. HIGH — включає підтягування резистора 20 кОм, що призведе до подачі напруги на виході в 5 вольт (або 3.3 вольти, для плати, яка працює на 3.3 вольтах). LOW — відключає підтяжку резистора, тому напруга на виході буде 0 вольт [16].

Для подання аналогового сигналу використовується функція analogWrite, яка приймає такі значення:

— вихід на який подається сигнал;

— коефіцієнт заповнювання.

Коефіцієнт заповнювання — це відношення періоду ввімкненого стану до періоду імпульсів, які прийнято, даний коефіцієнт може мати значення від 0 (завжди виключений) до 255 (завжди включений) [17]. На аналоговим виходах генерується ШІМ-сигнал з заданим коефіцієнтом заповнення, частота якого 490 Гц.

Функція руху назад(лістинг 2.6) має схожий опис до функції руху вперед. Ідея така ж сама, вмикаються одночасно два мотори, але головна відмінність в тому, що параметри швидкості повинні бути протилежними, для оберненого руху моторів.

Лістинг 2.6 — Опис функції руху назад

```
void Car_back() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, speeds_R);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, speeds_L);
  matrix_display(back);
}
```

Для того, щоб робот зупинився необхідно описати в функції виключення двох моторів, як показано в лістингу 2.7.

Лістинг 2.7 — Опис функції зупинки

```
void Car_Stop() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, 0);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, 0);
  matrix_display(STOP01);
}
```

Повороти вліво та вправо відбуваються за рахунок виконання роботом руху вперед одним мотором та рухом назад іншим. Для того, щоб пристрій міг виконувати поворот не змінюючи своє місце розташування, значення швидкості моторів повинні бути однаковими. Функції повороту наліво та направо показано в лістингах 2.8 та 2.9.

Лістинг 2.8 — Опис функції повороту ліворуч

```
void Car_left() {
  digitalWrite(MR_Ctrl, HIGH);
  analogWrite(MR_PWM, 255 - speeds_R);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, speeds_L);
  matrix_display(left);
}
```

Лістинг 2.9 — Опис функції повороту праворуч

```
void Car_right() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, speeds_R);
  digitalWrite(ML_Ctrl, HIGH);
  analogWrite(ML_PWM, 255 - speeds_L);
  matrix_display(right);
}
```

При керуванні пристроєм через Bluetooth модуль, через офіційний мобільний застосунок «KeyesRobot» відправляються повідомлення символьного типу.

Кожен символ відповідає за певну дію:

— «F», символ, який відправляється при натисненні кнопки для руху вперед;

— «B» символ, який відправляється при натисненні кнопки для руху назад;

— «L» символ, який відправляється при натисненні кнопки для повороту ліворуч;

— «R» символ, який відправляється при натисненні кнопки для повороту праворуч;

— «S» символ, який відправляється при натисненні кнопки для зупинки.

Для того, щоб мати змогу обробляти дані символи додано змінну символьного типу, яка буде зберігати отримані команди (лістинг 2.10).

Лістинг 2.10 — Оголошення змінної для зберігання команд Bluetooth-модуля

```
char ble_val;
```

Всі символи, які відправляються через Bluetooth-модуль, пристрій отримуватиме в Serial-моніторі та потім передавати в відповідну змінну. Тому для початку провіряємо чи Serial-монітор отримав якесь повідомлення, після чого йде запис отриманого повідомлення в змінну. Дане повідомлення виводиться на Serial-моніторі, для проведення майбутнього тестування. Саме повідомлення обробляється за допомогою оператора розгалуження switch, де в нього передається змінна в якій записано повідомлення і відповідно до того, яке повідомлення було надіслано виконується дія. Через те, що необхідно безперервно отримувати повідомлення про керування пристрою, все вище перелічене додано в функцію loop, яка безперервно працює протягом часу, коли пристрій ввімкнений. Реалізація обробки отриманого повідомлення показано на лістинг 2.11.

Лістинг 2.11 — Обробка отриманих повідомлень через Bluetooth-модуль

```
void loop() {
  if (Serial.available())
  {
    ble_val = Serial.read();
    Serial.println(ble_val);
    switch (ble_val) {
      case 'F': Car_front(); break;
      case 'B': Car_back(); break;
      case 'L': Car_left(); break;
      case 'R': Car_right(); break;
      case 'S': Car_Stop();  break;
```

Пристрій містить інфрачервоний приймач, який призначено для отримання повідомлень, через пульт керування пристроєм з інфрачервоним діодом. Пульт відповідно до натиснутої на ньому кнопки створює сигнал за допомогою

інфрачервоного світла відправляє сигнал у вигляді HEX-коду. Інфрачервоний приймач повинен отримувати даний сигнал і відповідно до нього пристрій повинен виконувати відповідні дії [18].

Для того щоб пристрій міг працювати з інфрачервоним датчиком, підключаємо бібліотеку «IRemote» (лістинг 2.12). Після чого створюємо об'єкт irrecv класу IRrecv. При створенні об'єкту передаємо в цей об'єкт значення, яке визначає вихід плати, до якого під'єднано ІЧ датчик. Також створюємо об'єкт results структури decode_results, яка буде зберігати розшифровані дані від ІЧ світла, та змінну ir_rec типу long для збереження числового значення прийнятого сигналу.

Лістинг 2.12 — Підключення бібліотеки «IRemote» та створення змінних для роботи з нею

```
#include <IRremote.h>
IRrecv irrecv(3);
decode_results results;
long ir_rec;
```

У безперервному циклі loop додамо умову, за якою перевіряється чи було отримано ІЧ сигнал. Дана умова реалізована за допомогою умовного оператора if(), який має умову irrecv.decode(&results). Ця умова виконується тоді, коли функція decode, яка знаходиться в об'єкті класу IRecv, аналізує сигнал та записує його в структуру results. Після чого сигнал, який перетворений через value записується у вигляді числового значення в змінну ir_rec.

Використовуючи оператор розгалуження switch в який передається змінна ir_rec порівнюється отримане значення з попередньо визначеними:

— 0xFF629D — значення, яке відповідає кнопці пульта, щоб виконати рух вперед;

— 0xFFA857 — значення, яке відповідає кнопці пульта, щоб виконати рух назад;

— 0xFF22DD — значення, яке відповідає кнопці пульта, щоб виконати поворот наліво;

— 0xFFC23D — значення, яке відповідає кнопці пульта, щоб виконати поворот наліво;

— 0xFF02FD значення, яке відповідає кнопці пульта, щоб виконати зупинку.

Якщо ж отриманий код не буде співпадати ні з одним з вище перелічених, тоді нічого не буде виконуватись. Також слід виконати скидання датчика, для того щоб підготувати його до отримання наступних сигналів, тому що без цього він не зможе обробляти нові сигнали. Програмна реалізація показана в лістингу 2.13.

Лістинг 2.13 — Реалізація обробки повідомлень з інфрачервоного пульта керування

```
 if (irrecv.decode(&results)) {
  ir_rec = results.value;
  Serial.println(ir_rec, HEX);
  switch (ir_rec) {
   case 0xFF629D: Car_front();   break;
   case 0xFFA857: Car_back();    break;
   case 0xFF22DD: Car_left();    break;
   case 0xFFC23D: Car_right();   break;
   case 0xFF02FD: Car_Stop();    break;
   default: break;
  }
  irrecv.resume();
 }
```

Для аналізу навколишнього середовища, використовуються ультразвуковий та визначення кольору датчики. Ультразвуковий датчик поставлено на сервопривід, який дозволяє охопити максимальну область для аналізу середовища. Сервопривід під’єднано до десятого виходу платформи, тому визначаємо даний пін в коді [19].

Також необхідно визначити виходи для ультразвукового датчика. Ультразвуковому датчику необхідно два виходи, один — для того щоб керувати відправкою ультразвукового імпульсу, інший — щоб отримувати відбитий від перешкоди сигнал. Визначення цих виходів показано в лістингу 2.14.

Для зберігання даних, отриманих від ультразвукового датчика додано змінні цілочисельного типу, які будуть зберігати відстань попереду робота, відстань визначена після повороту сервопривода ліворуч та відстань визначена після повороту сервопривода праворуч (лістинг 2.15).

Лістинг 2.14 — Визначення виходу для керування сервоприводом та виходів ультразвукового датчику

```
#define servoPin    10
#define Trig        12
#define Echo        13
```

Лістинг 2.15 — Створення змінних для зберігання відстаней виміряних ультразвуковим датчиком

```
int a;
int a1;
int a2;
```

Створимо функцію для керування сервоприводом. Ця функція повинна приймати значення бажаного кута повороту сервоприводом (від 0 до 180 градусів). Сервопривід керується ШІМ-сигналом, тому для перетворення отриманого бажаного кута в імпульси, додаємо змінну цілочисельного типу int pulsewidth. В цю змінну записується результат функції map(), яка виконує перетворення значення, яке є першим аргументом функції, в значення заданого діапазону, як другий та третій аргумент функції, також задаються обмеження для нового діапазону, як четвертий та п’ятий аргумент [20]. Виконується перетворення за такою формулою:

$$y = \frac{(x - x_{min}) \times (y_{max} - y_{min})}{x_{max} - x_{min}} + y_{min},$$

де x — вхідне значення;

y — перетворене значення;

x_{min} — мінімальне значення вхідного діапазону;

x_{max} — максимальне значення вхідного діапазону;

y_{min} — мінімальне значення нового діапазону;

y_{max} — максимальне значення нового діапазону.

Відповідно, першим значенням, яке передається в функцію map буде передаватись бажаний кут повороту, далі задається мінімальне та максимальне значення діапазону повороту сервопривода, що відповідає 0 та 180, а останніми будуть значення 500 та 1800 — стандартний діапазон часу імпульсу для сервопривода.

Для подання імпульсів використовується функція digitalWrite з параметрами визначеного виходу сервопривода та значенням, що подає напругу, після чого зазначається затримка функцією delayMiliseconds, яка приймає цілочисельне значення, яке означає час затримки в мілісекундах для визначення часу подання напруги, тобто значення яке ми отримали в змінну pulsewidth. Також додано затримку, яка буде чекати решту періоду та розраховується за формулою:

$$y = 20 - x/1000,$$

де y — час затримки;

x — тривалість імпульсу;

20 — час, який відповідає частоті 50 Гц.

Дані дії повинні виконуватись 5 разів, що буде забезпечувати стабільність позиціювання, тому вони будуть виконуватись в циклічному операторі for (лістинг 2.16).

Рисунок 2.16 — Реалізація функції керування сервоприводом

```
void procedure(int myangle) {
  int pulsewidth;
  pulsewidth = map(myangle, 0, 180, 500, 1800);
  for (int i = 0; i < 5; i++)
  {
    digitalWrite(servoPin, HIGH);
    delayMicroseconds(pulsewidth);
    digitalWrite(servoPin, LOW);
    delay((20 - pulsewidth / 1000));
  }
}
```

Реалізації методу обходу перешкод описано в функції Avoid. Дана функція використовує дві попередньо створенні функції виміру дистанції до перешкоди та керування сервоприводом. Для того, щоб пристрій міг переключатись між режимами ручного та автоматичного керування додано змінну flag логічного типу — bool, яка буде відповідати за вихід з режимів керування. Початкове значення цієї змінної рівне нулю. Додаємо цикл з передумовою, де подальші дії будуть

виконуватись, поки це значення буде незмінним. Для можливості виходу з даного циклу всередину додаємо керування через Bluetooth-модуль. Керування буде відбуватись через умовний оператор if, де умовою буде отримання команди у вигляді символу «S», через модуль. Після чого змінна flag отримує значення 1 та виконується функція Car_Stop(), яка зупиняє рух робота. Реалізація даного алгоритму показано в лістинг 2.17.

Лістинг 2.17 — Реалізація переключення між режимами керування пристрою

```
void Avoid()
{
 irrecv.disableIRIn();
 flag = 0;
 while (flag == 0)
 {
  if (Serial.available())
  {
   ble_val = Serial.read();
   if (ble_val == 'S')
   {
    flag = 1;
    Car_Stop();
   }
  }
```

Починається алгоритм зі сканування навколишнього середовища попереду робота, для цього викликаємо функцію checkdistance, результат виконання цієї функції записуємо в змінну a, яка була була створена раніше. З допомогою умовного оператора if перевіряємо чи є відстань меншою ніж 20 сантиметрів, якщо дана умова виконується — пристрій повинен зупинитись та повернути ультразвуковий датчик сервоприводом наліво, використовуючи функцію procedure, для сканування середовища ліворуч, потім повернути його праворуч, також для сканування. Відстані, які були виміряні праворуч та ліворуч, записуються в змінні a1 та a2 відповідно.

Лістинг 2.18 — Реалізація функції методу обходу перешкод

```
void Avoid()
{
 irrecv.disableIRIn();
```

```
flag = 0;
while (flag == 0)
{
  a = checkdistance();
  if (a < 20) {
    Car_Stop();
    delay(500);
    procedure(180);
    delay(500);
    a1 = checkdistance();
    delay(100);

    procedure(0);
    delay(500);
    a2 = checkdistance();
    delay(100);

    procedure(90);
    delay(500);
    if (a1 > a2) {
      Car_left();
      delay(700);
    } else {
      Car_right();
      delay(700);
    }
  }
  else {
    Car_front();
  }
  if (Serial.available())
  {
    ble_val = Serial.read();
    if (ble_val == 'S')
    {
      flag = 1;
      Car_Stop();
    }
  }
}
```

Після повного сканування середовища вибирається той напрямок, в якому перешкода знаходиться якомога далі, для цього порівнюються значення a1 та a2. Отже, якщо відстань до перешкоди ліворуч буде більшою — виконується поворот

ліворуч, а якщо праворуч відстань більша — виконується поворот праворуч. Якщо ж відстань до перешкоди попереду буде більша ніж 20 сантиметрів, пристрій буде виконувати рух вперед. Програмну реалізацію функції методу обходу перешкод показано в лістингу 2.18.

Передбачено, що пристрій може визначати колір перешкоди. Додаємо ще одну функцію уникнення перешкод, але вже з можливістю визначення кольорів. Дану функцію назвемо Avoid_color. Реалізація буде подібною до Avoid, але після того як буде знайдено перешкоду, пристрій наблизиться до неї на 5 сантиметрів, та виконає визначення кольору. Реалізація показано в лістингу 2.19.

Лістинг 2.19 — Реалізація наближення робота до перешкоди для визначення кольору

```
void Avoid_color()
{
 irrecv.disableIRIn();
 flag = 0;
 while (flag == 0) {
  a = checkdistance();
  if (a < 20) {
   while (a > 5) {
    Car_front();
    delay(100);
    a = checkdistance();
    if (Serial.available()) {
     ble_val = Serial.read();
     if (ble_val == 'S') {
      flag = 1;
      Car_Stop();
      return;
     }
    }
   }
   Car_Stop();
   delay(500);
   String detectedColor = detectColor();
```

В залежності від кольору перешкоди, пристрій буде виводити на дисплей назву кольору та виконувати дії, які передбачені заздалегідь відповідно певному кольору. Наприклад при визначенні червоного кольору — робот виконує розворот

на 180 градусів та виконувати рух вперед та перевіряє чи немає перешкоди після розвороту. Якщо ж колір перешкоди не визначено, тоді робот виконує звичайну поведінку уникнення перешкоди (лістинг 2.20).

Лістинг 2.20 — Реалізація обробки виміряного кольору датчиком та реакція на нього

```
void Avoid_color()
{
 irrecv.disableIRIn();
 flag = 0;
 while (flag == 0) {
  a = checkdistance();
  if (a < 20) {
   while (a > 5) {
    Car_front();
    delay(100);
    a = checkdistance();
    if (Serial.available()) {
     ble_val = Serial.read();
     if (ble_val == 'S') {
      flag = 1;
      Car_Stop();
      return;
     }
    }
   }
   Car_Stop();
   delay(500);
   String detectedColor = detectColor();

   if (detectedColor == "RED") {
      Car_Stop();
      delay(200);
      procedure(180);
      delay(500);
      Car_back();
      delay(1000);
      Car_Stop();
```

Продовження лістингу 2.20

```
   else if (detectedColor == "GREEN") {
      Car_left();
      delay(700);
      Car_Stop();
```

```
    }
    else if (detectedColor == "BLUE") {
       Car_right();
       delay(700);
       Car_Stop();
    }
    else {
       procedure(180);
       delay(500);
       a1 = checkdistance();
       delay(100);

       procedure(0);
       delay(500);
       a2 = checkdistance();
       delay(100);

       procedure(90);
       delay(500);
       if (a1 > a2) {
         Car_left();
         delay(700);
       } else {
         Car_right();
         delay(700);
       }
    }
  }
  else {
    Car_front();
  }
  if (Serial.available()) {
    ble_val = Serial.read();
    if (ble_val == 'S') {
      flag = 1;
      Car_Stop();
    }
  }
}
```

Після реалізації алгоритмів уникнення перешкод необхідно до оператора розгалуження, який відповідає за обробку команд отриманих через Bluetooth-модуль, додати дві нові умови, при їх виконанні буде включатись відповідний режим. Отримана команда ‘g’ — включає режим уникнення перешкоди, в той час

‘h’ — включає режим уникнення перешкод з визначенням кольору перешкоди (лістинг 2.21).

Лістинг 2.21 — Додані умови, що включають автономні режими керування

```
void loop() {
  if (Serial.available())
  {
    ble_val = Serial.read();
    Serial.println(ble_val);
    switch (ble_val) {
      case 'F': Car_front(); break;
      case 'B': Car_back(); break;
      case 'L': Car_left(); break;
      case 'R': Car_right(); break;
      case 'S': Car_Stop();  break;
      case 'g': Avoid(); break;
      case 'h': Avoid_color(); break;
      case 'u':
        speeds_l = Serial.readStringUntil('#');
        speeds_L = String(speeds_l).toInt();
        break;
      case 'v':
        speeds_r = Serial.readStringUntil('#');
        speeds_R = String(speeds_r).toInt();
        break;
```

2.2 Інтеграція сенсорів та обробка сигналів

Також додано функцію, яка буде повертати вимірювання відстані до перешкоди у вигляді числа з плаваючою крапкою, та використовувати ультразвуковий датчик для вимірювання. Для цього, створимо тимчасову змінну чисельного типу float, яка буде зберігати відстань до об’єктів. Щоб виконати вимірювання створюємо ультразвуковий імпульс, для цього подаємо на вихід плати, визначений як Trig, мінімальну напругу протягом 2 мікросекунд, це робиться для стабілізація датчика. Потім відправляється імпульс 5 вольт на той самий вихід, протягом 10 мікросекунд та вимикається подача напруги на вихід плати. Для того, щоб дізнатись відстань до об’єкта, на вихід, визначений як Echo, подається напруга в 5 вольт з допомогою функції pulseIn(), яка приймає як параметри номер виходу на платі та стан про очікування появи сигналу на цьому ж

виході (HIGH — очікується поява сигналу, LOW — зупиняється зчитування сигналу).

Дана функція повертає значення тривалості імпульсу в мікросекундах, а також визначити час очікування на імпульс в мікросекундах (за замовчуванням дане значення становить 1 секунду). Дані отриманні після виконання даної функції потрібно перевести в відстань.

Оскільки швидкість звука становить приблизно 343 м/с, де одна секунда рівна 29.1 мікросекундам, час який відповідає за 1 см відстані при вимірюваннях буде 58.2 мікросекунди, тому що звук повинен повернутись до цього датчика, тобто пройти відстань в два рази більшу. Програмна реалізація вимірювання відстані до перешкод за допомогою ультразвукового датчику показано в лістингу 2.22.

Рисунок 2.22 — Реалізація функції для вимірювання відстані до перешкоди

```
float checkdistance() {
  float distance;
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);
  distance = pulseIn(Echo, HIGH) / 58.20;
  delay(10);
  return distance;
}
```

Для того, щоб пристрій міг визначати колір перешкод, до нього під’єднано спеціальний датчик TCS34725. Для того щоб платформа вміла з ним працювати підключаємо спеціальну бібліотеку «Adafruit_TCS34725». Також необхідно підключити бібліотеку «Wire.h», вона є необхідною для роботи з цим датчиком, тому що він використовує інтерфейс I^2C для передачі даних з Arduino. I^2C — протокол зв’язку, через послідовну шину даних для зв’язку між платою та датчиком. Дана бібліотека реалізує роботу цього протоколу для платформи Arduino.

Після під’єднання необхідних бібліотек створюємо об’єкт tcs класу Adafruit_TCS34725.

Клас Adafruit_TCS34725 описаний в бібліотеці датчика та приймає при створенні два параметри:

— параметр часу інтеграції — час протягом якого датчик накопичує світло перед вимірюванням;

— коефіцієнт підсилення — підсилює сигнал з фотодіодів датчика, тобто збільшення його чутливості.

Задаються ці параметри константами, які визначені в самій бібліотеці. Параметри підбираємо так, щоб датчик визначав коректно кольори за невеликий проміжок часу, тому коефіцієнт підсилення буде кратний чотирьом, а час інтеграції — 50 мілісекунд (лістинг 2.23).

Рисунок 2.23 — Створення об’єкту класу датчика визначення кольору

```
Adafruit_TCS34725tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME
_50MS, TCS34725_GAIN_4X)
```

Для того, щоб датчик було ініціалізовано, виконуємо метод begin() цього класу в функції setup() та виводимо повідомлення при успішній ініціалізації датчика (лістинг 2.24).

Лістинг 2.24 — Ініціалізація датчика визначення кольору

```
void setup() {
  if (tcs.begin()) {
    Serial.println("Датчик кольору готовий!");
  } else {
    Serial.println("Помилка: датчик не знайдено!");
  }
```

Для визначення кольору створимо функцію рядкового типу detectColor(), яка буде повертати рядок з символьним значення кольору. В функції створюємо змінні цілочисельного типу, для зберігання значень визначених кольорів. Датчик має три фототранзистори, які визначають колір за колірною моделлю RGB (red, green, blue), тому створюємо такі змінні як показано в лістингу 2.25.

Лістинг 2.25 — Оголошення змінних для збереження замірів

```
String getDetectedColor() {
  uint16_t r, g, b, c;
  tcs.getRawData(&r, &g, &b, &c);
```

В ці змінні, як показано, записуються значення отримані від датчика після виклику функції getRawData(). Дана функція описана в класі Adafruit_TCS34725 і повертає значення RGB та значення загальної інтенсивності освітленості, тому було добавлено ще одну змінну, яка буде зберігати дане значення.

Після вимірювання датчиком, отримані результати були аномально великими, тому необхідно їх нормалізувати відповідно до значення інтенсивності освітленості (лістинг 2.26).

Лістинг 2.26 — Нормалізація значень відносно інтенсивності освітлення
```
uint8_t nr = (float)r / c * 255;
  uint8_t ng = (float)g / c * 255;
  uint8_t nb = (float)b / c * 255;
```

Отримавши нормалізовані значення переведемо їх в колірну модель HSV (hue — колірний тон, saturation — насиченість, value — яскравість кольору). З такою моделлю можна буде більш точно визначити колір, враховуючи відтінки кольорів, задаючи їх діапазон відповідно до шкали відтінків цієї моделі (рисунок 2.1) [21].

Рисунок 2.1 — Шкала відтінків колірної моделі HSV

Переведення значень з однієї моделі в іншу описано в функції rgbToHsv(). Реалізація даної функції показано в лістингу 2.28. Створюємо змінні для зберігання переведених даних після виконання цієї функції та на основі шкали відтінків моделі HSV створюємо умови для визначення кольорів (лістинг 2.29).

Лістинг 2.28 — Програмна реалізація переведення значень з RGB в HSV
```
void rgbToHsv(uint8_t r, uint8_t g, uint8_t b, float* h, float* s, float* v) {
  float fr = r / 255.0;
  float fg = g / 255.0;
  float fb = b / 255.0;

  float maxVal = max(fr, max(fg, fb));
  float minVal = min(fr, min(fg, fb));
  float delta = maxVal - minVal;
```

```
  *v = maxVal;
  *s = (maxVal == 0) ? 0 : delta / maxVal;

  if (delta == 0) {
    *h = 0;
  } else if (maxVal == fr) {
    *h = 60 * fmod(((fg - fb) / delta), 6);
  } else if (maxVal == fg) {
    *h = 60 * (((fb - fr) / delta) + 2);
  } else {
    *h = 60 * (((fr - fg) / delta) + 4);
  }

  if (*h < 0) *h += 360;
}
```

Лістинг 2.29 — Визначення кольору за шкалою відтінків HSV

```
String getDetectedColor() {
  uint16_t r, g, b, c;
  tcs.getRawData(&r, &g, &b, &c);
  uint8_t nr = (float)r / c * 255;
  uint8_t ng = (float)g / c * 255;
  uint8_t nb = (float)b / c * 255;
  float h, s, v;
  rgbToHsv(nr, ng, nb, &h, &s, &v);
  if ((h >= 0 && h < 15) || (h >= 345 && h <= 360)) return "RED";
  if (h >= 70 && h < 170) return "GREEN";
  if (h >= 200 && h < 260) return "BLUE";
  return "UNKNOWN";
}
```

Після цього можна вважати, що основа програми для кіберфізичного пристрою, є завершеною та готовою для експлуатації.

2.3 Випробування системи в реальних умовах

Важливим етапом для розробки кіберфізичного комплексу збору даних є тестування його працездатності за реальних умов. Метою даного етапу є перевірка стабільності роботи комплексу, ефективність алгоритмів обходу перешкод, а також коректність зібраних даних для передачі та обробки в динамічному середовищі.

Перед початком випробувань було підключено всі необхідні модулі: ультразвуковий датчик HC-SR04, ІЧ-приймач, Bluetooth-модуль, LED-матрицю та

датчик кольору. Згідно з програмним кодом, система здійснює ініціалізацію кожного модуля та переходить до очікування команд або автономної роботи залежно від обраного режиму.

Для початку було проведене тестування базового функціоналу руху, а саме — рух моторів та можливість керування моторами через ІЧ-пульт. У відповідь на натискання кнопок пульта пристрій починав рухатись у заданому пультом напрямку (вперед, назад, вліво, вправо), зупинявся та переключався між режимами. Результати показали, що всі команди, отримані від пульта приймались коректно, а затримка між натисканням кнопки та реагуванням пристрою була мінімальна (до 200 мс), що забезпечувало зручність при керуванні пристроєм.

Тестування Bluetooth-модулю показало, що команди отримуються вірно, а затримка в отриманні команд була мінімальною. Для проведення тестування використовувся офіційний додаток для керування через Bluetooth — «KeyesRobot». Було проведено перевірку всіх команд для керуванням пристрою [22].

Наступним етапом була перевірка автономного режиму з можливістю обходу перешкод. В цьому режимі активовувався ультразвуковий датчик, який безперервно вимірював відстань до об’єктів перед комплексом, якщо перешкода знаходилась на відстані меншій 20 см, пристрій автоматично змінював траєкторію руху, де не було перешкод. Під час тестування пристрій успішно уникав статичних перешкод у вигляді коробок, стін та книг. Даний режим працював стабільно, що свідчить про правильну інтеграцію ультразвукового датчику та сервоприводу з логікою керування.

Для перевірки режиму виявлення кольору, перед пристроєм розміщувались кольорові об’єкти з різними типами поверхонь, а саме: глянцева поверхня (кубик Рубика) (рисунок 2.2), матові поверхні (картон), поверхні з підсвіткою (монітор смартфону) (рисунок 2.3). Датчик розпізнавав кольори, після чого система реагувала відповідно до кольору.

Рисунок 2.2 — Проведення тестування датчику на кубику Рубика

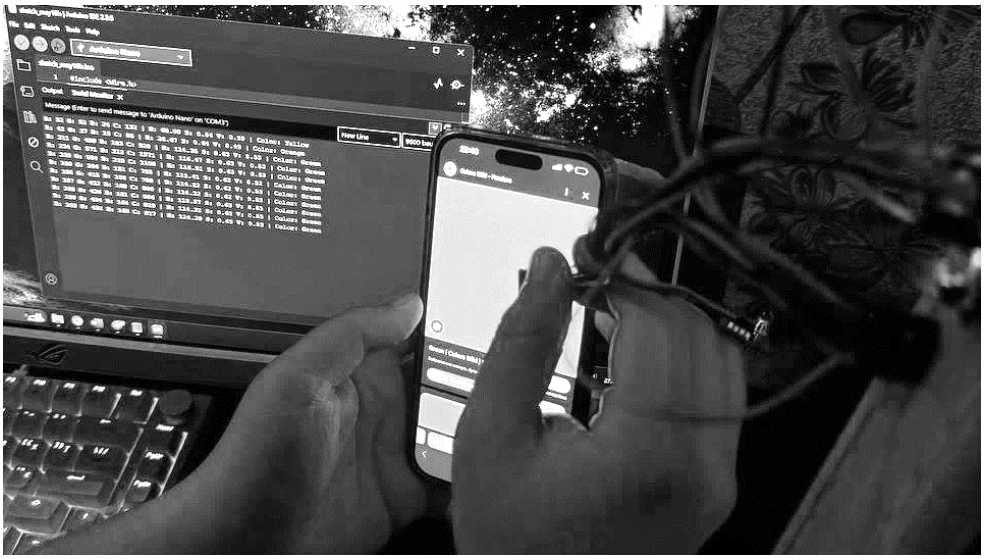

Рисунок 2.3 — Проведення тестування датчику на кубику дисплеї смартфону

Під час випробовування було визначено, що комплекс стабільно працює в усіх режимах. Живлення від акамуляторів забезпечувало безперервну роботу протягом, приблизно 2 годин. Найбільше енергоспоживання було зафіксоване при одночасній роботі Bluetooth-модуля, моторів та датчику визначення кольору. В

цілому, випробування підтвердили працездатність розробленої системи та її здатність до автономної навігації й взаємодії з користувачем.

3 ОПТИМІЗАЦІЯ ТА ПЕРСПЕКТИВИ РОЗВИТКУ

3.1 Оптимізація коду та підвищення ефективності

Сучасні роботизовані системи вимагають вимагають швидкість виконання команд та ефективне енергоспоживання. Затримки у виконанні команд можуть призвести до несвоєчасного реагування на зовнішні фактори, що є критичним для їх автономної роботи. Головним недоліком програми є досить велике використання функцій delay() та delayMicroseconds().

Ця функція призводить до зупинки виконання програми на вказаний проміжок часу. В програмі вона переважно використовується для створення додаткового часу, щоб завершити певну дію. Наприклад для виконання повороту пристрою (лістинг 3.1) чи виконання повороту ультразвукового датчика сервоприводом (лістинг 3.2).

Лістинг 3.1 — Використання затримки при повороті

```
if (detectedColor == "RED") {
        Car_Stop();
        delay(200);
        procedure(180);
        delay(500);
        Car_back();
        delay(1000);
        Car_Stop();
}
```

Лістинг 3.2 — Використання затримок при повороті сервоприводом

```
for (int i = 0; i < 5; i++)
  {
    digitalWrite(servoPin, HIGH);
    delayMicroseconds(pulsewidth);
    digitalWrite(servoPin, LOW);
    delay((20 - pulsewidth / 1000));
  }
```

В процесі виконання функцій delay() та delayMicroseconds(), операції зчитування даних з датчиків, математичні обчислювання чи керування виходами платформи Arduino не виконуються. При цьому дана функція не впливає на переривання, тому самі переривання, запис даних, які надходять через вихід RX

через послідовний інтерфейс а також ШІМ-сигнали, які формуються функцією analogWrite() будуть виконуватись [23].

Альтернативою для цих функцій є функція millis(), яка повертає кількість мілісекунд, які пройшли від часу початку виконання програми. Дане число може переповнитись і скинеться до нуля приблизно через 50 днів. Значення часу який повертається цією функцією має тип unsigned long, тому для того, щоб виконати будь-які математичні дії з цим значенням, його потрібно буде перевести в потрібний тип даних, наприклад int. З допомогою цієї функції можна створити заміну delay (лістинг 3.3):

Лістинг 3.3 — Функція для заміни delay()

```
unsigned long previousMillis = 0;
const long interval = 1000;
void nonBlockingDelay() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
  }
}
```

В змінній interval задається значення в мілісекундах, яке вказує на час затримки виконання певного коду. В змінній previousMillis записується початковий час виконання даної функції після чого різниця поточного значення millis (збережене в currentMillis) та початкового часу порівнюється з заданим інтервалом. Така функція дозволяє виконувати паралельно певні дії, але такий варіант реалізації затримок не є обов’язковим в розробленій програмі, через те що дії виконуються послідовного крок за кроком та немає необхідності зчитувати значення з безлічі датчиків.

Для збільшення енергоефективності програми можна реалізувати періодичне включення певних датчиків та вимкнення певних діодів, які можуть не використовуватись:

— датчик визначення кольору має особистий світлодіод, який постійно включено;

— інфрачервоний приймач постійно ввімкнений.

Датчик для визначення кольору досить залежить від освітленості середовища, тому що він отримує відбите світло від поверхні. Саме з допомогою світлодіоду на датчику, можна визначити колір перешкод при відсутності або дуже низькому зовнішньому освітленню.

Інфрачервоний приймач повинен постійно бути ввімкнений через необхідність моментально реагувати на подані команди через пульт, тому що при екстремальних умовах реакція на команди може привести до несвоєчасного реагування, що може призвести до ушкоджень приладу. Проте в режимах автономного керування, даний датчик можна вимкнути для енергозбереження приладу, тому в функції Avoid() та Avoid_color() додамо (лістинг 3.4):

Лістинг 3.4 — Команда вимкнення інфрачервоного приймача для енергоефективності

```
void Avoid()
{
 irrecv.disableIRIn();
```

Для ще більшого збільшення енергоефективності можна ввімкнути режим енергоспоживання процесора. Даний режим називається sleep-режим, він дозволяє виконати продовження програми після виконання переривання. Даний режим можна ввімкнути наступним чином (лістинг 3.5):

Лістинг 3.5 — Ввімкнення режиму енергоспоживання процесору

```
void enterSleep() {
   set_sleep_mode(SLEEP_MODE_IDLE);
   sleep_enable();
   sleep_mode(); // Перехід у режим очікування
   sleep_disable();
}
void loop() {
   if (/* умова активності */) {
      // Виконання основного коду
   } else {
      enterSleep();
   }
}
```

3.2 Можливості розширення функціоналу

Мікроконтролерна система побудована на платформі Arduino має великі можливості щодо масштабування та вдосконалення. Через відкриту архітектуру та наявність безлічі модулів, які сумісні з даною платформою, пристрій може бути суттєво розширитись як з боку апаратної, так і програмної частини.

До пристрою можуть додатись безліч нових датчиків, як от LIDAR (рисунок 3.6). Датчик з LIDAR побудований на технології отримання та обробки інформації про віддалені об’єкти за допомогою оптичних систем, такі системи використовують явища відбиття та розсіювання світла.

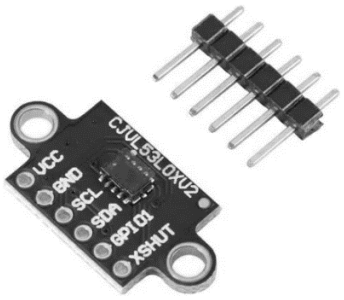

Рисунок 3.1 — Лазерний датчик відстані (LIDAR)

Використання даного датчику разом з ультразвуковим датчиком значно підвищить точність у визначенні перешкод. Також з допомогою LIDAR можна буде будувати тривимірні карти навколишнього середовища.

Для додаткового функціоналу пристрою, можна додати камеру (рисунок 3.7). З її допомогою можна буде впровадити функції комп’ютерного зору, наприклад розпізнавання певних об’єктів, ліній чи навіть QR-кодів. Поєднавши алгоритми обробки зображення, система зможе приймати рішення на основі візуальних даних в реальному часі.

Ще одним доповненням до пристрою може стати GPS-модуль (рисунок 3.8). Використовуючи його з розробленим пристроєм, можна буде будувати карту перешкод, та керувати пристроєм за допомогою координат, відстежувати переміщення пристрою та організувати роботу пристрою на відкритій місцевості.

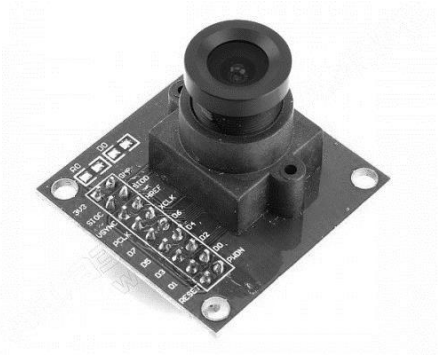

Рисунок 3.2 — Модуль камери OV7670

Рисунок 3.3 — Модуль GPS GY-NEO6MV2

Ще одним напрямком вдосконалення є вдосконалення та розширення каналів зв’язку для керування пристроєм. Замість ІЧ-пульта та Bluetooth-модулю, можна використовувати більш сучасні методи зв’язку — зокрема через мережу інтернет, використовуючи Wi-Fi або ж через мобільний зв’язок, використовуючи GMS-модуль. Це дозволить керувати пристроєм на більших дистанціях, через вебінтерфейс або ж через вдосконалений мобільний застосунок. Також можливе впровадження голосового асистенту, що дозволить керувати пристроєм голосовими командами. Таким чином взаємодія користувача може бути значно покращена та спрощена під час користування.

Іншим перспективним напрямком вдосконалення пристрою є використання алгоритмів штучного інтелекту. Проте через умови обмеженого обчислювального ресурсу платформи Arduino, алгоритми можуть бути реалізовані в спрощеному вигляді з використанням технології TinyML — це технологія машинного навчання на базі мікроконтролерів. Також можна реалізувати саму нейронну мережу на іншній більш продуктивній обчислювальній машині, з якою пристрій буде

спілкуватись, наприклад, через мережу інтернет. Тоді сам пристрій буде виступати в ролі комплексного датчика, який надсилає дані про навколишнє середовище у вигляді фото чи вимірів з інших датчиків, а обчислювальна машина буде отримувати виміряні дані, аналізувати їх та опираючись на них приймати рішення. Інтеграція нейронних мереж дозволить системі адаптуватись до змін середовища та приймати більш ефективні рішення без втручання користувача.

Отже, розроблена мікроконтролерна система має значний потенціал для вдосконалення та масштабування, інтегруючи нові датчики та технології.

3.3. Аналіз результатів

У процесі випробування автономного режиму роботи кіберфізичного комплексу було проведено оцінку точності роботи алгоритму уникнення перешкод, заснованого на показниках ультразвукового датчика HC-SR04. Основним критерієм точності в даному випадку є своєчасне виявлення перешкоди, коректне визначення відстані до неї та правильний вибір напрямку об’їзду.

Під час серії тестів пристрій рухався у напрямку до об'єктів, розміщених на різній відстані та під різними кутами. При досягненні відстані меншої за 20 см (порогове значення), система фіксувала наявність перешкоди та виконувала зміну траєкторії. Аналіз зібраних даних показав:

Середнє значення відстані, на якій відбувалося реагування системи: 18,4 см, що свідчить про невелике відхилення від заданого порогу, обумовлене часом обробки сигналу та інерційністю руху.

Середня похибка вимірювання відстані датчиком HC-SR04 у тестових умовах (порівняно з ручними вимірюваннями лінійкою) склала ±1,5 см, що є прийнятним результатом для задач короткодистанційної навігації.

Кількість успішно обійдених перешкод склала 96% від загальної кількості випробувань (24 з 25), що підтверджує високу ефективність алгоритму.

Також було протестовано зміну траєкторії у випадках, коли перешкоди розміщувалися зліва, справа або безпосередньо по центру. Система демонструвала здатність аналізувати вільні напрямки та змінювати курс руху у бік, де не фіксувалося перешкод. Це реалізовано шляхом простого аналізу даних від

сервоприводу, що обертає датчик, і дозволяє виконувати сканування простору під кількома кутами (наприклад, -45°, 0°, +45°).

Отримані дані дозволяють зробити висновок, що система має високу стабільність в умовах типових для внутрішніх приміщень, таких як кімнати або лабораторії. При цьому точність виявлення перешкод є достатньою для реалізації простих задач автономної навігації, таких як рух у коридорі, уникнення статичних об’єктів тощо.

Подальше покращення точності можна досягти за рахунок:

— використання кількох ультразвукових датчиків, що дозволяє охопити ширший кут огляду;

— фільтрації шумів показників датчика через алгоритми згладжування;

— поєднання ультразвукових вимірювань з оптичними або інфрачервоними датчиками для підвищення надійності.

Таким чином, точність роботи системи в режимі обходу перешкод відповідає очікуваним характеристикам для недорогих мобільних платформ та може бути вдосконалена у рамках подальших досліджень.

Результати випробувань підтвердили працездатність розробленого кіберфізичного комплексу збору даних. Система стабільно функціонує у всіх передбачених режимах, має зрозумілий інтерфейс керування та здатна до автономної навігації в середовищі з перешкодами. Інтеграція датчиків кольору й відстані, а також можливість перемикання між режимами роботи дозволяють використовувати пристрій у різноманітних задачах.

Серед можливих напрямків практичного застосування комплексу можна виділити:

— навчальні цілі — як платформа для ознайомлення з основами робототехніки та програмування;

— логістика на складі — для транспортування невеликих вантажів за кольоровими мітками;

— дослідницькі цілі — для розробки та тестування нових алгоритмів навігації й розпізнавання об'єктів;

— інтелектуальні іграшки або освітні набори для STEM-навчання.

У подальшому розвиток системи може передбачати додавання модулів зберігання та обробки даних, GPS-навігації, Wi-Fi-модуля для віддаленої передачі інформації, а також покращення енергоефективності та дизайну шасі для роботи на складніших поверхнях [25].

Таким чином, розроблений комплекс демонструє хороші результати як у керованому, так і в автономному режимах, що свідчить про доцільність його подальшого розвитку та адаптації до практичних завдань.

ВИСНОВКИ

У комплексній бакалаврській кваліфікаційній роботі було спроєктовано та реалізовано кіберфізичний комплекс збору даних, призначений для автономного моніторингу навколишнього середовища та взаємодії з користувачем через різні інтерфейси. Система включає керований модуль на базі платформи Arduino з підключеними датчиками відстані (HC-SR04), кольору (TCS34725), світла, Bluetooth-модулем, ІЧ-приймачем.

У ході роботи було проаналізовано сучасні підходи до побудови кіберфізичних систем, що дозволило обґрунтовано обрати склад апаратних компонентів та програмну архітектуру. Реалізовано підтримку декількох режимів роботи пристрою: ручне керування (через ІЧ-пульт та Bluetooth) і автономна навігація з можливістю уникнення перешкод та виявлення кольору. Для кожного режиму реалізовано окрему логіку, яка враховує вхідні дані від відповідних сенсорів.

Було реалізовано програмне забезпечення, що дозволяє пристрою отримувати команди в реальному часі від користувача, самостійно орієнтуватись у просторі, використовуючи дані ультразвукового сенсора для уникнення перешкод та виявляти кольори об'єктів та адаптувати поведінку залежно від кольору.

Проведено комплексне тестування роботи системи в реальних умовах. Результати випробувань підтвердили стабільність функціонування пристрою, високу точність розпізнавання перешкод та кольорів, а також зручність у керуванні. Було зафіксовано, що система коректно реагує на об’єкти, розташовані на відстані до 20 см, успішно обходить перешкоди та виконує обробку команд з мінімальною затримкою. Автономна робота системи можлива протягом 2 годин без підзарядки, що є достатнім для типових сценаріїв використання.

Таким чином, усі поставлені цілі бакалаврської кваліфікаційної роботи досягнуто. Розроблена система може бути використана як основа для створення більш складних мобільних роботизованих платформ, освітніх STEM-проєктів, систем розумного моніторингу або автономних пристроїв для дослідження середовища. У перспективі можливе подальше розширення функціональності за

рахунок інтеграції з хмарними сервісами, використання додаткових сенсорів та застосування штучного інтелекту для складніших сценаріїв поведінки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кіберфізичний комплекс збору даних з можливістю обходу перешкод / Богомолов С.В., Заїченко І.Г., Данилюк І.В. // Матеріали LIV наукової-технічної конференції підрозділів (Вінниця, 2025 р.) [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24075/19939

2. Інтегроване середовище розробки. Wikipedia: вебсайт. URL: https://uk.wikipedia.org/wiki/

3. Климчук В. Програмування мікроконтролерів AVR засобами Arduino. Львів : Львівська політехніка, 2020. ISBN 978-966-941-460-2.

4. Початок роботи з Arduino. ArduinoUa: вебсайт. URL: https://doc.arduino.ua/ru/guide/Windows

5. Об’єктно-Орієнтоване Програмування (ООП). Пояснюємо чотири основні принципи, SoftServeInc: вебсайт. URL: https://career.softserveinc.com/uk-ua/stories/what-is-object-oriented-programming-oop-explaining-four-major-principles

6. C++. Wikipedia: вебсайт. URL: https://uk.wikipedia.org/wiki/C%2B%2B

7. Функції. Acode: вебсайт. URL: https://acode.com.ua/urok-15-funktsiyi-i-operator-return/

8. Розмір типів даних. Acode: вебсайт. URL: https://acode.com.ua/urok-33-rozmir-typiv-danyh/

9. Arduino Robot. ArduinoUa: вебсайт. URL: https://doc.arduino.ua/ru/guide/Robot

10. Структурна теорема Бьома — Якопіні. Wikipedia: вебсайт. URL: https://uk.wikipedia.org/wiki/%D0%

11. Оператори в C та C++. Wikipedia: вебсайт. URL: https://uk.wikipedia.org/wiki/%D0%9E%D

12. Датчик. Wikipedia: вебсайт. URL: https://uk.wikipedia.org/wiki/%D0%94%D0%B0%D1%82%D1%87%D0%B8%D0%B

13. Системи обробки даних, інформаційні системи та їх класифікація. BulkLib: вебсайт. URL: https://buklib.net/books/23517/

14. Інтерполяція. Vntu.edu.ua: вебсайт. URL: https://web.posibnyky.vntu.edu.ua/fksa/2kvetnyj_komp'yuterne_modelyuvannya_system_procesiv/t1/612..htm

15. Апостолюк В. О. Інтелектуальні системи керування. Київ : НТУУ «КПІ», 2008.

16. Функція digitalWrite(). ArduinoUa: вебсайт. URL: https://doc.arduino.ua/ru/prog/DigitalWrite

17. Імпульсна і середня потужність. Wikipedia: вебсайт. URL: https://www.radartutorial.eu/01.basics/rb53.uk.html#:.

18. Інфрачервона (ІЧ) спектроскопія. Libretexts: вебсайт. URL: https://ukrayinska.libretexts.org/%D

19. Сервопривід. Vsepro: вебсайт. URL: https://vsepro.com.ua/scho-take-servopryvid-i-dlya-choho-vin-potriben/

20. Функція map(). ArduinoUa: вебсайт. URL: https://doc.arduino.ua/ru/prog/Map

21. Колірна модель HSV. Stud.com: вебсайт. URL: https://stud.com.ua/43365/informatika/kolirna_model

22. Огляд видів тестування. Qatestlab: вебсайт. URL: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/

23. Функція delay(). ArduinoUa: вебсайт. URL: https://doc.arduino.ua/ru/prog/Delay

24. Датчики для Arduino. ArduinoUa: вебсайт. URL: https://arduino.ua/ru/cat6-atchiki?srsltid=AfmBOorQ7fFqcAlLgt9qfi-G5batQGdQySoQyu0eaTL8hEyS4sO0Yp6H

25. Сфери використання інтернету речей. Imena: вебсайт. URL: https://www.imena.ua/blog/top-10-scope-iot/

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«21» березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи
«Кіберфізичний комплекс збору даних з можливістю обходу перешкод.
Частина 2. Програмне забезпечення»
08-54. КБКР.039.00.000 ТЗ

Науковий керівник: к.т.н., доц. каф. ОТ
Богомолов С.В

Студент групи 1СП-21б
Данилюк І.В.

м. Вінниця — 2025

1 Підстава для використання бакалаврської кваліфікаційної роботи (КБКР)

1.1 Кіберфізичні системи є сучасним технологічним рішенням, яке поєднує фізичні процеси з цифровими обчисленнями та взаємодію з допомогою мережевих технологій. Вони мають застосування в різних сферах промисловості, транспорту, сільського господарства та дослідження навколишнього середовища.

1.2 Наказ про затвердження теми кваліфікаційної роботи.

2 Мета КБКР і призначення розробки

2.1 Мета роботи — розробка програмного забезпечення для кіберфізичної системи на базі платформи Arduino, яка виконує автоматичне визначення та обхід перешкоди у фізичному середовищі й виконує збір обробку та аналіз даних з датчиків.

2.2 Призначення розробки — розробка програмного забезпечення для кіберфізичного комплексу збору даних на базі платформи Arduino, який виконує автоматичне визначення та обхід перешкоди.

3 Вихідні дані для виконання КБКР

3.1 Вибір платформи та середовища розробки.

3.2 Аналіз архітектуру програмного забезпечення.

3.3 Створення алгоритму збору даних та обходу перешкод.

3.4 Програмування контролера, інтеграція сенсорів та обробка сигналів.

3.5 Оптимізація програмного забезпечення та підвищення ефективності.

3.6 Випробування системи в реальних умовах та аналіз результатів.

4 Вимоги до виконання КБКР

Головна вимога — розробити програмне забезпечення для кіберфізичного комплексу збору даних на базі платформи Arduino з можливістю обходу перешкод.

5 Етапи КБКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

№ етапу	Назва етапу	Термін виконання	Очікувані результати
1	Аналіз існуючих платформ та середовища розробки	21.03.25 — 30.03.25	Аналітичний огляд
2	Реалізація та тестування	31.03.25 — 13.04.25	Розділ 2
3	Оптимізація та перспективи розвитку	14.04.25 — 27.04.25	Розділ 3
4	Оформлення пояснювальної записки, графічного матеріалу та презентації	28.04.25 — 11.05.25	Пояснювальна записка, графічний матеріал та презентація

6 Матеріали, що подаються до захисту КБКР

До захисту подаються: пояснювальна записка КБКР, графічні та ілюстративні матеріали, протокол попереднього захисту на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до КБКР українською та іноземною мовами, довідка про відповідність оформлення вимогам.

7 Порядок контролю виконання та захисту КБКР

Контроль виконання етапів здійснюється науковим керівником згідно з встановленими термінами. Захист БКР проходить на засіданні екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлення та порядок виконання КБКР

8.1 При оформленні БКР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Системне програмування») — кафедра обчислювальної техніки, ВНТУ 2023.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:

Кіберфізичний комплекс збору даних з можливістю обходу перешкод. Частина 2. Програмне забезпечення

Тип роботи: Бакалаврська кваліфікаційна робота

(БДР, МКР)

Підрозділ кафедра обчислювальної техніки

(кафедра, факультет)

Показники звіту подібності StrikePlagiarism

Оригінальність 99,5% Схожість 0,5%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _ Захарченко С. М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism щодо роботи.

Автор роботи _____ Данилюк І.В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Богомолов С.В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

ІЛЮСТРАТИВНА ЧАСТИНА

КІБЕРФІЗИЧНИЙ КОМПЛЕКС ЗБОРУ ДАНИХ З МОЖЛИВІСТЮ ОБХОДУ ПЕРЕШКОД. ЧАСТИНА 2. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

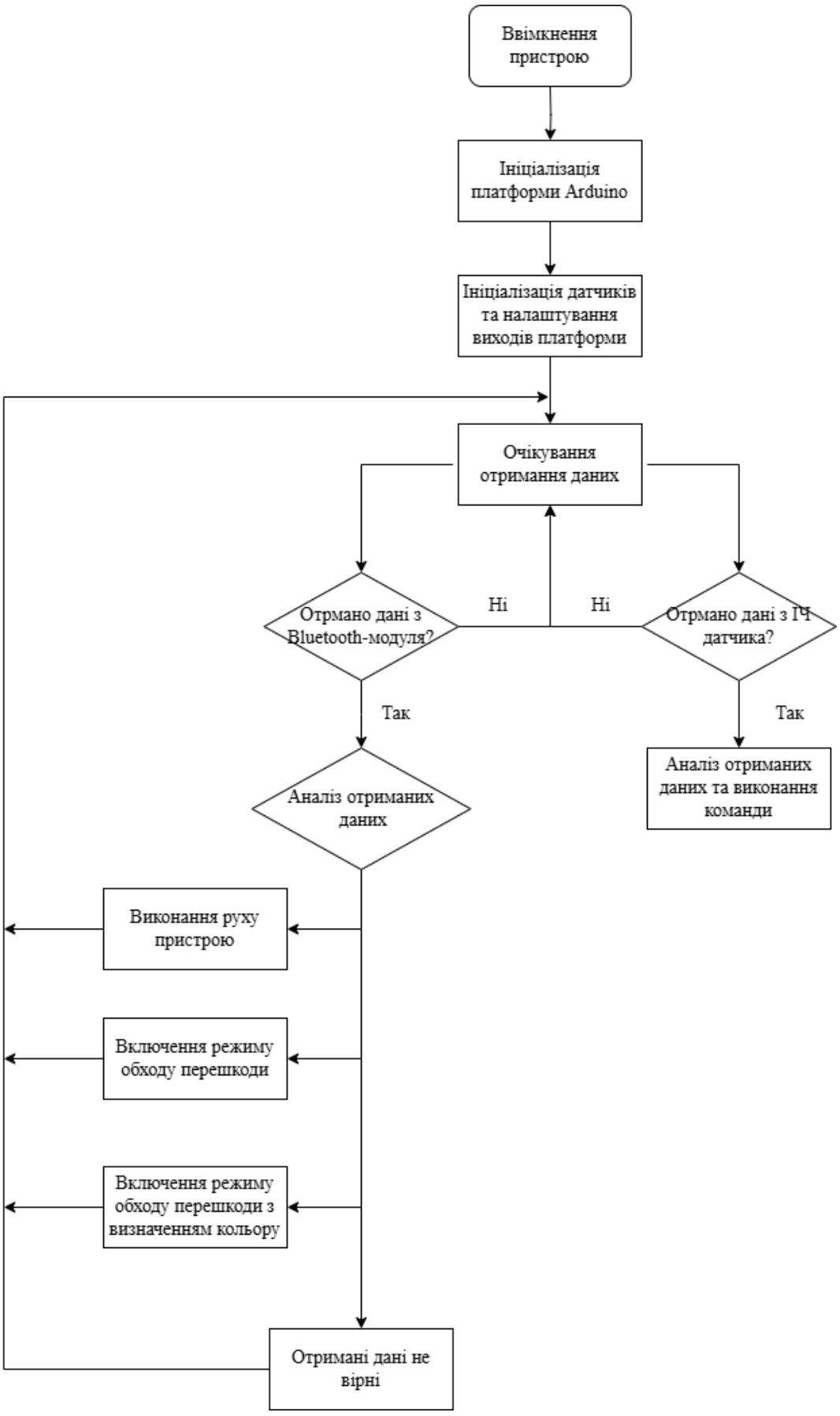

Рисунок В.1 – Структурна схема алгоритму

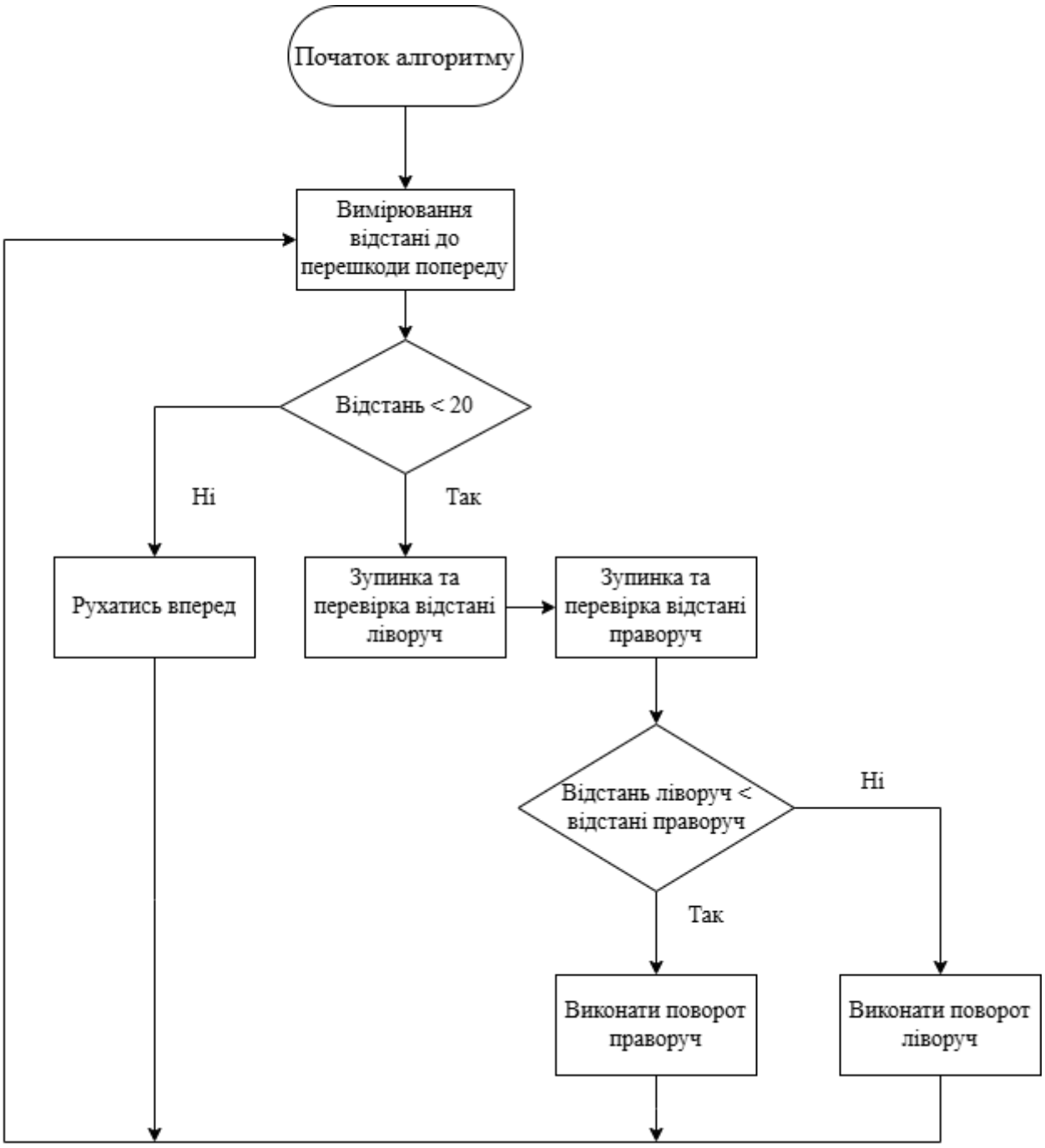

Рисунок В.2 — Алгоритм обходу перешкоди

ДОДАТОК Г

Лістинг програмного коду файлу KKcode.ino

```
#include <IRremote.h>
#include <Wire.h>
#include "Adafruit_TCS34725.h"
IRrecv irrecv(3);
decode_results results;
long ir_rec;
unsigned char start01[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x80,
0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};
unsigned char STOP01[] = {0x2E, 0x2A, 0x3A, 0x00, 0x02, 0x3E, 0x02, 0x00, 0x3E,
0x22, 0x3E, 0x00, 0x3E, 0x0A, 0x0E, 0x00};
unsigned char front[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x24, 0x12, 0x09, 0x12, 0x24,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
unsigned char back[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x24, 0x48, 0x90, 0x48, 0x24,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
unsigned char left[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x44, 0x28, 0x10, 0x44,
0x28, 0x10, 0x44, 0x28, 0x10, 0x00};
unsigned char right[] = {0x00, 0x10, 0x28, 0x44, 0x10, 0x28, 0x44, 0x10, 0x28, 0x44,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
unsigned char Smile[] = {0x00, 0x00, 0x1c, 0x02, 0x02, 0x02, 0x5c, 0x40, 0x40, 0x5c,
0x02, 0x02, 0x02, 0x1c, 0x00, 0x00};
unsigned char Disgust[] = {0x00, 0x00, 0x02, 0x02, 0x02, 0x12, 0x08, 0x04, 0x08,
0x12, 0x22, 0x02, 0x02, 0x00, 0x00, 0x00};
unsigned char Happy[] = {0x02, 0x02, 0x02, 0x02, 0x08, 0x18, 0x28, 0x48, 0x28,
0x18, 0x08, 0x02, 0x02, 0x02, 0x02, 0x00};
unsigned char Squint[] = {0x00, 0x00, 0x00, 0x41, 0x22, 0x14, 0x48, 0x40, 0x40, 0x48,
0x14, 0x22, 0x41, 0x00, 0x00, 0x00};
unsigned char Despise[] = {0x00, 0x00, 0x06, 0x04, 0x04, 0x04, 0x24, 0x20, 0x20,
0x26, 0x04, 0x04, 0x04, 0x04, 0x00, 0x00};
unsigned char Heart[] = {0x00, 0x00, 0x0C, 0x1E, 0x3F, 0x7F, 0xFE, 0xFC, 0xFE,
0x7F, 0x3F, 0x1E, 0x0C, 0x00, 0x00, 0x00};
unsigned char clear[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
#define SCL_Pin  A5
#define SDA_Pin  A4  /
#define ML_Ctrl 4
#define ML_PWM 6
#define MR_Ctrl 2
#define MR_PWM 5
Adafruit_TCS34725 tcs =
Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
TCS34725_GAIN_4X);
```

```
char ble_val;
byte speeds_L = 200;
byte speeds_R = 200;
String speeds_l, speeds_r;
#define light_L_Pin A1
#define light_R_Pin A2
int left_light;
int right_light;

#define servoPin    10  //servo Pin
#define Trig        12
#define Echo        13
float distance;

int a;
int a1;
int a2;
bool flag;
void setup() {
  if (tcs.begin()) {
    Serial.println("Датчик кольору готовий!");
  } else {
    Serial.println("Помилка: датчик не знайдено!");
  }
  tcs.setInterrupt(false);
  pinMode(ML_Ctrl, OUTPUT);
  pinMode(ML_PWM, OUTPUT);
  pinMode(MR_Ctrl, OUTPUT);
  pinMode(MR_PWM, OUTPUT);
  pinMode(L_pin, INPUT);
  pinMode(M_pin, INPUT);
  pinMode(R_pin, INPUT);
  matrix_display(clear);
  matrix_display(start01);
  pinMode(servoPin, OUTPUT);
  pinMode(light_L_Pin, INPUT);
  pinMode(light_R_Pin, INPUT);
  pinMode(Trig, OUTPUT);
  pinMode(Echo, INPUT);
  procedure(90);
}
void loop() {
  if (Serial.available())
  {
    ble_val = Serial.read();
```

```
    Serial.println(ble_val);
    switch (ble_val) {
      case 'F': Car_front(); break;
      case 'B': Car_back(); break;
      case 'L': Car_left(); break;
      case 'R': Car_right(); break;
      case 'S': Car_Stop();  break;
      case 'g': Avoid(); break;
      case 'h': Avoid_color(); break;
      case 'u':
        speeds_l = Serial.readStringUntil('#');
        speeds_L = String(speeds_l).toInt();
        break;
      case 'v':
        speeds_r = Serial.readStringUntil('#');
        speeds_R = String(speeds_r).toInt();
        break;
      case 'k': matrix_display(Smile);    break;
      case 'l': matrix_display(Disgust);  break;
      case 'm': matrix_display(Happy);    break;
      case 'n': matrix_display(Squint);   break;
      case 'o': matrix_display(Despise);  break;
      case 'p': matrix_display(Heart);    break;
      case 'z': matrix_display(clear);    break;
      default: break;
    }
  }
  if (irrecv.decode(&results)) {
    ir_rec = results.value;
    Serial.println(ir_rec, HEX);
    switch (ir_rec) {
      case 0xFF629D: Car_front();   break;
      case 0xFFA857: Car_back();    break;
      case 0xFF22DD: Car_left();    break;
      case 0xFFC23D: Car_right();   break;
      case 0xFF02FD: Car_Stop();    break;
      default: break;
    }
    irrecv.resume();
  }
}
float checkdistance() {
  float distance;
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
```

```
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);
  distance = pulseIn(Echo, HIGH) / 58.20;
  delay(10);
  return distance;
}
void procedure(int myangle) {
  int pulsewidth;
  pulsewidth = map(myangle, 0, 180, 500, 1800);
  for (int i = 0; i < 5; i++)
  {
    digitalWrite(servoPin, HIGH);
    delayMicroseconds(pulsewidth);
    digitalWrite(servoPin, LOW);
    delay((20 — pulsewidth / 1000));
  }
}
void Avoid()
{
  irrecv.disableIRIn();
  flag = 0;
  while (flag == 0)
  {
    a = checkdistance();
    if (a < 20) {
      Car_Stop();
      delay(500);
      procedure(180);
      delay(500);
      a1 = checkdistance();
      delay(100);
      procedure(0);
      delay(500);
      a2 = checkdistance();
      delay(100);
      procedure(90);
      delay(500);
      if (a1 > a2) {
        Car_left();
        delay(700);
      } else {
        Car_right();
        delay(700);
      }
```

```
    }
    else {
      Car_front();
    }
    if (Serial.available())
    {
      ble_val = Serial.read();
      if (ble_val == 'S')
      {
        flag = 1;
        Car_Stop();
      }
    }
  }
}
void Avoid_color()
{
  irrecv.disableIRIn();
  flag = 0;
  while (flag == 0) {
    a = checkdistance();
    if (a < 20) {
      while (a > 5) {
        Car_front();
        delay(100);
        a = checkdistance();
        if (Serial.available()) {
          ble_val = Serial.read();
          if (ble_val == 'S') {
            flag = 1;
            Car_Stop();
            return;
          }
        }
      }
      Car_Stop();
      delay(500);
      String detectedColor = detectColor();
      if (detectedColor == "RED") {
          Car_Stop();
          delay(200);
          procedure(180);
          delay(500);
          Car_back();
          delay(1000);
```

```
        Car_Stop();
      }
      else if (detectedColor == "GREEN") {
        Car_left();
        delay(700);
        Car_Stop();
      }
      else if (detectedColor == "BLUE") {
        Car_right();
        delay(700);
        Car_Stop();
      }
      else {
        procedure(180);
        delay(500);
        a1 = checkdistance();
        delay(100);
        procedure(0);
        delay(500);
        a2 = checkdistance();
        delay(100);
        procedure(90);
        delay(500);
        if (a1 > a2) {
          Car_left();
          delay(700);
        } else {
          Car_right();
          delay(700);
        }
      }
    }
    else {
      Car_front();
    }
    if (Serial.available()) {
      ble_val = Serial.read();
      if (ble_val == 'S') {
        flag = 1;
        Car_Stop();
      }
    }
  }
}
String getDetectedColor() {
```

```
  uint16_t r, g, b, c;
  tcs.getRawData(&r, &g, &b, &c);
  uint8_t nr = (float)r / c * 255;
  uint8_t ng = (float)g / c * 255;
  uint8_t nb = (float)b / c * 255;
  float h, s, v;
  rgbToHsv(nr, ng, nb, &h, &s, &v);
  if ((h >= 0 && h < 15) || (h >= 345 && h <= 360)) return "RED";
  if (h >= 70 && h < 170) return "GREEN";
  if (h >= 200 && h < 260) return "BLUE";
  return "UNKNOWN";
}
void rgbToHsv(uint8_t r, uint8_t g, uint8_t b, float* h, float* s, float* v) {
  float fr = r / 255.0;
  float fg = g / 255.0;
  float fb = b / 255.0;

  float maxVal = max(fr, max(fg, fb));
  float minVal = min(fr, min(fg, fb));
  float delta = maxVal — minVal;
  *v = maxVal;
  *s = (maxVal == 0) ? 0 : delta / maxVal;
  if (delta == 0) {
    *h = 0;
  } else if (maxVal == fr) {
    *h = 60 * fmod(((fg — fb) / delta), 6);
  } else if (maxVal == fg) {
    *h = 60 * (((fb — fr) / delta) + 2);
  } else {
    *h = 60 * (((fr — fg) / delta) + 4);
  }
  if (*h < 0) *h += 360;
}
void matrix_display(unsigned char matrix_value[])
{
  IIC_start();
  IIC_send(0xc0);
  for (int i = 0; i < 16; i++)
  {
    IIC_send(matrix_value[i]);
  }
  IIC_end();
  IIC_start();
  IIC_send(0x8A);
  IIC_end();
```

```
}
unsigned long previousMillis = 0;
const long interval = 1000;
void nonBlockingDelay() {
  unsigned long currentMillis = millis();
  if (currentMillis — previousMillis >= interval) {
    previousMillis = currentMillis;
  }
}
void Car_back() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, speeds_R);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, speeds_L);
  matrix_display(back);
}
void Car_front() {
  digitalWrite(MR_Ctrl, HIGH);
  analogWrite(MR_PWM, 255 — speeds_R);
  digitalWrite(ML_Ctrl, HIGH);
  analogWrite(ML_PWM, 255 — speeds_L);
  matrix_display(front);
}
void Car_left() {
  digitalWrite(MR_Ctrl, HIGH);
  analogWrite(MR_PWM, 255 — speeds_R);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, speeds_L);
  matrix_display(left);
}
void Car_right() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, speeds_R);
  digitalWrite(ML_Ctrl, HIGH);
  analogWrite(ML_PWM, 255 — speeds_L);
  matrix_display(right);
}
void Car_Stop() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, 0);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, 0);
  matrix_display(STOP01);
}
```