Міністерство освіти і науки України

Чернівецький національний університету імені Юрія Федьковича»

Відокремлений структурний підрозділ «Фаховий коледж Чернівецького національного університету імені Юрія Федьковича»

Природниче відділення

Циклова комісія комп'ютерної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»

зі спеціальності *123 Комп'ютерна інженерія*

підготовки за освітньо-професійною програмою «*Комп'ютерна інженерія*»

на тему: «**Програмна частина програмно-апаратного комплексу гусеничного керованого робота»**

Виконав (ла):
студент (ка) 4-го курсу *Ракул Катерина Олександрівна* _____
(прізвище, ім'я та по батькові) (підпис)
Керівник:
к. ф.-м. н., Тащук О.Ю. _____
(науковий ступінь, вчене звання, прізвище та ініціали) (підпис)
Рецензент:
Букурос О.В. _____
(науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

До захисту допущено:
Протокол засідання циклової комісії № _
від „___"_____ 2024 р.
Голова циклової комісії _____ Олександр ТАЩУК

Чернівці, 2024

Завдання та календарний план роботи

Міністерство освіти і науки України
Чернівецький національний університету імені Юрія Федьковича»
Відокремлений структурний підрозділ «Фаховий коледж
Чернівецького національного університету імені Юрія Федьковича»

Затверджую
Голова циклової комісії,
Олександр ТАЩУК.

_____
«____» _____2023 р.

Завдання

На кваліфікаційну роботу	Ракул Катерина Олександрівна
Тема кваліфікаційної роботи	Програмна частина програмно-апаратного комплексу гусеничного керованого робота
Постановка завдання в короткій формі	Моделювання, розробка та програмування програмної частини, апаратно-програмного комплексу гусеничного керованого робота.
Вихідні дані (2-3 адреси сайтів, матеріали яких рекомендує керівник кваліфікаційної роботи)	https://wiki.keyestudio.com/Ks0428_keyestudio_Mini_Tank_Robot_V2#Product_List https://docs.keyestudio.com/projects/KS0555/en/latest/arduino/arduino.html

Календарний план роботи

Дата отримання версії звіту керівником	Етап виконання роботи	Виконання (зазначає керівник)
10.10.23	Отримання завдання до кваліфікаційної роботи.	
12.11.23	Огляд літератури, присвяченої розробці ПЗ автономних мобільних роботів.	
16.12.23	Аналіз існуючих конструкторських рішень. Оформлення розділу 1.	
05.01.24	Вибір мови програмування, узгодження модулів системи.	
15.02.24	Написання теоретичного матеріалу по використовуваних модулях.	

11.03.24	Проектування програмної частини, підключення та монтаж деталей мобільного робота. Оформлення розділу 2.	
09.04.24	Тестування та перевірка, працездатності ПЗ. Оформлення розділу 3.	
18.05.24	Оформлення та редагування кваліфікаційної роботи.	
31.05.24	Подання роботи на перевірку Інтернет-сервісом Unicheck.	
31.05.24	Подання роботи на рецензію	
06.06.24	Представлення роботи на засіданні Циклової комісії	
за графіком	Захист кваліфікаційної роботи	

Дата видачі завдання _____.

Студент Катерина РАКУЛ

Керівник (П.І.Б) Олександр ТАЩУК

АНОТАЦІЯ

Ракул К.О. Програмна частина програмно-апаратного комплексу гусеничного керованого робота. Кваліфікаційна робота.

В роботі проведено огляд теоретичних основ, інструментів та технологій, що використовуються у розробці програмного забезпечення для автономних мобільних роботів, зроблено аналіз вимог що до проектування програмної частини, виконано реалізацію та інтеграцію програмної частини з апаратною та здійснено випробування програмно-апаратного комплексу гусеничного керованого робота

Актуальність роботи пояснюється тим, що мобільні роботи є універсальними механізмами, вони можуть відтворювати функції, які притаманні людини а також роботи здатні виконувати ті види діяльності які для людей є трудомісткими, важкими, монотонними, шкідливими для здоров'я і життя, також АМР застосовуються для автоматизації виробництва за рахунок чого зростає продуктивність праці та поліпшується якість продукції.

Конструювання мобільних роботів завжди вимагає вдосконалення як апаратної, так і програмної частини. У випадку коли якість апаратного забезпечення висока, але програмна частина недостатньо ефективна або нестабільна, це може призвести до непродуктивності робота або навіть до аварійних ситуацій.

Таким чином, актуальність роботи полягає в потребі розробки надійного, ефективного та легко налагоджуваного програмного забезпечення для керованого робота, що дозволить підвищити продуктивність, безпеку та функціональні можливості таких систем, це в свою чергу призведе до їх більш широкого застосування в різних галузях та підвищення ефективності робочих процесів.

Метою роботи є реалізація програмної частини яка включає в себе розробку основних модулів та алгоритмів керування, імплементацію комунікаційного протоколу між апаратною та програмною частинами, тестування та налагодження програмного забезпечення, а також інтеграцію програмного

продукту з апаратною частиною.

Об’єктом дослідження є програмна частина, апаратно-програмного комплексу гусеничного керованого робота.

Робота складається з 3 розділів, в яких проведено огляд теоретичних основ, інструментів та технологій, що використовуються у розробці програмного забезпечення для автономних мобільних роботів, зроблено аналіз вимог що до проектування програмної частини, виконано реалізацію та інтеграцію програмної частини з апаратною та здійснено випробування програмно-апаратного комплексу гусеничного керованого робота

Кваліфікаційна робота містить 57 сторінок, 5 рисунків та 15 посилань на літературні джерела.

Ключові слова: *Гусеничний робот, програмування робота, керування рухом, сенсорні системи, ультразвуковий датчик, датчики лінії, фоторезистори, алгоритми керування, автономні роботи, Arduino, Широтно-імпульсна модуляція, серійний монітор, автоматична навігація, робототехніка,*

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

AGV – (Automated Guided Vehicles) – автоматизовані керовані транспортні засоби.

AMR (Autonomous Mobile Robots) – Автономні мобільні роботи (АМР).

ОС – операційна система.

ПЗ – програмне забезпечення.

ПР – промисловий робот.

ГКР – гусеничний керований робот.

САПР – система автоматизованого проектування.

СП – система приводів.

ШІМ – широтно-імпульсна модуляція

ICR – миттєвий центр обертання (Instantaneous Center of Rotation)

Wi-Fi – технологія бездротової локальної мережі.

BLE – безпровідна технологія Bluetooth з низьким енергоспоживанням.

GPIO – інтерфейс для зв'язку між компонентами в системі комп’ютерів.

PCB – друкована плата

LCD- дисплей (Liquid Crystal) – рідкокристалічний дисплей.

ВСТУП

Однією з особливостей сучасної науково-технічної революції є впровадження роботів у різні сфери людської діяльності, роботизовані системи стали невід’ємною складовою сучасного світу і знаходять застосування у багатьох галузях, таких як промисловість, медицина, транспорт, наукові дослідження, виробничі та військові цілі, роботи можуть виконувати завдання, які не під силу людині або є небезпечними для життя.

Мобільні роботи являються універсальними машинами що здатні відтворювати рухові та інтелектуальні функцій людини а також можуть виконувати ті види діяльності які для людей є трудомісткими, важкими, монотонними, шкідливими для здоров'я і життя, в основному це допоміжні та основні виробничі операції; робота в екстремальних умовах та дослідженнях, вони також застосовуються для комплексної автоматизації виробництва, зростання продуктивності праці, поліпшення якості продукції[1].

Успішна реалізація мобільних роботів завжди вимагає вдосконалення як апаратної, так і програмної частини. У випадку коли гарантується висока якість апаратного забезпечення, але програмна частина недостатньо ефективна або нестабільна, це може призвести до непродуктивності робота або навіть до аварійних ситуацій.

Таким чином, актуальність проблеми полягає в потребі розробки надійного, ефективного та легко налагоджуваного програмного забезпечення для керованого робота. Що дозволить підвищити продуктивність, безпеку та функціональні можливості таких систем, що в свою чергу призведе до їх більш широкого застосування в різних галузях та підвищення ефективності робочих процесів.

Об’єктом дослідження є програмна частина, апаратно-програмного комплексу гусеничного керованого робота.

Предметом дослідження є програмна частина програмно-апаратного комплексу гусеничного керованого робота, яка включає у себе аналіз, розробку та вдосконалення програмного забезпечення, яке керує

функціональністю та роботою робота. Вивчення програмної частини а саме розгляд алгоритмів керування, обробки даних, взаємодії з апаратною частиною, а також впливу програмного забезпечення на ефективність та безпеку робота. Це дослідження спрямоване на створення оптимальних програмних рішень, які допоможуть досягти максимальної продуктивності, надійності та функціональності гусеничного керованого робота.

Мета роботи: моделювання та розробка програмної частини, апаратно-програмного комплексу гусеничного керованого робота.

Для досягнення поставленої мети необхідно виконати наступні задачі:

– Провести огляд інструментів та технологій, які використовуються при розробці програмного забезпечення.

– Змоделювати, принципи функціонування програмної частини робота.

– Проаналізувати основні модулі та алгоритмів керування роботом.

– Виконати імплементацію комунікаційного протоколу між апаратною та програмною частинами.

– Здійснити тестування та налагодження програмного забезпечення.

Методи дослідження: В роботі використано комбінаційні методи, та підходи які застосовуються для аналізу, розробки та валідації програмного забезпечення.

Практична цінність полягає в тому що, отримані в роботі результати можна застосувати для подальшої розробки великих автоматизованих транспортних систем з надійним, ефективним та легко налагоджуваним програмним забезпеченням, де вони матимуть значну практичну цінність.

ПОСТАНОВКА ЗАДАЧІ

1.1. Архітектура апаратно програмного - комплексу автономного керованого робота

Автономні мобільні роботи (AMR) в сучасному світі займають важливе місце в таких сферах, як промисловість, транспорт, медицина, наука та багато інших. Їх розробка та успішне функціонування забезпечує взаємодію та керування роботом котре базуються на інтеграції програмного та апаратного забезпечення в єдину систему, яка вимагає високої ефективності та надійності, що дозволяє роботам ефективно функціонувати в автономному режимі та виконувати завдання в складних умовах без постійного втручання людини.

Апаратна та програмна складові архітектури AMR рис.1.1. є надзвичайно важливими для забезпечення його ефективної та стабільної виконання дій в різних умовах. [2-4]

Рисунок 1.1 – Узагальнена структура архітектура апаратного – програмно комплексу автономного мобільного робота

Апаратна складова включає в себе різноманітні компоненти, які забезпечують його рух та функціонування. До складу основних елементів апаратної частини відноситься:

*Механічна платформа: М*еханічна структура робота, яка складається з рами, гусениць, двигунів, інших та механічних компонент, що забезпечують його рух та маневреність на різних поверхнях.

Електроніка: Система управління та контролю робота базується на вбудованих мікроконтролерах або мікропроцесорах, які відповідають за обробку сигналів від сенсорів, керування рухом та виконання програмних алгоритмів. Така система виконує роль "мозку" робота, об'єднуючи всі його компоненти і координуючи їхню діяльність надсилаючи команди від мікроконтролера до системи управління двигунами, СУ здійснює перетворення команди на механічні дії, які забезпечують рух робота. Це може включати управління гусеницями, колесами або іншими механізмами, залежно від типу робота.

Датчики: Гусеничний робот обладнаний різноманітними датчиками та сенсорами, які забезпечують збір інформації про стан робота та навколишнє середовище. Вони можуть включати гіроскопи, акселерометри, датчики температури, рівня освітленості та інші датчики, які допомагають роботові адаптуватися до змінних умов.

Актуатори: Двигуни та сервоприводи які відповідають за рух та маневреність робота, керуючи обертанням коліс гусениць та інших рухомих елементів.

Батарея: Забезпечує електропостачання для всіх компонентів системи. Ефективне управління енергією є ключовим фактором для довговічності та автономності робота.

Програмна складова Програмна частина містить в своєму складі комп'ютерні програми, алгоритми та штучний інтелект, які забезпечують функціонування робота. Це можуть бути алгоритми навігації, системи відслідковування об'єктів, системи вирішення завдань розпізнавання образів та системи прийняття рішень на основі зібраної інформації а також штучний інтелект. Програмну складову керованого робота можна розділити на такі компоненти як:

Система візуального сприйняття: Обробка відеодані з камер та інших

візуальних сенсорів для визначення об'єктів, перешкод та маршрутів.

Алгоритм мапування та локалізації: Використовуються для створення карт середовища та визначення місцезнаходження робота на карті.

Планування маршруту: Відповідає за вибір оптимального маршруту для досягнення цілі, враховуючи обмеження та перешкоди.

Система управління рухом: Забезпечує керування рухом робота з урахуванням визначеного маршруту та обмежень.

Штучний інтелект: Використовується для вдосконалення процесу вирішення завдань та адаптації до змін у навколишньому середовищі.

Інтеграція та взаємодія Архітектура програмно-апаратного комплексу керованого робота передбачає ефективну інтеграцію між апаратною та програмною складовими. Це включає в себе розробку інтерфейсів для передачі даних між різними компонентами, синхронізацію роботи апаратних та програмних елементів, а також розробку алгоритмів реагування на зміни в середовищі рис.1.2 [4,5].

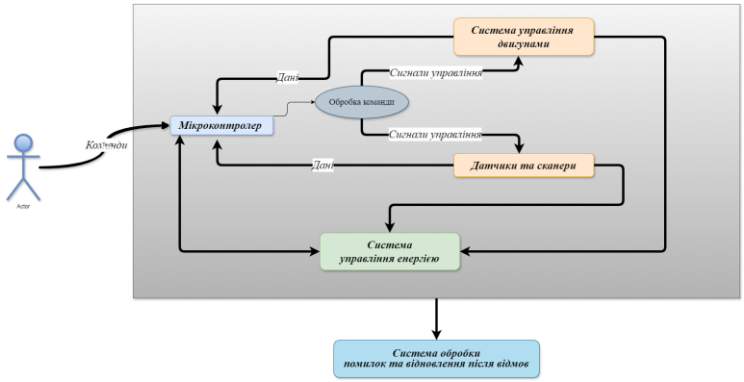


Рисунок 1.2 – Принципи взаємодії компонент архітектури апаратно – програмно комплексу автономного мобільного робота

Взаємодія компонент:

– *Команди та відповіді:* Мікроконтролер отримує команди від оператора або внутрішніх алгоритмів, обробляє їх та відправляє відповідні сигнали на системи управління двигунами та датчики.

– *Обробка даних:* Датчики надсилають дані на мікроконтролер, який обробляє їх та використовує їх для прийняття рішень щодо подальших дій.

– *Енергоефективність:* Система управління енергією моніторить використання батареї та оптимізує споживання енергії, щоб забезпечити максимальну автономність роботи.

– *Стійкість до помилок:* Програмна робота включає механізми для обробки помилок та відновлення після відмов, що забезпечує стійкість системи до непередбачуваних подій.

1.2. Принципи функціонування програмної частини робота

Автономні мобільні роботи (AMR) здобувають все більшу популярність в різних галузях, від промисловості до робототехніки та медицини. Їхній успіх значною мірою залежить від програмної частини, яка керує поведінкою роботів, взаємодією з оточуючим середовищем та приймає рішення в реальному часі,

До ключових компонент програмної частини автономних мобільних роботів та їх успішного функціонування відносяться наступні компоненти

Система сприйняття Першим і важливим етапом у функціонуванні автономного мобільного робота є його система сприйняття оточуючого середовища. Інформація, зібрана цими системами, обробляється програмним забезпеченням для подальшого аналізу і вирішення завдань, створення карт середовища та визначення місцезнаходження робота.

Обробка та аналіз даних Отримані від датчиків та сенсорів дані проходять через процес обробки та аналізу, за для забезпечення робота необхідною інформацією для подальшого функціонування. Ця система включає в себе фільтрацію шуму, видалення артефактів, визначення розташування об'єктів, виявлення перешкод та інші обчислювальні операції, які використовується для створення роботом моделі середовища та подальшого прийняття рішень.

Алгоритми мапування та локалізації. Одним з ключових етапів функціонування автономних мобільних роботів є створення карти середовища та визначення місцезнаходження робота на цій карті. Для цього використовуються алгоритми мапування, які на основі отриманої інформації

про середовище створюють його модель. Паралельно з цим використовуються алгоритми локалізації, які дозволяють роботу визначати своє поточне місцезнаходження на створеній карті [7].

*Планування траєкторії та маршруту***.** Після отримання інформації про середовище та визначення поточного місцезнаходження програмна частина робота визначає оптимальну траєкторію або маршрут для досягнення мети. Це включає в себе вибір оптимального шляху до цілі, врахування перешкод, обмежень та умов, таких як вимоги щодо часу, енергії, безпеки тощо [8].

Управління рухом та навігація. Після планування оптимального маршруту програмна частина робота переходить до етапу управління його рухом та навігацією. Це включає вибір оптимального шляху до цілі, врахування перешкод, обмежень та умов, таких як вимоги щодо часу, енергії, безпеки тощо. Планування маршруту може бути виконано за допомогою різних алгоритмів, таких як алгоритми пошуку найкоротшого шляху A*, RRT (Rapidly-exploring Random Tree), генетичні алгоритми, а також евристичні алгоритми [9].

Адаптація та взаємодія з оточуючим середовищем Одним з важливих аспектів програмної частини є здатність її адаптації до змін в оточуючому середовищі та умов роботи. Це включає в себе виявлення, ідентифікацію, взаємодію з новими об'єктами та системами, реакцію на зміни, освітленості, погодніх умов, чи інші фактори.

Програмна частина роботи включає в себе кілька ключових принципів, які становлять основу для ефективної та якісної розробки програмного забезпечення до них відносяться:

Модульність: Цей принцип передбачає створення програмного коду у вигляді взаємозалежних модулів або компонентів. Кожен модуль виконує певну функцію та може бути розроблений, тестований та підтримуваний окремо. Це полегшує розробку, вдосконалення та масштабування програми.

Читабельність: Програмний код повинен бути зрозумілим для інших розробників, які можуть працювати з ним у майбутньому. Це включає в себе

добре підібрані назви змінних, ясні коментарі та належну документацію.

Ефективність: Робочий код повинен бути оптимізованим щодо використання ресурсів системи, таких як пам'ять та обчислювальна потужність. Це допомагає забезпечити швидку та ефективну роботу програми.

Надійність: Програмна частина повинна бути надійною та відмовостійкою. Це означає, що вона повинна вміти ефективно обробляти непередбачувані ситуації та помилки, не зупиняючи свою роботу.

Ці принципи становлять фундаментальну основу для розробки високоякісної програмної частини роботи. Їх використання допомагає забезпечити ефективність, якість та надійність програмного забезпечення.

Розвиток нових технологій та алгоритмів у сфері розробки програмної частини АМР продовжить підвищувати їх автономність та функціональність адже це є складним і багатогранним комплексом, який вимагає глибоких знань в області комп'ютерних наук, механіки та системного аналізу. Розробка ефективної програми вимагає не тільки технічних навичок, але й творчого мислення, здатності до аналізу складних систем та постійного вдосконалення.

1.3.Інструменти та технології, що використовуються у розробці програмного забезпечення мобільних роботів.

Розробка програмного забезпечення для автономних мобільних роботів є складним і багатоступеневим процесом, який вимагає використання різноманітних інструментів та технологій для забезпечення ефективності, надійності та адаптивності систем. Основні інструменти та технології, що використовуються у сучасній практиці розробки програмного забезпечення для автономних мобільних роботів [10].

Системи управління версіями **(VCS)** є ключовими для координації роботи розробників, особливо у великих командах. Вони дозволяють відстежувати зміни в коді та спільно працювати над проектом. Основні VCS включають:

Git: Найпопулярніша система управління версіями, яка забезпечує можливість паралельної розробки та інтеграції змін. Інструменти, такі як GitHub і GitLab,

дозволяють зручно керувати проектами, організовувати код-рев'ю та безперервну інтеграцію.

Subversion (SVN): Централізовано керована система управління версіями, яка також використовується в багатьох проектах для забезпечення контролю версій.

Інтегровані середовища розробки (IDE) забезпечують інструменти для написання, тестування та налагодження коду. Деякі з найбільш популярних IDE включають:

Visual Studio Code: Легка і розширювана IDE, яка підтримує багато мов програмування та інтеграцію з різними інструментами для розробки роботів.

PyCharm: Потужне IDE для Python, яке часто використовується в розробці програмного забезпечення для роботів завдяки своїй інтеграції з бібліотеками для машинного навчання та обробки даних.

Eclipse: Відкрите IDE для Java, яке підтримує також інші мови програмування через додаткові плагіни, що робить його гнучким для різних проектів.

Системи управління проектами використовується для організації та керування процесом розробки, системи управління проектами допомагають координувати завдання, терміни та ресурси. Основні інструменти включають:

Jira: Потужний інструмент для управління проектами та відстежування завдань, широко використовуваний у розробці програмного забезпечення з використанням Agile-методологій.

Trello: Візуальний інструмент для організації завдань, що дозволяє створювати дошки, списки та картки для управління проектами.

Технології безперервної інтеграції та доставки (CI/CD) автоматизують процеси тестування та розгортання програмного забезпечення, забезпечуючи швидке та надійне оновленя. Основні інструменти включають:

Jenkins: Популярний сервер автоматизації для CI/CD, що підтримує інтеграцію з багатьма іншими інструментами та технологіями.

GitLab CI/CD: Вбудовані в GitLab інструменти для безперервної інтеграції та доставки, що дозволяють налаштовувати пайплайни для автоматизації

процесів.

Робототехнічні фреймворки забезпечують готові рішення для створення програмного забезпечення роботів. Основні фреймворки включають [11]:

Robot Operating System (ROS): Найпопулярніша платформа для розробки програмного забезпечення для роботів, яка забезпечує інструменти для моделювання, контролю, обробки даних та зв'язку між компонентами робота.

Microsoft Robotics Developer Studio: Набір інструментів для розробки, симуляції та тестування робототехнічних застосунків на базі .NET.

Системи симуляції дозволяють тестувати та відлагоджувати програмне забезпечення роботів у віртуальному середовищі, що знижує витрати на апаратні ресурси та прискорює процес розробки. Основні інструменти включають:

Gazebo: Відкрите середовище для симуляції робототехнічних систем, яке інтегрується з ROS та забезпечує реалістичну фізику для тестування алгоритмів управління.

Webots: Платформа для 3D-симуляції мобільних роботів, що підтримує різні типи роботів та сенсорів.

Інструменти для обробки даних та машинного навчання відіграють важливу роль у розробці автономних роботів, забезпечуючи можливість аналізу великих обсягів даних та прийняття рішень. Основні інструменти включають:

TensorFlow: Популярна бібліотека для машинного навчання, яка забезпечує інструменти для створення та тренування нейронних мереж.

OpenCV: Бібліотека для комп'ютерного зору, яка надає інструменти для обробки зображень та відео.

Хмарні платформи забезпечують інфраструктуру для зберігання даних, обчислень та аналітики, що дозволяє розробникам розгортати та масштабувати свої застосунки без необхідності управління фізичними серверами. Основні хмарні платформи включають:

Amazon Web Services (AWS): Найбільша хмарна платформа, що пропонує

широкий спектр сервісів для обчислень, зберігання даних, машинного навчання та багато іншого.

Microsoft Azure: Хмарна платформа від Microsoft, що надає послуги для розробки, тестування, розгортання та управління застосунками.

Google Cloud Platform (GCP): Хмарна платформа від Google, що пропонує сервіси для обчислень, зберігання, аналізу даних та машинного навчання.

Системи контролю якості та тестування є критично важливим етапом розробки. Для цього використовуються різноманітні інструменти та технології:

Selenium: Інструмент для автоматизованого тестування веб-застосунків, що дозволяє записувати та виконувати тести в різних браузерах.

JUnit: Фреймворк для модульного тестування на мові Java, який дозволяє розробникам писати та виконувати тести для окремих компонентів.

SonarQube: Платформа для автоматизованого аналізу якості коду, яка допомагає виявляти помилки, вразливості та інші проблеми.

Інструменти для спільної роботи та комунікації Ефективна комунікація та співпраця між членами команди є важливим аспектом успішної розробки програмного забезпечення. Основні інструменти включають:

Slack: Платформа для обміну повідомленнями та співпраці, яка дозволяє створювати канали для обговорення проектів та інтеграцію з іншими інструментами.

Microsoft Teams: Інструмент для співпраці, що включає чати, відеоконференції, обмін файлами та інтеграцію з іншими сервісами Microsoft.

Confluence: Вікі-платформа для спільної роботи та документування, що дозволяє створювати, редагувати та зберігати інформацію про проект.

Розробка програмного забезпечення для автономних мобільних роботів є складним та багатогранним процесом, який потребує використання широкого спектру інструментів та технологій для забезпечення ефективності, надійності та адаптивності систем. Від систем управління версіями та інтегрованих середовищ розробки до робото технічних фреймворків та хмарних платформ.

– всі ці елементи грають важливу роль у створенні високоякісних та надійних AMR комплексів

ПРОЕКТУВАННЯ ПРОГРАМНОЇ СКЛАДОВОЇ

2.1 Вимоги до функціональності та надійності програм

Програмне забезпечення являється однією з ключових компонент не тільки автономних мобільних роботів, а всіх автоматизованих систем та комплексів у цілому. Перед ПЗ ставляться вимоги що до забезпечення функціонування АМР, оскільки воно відповідає за управління роботом, його рух, навігацію, сприйняття інформації з сенсорів та взаємодію з оточуючим середовищем а також надійність.

Вимоги до функціональності

Навігація та локалізація одна з найбільш критичних функцій для автономних мобільних роботів. ПЗ повинно забезпечувати точне визначення місцезнаходження робота в просторі, а також планування та виконання маршрутів.

SLAM (Simultaneous Localization and Mapping): Алгоритми одночасної локалізації та картування які дозволяють роботу створювати карти невідомих середовищ та визначати своє місцезнаходження на них.

GPS-навігація: Для роботи в зовнішніх середовищах AMR повинні мати можливість використовувати глобальні навігаційні супутникові системи для точного позиціонування.

Виявлення та уникнення перешкод Безпека та ефективність руху робота залежать від здатності виявляти та уникати перешкоди.

Сенсори та датчики: Роботи використовують різні сенсори та датчики, такі як лідари, ультразвукові датчики та камери, для виявлення перешкод.

Алгоритми уникнення перешкод: Програмне забезпечення повинно мати алгоритми, які дозволяють роботу змінювати маршрут у реальному часі для уникнення зіткнень.

Збирання та обробка даних Автономні роботи часто виконують завдання, пов'язані зі збором даних про навколишнє середовище.

Обробка зображень: Використання технологій комп'ютерного зору для аналізу візуальних даних.

Аналіз даних у реальному часі: Програмне забезпечення повинно мати можливість обробляти та аналізувати дані в режимі реального часу для прийняття оперативних рішень.

Взаємодія з користувачем Для ефективної роботи роботи повинні мати інтерфейси для взаємодії з користувачем, які забезпечують легкість керування та отримання зворотного зв'язку.

Інтерфейси користувача (UI): Зручні та інтуїтивно зрозумілі інтерфейси для управління роботом та моніторингу його стану.

Віддалене управління: Можливість дистанційного керування та контролю робота через мережу.

Вимоги до надійності

Відмовостійкість. Надійність програмного забезпечення робота повинна забезпечувати безперебійну роботу навіть у разі виникнення помилок.

Резервні системи: Використання резервних компонентів та алгоритмів для забезпечення безперервної роботи у випадку збою основних систем.

Моніторинг стану: Постійний моніторинг стану системи та своєчасне виявлення потенційних проблем.

Стійкість до несприятливих умов Програмне забезпечення повинно забезпечувати стійкість роботи робота в різних умовах, включаючи екстремальні температури, вологість та механічні впливи.

Адаптивні алгоритми: Алгоритми, які можуть адаптуватися до змін у середовищі, забезпечуючи стабільну роботу.

Захист від збоїв: Використання технологій захисту від збоїв, таких як дублювання даних та перезапуск систем.

Безпека Ключовий аспектом для автономних мобільних роботів, особливо при роботі в громадських місцях або взаємодії з людьми.

Шифрування даних: Захист даних, що передаються між роботом і контролюючими системами, від несанкціонованого доступу.

Аутентифікація користувачів: Використання надійних методів аутентифікації для доступу до систем управління роботом.

Тестування та валідація Програмне забезпечення повинно проходити ретельне тестування та валідацію для забезпечення його відповідності вимогам функціональності та надійності.

Модульне тестування: Тестування окремих компонентів програмного забезпечення для виявлення та виправлення помилок.

Інтеграційне тестування: Перевірка взаємодії між різними компонентами системи для забезпечення їх сумісності та коректної роботи.

Тестування в реальних умовах: Проведення тестів у реальних умовах експлуатації для виявлення потенційних проблем, які можуть виникнути у процесі роботи.

Програмне забезпечення для автономних мобільних роботів повинно відповідати високим вимогам до функціональності та надійності. Це включає точну навігацію та локалізацію, виявлення та уникнення перешкод, збирання та обробку даних, а також забезпечення зручних інтерфейсів для взаємодії з користувачем.

Надійність програмного забезпечення досягається через відмовостійкість, стійкість до несприятливих умов, забезпечення безпеки та ретельне тестування. Всі ці аспекти є критично важливими для успішного функціонування автономних мобільних роботів у різних сферах застосування.

2.2. Вибір технологій та мов програмування

Програмне забезпечення для гусеничних роботів повинно забезпечувати високу продуктивність, особливо в режимі реального часу. Це важливо для виконання таких завдань, як обробка сенсорних даних, навігація та управління рухом.

Роботи часто працюють у критичних умовах, де будь-який збій може призвести до серйозних наслідків. Тому вибрані технології та мови програмування повинні підтримувати механізми для підвищення відмовостійкості та надійності. Також роботи повинні бути легко адаптовані до нових завдань та умов, що вимагає використання гнучких та

масштабованих технологій. Модульність програмного забезпечення є важливим аспектом для досягнення цієї вимоги.

Доступність інструментів, бібліотек, фреймворків та підтримка спільноти розробників є важливими факторами, які впливають на вибір технологій та мов програмування.

Мови програмування

C/C++: є високо продуктивними мовами низького рівня, які дозволяють розробникам ефективно управляти пам'яттю та ресурсами системи, що є критичним для реального часу. Ці мови мають велику кількість бібліотек, які забезпечують широкий спектр функціональності, від роботи з сенсорами до алгоритмів управління, через їх широке використання вони є стандартом у робототехніці, що забезпечує доступ до великої кількості ресурсів та прикладів. Як недоліки є складність цих мов адже вони мають високу складність, що може збільшити час розробки та відлагодження, а також менша безпека адже відсутність автоматичного управління пам'яттю може призвести до помилок, таких як витоки пам'яті.

Python є простим у використанні він являється мовою високого рівня з простим та читабельним синтаксисом, що пришвидшує розробку та відлагодження, для нього існує велика кількість бібліотек для робототехніки, зокрема, популярний фреймворк Robot Operating System (ROS). *Python* є гнучким: що дозволяє швидко прототипувати та тестувати нові ідеї. Як недолік цієї мови є те що вона менша продуктивна адже Python є інтерпретованою мовою, що може призвести до меншої продуктивності у порівнянні з C/C++.Також у нього обмежена підтримка реального часу: Для критичних завдань реального часу Python може бути менш підходящим через затримки в інтерпретації.

Java Портативність цієї мови дозволяє писати код один раз і запускати його на будь-якій платформі, що підтримує JVM (Java Virtual Machine). Java має широкий спектр інструментів для розробки, тестування та розгортання а також володіє автоматичним управлінням пам'яттю, що зменшує ризик її витоків, як

недолік Java може бути повільнішою, ніж C/C++, особливо для завдань, що вимагають високої продуктивності, налаштування JVM може бути складнішим порівняно з іншими мовами.

Технології та фреймворки

Фреймворк Framework, основа, платформа, структура, інфраструктура [12] Фреймворк є готовим комплексом програмних рішень, який включає у себе дизайн, логіку та базову функціональність системи або підсистеми, а також до його складу можуть входити допоміжні програми, бібліотеки коду, скрипти, які полегшують створення та поєднання різних компонентів ПЗ чи розробку готового програмного продукту.

Однією з переваг, використання таких технологій є те що вона являється каркасною, тобто такі програми мають стандартну структуру. У Каркасних застосунках частина програми може мати основу базових класів фреймворку, в основі яких покладені методи об'єктно-орієнтованого програмування, через структуру внутрішньої реалізації коду написання програмного продукту стало заздалегідь відомо і це дає змогу набагато простіше та швидше створювати засоби для автоматичної побудови графічних інтерфейсів.

1. *Robot Operating System (ROS)* Модульність ROS дозволяє розробляти програмне забезпечення у вигляді окремих модулів, які можуть легко інтегруватися та масштабуватися, ця технологія володіє широкою підтримкою спільнот, та наявністю безлічі бібліотек та пакетів, в ній можлива інтеграція з різними мовами: Підтримка як для C++, так і для Python, що дозволяє використовувати їх сильні сторони. Недоліком є Складність ROS полягає у тому що вона має круту криву навчання, особливо для новачків, залежить від ОС адже основна підтримка ROS здійснюється для Linux, що може обмежити використання на інших платформах [11].

OpenCV потужній інструмент для обробки зображень та комп'ютерного зору, вона підтримує різні мови програмування та платформи, що дозволяє використовувати її на багатьох платформах, також вона має високу продуктивність, через оптимізовані алгоритми для реального часу.

Як недоліки це складність яка вимагає знань у сфері обробки зображень та алгоритмів комп'ютерного зору, може потребувати налаштування залежностей, що може бути складним.

TensorFlow та PyTorch Підтримує різноманітні моделі машинного навчання, що дозволяє реалізовувати складні алгоритми, такі як розпізнавання об’єктів, класифікація, сегментація тощо, підтримується великою спільнотоюи розробників, що забезпечує постійне оновлення та поліпшення інструментів, легко інтегрується з іншими фреймворками та бібліотеками для розробки робототехнічних систем. Недоліками є ресурсоємність, адже використовується модель машинного навчання яка може вимагати значних обчислювальних ресурсів, що іноді є проблематичним для вбудованих систем також налаштування та навчання таких моделей можуть бути складними і потребувати значних знань у сфері машинного навчання.

Інструменти для розробки

Інтегровані середовища розробки (IDE) (integrated development environment - IDE), комплекс програмних засобів які використовується для розробки програмного забезпечення (ПЗ) та містять в собі такі елементи як текстовий редактор, транслятор (компілятор та/або інтерпретатор) та засоби автоматизації.

Інтегровані середовища розробки створені для, максимізації продуктивної діяльності програміста за рахунок зв’язку компонент з простими користувацькими інтерфейсами, із за чого при розробці програмного продукту у розробника є змога робити якомога менше дій при перемиканні режимів, що є неможливим у дискретних програмах.

Зазвичай інтегровані середовища у своїй наявності містять єдину програму, в якій можливо проводити всю розробку ПЗ у якій присутні різноманітні функцій для її створення, зміни, компілювання, розгортання та налагодження. Інтегроване середовище також поєднує у собі різні утиліти, які містяться в одному модулі, цей модуль дозволяє абстрагуватися від виконання допоміжних завдань, тим самим даючи змогу розробнику працювати над

вирішенням алгоритмічного завдання, та не витрачати час на типові технічні дії до прикладу, виклик компілятора через що, підвищується продуктивність, та швидкість розробки ПЗ. Така інтеграція завдань при розробці також підвищує продуктивність за рахунок можливості введення додаткових допоміжних функцій на проміжних етапах роботи. Також інтегроване середовище дає змогу аналізувати код і тим самим забезпечує миттєвий зворотний зв'язок та повідомити про наявність помилки.

Більшість сучасних інтегрованих систем володіють графічним інтерфейсом, у яки для спрощення розробки ПЗ також наявні засоби для інтеграції з системами керування версіями. На даний момент часу велика кількість середовищ розробки в своєму складі містять браузер класів, інспектор об'єктів та діаграми ієрархії класів що можна використовувати при об'єктно-орієнтованому програмуванні ПЗ.

Arduino IDE – інтегроване середовище розробки програмного забезпечення для ОС Windows, macOS і Linux, яка підтримує такі мови програмування як C і C ++ [4], основним призначенням якої є створення та завантаження програмного продукту на мікроконтролери чи інші плати управління [5] .

Visual Studio Code: Легке та потужне середовище, яке підтримує багато мов програмування через розширення. Підходить для Python, C/C++ та інших мов.

PyCharm: Спеціалізоване середовище для Python, яке має інструменти для зручного розробки, відлагодження та тестування.

Eclipse: Потужне середовище для розробки на Java з великим набором плагінів для розширення функціональності.

Системи контролю версій

Git: Найпоширеніша система контролю версій, яка дозволяє ефективно керувати змінами у вихідному коді, співпрацювати в команді та відстежувати історію змін.

GitHub ma GitLab: Онлайн платформи, що надають інструменти для спільної розробки, управління проектами та CI/CD.

Системи безперервної інтеграції та доставки (CI/CD)

Jenkins: Популярний інструмент для автоматизації збірки, тестування та розгортання програмного забезпечення.

Travis CI: Хмарна система CI, яка інтегрується з GitHub та забезпечує автоматизацію тестування та збірки проектів.

Вибір технологій та мов програмування для гусеничних роботів є важливим етапом у розробці робототехнічних систем. Враховуючи вимоги до продуктивності, надійності, гнучкості та підтримки, найбільш підходящими є мови C/C++ для завдань з високими вимогами до продуктивності та Python для швидкої розробки та прототипування. Використання фреймворків, таких як ROS, та інструментів для обробки зображень (OpenCV) та машинного навчання (TensorFlow, PyTorch) дозволяє створювати ефективні та надійні системи для різноманітних застосувань гусеничних роботів. Сучасні інструменти для розробки, контролю версій та автоматизації процесів розробки забезпечують ефективність та якість створюваного програмного забезпечення.

3. РЕАЛІЗАЦІЯ ПРОГРАМНОЇ ЧАСТИНИ

3.1. Розробка основних модулів та алгоритмів керування.

Arduino — це електронна платформа з відкритим вихідним кодом, заснована на простому у використанні апаратному та програмному забезпеченні. Він призначений для тих, хто створює інтерактивні проекти.

Arduino IDE – Інтегроване середовище розробки яке працює з різними ОС, призначенням якого є створення та завантаження програмного забезпечення на мікроконтролери Arduino та сумісні з ним плати [14]., такий програмний продукт вказує мікроконтролеру, як саме він повинен працювати та яку дію йому потрібно виконувати.

Вихідний код Arduino IDE містить загальнодоступну ліцензією GNU та підтримується такими мовами програмування, як C та C++ у яких виконуються спеціальні правила структурування коду [15], також Arduino IDE містить бібліотеку з проекту Wiring, яка надає безліч загальних процедур введення та виведення.

Написаний розробником код виконується на основі двох базових функції, які виконують запуск ескізу і основного циклу програми, вони компілюються і пов'язуються з заглушкою програми *main ()* в циклічну програму, що виконується Програма avrdude використовується для перетворення коду, який виконується, в текстовому файл з шістнадцятковим кодуванням, після чого закодований текстовий файл завантажується в плату Arduino програмою-завантажувачем у вбудованому програмному забезпеченні плати

Написання програмного продукту для виконання руху гусеничного керованого робота розробимо алгоритм його роботи у залежності від використовуваних датчиків на основі яких здійснюється керування.

Рух робота, який рухається за світлом на основі даних з фоторезистора

Першим етапом моделювання буде логіка руху робота, який рухається за світлом, керування моторних приводів здійснюється на основі даних, що надходять з фотоелемента.

У даному випадку використовується два модулі фотоелементів для виявлення

інтенсивності світла зліва та справа від робота, здійснюється зчитування відповідних аналогових значень, на основі яких здійснюється керування обертанням двох двигунів, щоб контролювати рухи робота.

Логіка руху робота, що слідує за світлом.

Лівий модуль фотоелемента → Світло _ліве;

Правий модуль фотоелемента → Світло _праве;

Умова	***Дія***
Якщо: Світло _ліве > 650 Світло _праве > 650	*Статус:* **Рух → вперед**
Якщо: Світло _ліве > 650 **Світло _праве ≤650**	*Статус:* **Рух → Поворот ліворуч;** **Обертання моторів → вліво;**
Якщо: **Світло _ліве ≤650** Світло _праве > 650	*Статус:* **Рух → Поворот праворуч;** **Обертання → вправо;**
Якщо: **Світло _ліве ≤650** **Світло _праве ≤650**	*Статус:* **Рух → Зупинка; СТІЙ**

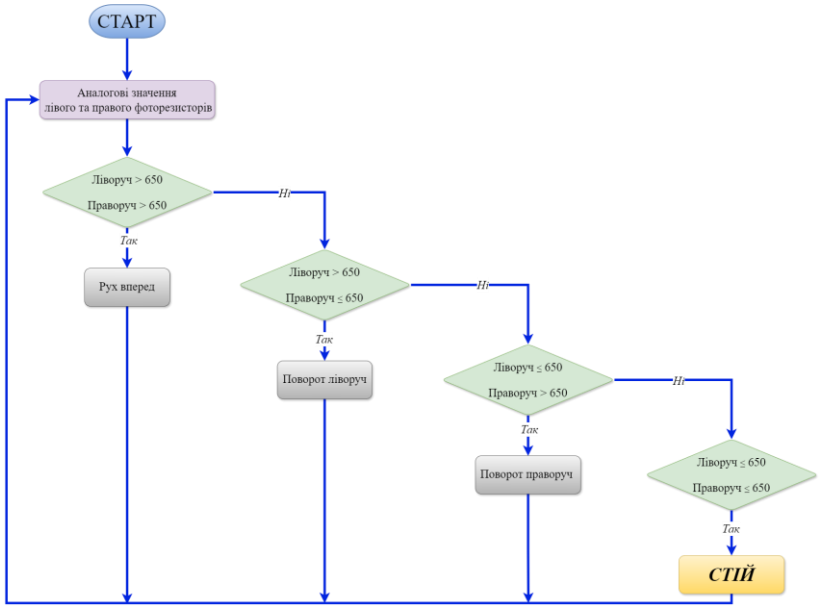


Рисунок. 3.1. Блок-схема логіки руху робота на основі дааних з фотоелемента

Рух робота, що рухається на основі даних ультразвукового датчика

Наступний крок моделювання руху робота на основі УЗ датчика, який використовується для визначити відстань між роботом та перешкодою попереду, керування обертанням двигунів здійснюється на основі даних отриманих з датчика. Для цього як початкові значення задаємо

Відстань, виміряна ультразвуковим датчиком між роботом і перешкодою попереду (кут сервоприводу = 90°, *а (см)*)

Відстань, між роботом і перешкодою ліворуч (кут сервоприводу = 160° a_1*(см)*)

Відстань між роботом і перешкодою праворуч (кут сервоприводу = 20° a_2*(см)*)

Умова	***Дія***
Якщо: $a < 20$	*Статус:* **Рух** → **Зупинка** 500 *мс*; **Кут сервоприводу** → 160°, **Інформація** → a_1, **затримка** → 100 *мс*; **Кут сервоприводу** → 0°, **Інформація** → a_2, **затримка** → 100 *мс*;
Якщо: $a_1 < 50$ $a_2 < 50$	*Статус:* **Порівняння** → a_1:a_2
Якщо: $a_1 > a_2$	*Статус:* **Рух** → **Поворот ліворуч через** 700 *мс* **Обертання моторів** → **у ліво**, **Кут сервоприводу** → 90°, **ШІМ** → 255; *Статус:* **Рух** → **вперед через** 0,5 **с**, **ШІМ** → 200.
Якщо: $a_1 < a_2$	*Статус:* **Рух** → **Поворот праворуч через** 700 *мс*, **Обертання моторів** → **у право**, **Кут сервоприводу** → 90°, **ШІМ** → 255; *Статус:* **Рух** → **вперед через** 0,5 **с**, **ШІМ** → 200.
Якщо: $a < 20$ $a_1 \geq 50$	*Статус:* **Рух** → **Поворот ліворуч через** 500 *мс*, **Обертання моторів** → **у ліво**, **Кут сервоприводу** → 90°, **ШІМ** → 255; *Статус:* **Рух** → **вперед через** 0,5 **с**, **ШІМ** → 200.

Якщо: $a < 20$ $a_2 \geq 50$	*Статус:* **Рух → Поворот праворуч через 500** *мс*, **Обертання моторів → у право,** **Кут сервоприводу → 90°, ШІМ → 255;** *Статус:* **Рух → вперед через 0,5 с, ШІМ → 200.**
Якщо: $a \geq 20$	*Статус:* **Рух → Вперед, ШІМ → 100.**

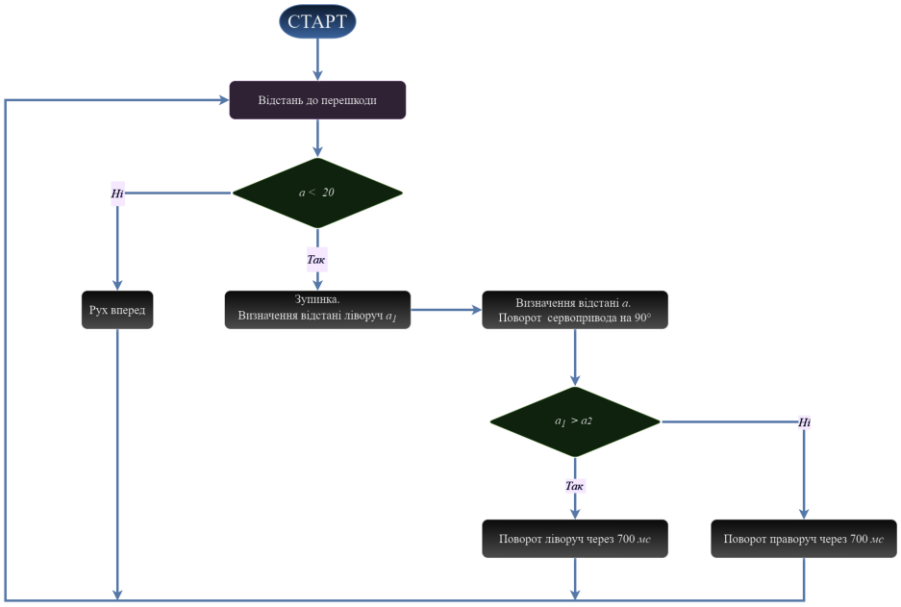


Рисунок. 3.2. Блок-схема логіки руху робота на основі даних з УЗ датчика.

Рух робота, на основі ІЧ датчика відстеження

Далі проведемо моделювання руху робота, який стежить за лінією.

Датчик відстеження лінії, застосовується для визначення, чорної лінії навколо робота, після чого контроль обертання двигунів здійснюється відповідно до результатів виявлення.

Датчик відстеження лінії посередині:

Виявлено чорну лінію → високий рівень;

Виявлено білу лінію → низький рівень;

Датчик відстеження лінії зліва:

Виявлено чорну лінію → високий рівень;

Виявлено білу лінію → низький рівень;

Датчик відстеження лінії праворуч:

Виявлено чорну лінію → високий рівень;

Виявлено білу лінію → низький рівень;

Логіка руху робота з відстеженням лінії має вигляд:

Умова	***Дія***
Датчиком посередині виявлено чорну лінію	
Якщо: Лівий датчик виявляє чорну лінію, а правий – білу	*Статус:* `Рух → Поворот ліворуч; Обертання моторів → вліво;`
Якщо: Лівий датчик виявляє білу лінію, а правий – чорну	*Статус:* `Рух → Поворот праворуч; Обертання → вправо;`
Якщо: Обидва датчики виявляють білу або чорну лінії	*Статус:* `Рух → вперед`
Датчиком посередині виявлено білу лінію	
Якщо: Лівий датчик виявляє чорну лінію, а правий – білу	*Статус:* `Рух → Поворот ліворуч; Обертання моторів → вліво;`
Якщо: Лівий датчик виявляє білу лінію, а правий – чорну	*Статус:* `Рух → Поворот праворуч; Обертання → вправо;`
Якщо: Обидва датчики виявляють білу або чорну лінії	*Статус:* `Рух → Зупинка; СТІЙ`

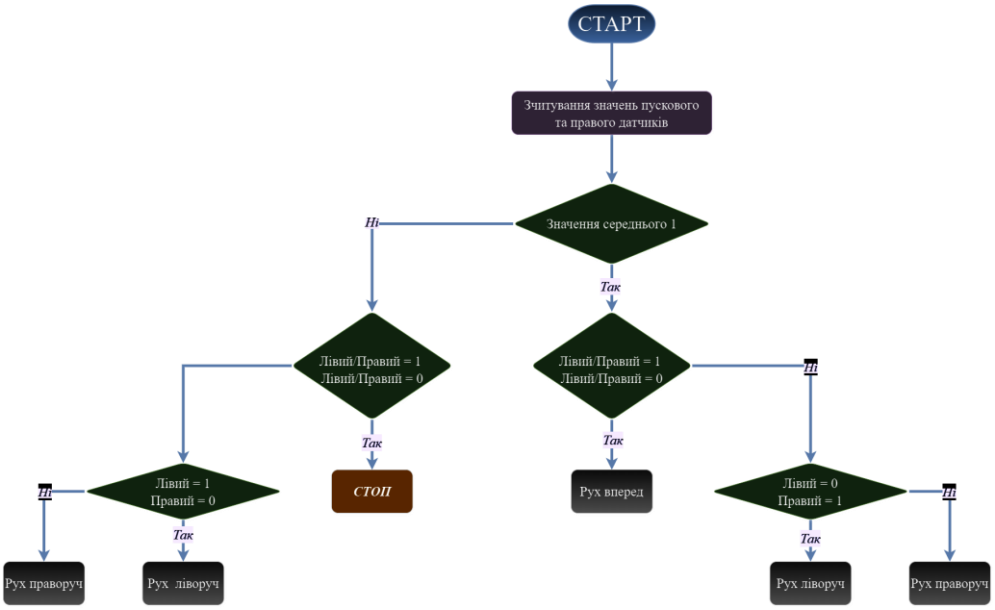


Рисунок. 3.3. Блок-схема логіки руху робота на основі даних з ІЧ датчика.

3.2. Тестування та налагодження програмного забезпечення

На основі розроблених модулів та алгоритмів керування гусеничним мобільним роботом, здійснюється створення та імплентація ПЗ для керування

роботом на основі використовуваних датчиків.

Програма керування рухом робота, на основі даних отриманих з фоторезистора:

Цей код призначений для керування мобільним гусеничним роботом, який використовує два фоторезистори для визначення напрямку руху залежно від інтенсивності світла. Робот має два двигуни, кожен з яких керується через штифти контролера.

1. Визначення макросів та глобальних змінних

```
#define light_L_Pin A1 // Значення лівого фоторезистора
#define light_R_Pin A2 // Значення правого фоторезистора
#define ML_Ctrl 13 // Значення штифта керування напрямком лівого
двигуна
#define ML_PWM 11 // Значення штифта керування ШІМ лівого
двигуна
#define MR_Ctrl 12 // Значення штифта керування напрямком
правого двигуна
#define MR_PWM 3 // Значення штифта керування ШІМ правого
двигуна

int left_light;
int right_light;
```

`light_L_Pin` та `light_R_Pin`: Визначають аналогові входи для лівого та правого фоторезисторів.

`ML_Ctrl, ML_PWM, MR_Ctrl, MR_PWM`: Визначають цифрові штифти для керування напрямком і швидкістю лівого та правого двигунів.

`left_light та right_light:` Змінні для збереження значень, зчитаних з фоторезисторів.

2. Налаштування штифтів в функції `setup`

```
void setup() {
  Serial.begin(9600);
  pinMode(light_L_Pin, INPUT);
  pinMode(light_R_Pin, INPUT);
  pinMode(ML_Ctrl, OUTPUT);
  pinMode(ML_PWM, OUTPUT);
  pinMode(MR_Ctrl, OUTPUT);
  pinMode(MR_PWM, OUTPUT);
```

```
}
```
`Serial.begin(9600)`: Ініціалізує серійне з'єднання з швидкістю 9600 бод для виведення інформації в серійний монітор.

`pinMode(...)`: Встановлює режими роботи штифтів: `INPUT` для фоторезисторів та `OUTPUT` для керування двигунами.

3. Головна функція циклу `loop`

```
void loop() {
  left_light = analogRead(light_L_Pin);
  right_light = analogRead(light_R_Pin);

  Serial.print("left_light_value = ");
  Serial.println(left_light);
  Serial.print("right_light_value = ");
  Serial.println(right_light);

  if (left_light > 650 && right_light > 650) {
    Car_front();
  } else if (left_light > 650 && right_light <= 650) {
    Car_left();
  } else if (left_light <= 650 && right_light > 650) {
    Car_right();
  } else {
    Car_Stop();
  }
}
```
`analogRead(...)`: Зчитує значення з фоторезисторів і зберігає їх у змінних `left_light` та `right_light`.

`Serial.print(...)` та `Serial.println(...)`: Виводять значення з фоторезисторів у серійний монітор для відлагодження.

`if` конструкції: Приймають рішення щодо напрямку руху робота на основі значень з фоторезисторів:

Якщо обидва значення більше 650: рух вперед.

Якщо лівий більше 650, а правий менше або дорівнює 650: поворот ліворуч.

Якщо лівий менше або дорівнює 650, а правий більше 650: поворот праворуч.

В іншому випадку: зупинка.

4. Функції керування рухом робота

```
void Car_front() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, 200);
```

```
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, 200);
}
void Car_left() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, 200);
  digitalWrite(ML_Ctrl, HIGH);
  analogWrite(ML_PWM, 200);
}
void Car_right() {
  digitalWrite(MR_Ctrl, HIGH);
  analogWrite(MR_PWM, 200);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, 200);
}
void Car_Stop() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, 0);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, 0);
}
```

`Car_front()`: Встановлює напрямок обох двигунів вперед (LOW) та задає швидкість за допомогою ШІМ сигналу 200.

`Car_left()`: Встановлює напрямок правого двигуна вперед (LOW), а лівого - назад (HIGH), задаючи швидкість 200.

`Car_right()`: Встановлює напрямок правого двигуна назад (HIGH), а лівого - вперед (LOW), задаючи швидкість 200.

`Car_Stop()`: Зупиняє обидва двигуни, встановлюючи ШІМ сигнал 0.

Код використовує просту логіку для керування рухом робота залежно від значень з фоторезисторів.

Функції `Car_front`, `Car_left`, `Car_right` та `Car_Stop` забезпечують базові дії для руху робота.

Для покращення функціональності можна додати додаткові умови або більш складні алгоритми обробки даних з фоторезисторів та інших сенсорів.

Програма керування рухом робота, на основі даних отриманих з УЗ датчика:

Даний код здійснює керування гусеничним роботом, який використовує

ультразвуковий датчик для визначення відстані до об'єктів.

На основі цієї інформації робот може рухатися вперед, назад, повертати або зупинятися. Код також включає функції для керування точковою матрицею та сервоприводом.

Глобальні змінні та макроси

`start01, front, back, left, right, STOP01, clear`: Массиви байтів для різних шаблонів, які можуть відображатися на точковій матриці.

`SCL_Pin, SDA_Pin`: Штифти для зв'язку по протоколу I2C.

`ML_Ctrl, ML_PWM, MR_Ctrl, MR_PWM`: Штифти для керування двигунами.

`Trig, Echo`: Штифти для ультразвукового датчика.

`servoPin`: Штифт для керування сервоприводом.

`distance, pulsewidth`: Змінні для зберігання відстані та ширини імпульсу для сервопривода.

Функція `setup` Ініціалізує серійне з'єднання, встановлює режими роботи штифтів, очищає та відображає початковий шаблон на точковій матриці, встановлює серво на 90 градусів.

Функція `loop`

Викликає `checkdistance` для отримання відстані до об'єкта.

В залежності від отриманої відстані викликає відповідну функцію для керування роботом:

`Car_front()` - рух вперед.

`Car_back()` - рух назад.

`Car_Stop()` - зупинка.

Функції керування двигунами

`Car_front()`: Встановлює обидва двигуни для руху вперед.

`Car_back()`: Встановлює обидва двигуни для руху назад.

`Car_left()`: Встановлює роботу двигунів для повороту наліво.

`Car_right()`: Встановлює роботу двигунів для повороту направо.

`Car_Stop()`: Зупиняє обидва двигуни.

Функції для роботи з точковою матрицею

`matrix_display()`: Відображає заданий шаблон на точковій матриці через I2C.

`IIC_start()`, `IIC_send()`, `IIC_end()`: Допоміжні функції для роботи з I2C.

Функція керування сервоприводом

`procedure(int myangle)`: Встановлює сервопривод на заданий кут.

Функція ультразвукового датчика

`checkdistance()`: Обчислює та повертає відстань до об'єкта за допомогою ультразвукового датчика.

Код забезпечує базову функціональність для керування гусеничним роботом, дозволяючи йому рухатися, зупинятися та відображати різні шаблони на точковій матриці.

Програма керування рухом робота, на основі даних отриманих з ІЧ датчика:

Код забезпечує керування рухом мобільного робота, який використовує три датчики лінії (лівий, середній і правий) для відстеження чорної лінії на поверхні. Робот здатен рухатися вперед, назад, повертати ліворуч, праворуч або зупинятися залежно від зчитаних даних з датчиків.

Глобальні змінні та макроси

`L_pin`, `M_pin`, `R_pin`: Штифти для підключення лівого, середнього і правого датчиків лінії.

`ML_Ctrl`, `ML_PWM`, `MR_Ctrl`, `MR_PWM`: Штифти для керування напрямком і швидкістю лівого і правого двигунів.

Функція `setup`

```
void setup() {
  Serial.begin(9600); // Встановлює швидкість передачі даних через серійний порт на 9600 бод.
  pinMode(L_pin, INPUT); // Встановлює всі штифти датчиків лінії
```

```
в режим входу.
  pinMode(M_pin, INPUT);
  pinMode(R_pin, INPUT);
  pinMode(ML_Ctrl, OUTPUT); // Встановлює всі штифти керування
двигунами в режим виходу.
  pinMode(ML_PWM, OUTPUT);
  pinMode(MR_Ctrl, OUTPUT);
  pinMode(MR_PWM, OUTPUT);
}
```

Функція `loop`

```
void loop() {
  L_val = digitalRead(L_pin); // Зчитує значення з лівого датчика.
  M_val = digitalRead(M_pin); // Зчитує значення з середнього
датчика.
  R_val = digitalRead(R_pin); // Зчитує значення з правого
датчика.

  if (M_val == 1) { // Якщо середній датчик виявляє чорну лінію.
    if (L_val == 1 && R_val == 0) { // Якщо чорна лінія виявлена
на лівому датчику, але не на правому, поворот ліворуч.
      Car_left();
    } else if (L_val == 0 && R_val == 1) { // Якщо чорна лінія
виявлена на правому датчику, але не на лівому, поворот праворуч.
      Car_right();
    } else { // Інакше, рух вперед.
      Car_front();
    }
  } else { // Якщо середній датчик не виявляє чорну лінію.
    if (L_val == 1 && R_val == 0) { // Якщо чорна лінія виявлена
на лівому датчику, але не на правому, поворот ліворуч.
      Car_left();
    } else if (L_val == 0 && R_val == 1) { // Якщо чорна лінія
виявлена на правому датчику, але не на лівому, поворот праворуч.
      Car_right();
    } else { // Інакше, зупинка.
      Car_Stop();
    }
  }
}
```

Функції керування рухом

`Car_front()`: Встановлює обидва двигуни на рух вперед зі швидкістю ШІМ 100.

`Car_back()`: Встановлює обидва двигуни на рух назад зі швидкістю ШІМ 150.

`Car_left()`: Встановлює правий двигун на рух вперед зі швидкістю ШІМ 100, а лівий двигун на рух назад зі швидкістю ШІМ 150.

`Car_right()`: Встановлює правий двигун на рух назад зі швидкістю ШІМ 150, а лівий двигун на рух вперед зі швидкістю ШІМ 100.

`Car_Stop()`: Зупиняє обидва двигуни, встановлюючи ШІМ 0.

Код забезпечує базову функціональність для роботів, що слідкують за лінією, дозволяючи їм автоматично коригувати свій напрямок руху, базуючись на зчитаних даних з трьох датчиків лінії.

ВИСНОВКИ

В результаті виконання роботи було проведено дослідження та реалізацію програмного забезпечення для керування гусеничним роботом з використанням різних сенсорних систем і алгоритмів керування. Головні результати роботи можна підсумувати таким чином:

Інтеграція сенсорів: Було успішно інтегровано сенсорні системи, включаючи фоторезистори для виявлення світла, ультразвуковий датчик для вимірювання відстані, а також датчики лінії для слідкування за лініями. Це дозволило роботу ефективно орієнтуватися в просторі та реагувати на зовнішні умови.

Алгоритми керування: Розроблено та реалізовано алгоритми для різних сценаріїв руху робота, включаючи рух вперед, назад, повороти ліворуч та праворуч, а також зупинку. Алгоритми базуються на аналізі даних з сенсорів та забезпечують адекватну реакцію робота на зміну умов середовища.

Програмна платформа: Використано мову програмування C++ та платформи Arduino для розробки та тестування програмного забезпечення. Це забезпечило високий рівень гнучкості та можливість подальшої модернізації системи.

Візуалізація та діагностика: Було використано серійний монітор для виведення діагностичних повідомлень, що дозволило здійснювати моніторинг стану робота та оперативно виявляти і виправляти помилки в роботі.

Енергоефективність: Програмне забезпечення оптимізовано для забезпечення тривалої роботи від батарей, що є критичним для автономних мобільних роботів.

Результати цієї роботи створюють основу для подальшого вдосконалення гусеничних роботів. Наступними кроками можуть бути:

Розширення функціональності за рахунок додавання додаткових сенсорів (наприклад, камер для комп'ютерного зору) для покращення сприйняття оточення.

Покращення алгоритмів через впровадження методів машинного навчання для вдосконалення ухвалення рішень та адаптації до складних умов середовища.

Розробка більш складних алгоритмів автономного руху та навігації для роботи в непередбачуваних середовищах.

Розробка модульної архітектури програмного забезпечення для полегшення оновлень та розширень функціональності робота.

Загалом, робота показала успішну реалізацію основних функцій керування гусеничним мобільним роботом, демонструючи потенціал для подальшого розвитку в цій галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Some Details of the Development of Mobile Robot Platforms [Електронний ресурс]. Режим доступу: https://link.springer.com/chapter/10.1007%2F978-1-4471-1273-0_15
2. Михайлов Є. Мобільні роботи: Навчальний посібник. – Одеса: ОНПУ, 2016. – 239 с.
3. Labrosse, J. J. MicroC/OS-II: The Real-Time Kernel. CMP Books. (2002).
4. Harrigan, R. (1994). Power and Drive Systems for Mobile Robots. Springer.
5. Zhang, Q., & Wong, J. Y. (1995). A comparison of the tractive performance of tracked and wheeled vehicles. Journal of Terramechanics, 32(3), 157-170.
6. Craig, J. J. (2005). Introduction to Robotics: Mechanics and Control. Pearson Prentice Hall.
7. Siciliano, B., & Khatib, O. (Eds.). (2016). Springer Handbook of Robotics. Springer.
8. Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). Introduction to Autonomous Mobile Robots. MIT Press.
9. Montemerlo, M., Thrun, S., Koller, D., & Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In Proceedings of the AAAI National Conference on Artificial Intelligence.
9. LaValle, S. M. (2006). Planning Algorithms. Cambridge University Press.
10. Borenstein, J., Everett, H. R., & Feng, L. (1996). Navigating Mobile Robots: Systems and Techniques. A. K. Peters, Ltd.
11. Quigley, M., Gerkey, B., & Smart, W. D. (2015). Programming Robots with ROS. O'Reilly Media.
11. Koubaa, A. (2016). Robot Operating System (ROS): The Complete Reference (Volume 1). Springer.
12 Г. Г. Півняк, Б. С. Бусигін, М. М. Дівізінюк та ін. — Тлумачний словник з інформатики Д., Нац. гірнич. ун-т, 2010. — 600 с.
13 Bequette, B. W. (2003). Process Control: Modeling, Design, and Simulation. Prentice Hall.

14. Steve Bush. Updated: Arduino announces FPGA board, ATmega4809 в Uno Wi-Fi mk2, cloud-based IDE and IoT hardware. Electronics Weekly
15. Jack J Purdum. Початок C для Arduino: learn C programming for the Arduino . - 2015. - ISBN 978-1-4842-0940-0 , 978-1-4842-0941-7

Додаток А

Лістинг програм

Програма керування рухом робота, на основі даних отриманих з фоторезистора:

```
#define light_L_Pin A1 // Значення лівого фоторезистора
#define light_R_Pin A2 // Значення правого фоторезистора
#define ML_Ctrl 13 // Значення штифта керування напрямком лівого
двигуна
#define ML_PWM 11 // Значення штифта керування ШІМ лівого двигуна
#define MR_Ctrl 12 // Значення штифта керування напрямком правого
двигуна
#define MR_PWM 3 // Значення штифта керування ШІМ правого двигуна

int left_light;
int right_light;

void setup() {
  Serial.begin(9600);
  pinMode(light_L_Pin, INPUT);
  pinMode(light_R_Pin, INPUT);
  pinMode(ML_Ctrl, OUTPUT); // Виправлено з "ВИВІД" на "OUTPUT"
  pinMode(ML_PWM, OUTPUT);  // Виправлено з "ВИХІД" на "OUTPUT"
  pinMode(MR_Ctrl, OUTPUT); // Виправлено з "ВИВІД" на "OUTPUT"
  pinMode(MR_PWM, OUTPUT);  // Виправлено з "ВИХІД" на "OUTPUT"
}

void loop() {
  left_light = analogRead(light_L_Pin);
  right_light = analogRead(light_R_Pin);

  Serial.print("left_light_value = ");
  Serial.println(left_light);
  Serial.print("right_light_value = ");
  Serial.println(right_light);
```

```
  if (left_light > 650 && right_light > 650) { // Дані отримані з
фоторезистора, Рух вперед
    Car_front();
  } else if (left_light > 650 && right_light <= 650) { // Дані
отримані з фоторезистора, Поворот ліворуч
    Car_left();
  } else if (left_light <= 650 && right_light > 650) { // Дані
отримані з фоторезистора, Поворот праворуч
    Car_right();
  } else { // інші ситуації, СТОП
    Car_Stop();
  }
}

void Car_front() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, 200);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, 200);
}

void Car_left() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, 200);
  digitalWrite(ML_Ctrl, HIGH);
  analogWrite(ML_PWM, 200);
}

void Car_right() {
  digitalWrite(MR_Ctrl, HIGH);
  analogWrite(MR_PWM, 200);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, 200);
}
void Car_Stop() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, 0);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, 0);
}
```

Програма керування рухом робота, на основі даних отриманих з УЗ датчика:

```
unsigned char start01[] =
{0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x80,0x40,0x20,0x10,0x0
8,0x04,0x02,0x01};
unsigned char front[] =
{0x00,0x00,0x00,0x00,0x00,0x24,0x12,0x09,0x12,0x24,0x00,0x00,0x0
0,0x00,0x00,0x00};
unsigned char back[] =
{0x00,0x00,0x00,0x00,0x00,0x24,0x48,0x90,0x48,0x24,0x00,0x00,0x0
0,0x00,0x00,0x00};
unsigned char left[] =
{0x00,0x00,0x00,0x00,0x00,0x00,0x44,0x28,0x10,0x44,0x28,0x10,0x4
4,0x28,0x10,0x00};
unsigned char right[] =
{0x00,0x10,0x28,0x44,0x10,0x28,0x44,0x10,0x28,0x44,0x00,0x00,0x0
0,0x00,0x00,0x00};
unsigned char STOP01[] =
{0x2E,0x2A,0x3A,0x00,0x02,0x3E,0x02,0x00,0x3E,0x22,0x3E,0x00,0x3
E,0x0A,0x0E,0x00};
unsigned char clear[] =
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0,0x00,0x00,0x00};

#define SCL_Pin A5 // Установка контакту годинника на A5
#define SDA_Pin A4 //Установка контакту даних на A4

#define ML_Ctrl 13 //визначте штифт керування напрямком лівого
двигуна
#define ML_PWM 11 //визначити штифт керування ШІМ лівого двигуна
#define MR_Ctrl 12 //визначте штифт керування напрямком правого
двигуна
#define MR_PWM 3 //визначте штифт керування ШІМ правого двигуна
#define Trig 5 //ультразвуковий тригер Pin
#define Echo 4 //ультразвукове exo Pin

int distance;
int pulsewidth;

#define servoPin 9 //servo Pin
```

```
void setup() {

  Serial.begin(9600);
  pinMode(SCL_Pin, OUTPUT);
  pinMode(SDA_Pin, OUTPUT);
  matrix_display(clear); //Очистити дисплей
  matrix_display(start01); //відображення шаблону початку
  pinMode(servoPin, OUTPUT);
  procedure(90); //встановити серво на 90°
  pinMode(Trig, OUTPUT);
  pinMode(Echo, INPUT);
  pinMode(ML_Ctrl, OUTPUT);
  pinMode(ML_PWM, OUTPUT);
  pinMode(MR_Ctrl, OUTPUT);
  pinMode(MR_PWM, OUTPUT);
}

void loop() {
  distance = checkdistance(); //призначити відстань, визначену
ультразвуковим датчиком, до відстані
  if (distance >= 20 && distance <= 60) { //діапазон для
переходу на передній план
    Car_front();
  } else if (distance > 10 && distance < 20) { //діапазон
зупинки
    Car_Stop();
  } else if (distance <= 10) { //діапазон для повернення
    Car_back();
  } else { // інші ситуації, стоп
    Car_Stop();
  }
}
```

```
/************функція для роботи двигуна****************/
void Car_front() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, 200);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, 200);
}

void Car_back() {
  digitalWrite(MR_Ctrl, HIGH);
  analogWrite(MR_PWM, 200);
  digitalWrite(ML_Ctrl, HIGH);
  analogWrite(ML_PWM, 200);
}

void Car_left() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, 200);
  digitalWrite(ML_Ctrl, HIGH);
  analogWrite(ML_PWM, 200);
}

void Car_right() {
  digitalWrite(MR_Ctrl, HIGH);
  analogWrite(MR_PWM, 200);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, 200);
}

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

```
/*********************точкова матриця*********************/

// функція матричного відображення
void matrix_display(unsigned char matrix_value[]) {
  IIC_start(); // виклик функції, яка запускає передачу даних
  IIC_send(0xc0); //Вибір адреси
  for (int i = 0; i < 16; i++) { //дані шаблону мають 16 біт
    IIC_send(matrix_value[i]); //дані для передачі шаблонів
  }
  IIC_end(); //кінець для передачі шаблону даних
  IIC_start();
  IIC_send(0x8A); //вибір ширини імпульсу4/16, керування
дисплеєм
  IIC_end();
}

//Умова початку передачі даних
void IIC_start() {
  digitalWrite(SCL_Pin, HIGH);
  delayMicroseconds(3);
  digitalWrite(SDA_Pin, HIGH);
  delayMicroseconds(3);
  digitalWrite(SDA_Pin, LOW);
  delayMicroseconds(3);
}

// передати дані
void IIC_send(unsigned char send_data) {
  for (char i = 0; i < 8; i++) { //Кожен байт має 8 біт
    digitalWrite(SCL_Pin, LOW); //витягніть контакт годинника
SCL Pin, щоб змінити сигнали SDA
    delayMicroseconds(3);
    if (send_data & 0x01) { //встановити високий і низький
рівень SDA_Pin відповідно до 1 або 0 кожного біта
      digitalWrite(SDA_Pin, HIGH);
    } else {
      digitalWrite(SDA_Pin, LOW);
    }
```

```
    delayMicroseconds(3);

    digitalWrite(SCL_Pin, HIGH); //підніміть контакт годинника
SCL_Pin, щоб припинити передачу даних
    delayMicroseconds(3);
    send_data = send_data >> 1; // виявляти біт за бітом, тому
переміщувати дані по одному
  }
}

//Ознака завершення передачі даних
void IIC_end() {
  digitalWrite(SCL_Pin, LOW);
  delayMicroseconds(3);
  digitalWrite(SDA_Pin, LOW);
  delayMicroseconds(3);
  digitalWrite(SCL_Pin, HIGH);
  delayMicroseconds(3);
  digitalWrite(SDA_Pin, HIGH);
  delayMicroseconds(3);
}

/*************** кінцевий матричний
дисплей******************/
//Функція керування сервоприводом
void procedure(int myangle) {
  for (int i = 0; i <= 50; i = i + (1)) {
    pulsewidth = myangle * 11 + 500;
    digitalWrite(servoPin, HIGH);
    delayMicroseconds(pulsewidth);
    digitalWrite(servoPin, LOW);
    delayMicroseconds(20 - pulsewidth / 1000);
  }
}
```

```
//Функція керування ультразвуковим датчиком Функція керування
ультразвуком
float checkdistance() {
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);
  float distance = pulseIn(Echo, HIGH) / 58.20; //58.20, тобто
2*29.1=58.2
  delay(10);
  return distance;
}
```

Програма керування рухом робота, на основі даних отриманих з ІЧ датчика:

```
// The wiring of line tracking sensor
#define L_pin  11  // left
#define M_pin  7   // middle
#define R_pin  8   // right
#define ML_Ctrl 4  // Define the direction control pin of the left
motor
#define ML_PWM 6   // Define the PWM control pin of the left motor
#define MR_Ctrl 2  // Define the direction control pin of the right
motor
#define MR_PWM 5   // Define the PWM control pin of the right motor
int L_val, M_val, R_val;

void setup() {
  Serial.begin(9600); // Set the baud rate to 9600
  pinMode(L_pin, INPUT); // Set all pins of the line tracking
sensor as input mode
  pinMode(M_pin, INPUT);
  pinMode(R_pin, INPUT);
  pinMode(ML_Ctrl, OUTPUT);
  pinMode(ML_PWM, OUTPUT);
  pinMode(MR_Ctrl, OUTPUT);
  pinMode(MR_PWM, OUTPUT);
}

void loop() {
  L_val = digitalRead(L_pin); // Read the value of the left sensor
  M_val = digitalRead(M_pin); // Read the value of the middle
sensor
  R_val = digitalRead(R_pin); // Read the value of the right sensor
```

```
  if (M_val == 1) { // The middle one detects black lines
    if (L_val == 1 && R_val == 0) { // If a black line is detected
on the left, but not on the right, turn left
      Car_left();
    } else if (L_val == 0 && R_val == 1) { // If a black line is
detected on the right, not on the left, turn right
      Car_right();
    } else { // Otherwise, go front
      Car_front();
    }


  } else { // The middle one doesn't detect black lines
    if (L_val == 1 && R_val == 0) { // If a black line is detected
on the left, but not on the right, turn left
      Car_left();
    } else if (L_val == 0 && R_val == 1) { // If a black line is
detected on the right, not on the left, turn right
      Car_right();
    } else { // Otherwise, stop
      Car_Stop();
    }
  }
}

// go front
void Car_front() {
  digitalWrite(MR_Ctrl, HIGH);
  analogWrite(MR_PWM, 100);
  digitalWrite(ML_Ctrl, HIGH);
  analogWrite(ML_PWM, 100);
}

// go back
void Car_back() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, 150);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, 150);
}
```

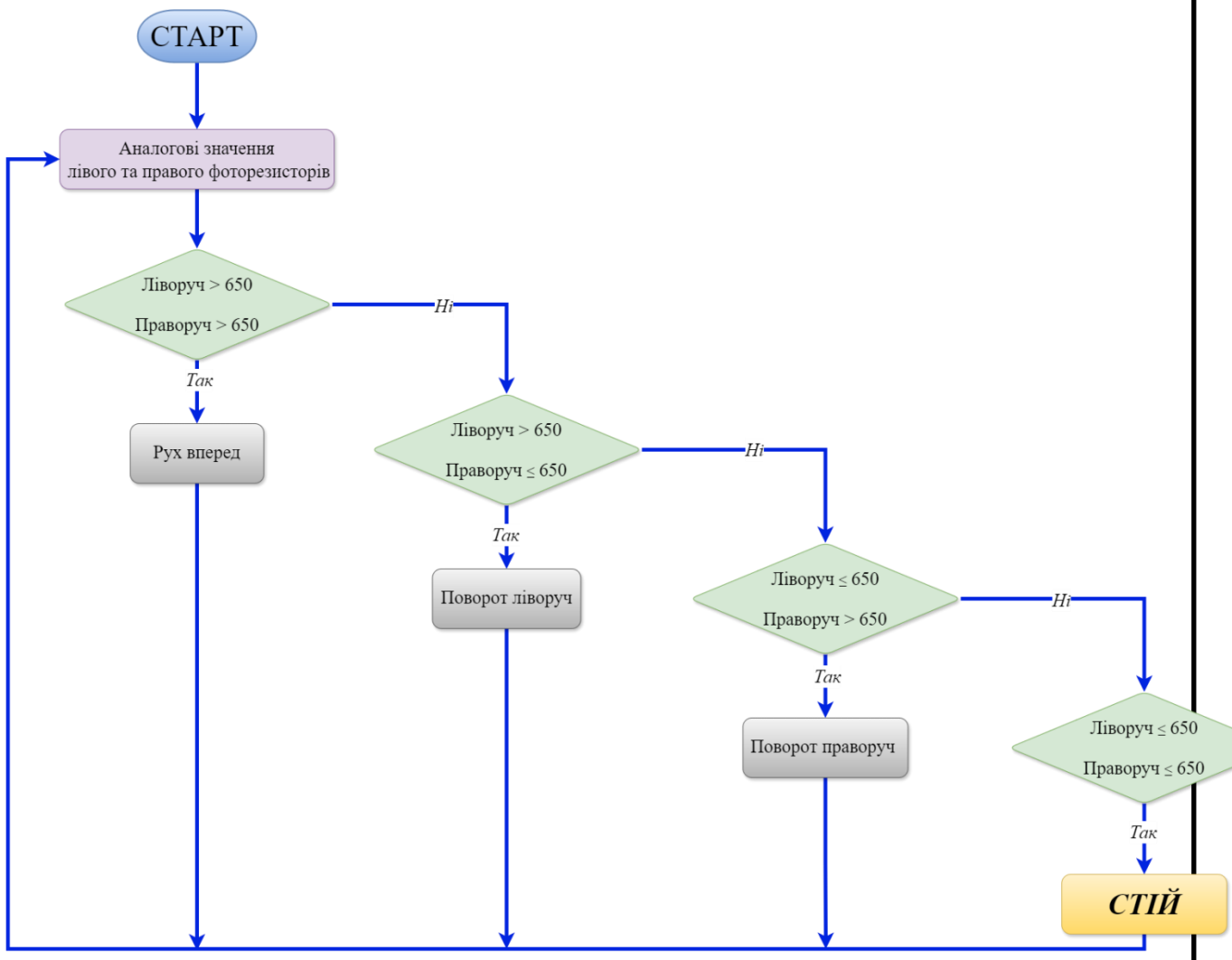
```
// turn left
void Car_left() {
  digitalWrite(MR_Ctrl, HIGH);
  analogWrite(MR_PWM, 100);
  digitalWrite(ML_Ctrl, LOW);
  analogWrite(ML_PWM, 150);
}

// turn right
void Car_right() {
  digitalWrite(MR_Ctrl, LOW);
  analogWrite(MR_PWM, 150);
  digitalWrite(ML_Ctrl, HIGH);
  analogWrite(ML_PWM, 100);
}

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

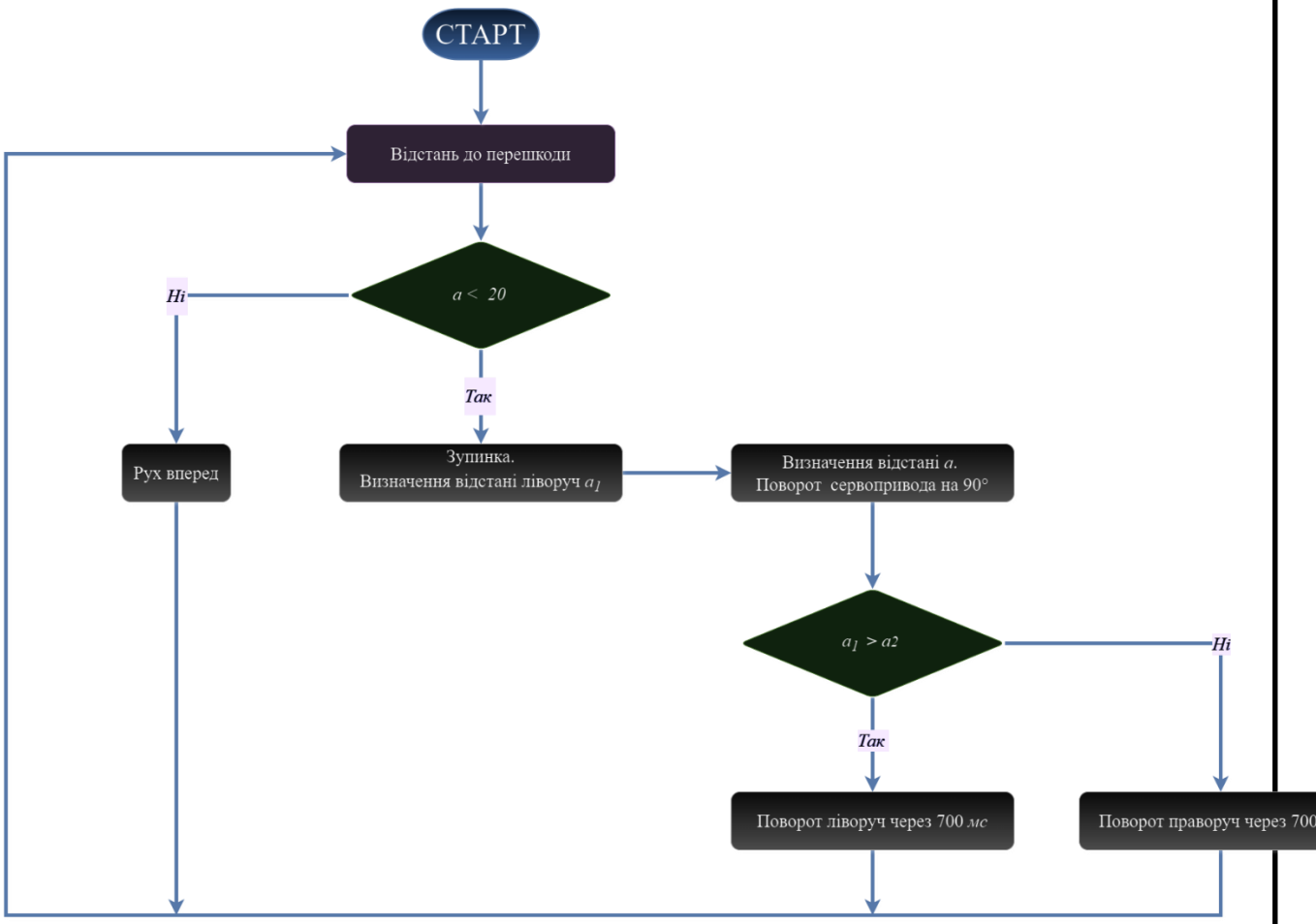
Додаток Б

Блок-схема логіки руху робота на основі даних з фотоелемента



Додаток В

Блок-схема логіки руху робота на основі даних з УЗ датчика.



Додаток Г

Блок-схема логіки руху робота на основі даних з ІЧ датчика.