Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ
В. о. завідувача кафедри,
канд. фіз.-мат. наук, доцент
_____ Я. М. Крайник
« __ » _____ 2022 р.

БАКАЛАВРСЬКА РОБОТА

Галузь знань: 12 Інформаційні технології
Спеціальність: 123 Комп'ютерна інженерія
Тема: **Програмно-апаратний комплекс для автономної навігації транспортних засобів у логістиці**

Шифр: 123 – КР.1 – 405.21810511

Виконав:
студент 4 курсу, групи 405,
спеціальності
123 Комп'ютерна інженерія
_____ М. О. Керекеслер

Керівник:
ст. викл.
_____ І. С. Бурлаченко

Миколаїв 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри,
канд. техн. наук, доцент
_____ Я. М. Крайник
« __ » _____ 2022 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Видано студенту групи 405 факультету комп'ютерних наук

Керекеслеру Максиму Олеговичу

(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи: Програмно-апаратний комплекс для автономної транспортних засобів у логістиці.

Затверджена наказом по ЧНУ від «__» _____ 2022 р. No _____

2. Строк представлення кваліфікаційної роботи «____» _____ 2022 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Прилад повинен отримувати інформацію з сенсорів та на її основі прокладати маршрут, вміти розпізнавати об'єкти та оминати їх.

4. Перелік питань, що підлягають розробці

Огляд технологій для системи з автономною навігацією. Моделювання прототипу, макетної та принципової схем. Збирання прототипу. Налаштування апаратної платформи. Створення серверної частини для обробки даних. Розробка програмного забезпечення приладу. Спеціальна частина з охорони праці. Висновки. Перелік джерел посилання.

5. Перелік графічних матеріалів

Зображення драйверів, двигунів, контролера, обчислювача, різних модулів

Зображення роботи нейронної мережі та серверної частини

Принципова схема компонентів та пристрою

Блок-схема принципу роботи системи.

6. Завдання до спеціальної частини

Розглянути основні державні норми України, щодо праці в умовах використання комп'ютерів та екраних пристроїв загалом, щодо вентиляції та кондиціонування, організація
повітрообміну, норм шумів та вібрацій. Ознайомитись з правами та нормами праці в умовах використання комп'ютерів та екраних пристроїв загалом.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
ст. викладач А. О. Алексєєва	кафедра екології Медичного інституту ЧНУ ім. Петра Могили	спеціальна частина з охорони праці

Керівник роботи _____старший викладач , І. С. Бурлаченко___ _____
(посада, прізвище, ім'я, по батькові) (підпис)

Завдання прийнято до виконання __Керекеслер Максим Олегович___ _____
(прізвище, ім'я, по батькові студента) (підпис)

Дата видачі завдання «____» _____ 20_____ р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Апаратно-програмний комплекс для людей з вадами дихальної системи

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КР	20.12.2021	05.01.2022	Виконано
2	Огляд літератури за темою роботи	08.01.2022	01.02.2022	Виконано
3	Складання календарного плану КР	02.02.2022	10.02.2022	Виконано
4	Аналіз предметної області	11.02.2022	17.02.2022	Виконано
5	Розробка проектних рішень	18.02.2022	18.03.2022	Виконано
6	Моделювання апаратної частини	19.03.2022	19.04.2022	Виконано
7	Розробка програмного забезпечення та тестування	20.04.2022	21.05.2022	Виконано
8	Відгук керівника КР	22.05.2022	01.06.2022	Виконано
9	Оформлення КР та презентації	02.06.2022	09.06.2022	Виконано
10	Попередній захист	10.06.2022	11.06.2022	Виконано
11	Рецензування	14.06.2022	18.06.2022	Виконано
12	Завершення оформлення КР та презентації	18.06.2022	25.06.2022	Виконано
13	Захист кваліфікаційної роботи	27.06.2022	28.06.2022	Виконано

Розробив студент_____Керекеслер Максим Олегович_____ _____

(прізвище, ім’я, по батькові) (підпис)

«__» _____ 2022 р.

Керівник роботи ___старший викладач , І. С. Бурлаченко_____ _____

(посада, прізвище, ім’я, по батькові) (підпис)

«__» _____ 2022 р.

АНОТАЦІЯ

бакалаврської роботи

«Програмно-апаратний комплекс для автономної навігації транспортних засобів у логістиці»

Студент: Керекеслер Максим Олегович

Керівник: ст. викл. І. С. Бурлаченко

Бакалаврська робота присвячена розробці програмно-апаратного комплексу для автономної навігації транспортних засобів у логістиці. Розглянуто наявні технології для автономної навігації та алгоритми будування. Практичне значення результатів дослідження та розроблення полягає в тому що, на підставі результатів можливе вдосконалення апаратно-програмного комплексу, у засвоєнні навичок процесу проектування, розробки, створення приладу, можливостях застосування розробленого програмно-апаратного комплексу.

Пояснювальна записка бакалаврської роботи складається зі вступу, трьох розділів, висновків та додатків. У вступі визначається актуальність теми, сформульовані мета, об’єкт, предмет та завдання дослідження та розроблення бакалаврської роботи. У першому розділі проводиться аналіз існуючих технологій для автономної навігації та обираються необхідні компоненти. У другому розділі розглянуто принцип роботи кожного елементу системи та принцип роботи всієї системи. У третьому розділі описано розробку програмного забезпечення системи з автономною навігацією. У висновках наведено аналіз виконаної роботи та отриманих результатів дослідження та розроблення.

У додатках А та Б наведено програмний код, що використовувався в проекті. В цілому, бакалаврська робота без додатків містить 81 сторінку, 29 рисунків, 3 таблиці, 28 джерел посилання.

Ключові слова: мікрокомп’ютер, мікроконтроллер, нейронна мережа, gps-модуль, компас, ультразвуковий сенсор.

ABSTRACT

of the Bachelor's Thesis

"Software and hardware complex for autonomous navigation of vehicles in logistics"

Student: Kerekesler Maxim Olegovich

Consultant: senior lecturer I.S. Burlachenko

The bachelor's thesis is devoted to the development of software and hardware for autonomous navigation of vehicles in logistics. The available technologies for autonomous navigation and construction algorithms are considered. The practical significance of the results of research and development is that, based on the results, it is possible to improve the hardware and software complex, in mastering the skills of the process of design, development, device creation, application of the developed software and hardware complex.

The explanatory note of the bachelor's thesis consists of an introduction, three sections, conclusions and appendices. The introduction determines the relevance of the topic, formulates the purpose, object, subject and objectives of research and development of the bachelor's thesis. The first section analyzes the existing technologies for autonomous navigation and selects the necessary components. The second section considers the principle of operation of each element of the system and the principle of operation of the entire system, as well as calculations for the power supply system. The third section describes the software development of the system with autonomous navigation. The conclusions provide an analysis of the work performed and the results of research and development.

Appendices A and B show the program code used in the project. In total, the bachelor's thesis without appendices contains 81 pages, 29 figures, 3 tables, 16 reference sources.

Keywords: microcomputer, microcontroller, neural network, gps module, compass, ultrasonic sensor.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

USB – Universal Serial Bus

VIN – Input Voltage

GND – Ground

VCC – Voltage Common Collector

IDE – Integrated Development Environment (укр. інтегроване середовище розробки)

I²C – Inter-Integrated Circuit

BLDC – Brushless Direct Current

API – Application programming interface(укр. інтерфейс прикладного програмування)

GPS – Global Positioning System

YOLO – You Only Look Once

COCO – Common Objects in Context

CNN – Convolutional Neural Network

ОС – Operating System

UART – Universal Asynchronous Receiver-Transmitter

ВСТУП

Мобільна робототехніка – галузь, що динамічно розвивається, спрямована на вирішення завдань у різних галузях – від промисловості до аерокосмічної сфери, від гірничої справи до медицини.

Поки що основною проблемою всіх мобільних апаратів, що нині існують, що переміщаються самостійно, без управління з боку людини, залишається навігація. Для успішної навігації в просторі бортова система робота повинна вміти будувати маршрут, управляти параметрами руху, правильно інтерпретувати відомості про навколишній світ, які отримують від датчиків, і постійно відстежувати власні координати.

Фахівці з робототехніки займаються складанням, програмуванням та обслуговуванням робототехнічних систем, здійснює технічне обслуговування роботів, виявляють та усувають несправності управління механічних та електричних частин, проводять дослідження щодо розширення можливостей роботів. Роботи, що створюються, можуть максимально точно виконувати задані функції, мати швидку окупність, працювати довгий час. Це сприяє підвищенню якості продукції.

Мета: Розробка апаратно-програмного комплексу пристроїв, які автономно виконують обчислення правильного маршруту до цілей, аналіз навколишнього середовища та оминають перешкоди.

Об'єкт: Система автономної навігації транспортних засобів в логістиці.

Предмет: Технології автономної навігації безпілотних наземних транспортних засобів.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

– Проаналізувати сучасні технології автономної навігації;

– Розробити прошивку для мікроконтролера Arduino для управління навігаційних модулів та модулів пересування(ультразвуковий сенсор, драйвер двигунів, тощо);

– Розробити застосунок для отримання даних з навігаційних модулів та побудови маршруту для мікрокомп’ютера;

– Розробити застосунок для побудови маршруту до цілі на основі інформації місцезнаходження отриманого з gps-модуля;

– Забезпечити систему живлення з необхідним запасом енергії для всіх компонентів;

– Виконання завдання з охорони.

Практичне значення отриманих результатів:

Даний пристрій можна використовувати в сфері доставки продукції з не великою вагою, наприклад доставка їжі. Також його можна використовувати для дослідження нейронних мереж з комп’ютерним зором.

Апробація Керекеслер М.О., Бурлаченко І. С. Аналіз характеристик комплектуючих для безпілотних наземних транспортних засобів. «Ольвійський форум-2022: Стратегії країн Причорноморського регіону в геополітичному просторі»: зб. тез доп. XVI Міжнародної наукової конференції. м. Миколаїв, 23–26 червня 2022 р. Миколаїв, 2022..

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА СКЛАДОВИХ СИСТЕМИ

1.1 Роботи в сфері доставки

Наземні мобільні роботи найчастіше використовуються в сегменті доставки. Ефективність використання наземних роботів-кур'єрів досить велика, так як роботи-кур’єри можуть знизити вартість доставки в порівнянні з використанням кур'єрів-людей, а значить, знизиться собівартість надання послуги, що послужить додатковим стимулом до впровадження таких пристроїв. Наземним мобільним роботам доведеться конкурувати з доставкою за допомогою безпілотників, що літають. Перед доставкою літаючими безпілотниками наземна доставка роботами має явну перевагу, що ні вантаж, ні кур'єр не впадуть людям на голову навіть у випадку, якщо "щось піде не так". Однак наземні роботи можуть зустрітися з проблемами крадіжки та вандалізму.

У 2016 році кілька компаній випробовують вуличних роботів-кур'єрів. З 2016 року розпочалося комерційне застосування вуличних роботів-кур'єрів. Одне з непростих завдань для створення таких роботів, - це розробка алгоритму, який дозволяв би роботу знаходити свій шлях у потоці людей, але так, щоб виключити ризик зіткнення з ними, принаймні з вини робота.

Роботи оснащені сенсорами, у тому числі камерами з машинним зором, радарами та ультразвуковими сенсорами, які виявляють тверді об'єкти, такі як бордюри та стіни.

В різних країнах світу існують свої рішення даних роботів. Наприклад:

- Marathon Robotics в Австралії;
- Piaggio в Італії;
- Eliport в Іспанії;
- Sidewalk в Литві;

– Kar-go в Об'єднаному Королівстві;
– Amazon Scout, SameDay, Starship в США.

Частіше всього компанії самі розробляють або замовляють спеціальні компоненти, які необхідні для роботи робота-кур’єра. Так, наприклад, компанія Startship розробила свою материнську плату обчислювача, свої камери та сенсори. Завдяки цьому апаратна частина буде робити ефективніше та споживати менше енергії.

1.2 Вибір комплектуючих

Перед тим як почати збирати прилад необхідно визначитися яким характеристикам та якій ціні мають відповідати комплектуючі. Частіше всього необхідні відмінні комплектуючі дуже дорого коштують, тому доводиться обирати дешеві аналоги.

Після порівняння аналогів, обрати найбільш підходящий варіант, відповідний усім необхідним критеріям, саме для вашого приладу.

Для даного приладу необхідно обрати такі комплектуючі: колеса та мотори для них, драйвер двигуна, мікроконтролер, мікрокомп’ютер, ультразвуковий сенсор, камера, gps-модуль.

1.2.1 Мотори-редуктори

Для руху робота необхідні колеса та мотори. Враховуючи, що сама платформа с приладами та вантаж матимуть велику вагу, то потрібно обирати колеса та мотори з достатньо великими розмірами, які зможуть витримати велике навантаження. Для цього можуть підійти мотор-колеса для електро-самокатів та гіроскутерів, які мають розміри 6,5-10 дюймів, або інші BLDC мотори.

Рис. 1.1 – Колесо мотор для гіроскутера на 10 дюймів

Однак такі мотор-колеса досить дорого коштують та споживатимуть багато енергії. В якості дешевої альтернативи було обрано звичайні мотори с редуктором Arduino та колеса до них. Дані мотори-редуктори дуже розповсюджені і їх легко використовувати. Вони не підійдуть для доставки товарів, але їх можна використовувати для перевірки працездатності робота.

Рис. 1.2 – Мотор-редуктор для Arduino з колесом

Шина має гумове покриття. Діаметр колеса з шиною: 6.6 см. Мотор з редуктором має такі характеристики:

- розміри: 64.2мм × 22.5мм × 18,8мм;
- робоча напруга: 3,0 v ~ 12.0v DC;
- кількість обертів в хвилину: 20 ~ 100;

Для переміщення робота знадобиться 4 колеса та мотора. За потребою їх кількість можна збільшити.

1.2.2 Драйвер двигуна

Для управління моторами ми будемо використовувати драйвера двигуна, так як підключати мотори напряму до мікроконтролера не можна. Драйвер буде регулювати швидкість моторів за допомогою ШИМ-сигналу, який подається з мікроконтролера.

Якщо б ми обрали великі мотор-колеса, то для їх управління нам знадобиться драйвер BLD-300B(рис. 1.3).

Рис. 1.3 – Драйвер BLD-300B

Цей драйвер може забезпечувати вихідну потужність 300VA max. Також він забезпечує високий крутний момент, низький рівень шуму, низький рівень вібрації. Працює він при напрузі 18-50V.

Однак для мотора-редуктора підійдуть інші моделі. Був обраний модуль l298n.

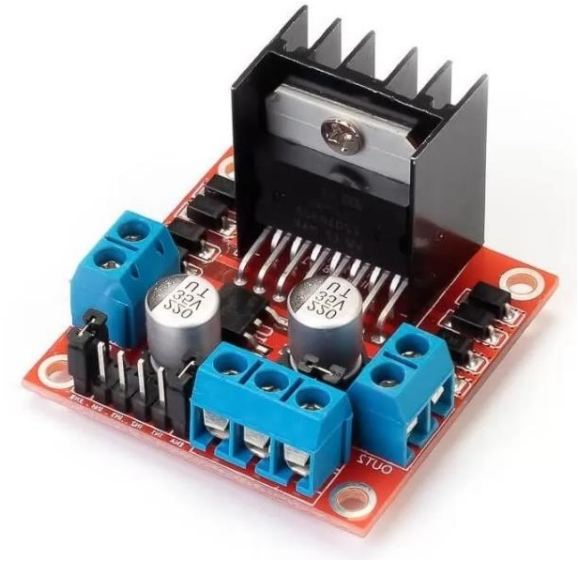

Рис. 1.4 – Драйвер l298n

Схема модуля, що складається з двох H-мостів, дозволяє підключати до нього один біполярний крокових двигун або одночасно два щіткові двигуни постійного струму. При цьому є можливість змінювати швидкість та напрямок обертання моторів. Управління здійснюється шляхом подання відповідних сигналів на командні входи. Основні характеристики модуля:

- Напруга живлення логіки: 5V;
- Споживаний логікою струм: 36mA;
- Напруга живлення двигунів: від 5V до 35V;
- Робочий струм драйвера: 2 А;
- Піковий струм драйвера 3 А;
- Максимальна потужність: 20 Вт (при температурі 75 ° С);
- Розміри модуля: 43.5 x 43.2 x 29.4 мм;

Модуль має роз'єми OUT1 і OUT2 для одного двигуна, та OUT3 і OUT4 для другого. Також модуль має вхід для живлення та вхід живлення логіки 5V. Також в модуль вбудований стабілізатор напруги, через який логіка модуля також може отримувати живлення. Контакти IN1, IN2 відповідають за управління першим двигуном, а контакти IN3, IN4 відповідають за управління другим двигуном. Контакти ENA, ENB відповідають за активацію та

деактивацію двигунів, також на них можна подати ШИМ-сигнал для регулювання швидкості обертання моторів.

1.2.3 Мікроконтролер

Для управління модулями було обрано один з мікроконтролерів Arduino. Arduino зручна платформа для розробки різних електронних пристроїв. Основними перевагами Arduino є: доступність, зручність, відкрита архітектура та програмний код, простота мови програмування. За допомогою
Arduino можна управляти сенсорами, різними модулями, в тому числі і драйвером двигунів. Існує безліч різновидів Arduino, однак частіше використовують Arduino UNO та Arduino Nano. Головна відмінність між ними це розмір. Arduino Nano має менші розміри та використовую кабель mini-USB, тоді як UNO використовує кабель USB. Однак обидві моделі використовують процесор ATmega 328P, тому вони можуть виконувати однакові програми. Для робота, який буде мати і так не маленькі габарити, розмір плати не має значення, тому було обрано Arduino UNO(рис. 1.5).

Рис. 1.5 – Arduino UNO

Також Arduino можна підключити до мікрокомп’ютера через UART інтерфейс або через USB. Це дає змогу обмінюватися інформацією між Arduino та мікрокомп’ютером.

1.2.4 Мікрокомп’ютер

Для обчислення маршруту робота-кур’єра необхідний мікрокомп’ютер з потужним процесором. Для цього було обрано одну одну з моделей Raspberry Pi.

Рис. 1.6 – Raspberry Pi

Raspberry Pi – це родина багатофункціональних мікрокомп’ютерів з розміром не більше кредитної карти. Девайс має чотириядерний процесор на борту та купу інтерфейсів, починаючи від Bluetooth та Wi-Fi, закінчуючи GPIO. Завдяки інтерфейсам і мережевим протоколам, міні-ПК можна з’єднати з настільними комп'ютерами, ноутбуками та смартфонами, а мережеві протоколи дозволяють здійснювати контроль над відеокамерами, сучасними TV і навіть дронами, що літають. Raspberry Pi має відносно невелику ціну і може використовуватися для автоматизації будь-яких невеликих задач. Також цей девайс не потребує великого досвіду програмування. Основною мовою програмування Raspberry Pi є Python, для якого написано безліч бібліотек для різних задач. Однак писати код можна і на інших мовах програмування, будь то C, C++, C#, Ruby, Java і т.д.. Прилад має велику популярність через доступність та невелику ціну. Raspberry Pi поставляється з 26 контактами GPIO, які дійсно дуже корисні для вбудованих проектів та інтерфейсного обладнання. Ці виходи дійсно корисні при вивченні взаємодії компонентів. Можна поєднати кілька цифрових сенсорів разом з великою кількістю контактів GPIO. Він підтримує майже всі периферійні пристрої, які

підтримують Arduino. Він підтримує кілька сенсорів одночасно. До нього можна підключати різні дисплеї, модулі, сенсори і т.д. Raspberry Pi постачається з швидким процесором, що дає хорошу продуктивність.

Існує багато моделей Raspberry Pi, такі як: Pi 1 A, Pi 1 A+, Pi 1 B, Pi 1 B+, Pi 2 B, Pi 3 A+, Pi 3 B, Pi 3 B+, Pi Zero, Pi Zero W, Pi 4 B. Вони відрізняються процесором та кількістю оперативної пам’яті, кількістю портів USB та наявністю модуля Wi-Fi. Порівняльна характеристика наведена в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика моделей Raspberry Pi

RPI model	**CPU**	**Кількість ядер**	**ОЗУ**	**Wi-Fi**
Pi 1 A	Broadcom BCM2835	1×700 МГц	256	-
Pi 1 A+	Broadcom BCM2835	1×700 МГц	256	-
Pi 1 B	Broadcom BCM2835	1x700 МГц	512	-
Pi 1 B+	Broadcom BCM2835	1x700 МГц	512	-
Pi 2 B	Broadcom BCM2836	4x900 МГц	1 Гб	-
Pi 3 A+	Broadcom BCM2837B0	4x1.4 ГГц	512	802.11ac 4.2
Pi 3 B	Broadcom BCM2837	4x1.2 ГГц	1 Гб	802.11n 4.1
Pi 3 B+	Broadcom BCM2837B0	4x1.4 ГГц	1 Гб	802.11ac 4.2
Pi Zero	Broadcom BCM2835	1x1 ГГц	512	-
Pi Zero W	Broadcom BCM2835	1x1 ГГц	512	802.11n 4.0
Pi 4 B	Broadcom BCM2711	4x1.5 ГГц	1 Гб 2 Гб 4 Гб	802.11ac 5.0

Для виконання завдання було обрано модель Pi 3 B тому що, вона є не поганим продуктом в порівнянні потужності та ціни.

1.3 Модулі для автономної навігації

Для запобігання зіткнення з перешкодами робота необхідно обладнати спеціальними модулями, які будуть сканувати навколишнє середовище та виявляти перешкоди. Для виявлення перешкод можна використати кілька сенсорів, такі як: сенсор зображення(камера), ультразвуковий сенсор, лідар, TimeOfFlight.

1.3.1 TimeOfFlight

Time Of Flight (ToF) – це метод вимірювання відстані між сенсором та об'єктом, заснований на різниці у часі між випромінюванням сигналу та його поверненням до сенсора після відображення від об'єкта. Різні типи сигналів можуть використовуватися з принципом часу проходження, найбільш поширеними є звук і світло. Всі сенсори ToF вимірюють відстані, використовуючи час, який потрібно фотонам для переміщення між двома точками, від випромінювача сенсора до мети, а потім до приймача сенсора.

3D-камери ToF можуть захоплювати дані глибини у трьох просторових вимірах. Для визначення дальності та визначення відстані ToF дуже ефективний при випромінюванні світла, а не звуку. У порівнянні з ультразвуком, він забезпечує набагато більший діапазон, більш швидкі показання та більшу точність, зберігаючи при цьому малий розмір, малу вагу та низьке енергоспоживання. Сенсори ToF використовуються для низки програм, включаючи навігацію роботів, моніторинг транспортних засобів, підрахунок людей і виявлення об'єктів. Сенсори відстані ToF використовують час, необхідний фотонам для переміщення між двома точками для розрахунку відстані між точками.

Для вимірювання відстані для роботів можна застосувати сенсори ToF Teraranger Evo та IND-TOF-1 від компанії Terabee.

Teraranger Evo – точний, високошвидкісний сенсор вимірювання відстані та виявлення об'єктів на близькій відстані. Такий сенсор ідеально підходить для програм мобільної робототехніки. Компанія Terabee створила декілька моделей даного сенсора, які відрізняються відстанню виявлення об’єктів. Максимальна відстань для виявлення об’єктів 60 метрів.

Рис. 1.7 – TeraRanger Evo

Сенсори TeraRanger Evo складаються з оптико-електронного сенсорного пристрою та задньої панелі, яка просто підключається, щоб забезпечити сенсор каналом зв'язку та можливостями керування живленням без необхідності використання адаптерів або складної проводки. Задню плату можна вибрати за власними потребами. Доступні задні плати USB, I2C та UART. Сенсор сумісний з Arduino, Raspberry Pi, Pixhawk та ROS. Однією з переваг є те, що він працює при слабкому освітленні і в повній темноті.

Також можна підключити декілька сенсорів до плати TeraRanger Hub Evo і таким чином створити масив сенсорних модулів з багатьма напрямками. Всього можна підключити до 8 сенсорів.

Сенсор IND-TOF-1 пропонує можливості виявлення на відстані 12,5 метра з використанням технології Time-of-Flight. Міцний корпус зі ступенем захисту IP65 забезпечує пило- та водонепроникність - в компактному (99 грам) та економічному виконанні. Сенсор забезпечує повідомлення про наближення

за допомогою класичного перемикаючого виходу NO/NC (0-24 В), а також передає відкалібровані дані про відстань через інтерфейс RS485. Шість вбудованих режимів роботи дозволяють використовувати різні програми моніторингу. IND-TOF-1 легкий (99 г) та компактний, що дозволяє встановлювати його у місцях, недоступних для великих та важких сенсорів.

1.3.2 LiDAR

Light Detection and Ranging(LiDAR) – це технологія дистанційного зондування, яка використовує лазерний імпульс для збору вимірювань, які потім можна використовувати для створення 3D-моделей, карт об'єктів і навколишнього середовища.

LiDAR працює аналогічно радару та сонару, але використовує світлові хвилі від лазеру замість радіо та звукових хвиль. Система лідар розраховує, скільки часу потрібно світлу, для попадання в об'єкт і відображення назад в сканер. Відстань розраховується з використанням швидкості світла. Системи можуть генерувати близько 1 000 000 імпульсів в секунду. Кожне з цих замірів або результатів може бути перетворено в тривимірну візуалізацію, яка являє собою хмару точок.

Хмари точок - це сукупність точок, що представляють тривимірну форму або об'єкт. Кожна точка має власний набір координат X, Y та Z, а в деяких випадках додаткові атрибути.

Система Lidar використовується для різних задач, однією з яких радар для автомобілів. Основною ідеєю роботи навігаційного лідара є те, що він випускає лазерні промені в простір навколо себе, вони відбиваються від різних перешкод і уловлюють спеціальними приймачами. Час, необхідний для повернення променя, говорить нам, як далеко знаходиться кожна перешкода від автомобіля. Отримані матеріали обробляє центральний процесор, який керує автомобілем. Також він на основі місцезнаходження завдяки системі GPS прокладає маршрут і рухається до точки призначення. Безпілотному

автомобілю потрібні дані про десятки об'єктів навколо. Тому Lidar крутиться навколо своєї осі, випромінюючи безліч світлових спалахів, і таким чином формує з «хмари точок» тривимірне 360-градусне зображення навколишнього оточення. І робити це може в будь-яких середовищах, погодних умовах і незалежно від часу доби. Щоб безпілотні автомобілі могли без участі людини пересуватися у просторі, їм потрібна комбінація відеокамер, радарів та лідарів. І лідар виконує критично важливу функцію - він дає автомобілю уявлення не тільки про власну локалізацію, а й місце розташування навколишніх об'єктів.

GPS у цьому випадку непридатно – воно визначає місце розташування з формуванням кола діаметром близько 5 м, а лідар робить це з точністю до 10 см.

Перевагами лідара є:

– Висока швидкість і точність збору.

– Висока проникність.

– Не залежить від інтенсивності світла в навколишньому середовищі і може використовуватися вночі або на сонці.

– Відсутність геометричних спотворень.

– Легко інтегрується с іншими методами збору;

– LIDAR має мінімальну залежність від людини, що добре в певних сферах.

На сьогоднішній день більшість лідарів – це дорогі рухливі пристрої. Звичайно, як і всі технології, лідари з часом стали дешевшими і зменшилися в розмірах. Але, поки вони коштують від $8000 до $80000, і швидко виходять з ладу.

А ось статичні лідари – інша річ. Зроблені головним чином з кремнію, в них немає рухомих частин і використовується лазер зі змінною довжиною хвилі. Такий пристрій легкий, маленький, споживає мало енергії (працює на батарейці AA), швидкий і точний, працює за будь-яких погодних умов, і

коштує не тисячі, а сотні доларів. На ринку є багато різних LiDAR сенсорів, які мають різні переваги.

LeddarTech Vu8 – це компактний напівпровідниковий лідар, який забезпечує високоточне виявлення кількох цілей у восьми незалежних сегментах. Виявивши цілі на відстані до 215 метрів і вагою всього 75 грамів, Vu8 забезпечує майже вдвічі більшу дальність при вдвічі меншому об'ємі порівняно з Leddar M16, від якого він надихається.

Рис. 1.8 – LeddarTech Vu8

У Vu8 використовується стаціонарне лазерне джерело світла, що значно підвищує надійність сенсора та економічність. Vu8 має високу стійкість до шуму і перешкод. Це означає, що на нього не впливають сигнали інших сенсорів, умови освітлення, включаючи пряме сонячне світло, і забезпечує надійне виявлення в різних погодних умовах, включаючи дощ і сніг.

LeddarOne є сенсорним модулем з повним променем, який повністю призначений для вимірювання в одній точці, що робить його ідеальним для таких додатків, як визначення рівня , безпека і спостереження і виявлення присутності .

LeddarOne можна легко інтегрувати практично у будь-яку систему. Компактний розмір модуля, низьке енергоспоживання та висока точність дають розробникам та інтеграторам широкі можливості для вдосконалення своїх продуктів.

Сенсор HDL-32E LiDAR невеликий, легкий, міцний та має до 32 лазерів у вертикальному полі зору 40 градусів. HDL-32E має розміри всього 5,7″ у висоту та 3,4″ у діаметрі, важить менше 2 кг і був розроблений, щоб перевершити вимоги найскладнішої автономної навігації у реальному світі, мобільного 3D-картографування та інших лідарних додатків.

Рис. 1.9 – HDL-32E LiDAR

Новий лідарний сенсор PUCK VLP-16 від Velodyne - це найменший, найновіший і просунутий продукт у лінійці 3D-лідарів Velodyne. Він більш економічний, ніж сенсори аналогічної вартості, та розроблений з урахуванням масового виробництва.

Він зберігає ключові особливості проривів Velodyne в області лідара: в режимі реального часу, 360-градусний горизонтальний кут огляду, тривимірна відстань та калібровані виміри відбивної здатності.

1.3.3 Ультразвукові сенсори

Ультразвукові сенсори відрізняються високою надійністю та неймовірною універсальністю застосування у різних галузях промисловості. Вони можуть використовуватися навіть для вирішення більш складних завдань, що включають розпізнавання об'єктів або вимірювання рівнів з точністю до міліметра, завдяки надійності роботи їх методу вимірювання, практично, у будь-яких умовах.

Ультразвукові сенсори довели свою надійність і високу точність практично у всіх галузях промисловості.

Для вимірювання відстані використовують високочастотні звукові імпульси, що випускаються сенсором. Ці імпульси випромінюються як конічного пучка і під час зустрічі з будь-якої поверхнею відбиваються від неї. Принцип роботи сенсорів ґрунтується на вимірі часу проходження сигналу. Це дозволяє виявляти об'єкти та вимірювати відстань від сенсора до них.

Ультразвук – це акустична хвиля дуже високої частоти, яка недоступна людському слуху. Оскільки вважається, що чутний діапазон частот знаходиться між 16 Гц і 20 кГц, ультразвук зазвичай означає акустичні хвилі вище 20 кГц.

Ультразвук має кілька характеристик, які роблять його таким корисним і сприяли його використанню в багатьох додатках електроніки. По-перше, він нечутий для людей і, отже, не виявляється користувачем. По-друге, ультразвукові хвилі можуть бути отримані з високою спрямованістю. По-третє, вони є компресійними коливаннями матерії (зазвичай повітря). Нарешті, вони мають нижчу швидкість поширення, ніж світло або радіохвилі.

Найбільш відомий ультразвуковий сенсор для Arduino це HC-SR04. Цей сенсор має велику популярність, доступність та невелику ціну. Діапазон дальності його виміру становить від 2 до 400 см. На його роботу не має істотного впливу електромагнітні випромінювання та сонячна енергія.

Існують і інші ультразвукові сенсори, які відрізняються відстанню вимірювань, робочою напругою, точністю вимірювань та ціною, такі як: URM37, HY-SRF05, URM08-RS485, US-015, US-100, JSN-SR04T-2 V3. Сенсор URM37 відстань вимірювань 8м та високу точність. Час вимірювань даного прибору 100мс. Сенсор URM08-RS485 більш дорожчий і також має велику точність вимірювань, але час вимірювань 70мс.

1.3.4 Gps-модулі

Для розпізнавання місцезнаходження можна використовувати gps-модуль. Gps-модулі встановлюють зв’язок з супутниками і отримують координати місцезнаходження. Найбільш розповсюджений gps-модуль являється u-blox NEO-6m-001. Для зв’язку з супутниками потрібен час. Холодний старт даного модуля від 5 до 20 хвилин, при умові що він знаходиться під відкритим небом або біля вікна. Гарячий старт близько 1 секунди. Даний модуль допомагає визначити місцезнаходження, але не досить точно.

Рис. 1.10 – NEO-6m-001

Для більш точних вимірювань потрібні більш дорогі моделі. Наприклад, модуль NEO-8m має частоту оновлення даних до 18 Гц(Один супутник) та 10 Гц(Багато супутників). Він може підключитися до 72 супутників, в той час як NEO-6m може підключитися лише до 12. Це дає більш точні дані місцезнаходження. Також час холодного старту даного модуля лише 26 секунд, що досить швидко.

Beitian BN-880 – це GPS-модуль, що добре зарекомендував себе. Модуль оснащений GPS-чіпом M8N та модулем компасу HMC5883l I2C, які разом забезпечують швидку фіксацію GPS 3D та точну навігацію. Модуль має флеш-

пам'ять для збереження конфігурації, але для більшості програм, включаючи iNav та Betaflight, він поставляється готовим до використання. Споживання енергії цього модуля 50 мА при 5 В. Цей модуль також може підключитися до 72 супутників. Час холодного старту 26 секунд, а гарячого – 3 секунди.

Рис. 1.11 – Beitian BN-880

Найбільш дорогим і точним GPS-модулем являється GPS-RTK2 – ZED-F9P.

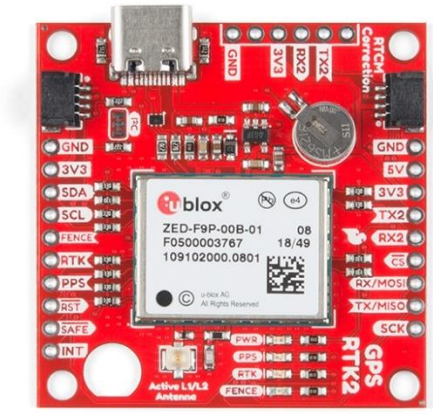

Рисунок 1.12 – GPS-RTK2 – ZED-F9P

ZED-F9P – це першокласний модуль для високоточних рішень позиціонування GNSS і GPS, включаючи RTK з точністю до 10 мм і тривимірною точністю. ZED-F9P є унікальним тим, що може працювати як з мобільним пристроєм, так і з базовою станцією.

Сенсор має резервну батарею, щоб остання конфігурація модуля та супутникові дані були доступні протягом двох тижнів. Ця батарея допомагає гарячому запуску модуля, значно скорочуючи час до першого виправлення.

1.3.5 Камера

Також для аналізу навколишніх об’єктів можна використовувати звичайну камеру, через яку обчислювач за допомогою нейронної мережі буде впізнавати різні об’єкти і на основі інформації про ці об’єкти та їхньої позиції на кадрі буде визначати в якому напрямку рухатись або зовсім не рухатись.

Для мікрокомп’ютера Raspberry Pi існує спеціальна Pi-камера High Quality Camera(Рисунок 1.12). Дану камеру легко підключити до Raspberry Pi та налаштувати. Вона має матрицю-сенсор Sony IMX477R. Для роботи потрібні об'єктиви за допомогою адаптерів типу C або CS. Модуль має 12,3-мегапіксельний сенсор. Максимальна роздільна здатність Raspberry Pi High Quality Camera становить 4056 x 3040 пікселів (5K).

Рис. 1.13 – Pi High Quality Camera без об'єктиву

1.3.6 Акселерометр, магнітометр

Також для визначення сторони світу на яку направлений пристрій знадобиться акселерометр та магнітометр.

Акселерометр – це прилад, який вимірює прискорення (величину зміни швидкості). Вимірювання величини динамічного прискорення дозволяє визначити, наскільки швидко та у якому напрямку рухається пристрій з акселерометром.

Магнітометр - це пристрій, який вимірює напруженість магнітного поля. Всі сучасні електронні магнітометри виготовляються за технологією МЕМС і дозволяють проводити вимірювання відразу по трьох перпендикулярних осях. Прилад видає проекції магнітного поля на три осі у системі координат магнітометра.

За допомогою акселерометра можна визначити положення пристрою в просторі, а за допомогою магнітометра визначити напрямок пристрою.

Gps-модуль може також показувати направлення, але робить це він не досить точно. Для цієї задачі підійде сенсор триосьового акселерометра + магнітометра (компаса) – GY-511 LSM303DLHC (Рисунок 1.13). Цей компактний сенсор використовує I2C для зв'язку та дуже простий у використанні. Робоча температура даного модуля від -40 °C до +85 °C. LSM303DLHC має повну шкалу лінійного прискорення ±2g/±4g/±8g/±16g та повну шкалу магнітного поля ±1,3/±1,9/±2,5/±4,0/±4,7/±5, 6/±8,1 Гс.

Рис. 1.14 – GY-511 LSM303DLHC

Інший аналог HMC5883L здатний вимірювати напруженість магнітного поля вздовж усіх трьох осей. Працює на шині I2C. Для створення 3D-компаса відмінно підійде. Він зможе послужити сенсором орієнтації пристрою у просторі, пристрій пошуку магнітних аномалій. Напруга живлення – 3.3-5

вольт. Компас невеликий, його габарити складають 18x14x5 мм. В загальному сенсори LSM303DLHC та HMC5883L дуже схожі між собою.

1.4 Висновки до розділу 1

Наземні роботи-кур’єри зараз розповсюджуються по всьому світу. Багато країн створили власні аналоги даного пристрою. Головним завданням робота є знайти шлях до цілі, а також минати перешкоди. Для цього було обрано необхідні комплектуючі. Для управлінням робота використовуються мікроконтролер та мікропроцесор. Мікроконтролер потрібен для управління різними модулями та моторами, а процесор потрібно для обчислення маршруту. Для навігації робота були обрані спеціальні модулі. LiDAR система використовує лазер для пошуку відстані до об’єктів, а ультразвукові сенсори використовують високочастотні звукові імпульси. Дані сенсорів дозволяються виявляти перешкоди. Для пошуку місцезнаходження використовується gps-модуль. Усі ці компоненти дозволяють створити систему автономної навігації.

РОЗДІЛ 2
РОЗРОБКА АПАРАТНОЇ ЧАСТИНИ

2.1 Апаратне забезпечення для визначення позиції та орієнтації в просторі

2.1.1 GPS-модуль

GPS-модуль – це пристрій, що знаходить супутники і підключається до них та на основі отриманих сигналів розраховує поточне місце розташування пристрою в просторі. Інформацію про місце знаходження модуль надає у вигляді координат довготи та широти.

Супутникова система складається з 24 супутників, які знаходяться в шести орбітальних площинах с центром на Землі, в кожній з яких по 4 супутника. Супутники обертаються над Землею на висоті 20000 км і рухаються зі швидкістю 14000 км в час. GPS працює за допомогою методу, що називається трилатерацією. Трилатерація, що використовується для розрахунку розташування, швидкості та висоти, збирає сигнали з супутників для виведення інформації про місцезнаходження. Його часто помилково сприймають як тріангуляцію, яка використовується для вимірювання кутів, а не відстаней. Кожен супутник передає унікальний сигнал і параметри орбіти, які дозволяють приладам GPS декодувати та обчислювати поточне місцезнаходження супутника. Приймач GPS використовує цю інформацію для розрахунку точного місцезнаходження. По-суті, GPS вимірює відстань до кожного супутника по кількості часу, необхідного для прийому переданого сигналу. Щоб обчислити координати свого місцезнаходження, а також відстежувати рух, GPS-модуль повинен з'єднатися мінімум 3 трьома супутника.

Основним джерелом похибок визначення GPS координат є спотворення сигналу від супутника високими будівлями. В великих містах з високими

будівлями GPS-модулю складно піймати сигнал. Сигнал дістається до приймача вже відбити від поверхні споруд. Таким чином довжина шляху збільшується і разом з цим збільшується похибка вимірювань. Також модуль не зможе працювати в будинку, тому що не зможе отримати сигнал.

Розрізняють «холодний», «теплий» та «гарячий» старт GPS-приймача. При включенні GPS-пристрій отримує з супутників дані альманаху та ефімерідів, які зберігаються в його пам'яті для подальшої роботи. Альманах - загальні дані про параметри орбіти навігаційних супутників. Ефемеріди - дані, що уточнюють ці параметри для конкретного супутника у конкретний момент часу.

Якщо прилад не вмикався довгий час, то під час увімкнення буде виконуватися «холодний» старт. Це відбувається тому що дані альманаху і ефемерідів застарілі і сенсор завантажує їх знову. Цей процес не швидкий і може зайняти багато хвилин.

Якщо прилад був увімкнений більше чим через приблизно 30 хвилин після вимкнення, то відбувається «теплий» старт. Дані альманаху у цьому випадку ще є актуальними, однак дані ефемерідів стали застарілими. Тому в даному випадку сенсор завантажує дані ефемерідів на основі альманаху.

Прилад, який був увімкнений менше чим через 30 хвилин після вимкнення, запускається в режимі «гарячого» старту. Він завантажує з своєї пам’яті дані альманаху та ефемерідів, які ще не встигли втратити свою актуальність. При цьому пристрій запускається дуже швидко і миттєво починає працювати.

GPS-модуль NEO-6M GY-GPS6MV2 є добре працюючим комплектним GPS-приймачем з вбудованою керамічною антеною. Прилад має вбудовану резервну батарею, завдяки якій він може зберігати дані при випадковому вимкненню живлення. Модуль має світлодіод, який загорається і починає блимати коли вдалося знайти супутники і відбувається встановлення зв’язку з ними. Коли зв’язок встановлено світлодіод перестає блимати і просто

включений. Необхідне живлення від 3,3V до 5V. Принципова схема модуля наведена на рисунку 2.1.

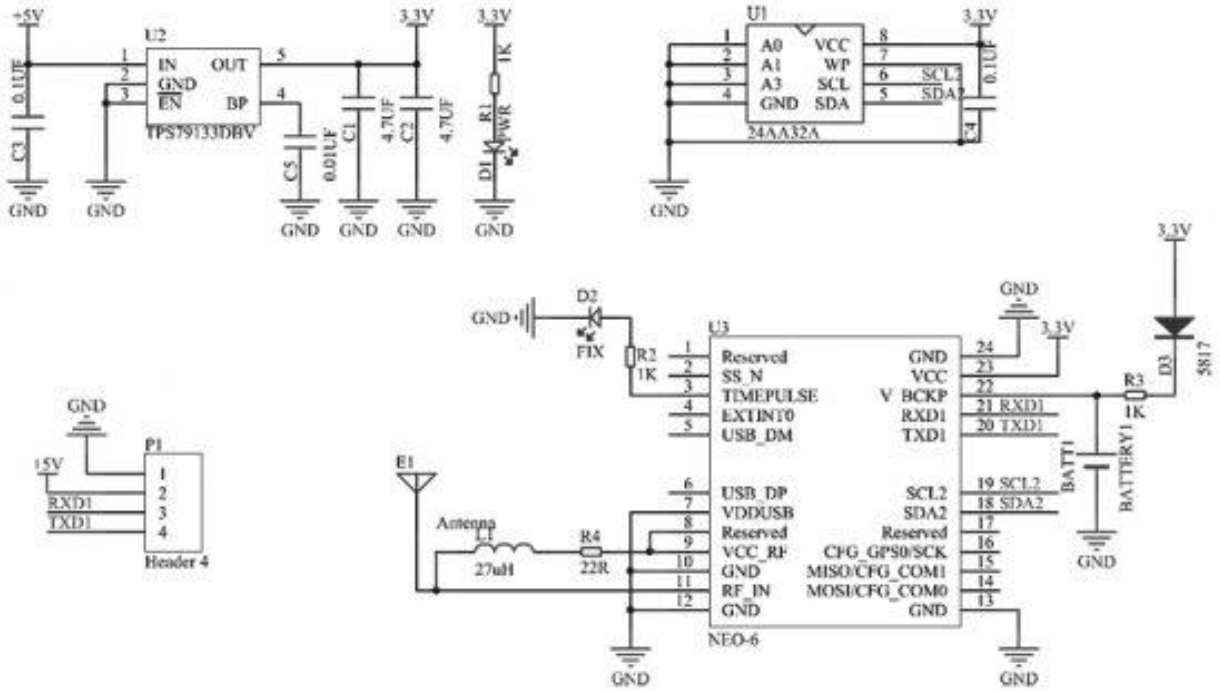

Рис. 2.1 – Принципова схема модуля GY-GPS6MV2

Модуль включає в себе один інтерфейс UART для послідовного зв'язку.

UART(Universal Asynchronous Receiver-Transmitter) – це фізичний пристрій прийому та передачі даних по двох дротах. Воно дозволяє двом пристроям обмінюватись даними на різних швидкостях. Оскільки інтерфейс асинхронний, то велике значення має швидкість передачі – і в приймача, і в передавача вона має бути однаковою. Швидкість вимірюється в бітах за секунду, або коротко – у бодах. Для GPS-модуля використовується 9600 бод. UART використовує два піна контролера - RX і TX, де перші літери позначають, відповідно, Receiver і Transmitter. Тому в модуля є піни RX і TX, а також піни VCC та GND для живлення. Для зв'язку двух приладів потрібно підключати TX одного приладу до RX другого приладу.

Було вирішено підключити Gps-модуль напряму до мікрокомп'ютера Raspberry Pi, а не до мікроконтролера. Таким чином мікрокомп'ютер буде швидко отримувати інформацію про місцезнаходження. Так як Raspberry Pi має 4 usb-порта, gps-модуль буде підключений до одного з портів за

допомогою адаптера USB to UART на базі чіпу CP2102, що дає змогу підключити ще будь-який пристрій з інтерфейсом UART до мікрокомп’ютера

2.1.2 Компас на базі модуля GY-511 LSM303DLHC5

Модуль GY-511 використовує мікросхему LSM303DLHC для виявлення та напряму магнітних полів. Протокол зв'язку цього модуля — I2C, і його можна підключити до різних процесорів, включаючи плату Arduino, використовуючи контакти SCL і SDA.

I2C — це протокол послідовного зв'язку, тому дані передаються біт за бітом по одному проводу (лінії SDA). I2C є синхронним, тому вихід бітів синхронізується з вибіркою бітів за допомогою тактового сигналу, що розділяється між Master та Slave. Тактовий сигнал завжди контролюється Master.

Модуль GY-511 має 8 контактів:

– VIN: модуль живлення від 3,3 до 5 вольт.

– 3V: На модулі є регулятор від 5 до 3,3 вольт. Цей контакт може використовуватися для живлення іншої частини схеми, для якої потрібно 3,3 вольта.

– GND: Земля.

– SCL: Clock Pin для зв'язку I2C

– SDA: вихід даних для зв'язку I2C

– INT2 : Переривання 2

– INT1: Переривання 1

– DRDY (DataReady): цей вихід активується, коли нові значення вимірюються сенсором і повертаються на вихід.

Принципова схема модуля наведена на рисунку 2.2.

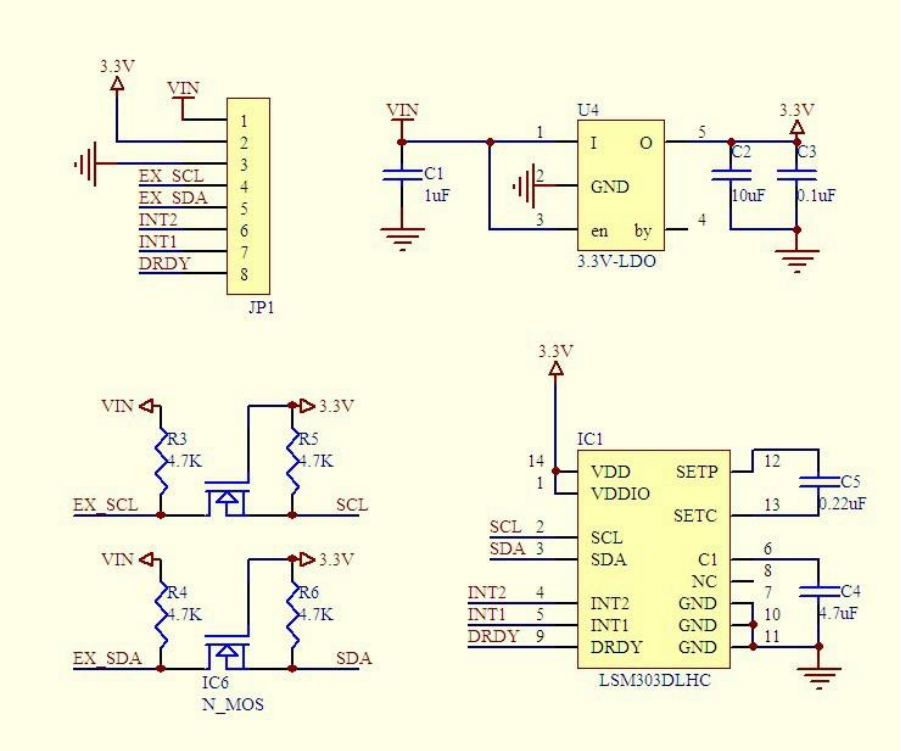

Рис. 2.2 – Принципова схема модуля GY-511

LSM303DLHC поєднує в собі цифровий 3-осьовий акселерометр та 3-осьовий магнітометр в одному корпусі, який ідеально підходить для створення компаса з компенсацією нахилу. LSM303DLHC включає I2C інтерфейс, який підтримує стандартний і швидкий режими 100 кГц і 400 кГц. Система може бути налаштована на генерацію сигналів переривання за інерцією події пробудження/вільного падіння, а також за позицією самого пристрою. Блоки магнітні та акселерометра можуть бути увімкнені або переведені в режим вимкнення живлення окремо.

Магнітометр – це сенсорна система для вимірювання магнітного поля.

Магнітний струм та поле безпосередньо пов'язані один з одним. Коли у дроті тече струм (електрони починають рухатися в одному напрямку), створюється магнітне поле. На цьому взаємозв'язку засновано основну ідею сенсорів компасу. Напрямок магнітного поля Землі впливає потік електронів у сенсору. Вимірюючи ці зміни струму, сенсор зможе визначати напрямки.

Коли цей модуль міститься в магнітне поле, у його мікроскопічній котушці за законом Лоренца індукується струм збудження. Модуль компаса

перетворює цей струм у диференціальну напругу для кожного напряму координат. Використовуючи цю напругу, можна розрахувати магнітне поле в кожному напрямку та отримати географічне положення.

Акселерометр – це пристрій для визначення проекції відносного прискорення, як різниці між реальним прискоренням об’єкта та гравітаційним прискоренням Землі. В загальному, вимірювальна частина сенсора являє собою чутливу масу, закріплену напівжорстким способом до корпуса. Напівжорсткість відбувається за рахунок використання пружини. Таким чином зміщення чутливої маси від положення її стану спокою показує величину прискорення.

Модуль підключається до мікроконтролера за допомогою пінів SCL та SDA.

Перед використанням модуля його потрібно відкалібрувати(якщо він не був калібрований до цього), а значить встановити діапазон виміру від 0 до 360 градусів. Для цього потрібно підключити пристрій до Arduino та виконати певний код. Під час виконання коду у вікні послідовного монітора можна побачити мінімальне та максимальне значення діапазону вимірювання для осей X, Y та Z. Для отримання максимально можливих значень потрібно повільно повністю повертати модуль по всім осям декілька разів. Після цього отриманні значення потрібно записати в код програми з компасом.

2.1.3 Ультразвуковий сенсор

Ультразвуковий сенсор – це пристрій, який вимірює за допомогою ультразвукових хвиль відстань до об’єкта. Ультразвуковий сенсор використовує перетворювач для відправлення та прийому ультразвукових імпульсів, що передають інформацію про близькість об'єкта. Високочастотні звукові хвилі відбиваються від кордонів, створюючи чіткі ехо-сигнали.

Щоб розрахувати відстань між сенсором та об'єктом, сенсор вимірює час, що проходить між випромінюванням звуку передавачем та його контактом з приймачем. Формула для цього розрахунку:

$$D = {}^{1}/_{2}\, T \times C \quad (1)$$

Де D – дистанція,

T – Час,

C – швидкість звуку(~ 343 метра в секунду)

Ультразвук – це акустична хвиля дуже високої частоти, яка недоступна людському слуху. Оскільки вважається, що чутний діапазон частот знаходиться між 16 Гц і 20 кГц, ультразвук зазвичай означає акустичні хвилі вище 20 кГц.

Для роботи ультразвукового сенсора необхідні дві частини: передавач та приймач. Частіше всього вони знаходяться дуже близько один до одного. Коли приймач розташований близько до передавача, звук буде розповсюджуватися по більш прямої лінії від передавача до виявленому об'єкту і назад до приймача. Це дає менші похибки в вимірюваннях.

З передавача виходять акустичні хвилі, які по формі схожі на світло з ліхтарика, чим з лазера, тому потрібно враховувати поширення та кут променя. Область виявлення буде збільшуватися в поперечному та вертикальному напрямках в залежності в того, як далеко звукові хвилі виходять з передавача.

Якщо промінь буде вузьким, то дальність виявлення буде більшою, так як енергія ультразвукового імпульса буде більш сфокусована і зможе пройти далі, перед тим як розсіється до непридатних для використання рівнів. А широкий промінь буде поширювати цю енергію по більш широкій дузі і тим самим дальність виявлення буде меншою.

Ультразвуковий далекомір HC-SR04 має такі технічні параметри:

- Напруга для живлення: 5В;
- Робочий параметр сили струму: 15 мА;
- Сила струму в пасивному стані < 2 мА;

– Оглядовий кут – 15 °;

– Вимірювальний кут - 30 °;

– Ширина імпульса – $\mathbf{10^{-6}}$ с.

На рисунку 2.3 наведено принципову схему ультразвукового далекоміра HC-SR04.

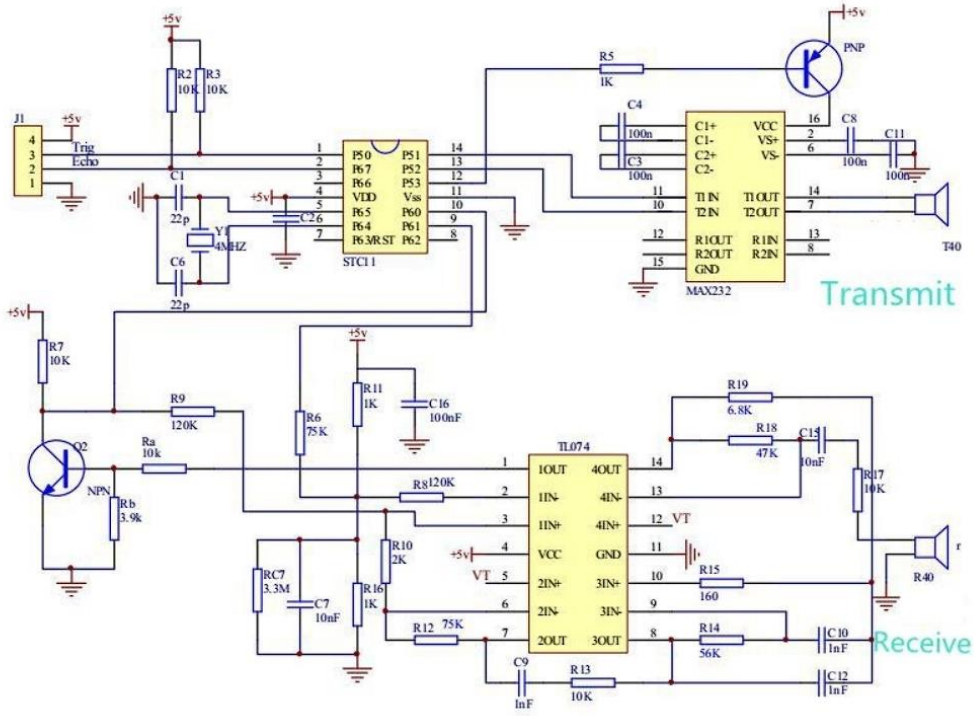

Рис. 2.3 – Принципова схема модуля ультразвукового далекоміра HC-SR04

Точність сенсора залежить від декількох факторів, таких як: температура та вологість повітря, відстань до об’єкту.

Сенсор має 4 піни: контакт живлення 5 В, Trig – сигнал входу, Echo – сигнал виходу, GND – заземлення.

Послідовність дій для отримання даних така:

– подаємо імпульс тривалістю 10 мкс на виведення Trig;

– всередині далекоміра вхідний імпульс перетворюється на 8 імпульсів частотою 40 кГц і посилається вперед через випромінювач T;

– коли надіслані імпульси доходять до перешкоди, вони відбиваються і приймаються приймачем R, в результаті отримуємо вихідний сигнал на виведенні Echo;

– отриманий сигнал переводиться в відстань контролером.

2.2 Апаратне забезпечення для руху

2.2.1 Принцип роботи драйвера BLD-300B та безщіткових двигунів

Безщітковий двигун – це електродвигун постійного струму(DC), який працює без механічних щитків і колектора традиційного щіткового двигуна. Це дозволяє уникнути певних технічних проблем та запобігти необхідності їх заміни Безщітковий двигун може називатися двигуном BLDC або BL.

У безщітковому двигуні постійного струму ротор, виконаний з постійного магніту, рухається магнітною силою ланцюга обмотки статора. У той час як щітковий двигун постійного струму використовує щітку та комутатор для перемикання струму, безщітковий двигун постійного струму використовує сенсор та електронну схему для перемикання струму.

Мотор-колесо від гіроскутера – це безщітковий трифазний двигун, який має три сенсора Холла, по яким двигун дізнається, в якому положенні в даний момент часу перебуває ротор і подає напругу на певні фази. Мотор-колеса підключаються до драйвера BLD-300B для управління ними.

BLD-300B це 3-фазний драйвер безщіткових двигунів постійного струму.

BLD-300B має такі технічні параметри:

– Напруга для живлення: 18-50 В;
– робочий параметр сили струму: до 10 А;
– потужність мотора: до 300 Вт;
– робоча температура: -40-85 °C;
– діапазон швидкостей: 0-3000 об/хв.;
– тип мотора: двигун постійного струму;
– Управління аналоговим сигналом 0..5 В.

Драйвер має 17 входів, з яких 8 з'єднуються з моторами. Значення входів наведено в таблиці 2.1.

Таблиця 2.1 – Входи драйвера BLD-300B та їх значення

Мітка	**Значення**
DC+/DC-	Вхід живлення постійного струму
U,V,W	Входи для підключення фаз мотора
Hu,Hv,Hw	Підключення сенсорів Холла
REF+	Живлення сенсора Холла +
REF-	Живлення сенсора Холла -
SV	ШИМ от 1-10КГц
COM	Загальний порт (опорний рівень 0 В)
F/R	Управління напрямком обертання
EN	Вмикання драйвера
BRK	Швидке гальмування
SPEED	Швидкість вхідного сигналу

На входи DC+/DC- подається живлення від акумулятора, так як для живлення двигунів потрібно багато енергії. Тому драйвер з моторами буде отримувати живлення окремо від інших компонентів всього пристрою.

Управління швидкістю виконується контролером через ШИМ-сигнал на вхід SV. Для різкого гальмування можна використовувати вхід BRK. Для регулювання направленням можна використовувати вхід F/R.

Для підключення моторів використовуються входи REF+, REF-, Hu, Hv, Hw, U, V, W. На ці входи можна підключити відразу декілька моторів. Однак вони будуть працювати однаково. Для того щоб регулювати швидкість та направлення між різними моторами, доведеться використовувати ще один такий драйвер. Схема підключення мотора наведена на рисунку 2.4.

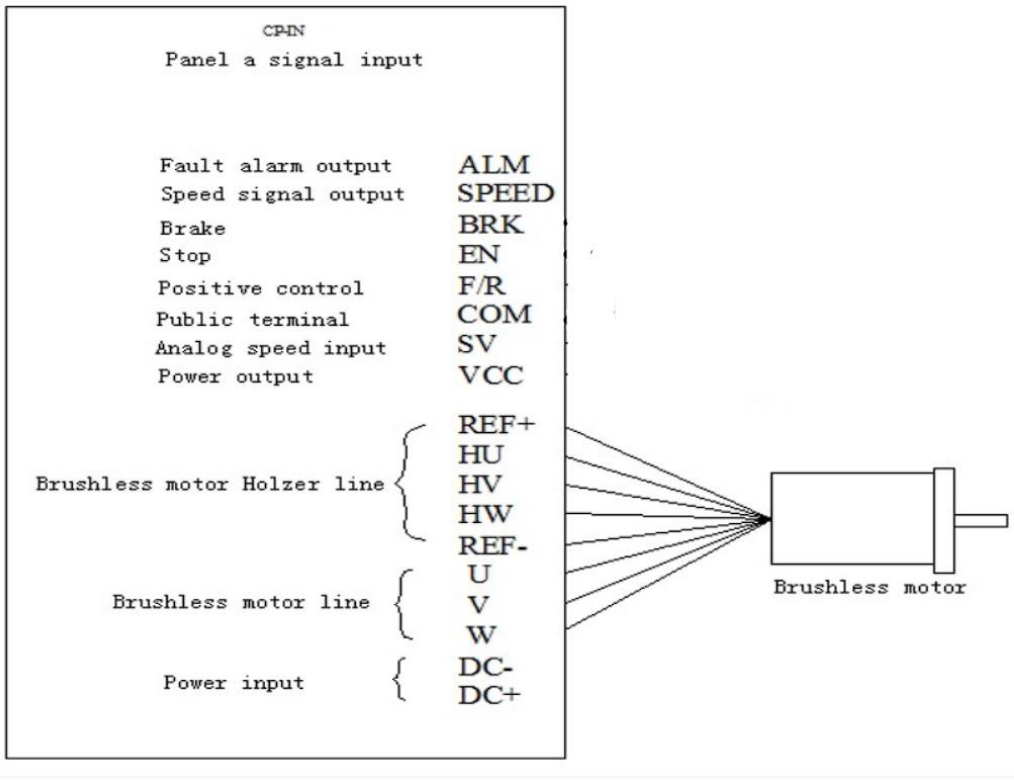

Рис. 2.4 – Схема підключення мотора до драйвера BLD-300B

Також драйвер має тумблер, який дозволяє вмикати або вимикати контроль швидкості.

Рис. 2.5 – Тумблер контроля швидкості.

При увімкненому контролі швидкості, драйвер буде регулювати швидкість мотора враховуючи опір. Наприклад, якщо при вимкненому контролі швидкості колесо буде крутитися на малих швидкостях, то його

можна легко зупинити рукою, а якшо контроль швидкості буде увімкнений, то колесо буде важко зупинити.

2.2.2 Принцип роботи драйвера l298n та моторів-редукторів

Мотор-редуктор є парою редуктора і електродвигуна змінного або постійного струму. Редуктор та двигуни об'єднані в один блок.

Мотор-редуктор забезпечує високий обертаючий момент при низькій потужності або низькій швидкості. В таких двигунах використовуються шестерні, які збираються в виді редуктора, для зниження швидкості, що надає великий обертаючий момент. Такі мотори частіше всього використовуються там, де необхідні великі зусилля для переміщення важких предметів.

Частіше всього мотор-редуктори використовують двигуни перемінного струму. Дані електромотори використовують багато струму. Два важливих параметри, які необхідно контролювати це направлення та швидкість обертання. Для цього підходять мікроконтролери, однак мікроконтролери не можуть забезпечити достатньо струму для запуска двигуна, і таким чином можем пошкодити мікроконтролер. Наприклад, контакти Arduino обмежені струмом 40 мА, а для моторів необхідний струм не менше ніж 100 мА. Для цього використовується драйвер l298n.

L298N – це модуль високовольтного подвійного мостового драйвера двигуна для управління двигуном постійного струму і кроковим двигуном. За допомогою нього можна контролювати швидкість, а також напрямок обертання двох двигунів постійного струму. Цей модуль складається з двоканального IC драйвера двигуна H-Bridge L298. Модуль дає змогу керувати швидкістю та напрямком обертання двигунів постійного струму за допомогою двох методів. Перший метод – це ШИМ, за допомогою нього можна регулювати швидкість. Другий метод – це H-Bridge, який дає змогу змінювати напрям обертання. Даний пристрій дозволяє керувати одночасно двома

двигунами постійного струму і змінювати напрямок і швидкість кожного двигуна окремо.

IC драйвер двигуна L298 являє собою високовольтну, сильноточну подвійну повномістну мікросхему драйвера двигуна. Він приймає стандартні логічні рівні TTL (логіка управління) та керує індуктивними навантаженнями, такими як реле, соленоїди, двигуни постійного струму та крокові двигуни. Це 15-контактна мікросхема. Робоча напруга даного модуля становить від 5 до 46 В, а максимальний струм, що протікає через кожен вихід, становить 3 А. Для включення або відключення пристрою незалежно від вхідних сигналів передбачені два входи включення.

Також до цієї мікросхеми прикріплений радіатор, який передає тепло, яке виділяється з електронного пристрою, поточному середовищу.

В модуль вбудований регулятор напруги 78M05 5V. Якщо встановлена перемичка 5V Enable, то регулятор напруги буде працювати. При цьому внутрішня схема буде отримувати живлення от регулятора напруги, якщо джерело живлення менше або рівно 12 В, а контакт 5В можна використовувати для живлення мікроконтролера. Якщо зняти перемичку, то живлення буде використовуватися тільки для моторів, а для живлення мікросхеми необхідно буде окремий сигнал на контакт 5В.

Значення контактів модуля наведено в таблиці 2.2.

Таблиця 2.2 – Входи драйвера L298n та їх значення

Мітка	**Значення**
5V	Контакт для живлення мікросхеми, якщо перемичка не встановлена. Якщо перемичка встановлена, то контакт використовується для живлення мікроконтролера або іншої схеми.
GND	Заземлення.
VCC	Вхід живлення. Напруга від 5 до 35В
ENA	Контакт для управління швидкістю двигуна А.

ENB	Контакт для управління швидкістю двигуна B.
IN1, IN2	Контакти для управління напрямком обертів двигуна A
IN3, IN4	Контакти для управління напрямком обертів двигуна B
OUT1, OUT2	Вихід для двигуна A
OUT3, OUT4	Вихід для двигуна B

На контакти ENA та ENB можна встановити перемичку, тоді швидкість двигунів A та B буде максимальною. Для управління напрямком на контакти IN1 та IN2 подати один високий та один низький сигнали відповідно, то двигун A почне крутитись в одному напрямку, а якщо подати сигнали навпаки, то двигун буде рухатись в іншому напрямку. Якщо на обидва входи подати високий або низький сигнал, то двигун зупиниться. Теж саме працює для входів IN3 та IN4, які відповідають за двигун B. Схема підключення двигунів наведена на рисунку 2.6.

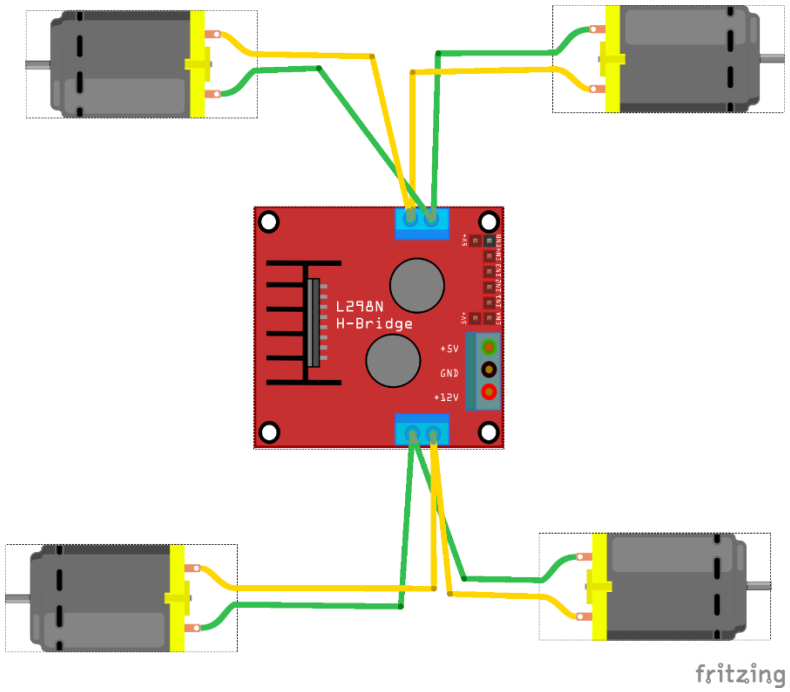

Рис. 2.6 – Схема підключення мотор-редукторів до модуля L298n

Модуль драйвера двигуна L298n використовує технологію H-Bridge для керування напрямком обертання двигуна постійного струму. У цьому методі Н-міст керував напрямком обертання двигуна постійного струму, змінюючи полярність його вхідної напруги.

Схема Н-моста містить чотири перемикаючі елементи, таких як транзистори (BJT або MOSFET), з двигуном у центрі, що утворює Н-подібну конфігурацію. Вхідні контакти IN1, IN2, IN3 та IN4 фактично управляють перемикачами схеми H-Bridge усередині L298N IC.

Ми можемо змінити напрямок струму, одночасно активувавши два певні перемикачі, таким чином, ми можемо змінити напрямок обертання двигуна.

Принципову схему модуля наведено на рисунку 2.7.

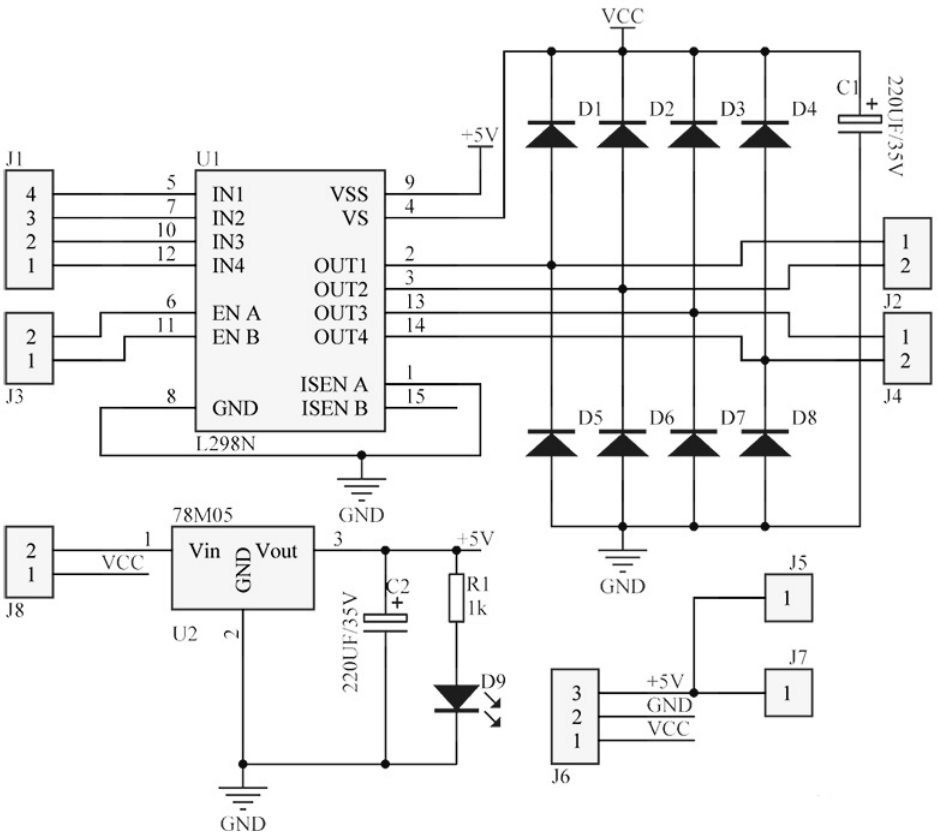

Рис. 2.7 – Принципова схема модуля L298n

2.3 Взаємодія всіх компонентів в системі

Головним елементом системи є мікрокомп'ютер Raspberry Pi 3 B, який являється «головним мозком» робота. Щоб ввімкнути весь пристрій,

достатньо ввімкнути лише мікрокомп’ютер, так як від нього починається ланцюгова реакція включення всіх інших компонентів. Зробити це можна під’єднавши джерело живлення до Raspberry Pi через мікро usb або підключивши до пінів 5V та Ground.

Для роботи Raspberry Pi необхідна спеціальна оперативна система на базі Linux, яка знаходиться на карті пам’яті ємністю 16 Гб. Завантажити ОС можна на офіційному сайті Raspberry і після цього встановити її на карту пам’яті. Карта пам’яті легко вставляється в Raspberry Pi і на ній буде зберігатися вся інформація.

Мікрокомп’ютер виконує багато обчислень і може нагріватися, особливо влітку при високій жарі. Для того щоб процесор мікрокомп’ютера не перегрівався, на нього встановлено кулер та радіатор охолодження. Таким чином температура процесора при активній роботі не перевищує 42 °C при температурі на вулиці близько 30 °C, що є дуже непоганим результатом. Максимальна температура при якій мікрокомп’ютер може працювати – 80 °C.

Після того як увімкнувся мікрокомп’ютер вмикаються під’єднані до нього gps-модуль GY-GPS6MV2 та мікроконтролер Arduino UNO. Arduino підключений до Raspberry Pi через usb, що допомагає їм обмінюватися інформацією через serial порт, а також дає живлення Arduino. Raspberry Pi та Arduino також можна з’єднати через UART інтерфейс, однак у Raspberry Pi логічний рівень 3.3V, а у Arduino 5V. Тому потрібно використовувати перетворювач логічних рівнів і виконувати підключення цих приладів між собою через нього.

І Arduino, і gps-модуль підключаються до Raspberry використовуючи UART інтерфейс. Однак Raspberry Pi має лише один такий інтерфейс. Тому для вирішення цієї проблеми можна підключити Arduino через usb і таким чином в Arduino залишається один вільний UART інтерфейс. Gps-модуль підключається до Raspberry Pi, а не до Arduino, тому що так мікрокомп’ютер зможе відразу отримувати без затримки актуальні дані про місцезнаходження.

Gps було вирішено також приєднати до мікрокомп’ютера через usb за допомогою адаптера USB to UART на базі чіпу CP2102, так як Raspberry Pi має 4 USB-порта. Внаслідок цього мікрокомп’ютер має 2 вільних USB-порта і один UART інтерфейс.

Після увімкнення Arduino вмикаються всі модулі, які приєднані до нього. Arduino потрібен для керування всіма модулями, і таким чином мікрокомп’ютер не навантажується додатковими процесами.

Ультразвуковий сенсор HC-SR04 підключається таким чином: Trig приєднується до 5 піна, Echo приєднується до 6 піна, а також використовує вільні входи живлення та заземлення Arduino. Модуль GY-511 LSM303DLHC підключається до аналогових входів: SCL до A5, SDA до A4. Модуль підключається саме до цих входів тому, що з ними працює бібліотека для модуля. Входи для лівого двигуна драйвера L298n IN1 та IN2 підключаються до пінів 8 та 7 відповідно, а IN3 та IN4 до пінів 3 та 4 відповідно. Всі перемички модуля зняті, тому ENA та ENB потрібно підключити до пінів 11 та 10 відповідно. Модуль l298n отримує живлення від акумулятора, але воно використовується для моторів. Для роботи схеми модуля потрібно підключити 5V та GND до Arduino. Хоть модуль і приєднаний до джерела живлення, він не буде працювати поки не увімкнеться мікроконтролер. Принципова схема всього пристрою наведена на рисунку 2.8.

Так як у Arduino використовуються майже всі піни можна використати макетну плату, однак вона може займати лишнє місце на платформі пристрою. Іншим способом для підключення додаткових модулів є sensor shield v5.0 для Arduino. Цей екран виводить стандартні контакти введення-виводу Arduino на роз'єм разом з виділеними контактами заземлення та живлення для кожного введення-виводу, щоб полегшити підключення сенсорів до інших пристроїв.

Даний екран підключається напряму до Arduino UNO. На його зворотній стороні є спеціальні контакти для прямого підключення. Цей екран займає

майже всі піни Arduino UNO, але дає ще більше контактів, що дозволяє підключати багато приладів.

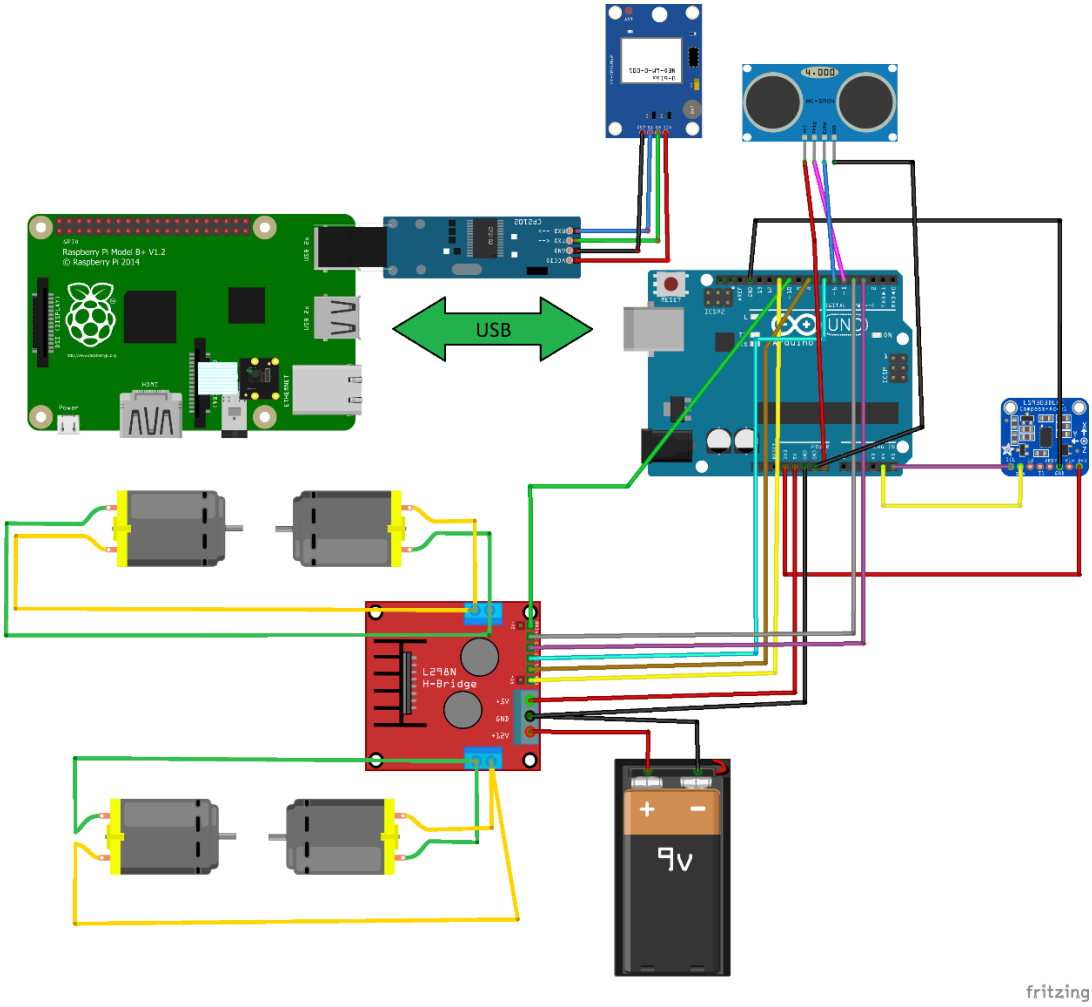

Рис. 2.8 – Принципова схема пристрою

Після того як вся система запустилася Raspberry Pi з’єднується з інтернетом та запускає скрипт виконання програмної частини. Спочатку мікрокомп’ютер намагається підключитися до сервера, який надає інформацію про місце призначення та буде виконувати додаткові обрахунки. В цей час gps-модуль виконує пошук супутників. Виконання решти програмного забезпечення почнеться тільки після того як буде встановлений зв’язок з сервером та супутниками. Всі інші модулі уже працюють. Мікроконтролер весь час отримує інформацію з модулів та передає її через serial порт мікрокомп’ютеру. Мікрокомп’ютер, в свою чергу, постійно отримує ще інформацію з gps-модуля та камери, яка приєднана до нього. Кадри з камери

передаються серверу для виявлення об’єктів, які можуть бути перешкодами. Сервер виявляє об’єкти та передає комп’ютеру інформацію про них. Raspberry Pi має не достатньо потужний процесор для таких обчислень кадру, тому цю задачу виконує сервер. Використовуючи всі ці дані, мікрокомп’ютер обчислює швидкість роботи моторів і передає її мікроконтролеру. Таким чином виконуються прокладання маршруту, розпізнавання та обминання перешкод. Принцип роботи системи наведено на рисунку 2.9.

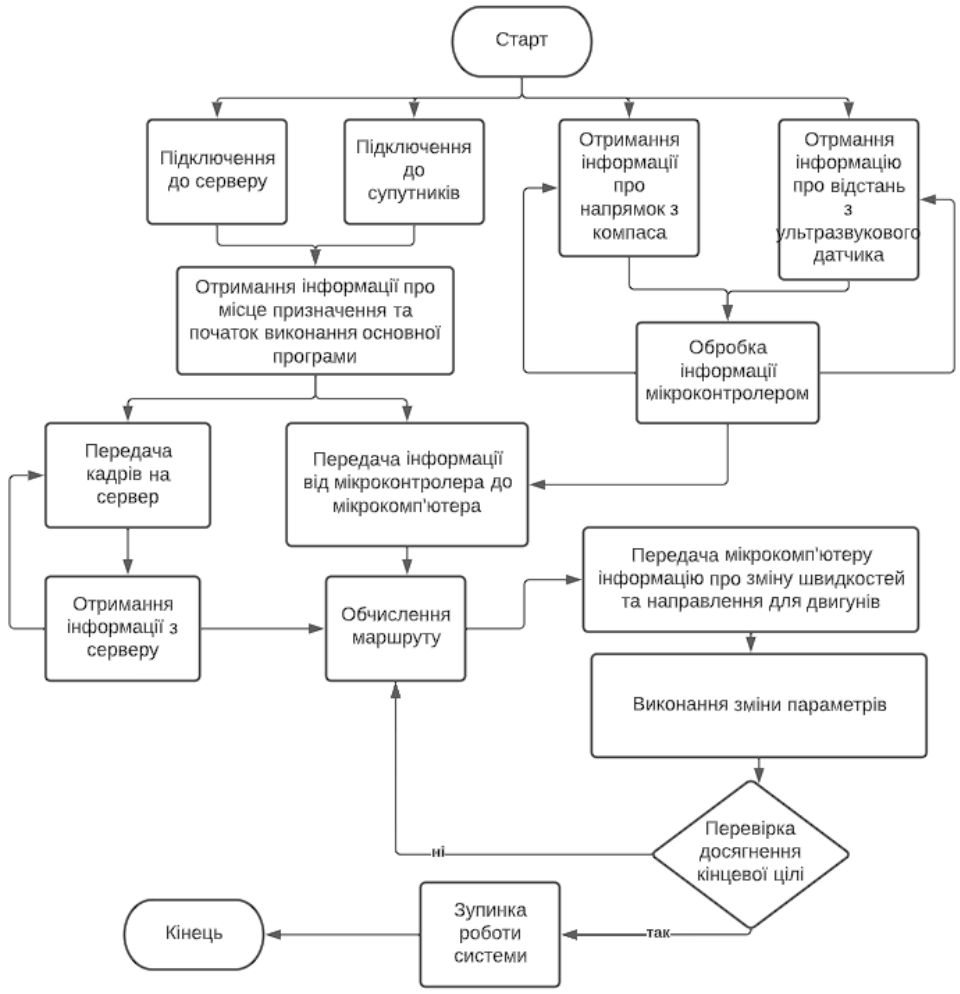

Рис. 2.9 – Принцип роботи системи

2.4 Розрахунок системи живлення

Для забезпечення функціонування роботам необхідне живлення – більшість роботів використовують для цього електрику. Основними джерелами електроенергії для роботи є акумулятори. Тип батареї, яка використовується для робота, залежить від безпеки, життєвого циклу та ваги.

Для правильної та максимально ефективної роботи всіх елементів системи необхідно багато енергії. Аби кожен елемент працював на повну потужність, потрібно розрахувати систему живлення.

Також слід враховувати, що робот долатиме кілометри або десятки кілометрів. Для подолання таких відстаней знадобиться багато часу. Тому робот повинен бути забезпечений як можна більшою кількістю енергії для того щоб він не вимкнувся по середині шляху.

Потрібно вирахувати кількість споживаного струму в годину. Для цього потрібно виявити скільки мА/год споживає кожен елемент системи, а потім додати їх. Споживання приладами мА:

– Raspberry Pi 3 B при високому навантажені максимально може споживати 2500-3000мА;

– Arduino UNO: 40мА;

– Драйвер Lsm298n: 36мА. Робочий струм драйверів A та B від 2000мА до 3000мА;

– Кожен Мотор-редуктор(4 шт.): від 200мА до 1000мА;

– Ультразвуковий сенсор HC-SR04: 15мА;

– Gps-модуль GY-GPS6MV2: 20мА;

– Компас GY-511 LSM303DLHC: 20мА;

$$\mathbf{3000 + 40 + 36 + (1000 * 4) + 15 + 20 + 20 = 7131}\ \text{мА}$$

Як можна побачити, найбільше струму споживають мікрокомп’ютер та двигуни. Оскільки мікрокомп’ютер та двигуни мають різну напругу, можна використовувати стабілізатор напруги XL4005.

Для роботи робота знадобиться мінімум 12 годин, а для непередбачених ситуацій необхідний запас 24 години. Тепер можна розрахувати ємність необхідного акумулятора $7131 \times 24 = 171144\ \text{mAh}$. Для роботи системи потрібен акумулятор що найменше 180 Ah. Однак це були розрахунки з не дуже потужними мотор-редукторами.

Якщо брати колеса-мотори від гіроскутера та драйвер BLD-300B. Драйвер BLD-300B(2 шт.): максимальний струм 15A(Скільки споживає струму сама мікросхема не уточнюється). Оскільки колеса-мотори споживають по 2-4 А, то їх кількість може бути до 3 штук на кожний драйвер. Але якщо взяти 4 колеса-мотора по 2А та два драйвера, мікросхеми яких будуть споживати приблизно по 1А, то вийдуть такі розрахунки:

$$\mathbf{3000 + 40 + (4 * 2000) + (1000 \times 2) + 15 + 20 + 20 = 13095}\ \text{мА}$$

Для драйвера BLD-300B з мотор колесами знадобиться акумулятор побільше, тому що тепер система споживає значно більше струму. Ємність необхідного акумулятора $13095 \times 24 = 314280$ mAh. Для цього знадобиться акумулятор що найменше 320 Ah.

2.5 Висновок до розділу 2

У розділі 2 визначено які модулі необхідні для системи з автономною навігацією. Було визначено як працює кожен модуль окремо і як він взаємодіє зі всією системою.

Визначено принцип роботи драйверів та двигунів. Було розглянути дві моделі драйверів та дві моделі двигунів. Вони відрізняються потужністю та ціною. Тому один драйвер використовується для дослідження та розробки, а другий уже може використовуватися для перевезення вантажу.

Було розроблено схему пристрою зі всіма підключеними сенсорами, контролером та мікрокомп'ютером. Був розроблений принцип роботи всієї системи з автономною навігацією.

Для довгої роботи системи з автономною навігацією було розраховано систему живлення, а саме ємність необхідного акумулятора. Оскільки найбільше енергії споживають двигуни, необхідний акумулятор ємністю не менше 180 Ah.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОЇ ЧАСТИНИ

3.1 Середовище розробки

Оскільки в даному проекті використовується плата Arduino UNO, для даного проекту обрано середовище розробки Arduino IDE.

Плата Arduino підключається до ПК через USB, де вона підключається до середовища розробки Arduino IDE.

Arduino IDE — це програмне забезпечення для користувачів, яке дозволяє їм писати власні програми (ескізи) для платформи Arduino. Ця платформа в першу чергу орієнтована на дизайнерів-любителів, які використовують Arduino для створення простих систем автоматизації та робототехніки.

Arduino IDE високо цінується користувачами за простоту використання. Він може виконувати складні процеси, мінімізуючи ресурси комп'ютера. Цей інструмент полегшує користувачам доступ до бібліотек. У той же час він пропонує оновлену підтримку останніх плат Arduino, які можуть допомогти користувачам створювати ескізи за допомогою останньої версії IDE.

Він доступний для всіх операційних систем, тобто MAC, Windows, Linux, і працює на платформі Java, яка поставляється з вбудованими функціями та командами, які відіграють важливу роль у налагодженні, редагуванні та компіляції коду.

Основні переваги Arduino IDE можна побачити в його здатності функціонувати як локальна програма та як онлайн-редактор, прямому створенні ескізу, параметрах модулів плати та інтегрованих бібліотеках. Зокрема, ось переваги, які користувачі можуть очікувати від системи:

- Параметри модуля дошки – інструмент оснащений модулем керування платою, де користувачі можуть вибрати, яку плату вони хочуть використовувати. Якщо потрібна інша дошка, вони можуть легко вибрати інший варіант зі спадного меню. Дані порту

оновлюються автоматично, коли в платі вносяться зміни або коли вибирається нова плата.

- Документація - цей інструмент дозволяє користувачам документувати свої проекти. Ця функція дозволяє їм відстежувати свій прогрес і знати будь-які внесені зміни. Крім того, документація дозволяє іншим програмістам використовувати ескізи на власних дошках.
- Спільний доступ до ескізів — Arduino IDE дозволяє користувачам ділитися своїми ескізами з іншими програмістами. Кожен ескіз має власне онлайн-посилання, яким користувачі можуть поділитися зі своїми колегами чи друзями. Ця функція доступна лише в хмарній версії.
- Інтегровані бібліотеки - програма має сотні вбудованих бібліотек. Ці бібліотеки були створені спільнотою Arduino і відкрито доступні для них. Користувачі можуть використовувати це для своїх проектів без використання сторонніх установок.
- Хоча сам інструмент розроблений для плат Arduino, він також має вбудовану підтримку підключення до стороннього обладнання. Це забезпечує широке використання Arduino IDE без прив'язки до пропрієтарних плат.

Мова програмування Arduino є стандартною мовою C++ (з використанням компілятора AVR-GCC) з деякими функціями, які полегшують новим користувачам писати програми в цьому бізнесі.

C++ — це багатоцільова мова програмування, яка широко використовується в усьому світі. Безсумнівно, незважаючи на те, що це дуже стара мова, вона є однією з найефективніших мов програмування.

Оскільки C++ — це мова програмування на основі компілятора; нам не потрібно встановлювати спеціальне середовище виконання під час роботи

програми. Тому вони попередньо інтерпретуються, що робить код швидшим і потужнішим.

Навіть компіляція та виконання швидші, що дозволяє створювати кілька типів програм.

C++ забезпечує хороший набір вбудованих бібліотек. Вони допомагають прискорити розробку програмного забезпечення та дозволяють користувачеві робити більше за меншу вартість.

В проекті також використовується мікрокомп'ютер Raspberry Pi. Для роботи на мікрокомп'ютер Raspberry Pi потрібно встановити ОС на базі Linux. Її можна безкоштовно завантажити на офіційному сайті www.raspberrypi.com. Встановлення ОС Raspberry Pi виконується дуже легко і інтуїтивно зрозуміло.

Основною мовою програмування для Raspberry Pi є Python. Хоча ми можна використовувати інші мови програмування.

Python – високорівнева мова програмування загального призначення, орієнтований на підвищення продуктивності розробника та читання коду. Синтаксис ядра Python є мінімалістичним. У той же час стандартна бібліотека даної мови програмування включає великий обсяг корисних функцій.

Переваги:

- Активно підтримується та постійно розвивається;
- Наявність великої стандартної бібліотеки;
- Наявність великої спільноти розробників;
- Використовується для розробки систем на базі штучного інтелекту, тому має широкі можливості по обробці даних;
- Висока швидкість розробки програм;
- Можливість використання елементів функціонально програмування;
- Наявність інтерактивного режиму;
- Наявність великої бази офіційної документації.

Недоліки:

- Відносно повільна швидкодія;
- Складна та не завжди надійна реалізація багатопоточності.

Основним середовищем розробки для Raspberry Pi є Thonny IDE або Geany.

Основні особливості Thonny:

- Він автозаповнює код.
- Він перевіряє код, щоб забезпечити відповідність скобок і виділити помилки.
- З ним легко почати, так як його установник також встановлює Python 3.7.
- Його відладчик простий у використанні, оскільки не вимагає знань точок зупинки.
- Він дозволяє користувачам перейти до функції виклику, надати зведення про локальні змінні та відображати код вказівника.
- Він має простий інтерфейс для встановлення пакетів. Це робить його дуже підходящим для початківців.

Переваги Thonny:

- Він має простий у використанні інтерфейс користувача.
- Інтерфейс користувача не містить жодних відволікаючих факторів для новачків.

Недоліки:

- Він пропонує базову функціональність на відміну інших просунутих IDE (наприклад, PyCharm).

Користувачі можуть зіткнутися з деякими проблемами, для яких недоступне швидке вирішення.

3.2 Програмна частина Arduino

Програмна частина для Arduino досить проста, оскільки підключено не дуже багато модулів. А бібліотеки знадобляться лише для компаса. Для

калібрування компаса потрібно завантажити бібліотеку LSM303. Також потрібна бібліотека Wire. Для початку потрібно підключити ці бібліотеки.

```
#include <Wire.h>
#include <LSM303.h>
```

Далі потрібно створити компас, масив для даних та вектор калібрувальних даних:

```
LSM303 compass;
LSM303::vector<int16_t> running_min = {32767, 32767, 32767}, running_max = {-32768, -32768, -32768};
char report[80];
```

Ініціалізація компасу:

```
void setup() {
  Serial.begin(9600);
  Wire.begin();
  compass.init();
  compass.enableDefault();
}
```

Далі в циклі відбувається отримання з модуля максимальних та мінімальних значень за трьома осями та виведення їх на монітор порта:

```
void loop() {
  compass.read();
  running_min.x = min(running_min.x, compass.m.x);
  running_min.y = min(running_min.y, compass.m.y);
  running_min.z = min(running_min.z, compass.m.z);
  running_max.x = max(running_max.x, compass.m.x);
  running_max.y = max(running_max.y, compass.m.y);
  running_max.z = max(running_max.z, compass.m.z);
  snprintf(report, sizeof(report), "min: {%+6d, %+6d, %+6d}    max: {%+6d, %+6d, %+6d}",
    running_min.x, running_min.y, running_min.z,
    running_max.x, running_max.y, running_max.z);
  Serial.println(report);
  delay(100);
}
```

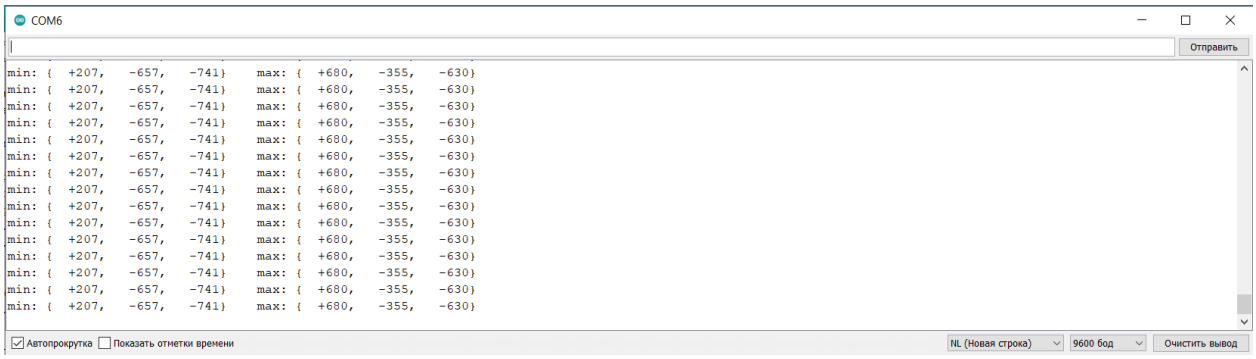

Рис. 3.1 – Дані модуля для калібрування

Модуль потрібно повільно обертати по кожній осі декілька разів щоб він зафіксував максимальні та мінімальні значення. Після цього ці значення потрібно записати для подальшого використання модулю. Після калібрування можна використовувати модуль в проекті.

Далі потрібно створити основний програмний застосунок, за допомогою якого будуть зчитуватися дані з сенсорів та будуть керуватися мотори.

Для початку треба додати попередні дві бібліотеки, а також створити змінні:

```
LSM303 compass;
const int LEFT_MOTOR_IN = 11;
const int RIGHT_MOTOR_IN = 10;
const int LEFT_MOTOR1 = 8;
const int LEFT_MOTOR2 = 7;
const int RIGHT_MOTOR1 = 4;
const int RIGHT_MOTOR2 = 3;
String one = "0  ";
String two = "0  ";
String r1 = "1";
String r2 = "2";
int distance = 0;
```

Змінні LEFT_MOTOR1, LEFT_MOTOR2 та RIGHT_MOTOR1, RIGHT_MOTOR2 відповідають за напрямок лівого та правого моторів відповідно. LEFT_MOTOR_IN та RIGHT_MOTOR_IN відповідають за швидкість лівого та правого двигунів відповідно. В зміну distance записується дистанція, яка отримується з ультразвукового сенсора. В змінні one, two, r1, r2

будуть записуватися дані швидкості та напрямку, які отримуються з мікрокомп’ютера. Також створюється компас.

Далі необхідно створити функцію роботи ультразвукового сенсора. В цій функції trigger посилає сигнал і через 10 мікросекунд echo отримує вихідний сигнал. Виглядає ця функція наступним чином:

```
long readUltrasonicDistance(int triggerPin, int echoPin)
{
  pinMode(triggerPin, OUTPUT);  // Clear the trigger
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  pinMode(echoPin, INPUT);
  return pulseIn(echoPin, HIGH);
}
```

Також потрібно створити функції для руху моторів вперед да назад:

```
void left_motor_foward(){
  digitalWrite(LEFT_MOTOR1, LOW);
  digitalWrite(LEFT_MOTOR2, HIGH);
}
void left_motor_back(){
  digitalWrite(LEFT_MOTOR1, HIGH);
  digitalWrite(LEFT_MOTOR2, LOW);
}
void right_motor_foward(){
  digitalWrite(RIGHT_MOTOR1, HIGH);
  digitalWrite(RIGHT_MOTOR2, LOW);
}
void right_motor_back(){
  digitalWrite(RIGHT_MOTOR1, LOW);
  digitalWrite(RIGHT_MOTOR2, HIGH);
}
```

Далі ініціалізуємо мотори та інші модулі в setup, а також створюється вектор з даними які були отримані під час калібрування:

```
Wire.begin();
compass.init();
compass.enableDefault();
pinMode(LEFT_MOTOR1, OUTPUT);
pinMode(LEFT_MOTOR2, OUTPUT);
pinMode(RIGHT_MOTOR1, OUTPUT);
pinMode(RIGHT_MOTOR2, OUTPUT);
pinMode(LEFT_MOTOR_IN, OUTPUT);
pinMode(RIGHT_MOTOR_IN, OUTPUT);
left_motor_foward();
right_motor_foward();
compass.m_min = (LSM303::vector<int16_t>){-264, -786, -682};
compass.m_max = (LSM303::vector<int16_t>){+785, +392, +463};
```

Отримання азимуту направлення з компаса:

```
compass.read();
  float heading = compass.heading() + 15;
  if(heading > 360) {
    heading = heading - 360;
  }
```

Далі відбувається перевірка на наявність даних в serial порту, які передаються обчислювачем. Якщо вони є, то швидкість моторів записується в змінні one та two, а напрямок в r1 та r2. Швидкість може бути від 0 до 255. Направлення має два значення: 1 – вперед, 0 – назад. Після чого значення швидкості застосовуються для моторів, а для направлення виконується відповідною функцією. Наступний код виконує ці дії:

```
if( Serial.available() >0 ) {
    String data = Serial.readStringUntil('\n');
    one = data.substring(0,3);
    two = data.substring(3,6);
    r1 = data.substring(6,7);
    r2 = data.substring(7,8);
    if (r1.toInt() == 1){
      left_motor_foward();
    }else{
      left_motor_back();
```

```
    }
    if (r2.toInt() == 1){
      right_motor_foward();
    }else{
      right_motor_back();
    }
    analogWrite(LEFT_MOTOR_IN, one.toInt());
    analogWrite(RIGHT_MOTOR_IN, two.toInt());
  }
```

Потім виконуємо розрахунок дистанції та передаємо все в serial порт:

```
distance = 0.01723 * readUltrasonicDistance(5, 6);
Serial.println(String(distance) + "|"+ String(heading,2));
```

3.3 Навчання моделі нейронної мережі

Для розпізнавання об'єктів також використовується камера. Однак для розпізнавання потрібно створити нейромережу.

Нейронна мережа є мережею математичних рівнянь. Вона приймає одну або кілька вхідних змінних і, проходячи через мережу рівнянь, отримує одну або декілька вихідних змінних. Також можна сказати, що нейронна мережа приймає вектор вхідних даних і повертає вектор вихідних даних.

У нейронній мережі є вхідний шар, один або кілька прихованих шарів та вихідний шар. Вхідний шар складається з однієї або декількох однакових змінних (або вхідних змінних або незалежних змінних), позначених як x1, x2, …, xn. Прихований шар складається з одного або кількох прихованих вузлів чи прихованих блоків. Вузол - це просто один з кіл на рисунку 3.2. Так само вихідна змінна складається з однієї або декількох одиниць виведення.

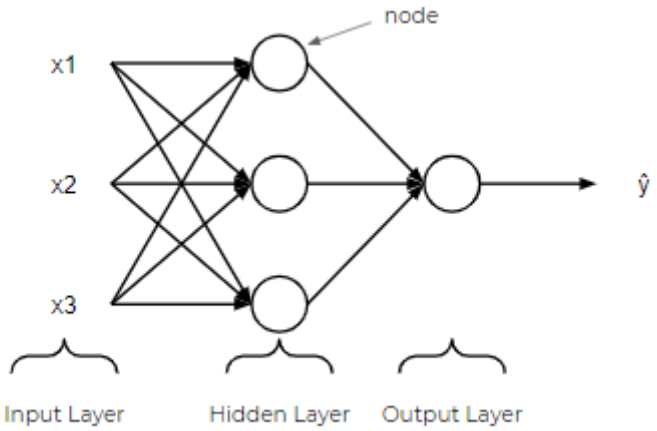

Рис. 3.2 – Схема роботи нейромережі

Шар може мати багато вузлів. Крім того, нейронна мережа може мати багато шарів. Як правило, більша кількість вузлів та шарів дозволяє нейронній мережі виконувати набагато складніші обчислення.

Вхідні функції (x) передаються в лінійну функцію кожного вузла, у результаті виходить значення z. Потім значення z передається у функцію активації, яка визначає, включається вимикач світла чи ні (між 0 та 1). Далі значення передаються в кожен вузол наступного шару і так відбувається поки не дійде до вихідного шару.

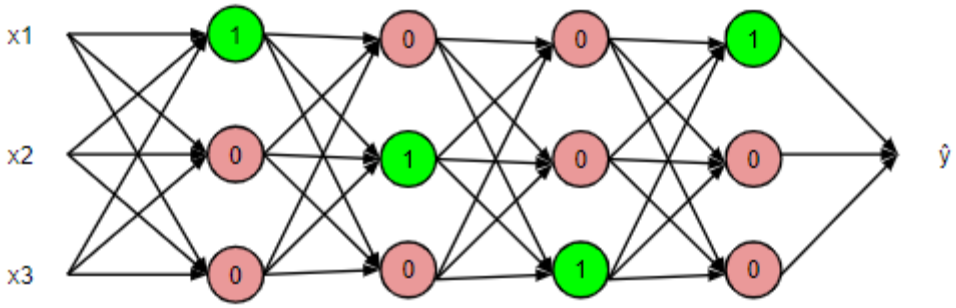

Рис. 3.3 – Принцип роботи нейромережі

Таким чином, кожен вузол зрештою визначає, які вузли на наступному рівні будуть активовані, доки не досягне виходу. Концептуально у цьому суть нейронної мережі.

Розпізнавання об'єктів — це загальний термін опису набору пов'язаних завдань комп'ютерного зору, які включають ідентифікацію об'єктів на цифрових фотографіях.

Класифікація зображень включає передбачення класу одного об'єкта на зображенні. Локалізація об'єкта відноситься до визначення розташування одного або декількох об'єктів на зображенні та малюванні великого прямокутника навколо їх меж. Виявлення об'єктів поєднує ці дві задачі та локалізує та класифікує один або кілька об'єктів на зображенні.

Одним з ключових компонентів більшості додатків комп'ютерного зору, заснованих на глибокому навчанні, є нейронна мережа (CNN). Винайдені в 1980-х роках піонером глибокого навчання Яном Лекуном, CNN є типом нейронної мережі, яка ефективно фіксує закономірності в багатовимірних просторах. Це робить CNN особливо зручним для зображень, хоча вони також використовуються для обробки інших типів даних.

Кожна згорткова нейронна мережа складається з одного або кількох згорткових шарів - програмного компонента, що здобуває значущі значення з вхідного зображення. І кожен шар згортки складається з кількох фільтрів, квадратних матриць, які ковзають зображення і реєструють зважену суму значень пікселів у різних місцях. Кожен фільтр має різні значення та витягує різні функції з вхідного зображення. На виході згортковий шар є набором «карт об'єктів».

При накладенні один на одного згорткові шари можуть виявляти ієрархію візуальних патернів. Наприклад, нижні шари будуть створювати карти об'єктів для вертикальних та горизонтальних країв, кутів та інших простих шаблонів. Наступні шари можуть виявляти складніші візерунки, такі як сітки та круги. У міру просування вглиб мережі шари виявлятимуть складні об'єкти, такі як автомобілі, будинки, дерева та люди.

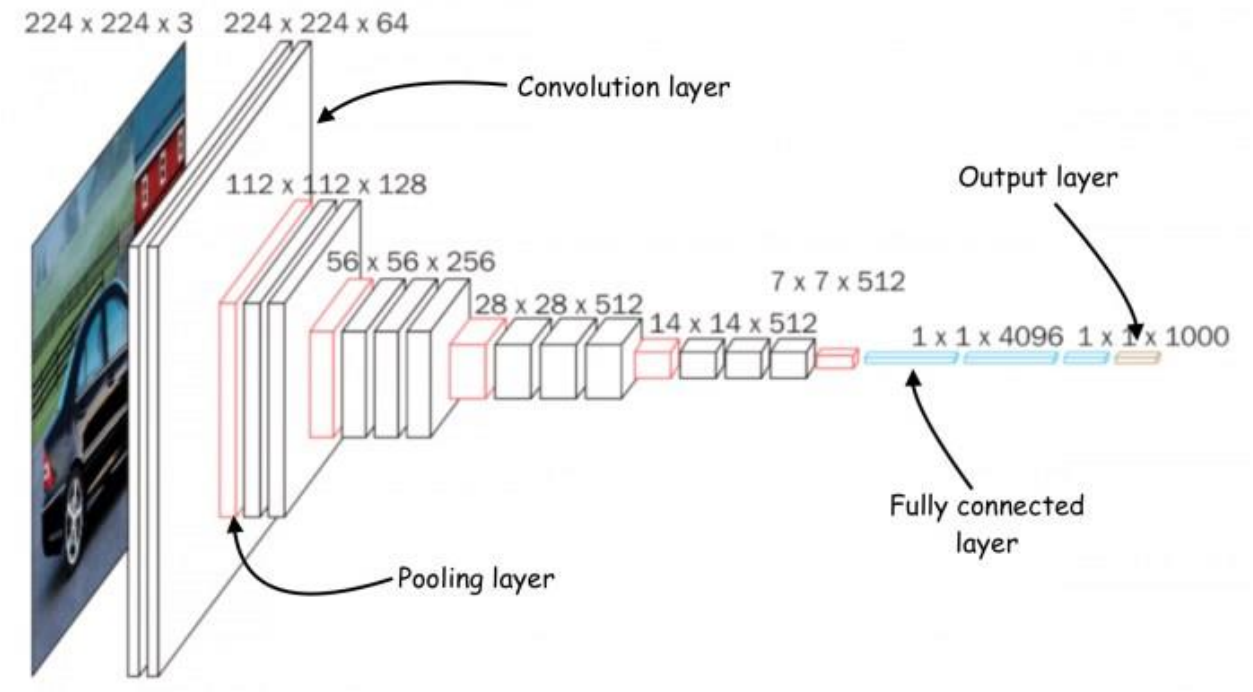

Рис. 3.4 – Архітектура згорткової нейронної мережі (CNN)

Сьогодні існує багато алгоритмів комп’ютерного зору.

Згорткова нейронна мережа на основі регіонів (R-CNN) була запропонована дослідниками II з Каліфорнійського університету в Берклі в 2014 році. R-CNN складається з трьох ключових компонентів.

По-перше, селектор областей використовує вибірковий пошук, алгоритм, який знаходить області пікселів на зображенні, які можуть представляти об'єкти, також звані областями інтересу (RoI). Селектор регіонів генерує близько 2000 областей інтересу кожного зображення.

Потім RoI деформуються до заданого розміру і передаються в нейронну мережу. CNN обробляє кожен регіон окремо, витягуючи ознаки за допомогою серії операцій згортки. CNN використовує пов'язані шари для кодування карт об'єктів в одновимірний вектор числових значень.

Нарешті, модель машинного навчання класифікатора зіставляє закодовані функції, отримані CNN, з вихідними класами. Класифікатор має окремий вихідний клас для фону, який відповідає всьому, що не є об'єктом.

Fast R-CNN – нова архітектура, яка вирішила деякі проблеми свого попередника. Fast R-CNN поєднує витяг ознак і вибір області в єдину модель машинного навчання.

Fast R-CNN отримує зображення та набір RoI і повертає список рамок, що обмежують, і класів об'єктів, виявлених на зображенні.

Одним з ключових нововведень у Fast R-CNN був «рівень об'єднання областей інтересу», операція, яка бере карти характеристик CNN та області інтересу для зображення та надає відповідні функції для кожної області. Це дозволило Fast R-CNN отримувати ознаки для всіх областей зображення за один прохід, на відміну від R-CNN, який обробляв кожну область окремо. Це призвело до значного збільшення швидкості.

You Only Look Once (YOLO) - сімейство нейронних мереж, які підвищили швидкість і точність виявлення об'єктів за допомогою глибокого навчання.

Основним поліпшенням YOLO є інтеграція всього процесу виявлення та класифікації об'єктів у єдину мережу. Замість того, щоб вилучати функції та регіони окремо, YOLO виконує все за один прохід через єдину мережу, звідси і назва «Ви тільки подивіться один раз».

YOLO може виконувати виявлення об'єктів з частотою кадрів потокового відео та підходить для додатків, що потребують логічного виводу в реальному часі.

За останні кілька років виявлення об'єктів глибокого навчання пройшло довгий шлях, перетворившись з ковдри з різних компонентів на єдину нейронну мережу, яка працює ефективно. Сьогодні багато програм використовують мережі виявлення об'єктів як один зі своїх основних компонентів. Через ряд переваг було використано саме цю архітектуру.

Для навчання нейронної мережі уже існує багато бібліотек для Python, такі як Keras та TenserFlow. Keras - це бібліотека для мови програмування Python, призначена для глибокого машинного навчання. Вона дозволяє

швидше створювати та налаштовувати моделі — схеми, якими поширюється і підраховується інформація під час навчання. TensorFlow — відкрита програмна бібліотека для машинного навчання, розроблена Google для вирішення завдань побудови та тренування нейронної мережі з метою автоматичного знаходження та класифікації образів, досягаючи якості людського сприйняття.

Для навчання нейронної мережі використовуватиметься архітектура YOLOv4. YOLOv4 – це модель виявлення об'єктів у реальному часі, опублікована у квітні 2020 року, яка досягла найсучаснішої продуктивності у наборі даних COCO. Він працює, розбиваючи завдання виявлення об'єкта на частини: регресію визначення позиціонування об'єкта з допомогою обмежувальних рамок і класифікацію визначення класу об'єкта. Ця реалізація YoloV4 використовує платформу Darknet.

Однак навчати нейромережу в домашніх умовах проблематично. Для навчання знадобиться потужний процесор або графічний процесор. В залежності від потужності процесора залежить скільки часу знадобиться для навчання нейромережі.

Виконати навчання нейромережі можна в Colab. Colab - це безкоштовне інтерактивне хмарне середовище для роботи з кодом від Google. Принцип у неї такий самий, як у решти онлайн-офісів компанії: вона дозволяє одночасно з колегами працювати з даними. Також цей сервіс надає не погані процесори, які підійдуть для навчання нейромережі.

Для навчанні нейромережі необхідні дані, по яким вона буде навчатися. Такі дані можна скачати з COCO. COCO (Common Objects in Context) - це великомасштабний набір даних для виявлення об'єктів, сегментації, виявлення ключових точок та підписів. Набір даних складається з 328 тис. зображень.

Для початку потрібно завантажити dataset на google drive. Далі потрібно підключити свій google drive до colab. Для цього потрібно вписати:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Після цього клонуємо з репозиторія github архітектуру навчання:

```
!git clone https://github.com/AlexeyAB/darknet
```

Далі потрібно змінити makefile, щоб увімкнути GPU та OPENCV:

```
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
```

Далі потрібно збудувати проект за допомогою команди `!make`.

Для навчання потрібно завантажити вже заздалегідь підготовлені ваги. Існує багато різних заздалегідь підготовлених ваг. У нашому випадку в даних будуть малі та великі об'єкти, і офіційне репо радить використовувати yolov4.conv.137 для ініціалізації.

```
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137
```

Потім потрібно завантажити датасет і розархівувати.

```
!cp /content/gdrive/MyDrive/neural/dataset1.v1i.darknet.zip ../
!unzip ../dataset1.v1i.darknet.zip -d data/
```

Нам потрібні додаткові файли з описом набору даних. Для цих файлів знадобиться файл .txt з шляхами до файлів у розділенням train та validation.

```
def add_txt_files_with_names_of_images(split = "train"):
  import os
  image_files = []
  os.chdir(os.path.join("data", split))
  for filename in os.listdir(os.getcwd()):
      if filename.endswith(".jpg"):
          image_files.append(f"data/{split}/" + filename)
  os.chdir("..")
  with open(f"{split}.txt", "w") as outfile:
      for image in image_files:
          outfile.write(image)
          outfile.write("\n")
      outfile.close()
  os.chdir("..")

for split in ["train", "valid"]:
  add_txt_files_with_names_of_images(split)


!ls data/
```

Потрібно створити папку backup на google drive для зберігання туди ваг нейромережі.

Далі потрібно скопіювати конфігурацію та змінити її:

```
!cp cfg/yolov4-custom.cfg /mydrive/YOLOv4Files/yolov4-obj.cfg
```

width=320

height=320

max_batches = 38000 *((кількість класів) x 2000 (не менше 6000))*

steps= 30400,34200 *((80% від max_batches), (90% від max_batches))*

filters = 72 *((кількість класів + 5) x 3, скрізь перед шаром йоло, 3 рази)*

classes = 19 *(у кожному шарі йоло)*

Необхідно створити файли з описом даних. Треба зробити на диску Google два файли: obj.data і obj.names. У obj.names ми запишемо імена класів, які будуть передбачені. У obj.data ми запишемо таку інформацію:

classes = 19

train = data/train.txt

valid = data/valid.txt

names = data/obj.names

backup = /mydrive/neural/backup

Далі копіюємо файли з gdrive в colab:

```
!cp /content/gdrive/MyDrive/yolov4-obj.cfg cfg/yolov4-obj.cfg
!cp /content/gdrive/MyDrive/obj.names ./data
!cp /content/gdrive/MyDrive/obj.data  ./data
```

Тепер можна виконувати навчання нейронної мережі:

```
!./darknet detector train data/obj.data cfg/yolov4-
obj.cfg yolov4.conv.137 -dont_show -map
```

Даний процес виконується досить довго і займає багато часу. Час виконання залежить від кількості класів та розміру датасету. Після закінчення навчання ваги нейромережі зберігаються в папку backup.

3.4 Створення серверної частини

Робота нейронної мережі дуже ресурсозатратне завдання. Процесор Raspberry Pi не зможе справлятися з такою кількістю розрахунків. Тому потрібно створити сервер, на якому будуть виконуватися розрахунки.

Raspberry Pi буде відправляти на сервер кадри з камери, які будуть оброблятися нейронною мережею. Отримані дані від нейронної мережі будуть відправлятися до Raspberry Pi. Для створення серверу знадобиться ряд бібліотек python. Для встановлення бібліотек використовується команда pip install. Ось бібліотеки які необхідно встановити:

– opencv-python;
– numpy;
– pickle;
– imagezmq;

Для початку імпортуємо бібліотеки:

```
import cv2
import os
import numpy as np
import time
import pickle
import socket
import imagezmq
```

Далі необхідно завантажити конфігурацію, ваги нейронної мережі та класи:

```
configPath = os.path.join("yolov4.cfg")
modelPath =  os.path.join("yolov4.weights")
classesPath = os.path.join("coco.names")
```

Використовуємо сокети для створення серверу:

```
ip = socket.gethostbyname(socket.gethostname())
port = 7777
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((ip, port))
connections = []
s.listen(1)
```

Для передачі кадрів використовується бібліотека imagezmq. Кадри можна передавати також через сокети, однак це не дуже ефективно і бібліотека справиться з цим завданням краще. Запуск сервера imagezmq:

```
image_hub = imagezmq.ImageHub()
```

Далі необхідно створити модель нейромережі та налаштувати її:

```
net = cv2.dnn_DetectionModel(modelPath, configPath)
net.setInputSize(200,200)
net.setInputScale(1.0/300)
net.setInputMean((300, 300, 300))
net.setInputSwapRB(True)
with open(classesPath, 'r') as f:
        classesList = f.read().splitlines()
```

Введення місця призначення, яке передається пристрою:

```
place = input("Enter your Address: ")
```

Далі створюється змінні, де будуть зберігатися адреса пристрою:

```
c = 0
addr = 0
```

Запускаємо цикл while для роботи серверу. В тілі циклу будуть виконуватися основні завдання. Спочатку перевіряємо підключення до серверу. Якщо підключень немає, то основні завдання не виконуються і відразу починається наступна ітерація циклу. Якщо підключення є, то на клієнт, який підключився відправляється назва місця призначення, а також виконання коду продовжується. Робиться це наступним чином:

```
 if c == 0:
      c, addr = s.accept()
if c != 0 and len(connections) == 0:
        connections.append(c)
        print(str(addr) + " connected.")
        c.sendall(place.encode(encoding='utf8') )
if len(connections) == 0:
        continue
```

Далі отримуємо кадри від клієнта:

```
 rpi_name, image = image_hub.recv_image()
image_hub.send_reply(b'OK')
```

Після обчислення нейронною мережею, яка записує знайдені класи, їх ймовірність, та координати розташування на кадрі:

```
classLabelIDs, confidences, bboxs = net.detect(image, confThreshold = 0.5)
```

```
            bboxs = list(bboxs)
            confidences = list(np.array(confidences).reshape(1,-1)[0])
            confidences = list(map(float, confidences
            bboxIdx = cv2.dnn.NMSBoxes(bboxs, confidences, score_threshold =
0.5, nms_threshold = 0.2)
            objects = []
```

Якщо масив об'єктів не пустий, то виконується обробка отриманих даних, а оброблені дані додаються в масив object:

```
    if len(bboxIdx) != 0:
                for i in range(0, len(bboxIdx)):
                        bbox = bboxs[np.squeeze(bboxIdx[i])]
                        classConfidence=
confidences[np.squeeze(bboxIdx[i])]
                        classLabelID=
np.squeeze(classLabelIDs[np.squeeze(bboxIdx[i])])
                        classLabel = classesList[classLabelID - 1]
                        objects.append([bbox,classLabel])
```

Дані архівуються за допомогою бібліотека pickle для того щоб їх можна було відправити, а потім відправляються клієнту:

```
    objects = pickle.dumps(objects)
            try:
                    c.sendall(objects)
            except:
                    connections = []
                    c.close()
                    c = 0
                    cv2.destroyAllWindows()
                    continue
```

Якщо не вдалося відправити, то сервер від'єднує клієнт.

Результат роботи серверу наведений на рисунку 3.5.

Рис. 3.5 – Результат роботи серверу

3.5 Програмне забезпечення апаратної частини

Для початку потрібно встановити на мікрокомп’ютер ряд бібліотек Python. Такі як:

– opencv-python;
– opencv-contrib-python;
– gps;
– numpy;
– pickle;
– imagezmq;
– geographiclib;
– googlemaps;
– imutils;

Для початку імпортуємо необхідні бібліотеки:

```
from gps import *
import time
from datetime import datetime, timedelta
```

```
from geographiclib.geodesic import Geodesic
import serial
import threading
import pickle
import googlemaps
import socket
import imagezmq
from imutils.video import VideoStream
```

Необхідно створити глобальні змінні, які використовуються для навігації:

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ip = '25.1.148.91' #ip серверу
port = 7777 #порт серверу
place = "" #зміна місця призначення
prev_lat = 0 #зміна широти
prev_long = 0 # зміна довготи
azimuth = 0 # зміна куда направлений пристрій
dist = 0 #дистанція до об'єкту
mode = True #перевірка на підключення gps до супутників
cone = True #перевірка підключення до серверу
objectt = False #перевірка на наявність об'єктів
azi = # необхідний азимут до цілі
```

Також підключаємо камеру:

```
rpi_name = socket.gethostname()
picam = VideoStream(usePiCamera=True).start()
time.sleep(2.0)
```

Підключення до контролера за допомогою бібліотеки Serial:

```
ser = serial.Serial('/dev/ttyUSB0', 9600, timeout=1)
ser.reset_input_buffer()
```

Далі запускається потік з функцією зчитування даних з модулів(опис самої функції буде далі):

```
thread1 = threading.Thread(target=getPosData).start()
```

Тепер запускаємо цикл, який закінчиться тоді, коли відбудеться підключення до серверу та встановлення gps-модулем зв'язку з супутниками:

```
while mode or conn:
```

```
        try:
            s.connect((ip,port))
            sender                                                    =
imagezmq.ImageSender(connect_to='tcp://25.1.148.91:5555')
            conn = False
        except:
            print("Couldn't connect to server")
```

Треба запустити потоки передачі кадрів з камери на сервер та отримання від сервера даних про об'єкти(функції будуть описані далі):

```
thread2 = threading.Thread(target=listen).start()
thread3 = threading.Thread(target=cam).start()
```

Для отримання координат цілі та маршруту використовується api googlemaps. Для отримання ключа api необхідно створити проект в google cloud, зайти в вкладку облікові дані і створити ключ(рисунок 3.6).

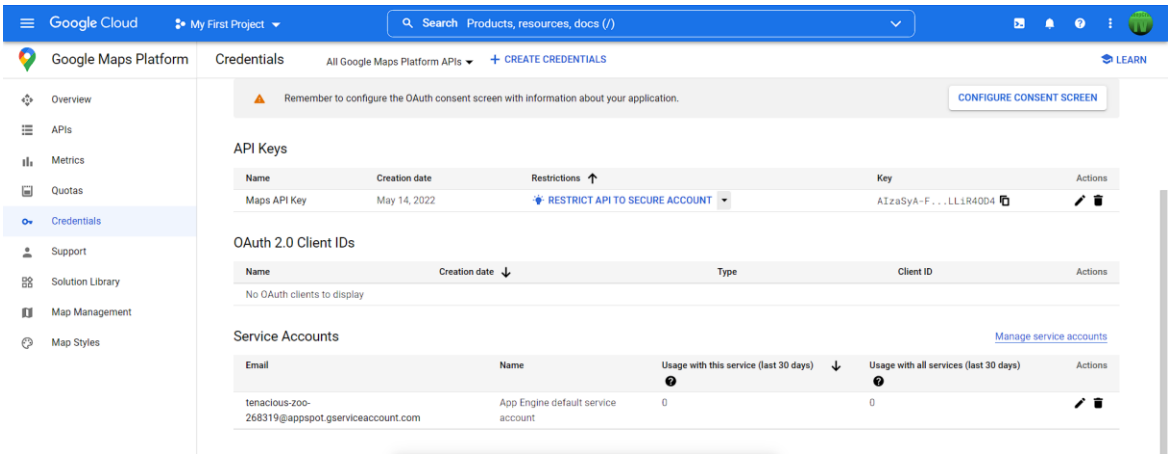

Рис. 3.6 – Отримання ключа для використання Google Maps Api

Далі вставляємо цей ключ в код:

```
gmaps=googlema5ps.Client(key='AIzaSyA-
8FubXjrZ5AViB3B22wLjNmkULLiR4OD4')
```

Потім за допомогою google maps прокладаємо маршрут. Спочатку отримуються координати місця призначення за його назвою:

```
geocode_result = gmaps.geocode(place)
```

Отримання місця знаходження пристрою за допомогою координат з gps-модуля:

```
reverse_geocode_result = gmaps.reverse_geocode((prev_lat,prev_long))
```

Функції google maps допомагають автоматично розрахувати маршрут до місця призначення:

```
results=
gmaps.directions(reverse_geocode_result[0]['formatted_address'],
geocode_result[0]["formatted_address"],mode="driving",
arrival_time=datetime.now()+timedelta(minutes=0.5))
```

В result записані всі точки маршрута та інформація про них.

Залишилося дістати з result координати кожної точки:

```
marker_points = []
waypoints = []
for leg in results[0]["legs"]:
    leg_start_loc = leg["start_location"]
marker_points.append(f'{leg_start_loc["lat"]},{leg_start_loc["lng"]}')
    for step in leg["steps"]:
        end_loc = step["end_location"]
        waypoints.append(f'{end_loc["lat"]},{end_loc["lng"]}')
last_stop = results[0]["legs"][-1]["end_location"]
marker_points.append(f'{last_stop["lat"]},{last_stop["lng"]}')
```

Тепер розглянемо основні функції. Функція передачі кадрів з камери на сервер:

```
def cam():
    while 1:
        image = picam.read()
        sender.send_image(rpi_name, image)
```

Функція отримання азимуту між двома точками за допомогою координат:

```
def get_azimuth(lat1,lat2,long1,long2):
    brng = Geodesic.WGS84.Inverse(lat1,long1,lat2,long2)['azi1']
    if brng < 0:
        brng = 360 + brng
    return brng
```

За допомогою бібліотеки geodesic та координат двох точок можна визначити азимут від однієї точки до другої

Функція для визначення в яку сторону ближче повертати відносно двох азимутів:

```
def get_direct(x1, x2):
    x1 = float(x1)
    x2 = float(x2)
    left = x1-x2
    if left < 0:
        left = (360 - x2)+x1
    right = x2-x1
    if right <0:
        right =(360-x1)+x2
    if right < left:
        return True,right
    else:
        return False,left
```

Якщо азимут пристрою не збігається з потрібним азимутом маршруту, то за допомогою цієї функції можна розрахувати в яку сторону найближче повертати.

Функція отримання даних з сервера:

```
def listen():
    global place
    global objectt
    while 1:
        if place == "":
            place = s.recv(1024)
        try:
            new_data = s.recv(1024)
        except:
            break
        data = pickle.loads(new_data)
        if len(data) != 0:
            objectt = True
        else:
            objectt = False
```

На початку отримуємо назву місця призначення та записуємо її в глобальну зміну. Потім отримуються дані з сервера та дістаються за допомогою бібліотеки pickle.

Функція для отримання значень з сенсорів:

```
def getPosData():
    gpsd = gps(mode=WATCH_ENABLE|WATCH_NEWSTYLE)
    while running:
        nx = gpsd.next()
         if nx['class'] == 'TPV':
             if nx['mode'] == 3:
                 global mode
             if mode == False:
                 mode = True
             latitude = getattr(nx, 'lat', "Unknown")
             longitude = getattr(nx, 'lon', "Unknown")
             global prev_lat
             global prev_long
            global azimuth
            global dist
            azi = ser.readline().decode('utf-8').rstrip().split("|")[1]
            dist              =              ser.readline().decode('utf-
8').rstrip().split("|")[0]
            if azi != '''':
               azimuth = azi
            prev_lat = latitude
            prev_long = longitude
```

Спочатку, поки gps-модуль намагається встановити зв’язок з супутниками, тіло функції не виконується. Коли зв’язок встановлений записуються довгота ти широта. В зміну azimuth записується азимут отриманий з компаса. В зміну dist записується відстань до об’єктів, яка отримана з ультразвукового сенсора.

В кінці незалежно від потоків починається цикл основної програми. В тілі цього цикла виконуються основні задачі. Спочатку отримуємо азимут між поточним розташуванням та місцем призначення:

```
          azi=get_azimuth(prev_lat,       float(waypoints[0].split(',')[0]),
prev_long, float(waypoints[0].split(',')[1]))
```

Якщо різниця між ними більше 16, то виконується функція знаходження найближчої сторони для повороту:

```
        r = float(azimuth) - azi
            byte = '25525511'
            if abs(r) >= 16:
                right, deg = get_direct(azimuth, azi)
                if right:
                    if deg < 20:
                        byte = '25515010'
                    if deg >= 50 and deg < 80:
                        byte = '25520010'
                    if deg >= 80 and deg < 180:
                        byte = '25525510'
                else:
                    if deg < 20:
                        byte = '15025501'
                    if deg >= 50 and deg < 80:
                        byte = '20025501'
                    if deg >= 80 and deg < 180:
                        byte = '25525501'
```

В змінній byte знаходяться 8 цифр, вони відповідають за рух моторів. Перші три цифри відповідають за швидкість першого мотору, наступні 3 цифри відповідають за швидкість другого мотору. Сьома цифра відповідає за напрямок лівого мотору, а восьма за напрямок правого мотору. Якщо 1 – рух вперед, 0 – рух назад.

В кінці зміна byte записується в serial порт та буде отримана мікроконтролером:

```
ser.write(byte.encode())
time.sleep(2.0)
ser.reset_input_buffer()
```

3.6 Висновок до розділу 3

У розділі 3 обрано технологію та мову програмування. Для розроблення програмної частини було обрано середовище розробки Arduino IDE оскільки у даному проекті використовується плата Arduino UNO. Для Arduino була написана прошивка для управління різними модулями та спілкування з мікрокомп’ютером. Також використовувалася мова програмування python для основних обчислень. Надано опис використаних функцій. Також обрано та використано необхідні бібліотеки.

Була створена та використана нейронна мережа для виявлення об’єктів. Для обчислень нейронної мережі була створена серверна частина, яка отримує кадри в камери пристрою та повертає дані про виявленні об’єкти на клієнт.

Було розроблене програмне забезпечення для обчислювача. За допомогою нього пристрій отримує дані з сенсорів, а також будує маршрут до цілі.

ВИСНОВКИ

Згідно мети бакалаврської роботи розроблено програмно-апаратний комплекс пристроїв, які автономно виконують обчислення правильного маршруту до цілей, аналіз навколишнього середовища та оминають перешкоди.

Було проаналізовано та обрано необхідні компоненти для створення системи автономної навігації. Багато необхідних модулів навігації з більш точними вимірюваннями мають більшу ціну, тому було обрано пристрої з достатньо високою точністю та задовільною ціною.

Було розраховано необхідну кількість енергії для кожного модулю. Найбільше енергії споживаються обчислювач та двигуни. Чим більш потужні двигуни, тим більше буде споживатися енергія.

Для виявлення об’єктів було вирішено використати камеру та за допомогою нейронної мережі знаходити об’єкти на камері. Проаналізовано всі архітектури нейронної мережі. Для обчислень нейронної мережі потрібний потужний процесор. Було вирішено створити серверну частину для цієї цілі.

Для роботи апаратної частини було створено програмні застосунки. Прошивки для Arduino розроблялися в Arduino IDE. Робота мікрокомп’ютера була написана на мові python, так як для неї передбачено багато бібліотек для різних цілей. Для прокладання маршруту було використано Google Maps Api.

Розроблений робот має такі переваги:

– Не великі габарити;
– Розпізнавання великої кількості об’єктів;
– Можливість додавати ще модулі;
– Висока швидкість виконання програмної частини.

До недоліків можна віднести:

– Мала швидкість руху;
– Не велика кількість модулів для аналізу навколишнього середовища.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. "Arduino: The Popular Microcontroller Board! Its History and to How to Use It," [Online]. Available: https://www.deviceplus.com/arduino/arduino-the-popular-microcontroller-board-its-history-and-to-how-to-use-it/. [Accessed 5 травень 2022].
2. "Google Maps Platform," Google, [Online]. Available: https://developers.google.com/maps/documentation?_gl=1*hc8wxy*_ga*MTYyNzcwNzY5Mi4xNjU2MDg1Nzkx*_ga_NRWSTWS78N*MTY1NjI3NzU0Ny4yLjAuMTY1NjI3NzU0Ny4w. [Accessed 6 травень 2022].
3. "OpenCV-Python Tutorials," Google, [Online]. Available: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html. [Accessed 6 травень 2022].
4. "Raspberry Pi: A cheat sheet," [Online]. Available: https://www.techrepublic.com/article/raspberry-pi-cheat-sheet/. [Accessed 5 Травень 2022].
5. "What is an Arduino?," [Online]. Available: https://learn.sparkfun.com/tutorials/what-is-an-arduino/all. [Accessed 6 травень 2022].
6. C. C. Aggarwal, Neural Networks and Deep Learning: A Textbook, 2018.
7. E. R. Davies, Computer Vision: Principles, Algorithms, Applications, Learning.
8. F. Z. Gerald Cook, Mobile Robots: Navigation, Control and Sensing, Surface Robots and AUVs, 2nd Edition.
9. O. R. G. Luis Payá, Mobile Robots Navigation, 2020.

10. S. E. Mathe, A. C. Pamarthy, H. K. Kondaveeti and S. Vappangi, "A Review on Raspberry Pi and its Robotic Applications," [Online]. Available: https://ieeexplore.ieee.org/document/9760590. [Accessed 5 Травень 2022].
11. P. Pedamkar, "Uses Of Raspberry Pi," [Online]. Available: https://www.educba.com/uses-of-raspberry-pi/. [Accessed 5 Травень 2022].
12. D. E. F. L. Y. C. Peiyuan Jiang, "A Review of Yolo Algorithm Developments," [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050922001363. [Accessed 6 травень 2022].
13. I. R. N. D. S. Roland Siegwart, Introduction to Autonomous Mobile Robots (Intelligent Robotics and Autonomous Agents series) second edition.
14. M. Yusro, "Utilization of microcontroller technology using Arduino board for Internet of Things (a systematic review)," [Online]. Available: https://aip.scitation.org/doi/10.1063/5.0041705. [Accessed 6 травень 2022].
15. Ш. Франсуа, Глубокое обучение на Python, 2017.
16. Я. Эрик, Programming Computer Vision with Python: Tools and Algorithms for Analyzing Images, 2012.
17. "SSH Documentation," [Online]. Available: https://man.openbsd.org/ssh. [Accessed 2 червень 2022].
18. "Raspberry Pi 3B+ 32 bit and 64 bit Benchmarks and Stress Tests," [Online]. Available: https://www.researchgate.net/publication/327467963_Raspberry_Pi_3B_32_bit_an. [Accessed 6 червень 2022].
19. "Raspberry Pi Camera Board v1.3 Datasheet," [Online]. Available: https://uk.pi-supply.com/products/raspberry-pi-camera-board-v1-3-5mp-1080p. [Accessed 6 червень 2022].
20. "Raspberry Pi 3 Model B+ Datasheet," [Online]. Available: https://raspberrypi.in.ua/wp-

content/uploads/2018/03/datasheet_raspberry_pi_3_model_b.pdf. [Accessed 6 червень 2022].

21. "Camera Serial Interface," [Online]. Available: https://www.mipi.org/sites/default/files/Camera%20Serial%20Interface_CSI2_CSI. [Accessed 6 червень 2022].
22. "R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms," [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e. [Accessed 6 червень 2022].
23. "Raspbian OS," [Online]. Available: http://www.raspbian.org/. [Accessed 6 червень 2022].
24. "imageZMQ: Transporting OpenCV images," [Online]. Available: https://pythonlang.dev/repo/jeffbass-imagezmq/. [Accessed 6 червень 2022].
25. "socket — Low-level networking interface," [Online]. Available: https://docs.python.org/3/library/socket.html. [Accessed 6 червень 2022].
26. "LSM303 Accelerometer + Compass Breakout," [Online]. Available: https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout/coding. [Accessed 6 червень 2022].
27. "Interface NEO-6M GPS Module with Arduino," [Online]. Available: https://microcontrollerslab.com/neo-6m-gps-module-arduino-tutorial/. [Accessed 6 червень 2022].
28. "Introduction to L298N Motor Driver," [Online]. Available: https://www.electroduino.com/introduction-to-l298n-motor-driver-how-its-work/#:~:text=L298n%20motor%20driver%20module%20uses,polarity%20of%20its%20input%20voltage.. [Accessed 6 червень 2022].

Додаток А

Код програми для мікроконтроллера

```
#include <Wire.h>
#include <LSM303.h>

LSM303 compass;
const int LEFT_MOTOR_IN = 11;
const int RIGHT_MOTOR_IN = 10;
const int LEFT_MOTOR1 = 8;
const int LEFT_MOTOR2 = 7;
const int RIGHT_MOTOR1 = 4;
const int RIGHT_MOTOR2 = 3;
String one = "0  ";
String two = "0  ";
String r1 = "1";
String r2 = "2";
int distance = 0;

long readUltrasonicDistance(int triggerPin, int echoPin)
{
  pinMode(triggerPin, OUTPUT);  // Clear the trigger
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  // Sets the trigger pin to HIGH state for 10 microseconds
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  pinMode(echoPin, INPUT);
  // Reads the echo pin, and returns the sound wave travel time in microseconds
  return pulseIn(echoPin, HIGH);
}

void left_motor_foward(){
  digitalWrite(LEFT_MOTOR1, LOW);
  digitalWrite(LEFT_MOTOR2, HIGH);
}
void left_motor_back(){
  digitalWrite(LEFT_MOTOR1, HIGH);
  digitalWrite(LEFT_MOTOR2, LOW);
```

```
}
void right_motor_foward(){
  digitalWrite(RIGHT_MOTOR1, HIGH);
  digitalWrite(RIGHT_MOTOR2, LOW);
}
void right_motor_back(){
  digitalWrite(RIGHT_MOTOR1, LOW);
  digitalWrite(RIGHT_MOTOR2, HIGH);
}

void setup() {
Serial.begin(9600);
Wire.begin();
compass.init();
compass.enableDefault();
pinMode(LEFT_MOTOR1, OUTPUT);
pinMode(LEFT_MOTOR2, OUTPUT);
pinMode(RIGHT_MOTOR1, OUTPUT);
pinMode(RIGHT_MOTOR2, OUTPUT);

pinMode(LEFT_MOTOR_IN, OUTPUT);
pinMode(RIGHT_MOTOR_IN, OUTPUT);
left_motor_foward();
right_motor_foward();

compass.m_min = (LSM303::vector<int16_t>){-264, -786, -682};
compass.m_max = (LSM303::vector<int16_t>){+785, +392, +463};
}
void loop() {
  compass.read();
  float heading = compass.heading() + 15;
  if(heading > 360) {
    heading = heading - 360;
  }
  if( Serial.available() >0 ) {
    String data = Serial.readStringUntil('\n');
    //Serial.print(data);
    one = data.substring(0,3);
    two = data.substring(3,6);
    r1 = data.substring(6,7);
    r2 = data.substring(7,8);
```

```
    if (r1.toInt() == 1){
      left_motor_foward();
    }else{
      left_motor_back();
    }
    if (r2.toInt() == 1){
      right_motor_foward();
    }else{
      right_motor_back();
    }
    //Serial.println(two.toInt());
  }
  analogWrite(LEFT_MOTOR_IN, one.toInt());
  analogWrite(RIGHT_MOTOR_IN, two.toInt());
  distance = 0.01723 * readUltrasonicDistance(5, 6);
  Serial.println(String(distance) + "|"+ String(heading,2));
  //Serial.println("Hello from Arduino!");
delay(500);
}
```

Додаток Б

Код програми для мікрокомп’ютера

```
from gps import *
import time
from datetime import datetime, timedelta
from geographiclib.geodesic import Geodesic
import serial
import threading
import pickle
import googlemaps
import socket
import imagezmq
from imutils.video import VideoStream

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ip = '25.1.148.91' #ip серверу
port = 7777 #порт серверу
place = "" #зміна місця призначення
running = True
prev_lat = 47.628510333 #зміна широти
prev_long = 30.999817667 # зміна довготи
azimuth = 0 # зміна куда направлений пристрій
dist = 0 #дистанція до об'єкту
mode = True #перевірка на підключення gps до супутників
conn = True
objectt = False #перевірка на наявність об'єктів
azi = # необзідний азимут до цілі
## lon = 30.999817667, lat = 47.628510333

def cam():
    while 1:
        image = picam.read()
        sender.send_image(rpi_name, image)
def get_azimuth(lat1,lat2,long1,long2):
    brng = Geodesic.WGS84.Inverse(lat1,long1,lat2,long2)['azi1']
    if brng < 0:
        brng = 360 + brng
    return brng
```

```
def get_direct(x1, x2):
    x1 = float(x1)
    x2 = float(x2)
    left = x1-x2
    if left < 0:
        left = (360 - x2)+x1
    right = x2-x1
    if right <0:
        right =(360-x1)+x2
    if right < left:
        return True,right
    else:
        return False,left
def listen():
    global place
    global objectt
    while 1:
        if place == "":
            place = s.recv(1024)
            print(place)
        try:
            new_data = s.recv(1024)
        except:
            break
        data = pickle.loads(new_data)
        print(data)
        if len(data) != 0:
            objectt = True
        else:
            objectt = False
def getPosData():
    gpsd = gps(mode=WATCH_ENABLE|WATCH_NEWSTYLE)
    while running:
        nx = gpsd.next()
         if nx['class'] == 'TPV':
             if nx['mode'] == 3:
                 global mode
             if mode == False:
                 mode = True
             latitude = getattr(nx, 'lat', "Unknown")
             longitude = getattr(nx, 'lon', "Unknown")
```

```
                    global prev_lat
                    global prev_long
                   global azimuth
                   global dist
                   azi = ser.readline().decode('utf-8').rstrip().split("|")[1]
                   dist = ser.readline().decode('utf-8').rstrip().split("|")[0]
                   if azi != '':
                      azimuth = azi
                   print("azimuth: ",azimuth)
                   print("Distance: ",dist)
                   prev_lat = latitude
                   prev_long = longitude
       ser = serial.Serial('/dev/ttyUSB0', 9600, timeout=1)
       ser.reset_input_buffer()
       gmaps = googlemaps.Client(key='AIzaSyA-FubXjrZ5AViB3B22wLjNmkULLiR4OD4')
       thread1 = threading.Thread(target=getPosData).start()
       while mode or conn:
           try:
               s.connect((ip,port))
               sender = imagezmq.ImageSender(connect_to='tcp://25.1.148.91:5555')
               conn = False
               print("Connected")
           except:
               print("Couldn't connect to server")

       thread2 = threading.Thread(target=listen).start()
       rpi_name = socket.gethostname()
       picam = VideoStream(usePiCamera=True).start()
       time.sleep(2.0)
       thread3 = threading.Thread(target=cam).start()
       #Get Way ...
       geocode_result = gmaps.geocode(place)
       reverse_geocode_result = gmaps.reverse_geocode((prev_lat,prev_long))
       results = gmaps.directions(reverse_geocode_result[0]['formatted_address'],
                                           geocode_result[0]["formatted_address"],
                                           mode="driving",
                                           arrival_time=datetime.now()             +
timedelta(minutes=0.5))
       marker_points = []
       waypoints = []
       for leg in results[0]["legs"]:
```

```
            leg_start_loc = leg["start_location"]
            marker_points.append(f'{leg_start_loc["lat"]},{leg_start_loc["lng"]}')
            for step in leg["steps"]:
                end_loc = step["end_location"]
                waypoints.append(f'{end_loc["lat"]},{end_loc["lng"]}')
        last_stop = results[0]["legs"][-1]["end_location"]
        marker_points.append(f'{last_stop["lat"]},{last_stop["lng"]}')

        markers = [ "color:blue|size:mid|label:" + chr(65+i) + "|"
                   + r for i, r in enumerate(marker_points)]
        print(waypoints)
        print(last_stop)
        result_map = gmaps.static_map(
                        center = waypoints[0],
                        scale=2,
                        zoom=16,
                        size=[640, 640],
                        format="jpg",
                        maptype="roadmap",
                        markers=markers,
                        path="color:0x0000ff|weight:2|" + "|".join(waypoints))

        with open("driving_route_map.png", "wb") as img:
            for chunk in result_map:
                img.write(chunk)
        #...Get Way
        try:
            print("App started!")

            while running:
                azi    =    get_azimuth(prev_lat,    float(waypoints[0].split(',')[0]),
prev_long, float(waypoints[0].split(',')[1]))
                print("azi ", azi)
                r = float(azimuth) - azi
                byte = '0  0  11'
                if abs(r) >= 16:
                    right, deg = get_direct(azimuth, azi)
                    print(right, deg)
                    if right:
                        if deg < 20:
                            byte = '25510011'
```

```
                    if deg >= 20 and deg < 80:
                        byte = '25505011'
                    if deg >= 80 and deg < 180:
                        byte = '25502011'
                else:
                    if deg < 20:
                        byte = '10025511'
                    if deg >= 20 and deg < 80:
                        byte = '05025511'
                    if deg >= 80 and deg < 180:
                        byte = '02025511'

            ser.write(byte.encode())
            time.sleep(2.0)
            ser.reset_input_buffer()
            way1                                                                    =
gmaps.reverse_geocode((float(waypoints[0].split(',')[0]),float(waypoints[0].split(','
)[1])))
            dist                                                                    =
gmaps.distance_matrix(origins=reverse_geocode_result[0]['formatted_address'],
                          destinations=way1[0]["formatted_address"],
                          departure_time=datetime.now())
            distance = dist['rows'][0]['elements'][0].get('distance')['text']
            if float(distance.split(" km")[0]) < 0.05:
                waypoints.pop(0)
            print("dist: ", distance)

    except(KeyboardInterrupt):
        byte = '0  0  11'
        ser.write(byte.encode())
        running = False
        print("App closed!")
```