Міністерство освіти і науки України
Державний університет «Одеська політехніка»

Інститут штучного інтелекту та робототехніки
Кафедра «Комп'ютерних систем»

Парфенюк Даніїл В'ячеславович,
студент групи УК-162

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Дослідження методів підвищення ефективності систем автоматичного паркування

Спеціальність: 123 – "Комп'ютерна інженерія"
Спеціалізація: Спеціалізовані комп'ютерні системи

Керівник:
Стрельцов Олег Васильович,
кандидат техн. наук, доцент

Одеса — 2021

Міністерство освіти і науки України
Державний університет «Одеська політехніка»

Інститут штучного інтелекту та робототехніки
Кафедра комп'ютерних систем

Рівень вищої освіти другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
(шифр і назва)
Спеціалізація / освітня програма Спеціалізовані ком'ютерні системи

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____

“____” _____ 2021 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Парфенюк Данііл В’ячеславович
(прізвище, ім’я, по батькові)

1. Тема роботи Дослідження методів підвищення ефективності систем автоматичного паркування
Керівник роботи Стрельцов Олег Васильович, кандидат техн. наук, доцент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ректора ОНПУ від “___” _____ року № _____
2. Зміст роботи
Існуючі системи та методи, Автономні транспортні засоби та автомобілі-роботи, Система автоматичного та автономного паркування автомобіля, Проблеми автопілота, Методи ідентифікації об’єктів у реальному часі, Методи ідентифікації відстані, Висновки за розділом 1, Побудова методів для підвищення ефективності, Побудова методу розпізнавання об’єктів у реальному часі, Побудова методу ідентифікації відстані, Висновки за розділом 2, Тестування розробленої системи, Реалізація руху стенду на Raspberry Pi4, Тестування методу розпізнавання об’єктів у реальному часі, Тестування методу ідентифікації відстані, Тестування інтегрованої системи, Висновки за розділом 3, Висновки, Список використаної літератури, Додаток А.

3. Перелік ілюстративного матеріалу

презентація

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

5. Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Існуючі системи та методи		виконано
2	Автономні транспортні засоби та автомобілі-роботи		виконано
3	Система автоматичного та автономного паркування автомобіля		виконано
4	Проблеми автопілота		виконано
5	Методи ідентифікації об’єктів у реальному часі		виконано
6	Методи ідентифікації відстані		виконано
7	Побудова методів для підвищення ефективності		виконано
8	Побудова методу розпізнавання об’єктів у реальному часі		виконано
9	Побудова методу ідентифікації відстані		виконано
10	Тестування розробленої системи,		виконано
11	Реалізація руху стенду на Raspberry Pi4		виконано
12	Тестування методу розпізнавання об’єктів у реальному часі		виконано
13	Тестування методу ідентифікації відстані		виконано
14	Тестування інтегрованої системи		виконано

Здобувач вищої освіти _____ Парфенюк Д.В.

Керівник роботи _____ Стрельцов О.В.

ЗМІСТ

ВСТУП

Питання парковки для великих міст це одна з найбільш актуальних та проблемних тем. Щорічно збільшується численність авто проте темпи побудови паркінгів значно нижчі. Пошук вільного місця для парковки може затягнутися на довгі часи через вузьку поїзджу частину створену протиправною парковою на крайніх полосах, або постійні ДТП, або через невідомість вільних місць на паркінгах. Паркування с автоматизованою системою управління допоможе зберегти час та нерви клієнтів. Завдяки цифровим табло, та камерам водії зможуть орієнтуються про наявність місць прямо при в'їзді на паркінг. Камери відео можуть вивести кількість вільних місць та локації на бортові системи автомобіля або монітори на паркінгі й авто само припаркуется на вільному місці. Зазвичай використовується навігаційна система: GPS. Її робота забезпечується спеціальними супутниками. Пристрої, наприклад, навігатори або мобільні телефони з підтримкою цих систем отримують сигнал супутника і визначають своє місце розташування в прив'язці до географічних координат на основі завантажених у пристрій карт місцевості. Точність роботи глобальних навігаційних систем достатня в масштабах великих територій, мова йде про кілометри або ж сотні метрів в масштабах кварталів, міста або країни, та іноді похибка може досягати 5-20 метрів в залежності від перешкод та якості сигналу. У випадку з підземними паркінгами або паркінгами всередині великих торгівельних центрів, точно визначити місце знаходження за допомогою глобальних систем практично неможливо. Залізобетонні будівлі, підвали, тунелі, сильно впливають на кінцеву точність сигналу, також можуть впливати несприятливі погодні умови та інші подібні фактори. Саме цих недоліків позбавлені системи локального позиціонування, їх вкрай очевидні переваги в порівнянні з глобальними навігаційними системами стають у нагоді при

побудові систем локації та зв'язку в межах певних територій і приміщень, а саме в межах підземних паркінгів, що нас і цікавить. Системи локального позиціонування здатні вирішувати завдання з визначенням місце розташування людей, техніки, інших фізичних об'єктів в невеликих масштабах. В межах цілком конкретної території або всередині будівлі з високою точністю приблизно (від 5м до 0.1м). Такі комплекси зазвичай складаються із стаціонарних та пристроїв які знаходяться на рухомих об’єктах, блоку(ів) керування, сервера і спеціального програмного забезпечення, структура може видозмінюватись в залежності від потреб.

Мета: Підвищення ефективності паркування авто за рахунок класифікації динамічного моніторингу розпізнавання образів та ідентифікації відстані до об’єктів за пріоритетністю.

Задачі дослідження. Для досягнення поставленої мети визначені наступні задачі, які необхідно вирішити:

1) дослідити існуючі рішення в даній області;

2) розробити інтегрований метод «розумного» паркування;

3) експериментально перевірити роботу інтегрованого методу на Raspberry PI 4.

Об’єктом дослідження є процес «розумного» паркування.

Предмет дослідження є методи підвищення ефективності позиціювання об’єктів.

Інноваційне рішення полягає в інтеграції апаратно-програмних модулів для підвищенні точності та ефективності «розумного» паркування.

ІСНУЮЧІ СИСТЕМИ ТА МЕТОДИ

1.1 Автономні транспортні засоби та автомобілі-роботи

Підвищення автономності автомобілів стали розглядати відносно давно і перші дослідження в цьому напрямку проводилися в Японії у 1970 році. Сьогодні ця область продовжує процвітати, у всьому світі йдуть активні розробки. «Автономні транспортні засоби вже проїхали багато сотень кілометрів, і творці даної технології стверджують, що вона допоможе знизити кількість заторів на дорогах, збільшити пропускну спроможність доріг та зробити рух безпечнішим, без участі водія, оскільки за статистикою близько 90 % усіх дорожньо-транспортних засобів засобів пригод відбувається з вини людини»[1].

Робототехніка – галузь, якій відводиться ключова роль більшості експертних прогнозів розвитку провідних країн світу. «Основним напрямком сучасного динамічного прогресування цієї галузі служить не лише кількісне зростання інноваційних рішень та інвестицій, а й їхня переорієнтація від сектора промислової робототехніки, до якої традиційно відносять різні програмовані маніпулятори для сварки, сортування та інше, в сектор когнітивних роботів, можливості яких уже дозволяють вийти межі природних їм ринків»[1].

У галузі проектування безпілотних легкових автомобілів-роботів оптимального успіху досягла фірма Google (США). Переміщення Google-мобіля контролюється згідно з дорожньою картою, яку становить концепція керування. Згідно з дорогами Каліфорнії машини Google у пробному режимі проїхали понад 300 тис. миль, у цій кількості відповідно до основної магістралі серед Сан Франциско та Лос Анджелеса. Значна доля цієї дороги освоєна повністю в автоматичному режимі із працівником фірми у пасажирському сидінні.

Автопарк компанії Google має 20 безпілотних автомобілів моделей Toyota Prius, Audi TT та інші (Рисунок 1.1).

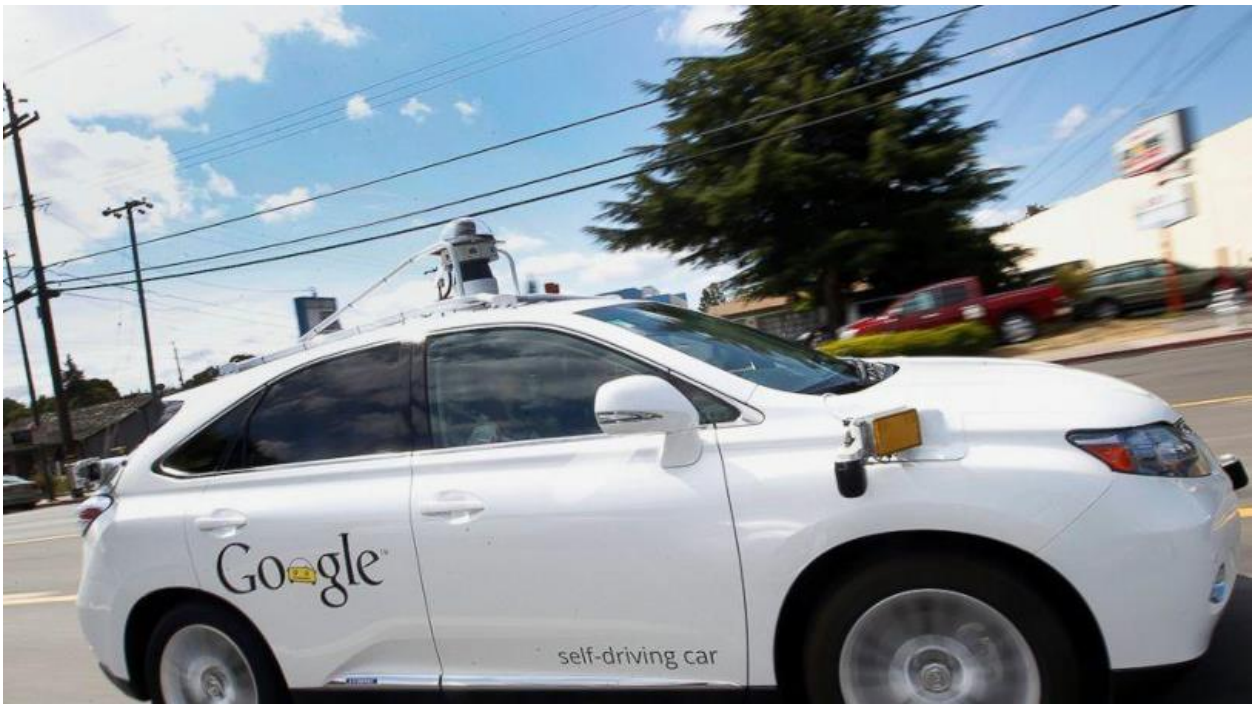

Рисунок 1.1 – Безпілотний автомобіль Google

В даний час встановлено велику кількість міжнародних планів згідно з дослідженням безпілотних транспортних засобів. З-поміж них можна відзначити численні проекти.

«Беспілотний засіб (БТС) під назвою Adaptive und KooperativeTechnologienfurdenIntelligentenVerkehr (скорочено AKTIV) розробляють 28 відомих європейських компаній. До них увійшли BMW, Siemen, Volkswagen, Bosch, Vodafone та ще 23 компанії » [2]. цією метою для AKTIV вважається концепція спостереження за безпекою на шляху, що дозволяє «убезпечити» водія у складній ситуації. У цьому періоді AKTIV здійснюватиме контроль над плавністю розгону і гальмування, синхронізуючи дані дії, зокрема й у всій категорії автомобілів. Крім того, вона містить аварійну систему гальмування при позаштатних ситуаціях. Автомобіль AKTIV під час тестування у безпілотному режимі на автодромі формує швидкість до 180 км/год.

Інший німецький проект із розробки безпілотного автомобіля під назвою Leonie. Його концепція управління здатна здійснювати контроль за станом 12 автомобілів у загальному автотранспортному потоці внаслідок великої кількості датчиків, далекомірів та тепловізорів. Автомобіль-робот має можливість переміщатися як у пустельній місцевості, так і в жвавих магістралях, долаючи великі дистанції та тунелі. Крім цього, німецькі конструктори ведуть дослідження згідно з формуванням ще одного безпілотного автомобіля-робота, який має назву Made in Germany (MIG).

У реальний період автомобіль MIG здатний здійснювати контроль над навколишньою ситуацією в радіусі 70 м і навчається їзді перехрестями та пішохідними переходами. Як сенсорні системи застосовуються всі ці ж датчики і далекоміри.

У конференції HAVEit німецьке об'єднання Volkswagen показало технологічні процеси Temporary AutoPilot "тимчасовий автопілот" на базі автомобіля Volkswagen Passat. Аналогічно літакам, водій цього автомобіля має право включати функцію автоматичного керування машиною в незавантажених зонах колії. Тимчасовий автопілот Volkswagen працює під час переміщення на магістралях зі швидкістю до 120 км/год. Перевага даної концепції управління у тому, що вона вже розташована до застосування на масових автомобілях. Створення Volkswagen вважається поступовим удосконаленням існуючих технологій, із якими комплектуються сучасні серійні машини, до яких відносяться, зокрема, адаптаційний круїз контроль та системи стеження за смугою переміщення.

Автопілот машини Volkswagen здатний керуватися відповідно до своєї смуги переміщення і зберігати стійку дистанцію за автотранспортним засобом, що пересувається спереду, і в разі необхідності зменшувати швидкість руху (Рисунок 1.2).

Рисунок 1.2 – БАС, що розробляється фірмами Volkswagen та Bosch, Vodafone

Ця концепція, ще, може розрізняти дорожні знаки і відповідати їх зміна швидкості руху. Крім магістралей автопілот Volkswagen здатний переміщатися в пробках, повторюючи однорідний цикл старт – зупинка. Однак, пересування цього безпілотного автомобіля в даний час обмежене, і він не здатний подолати встановлений маршрут без «підтримки» водія, на відміну від рухів автомобілів фірми Google.

Керівник японської автомобілебудівної фірми Nissan Карлос Гон заявив про те, що приблизно в 2020 році намічається масове виготовлення автомобілів з безпілотним керуванням. Відомості про виготовлення безпілотних машин було оголошено керуючим Nissan у Каліфорнії у місті Ірвайні, в якому розмістився центр Nissan. Формуватися роботизовані автомобілі будуть з використанням

технології «Autonomous Drive» (Рисунок 1.3), щоб керувати таким авто, немає необхідності присутності водія.

Нове виробництво буде включати лазерні радари, а також камери, відомості з яких буде потрапляти в комп'ютер, швидко пристосовується до поточної дорожньої ситуації.

Рисунок 1.3 – Безпілотний автомобіль компанії NISSAN.

З 2012 р., згідно з завданням Мінпромторгу, ФГУП «НАМІ» здійснює НДР щодо формування розумових самоврядних безпілотних вантажопасажирських АТС цивільного спрямування. В рамках плану розробляється приклад безпілотної АТС в основі автомобіля Лада-Калина з механічною коробкою передач (Рисунок 1.4).

У структуру технологічного зору входить комплект камер, радарів далекого та близького впливу, лазери та приймачі GNSS. За допомогою концепції технологічного зору БАС згідно з розроблюваними методами та програмним забезпеченням досліджує ситуацію навколо себе на дистанції до 200

м, розпізнає дорожню розмітку, дорожні знаки, світлофори, встановлює постійні та динамічні об'єкти. Згідно з записаним маршрутом, а також за допомогою концепції GNSS, автомобіль вже здатний переміщатися поза шляхами загального використання з місця А в місце Б без участі водія.

Рисунок 1.4 – Макетний зразок БАС на базі автомобіля Лада-Калина

«На макетному зразку автомобіля Лада-Калина забезпечена працездатність приводів педалі гальма, дросельної заслінки, кермового керування та селектора перемикання режимів роботи АККП, керованих за допомогою електронних пультів системи керування рухом»[3].

З підтримкою концепції технологічного зору, що надходить у структуру концепції управління БАС, водночас знаходять рішення ключові завдання підвищення безпеки дорожнього руху. Наприклад, у макетному стандарті БАС відпрацьовуються такі концепції: запобігання виїзду зустрічну

смугу руху; визначення дорожніх знаків, виявлення перешкод на шляху (автомобіль, людина, тварина тощо) та аварійної зупинки машини.

Першорядна концепція вважається особливо важливою збільшення безпеки дорожнього руху, т.к. Приблизно 80% смертельних випадків при ДТП у Російській Федерації відбуваються внаслідок виїзду АТС на зустрічну смугу руху. Метод та програмне надання концепції запобігання виїзду на зустрічну смугу руху розраховуються з урахуванням алгоритмів та програмних забезпечення концепцій визначення дорожніх знаків, виявлення перешкод на шляху.

"Введення "безпілотних" автомобілів дозволить ефективно вирішувати завдання підвищення безпеки АТС, зниження кількості пробок на дорогах, ДТП, травм та смертей, зниження витрати палива, викиду шкідливих речовин, парникових газів в атмосферу та підвищення рівня комфорту для пасажирів" [4].

Базові промислові постанови щодо безпілотних машин можна адаптувати і впроваджено на серійних АТС. Безпілотний автомобіль вважається перспективним проектом з метою цивільного та військового спрямування.

1.2 Система автоматичного та автономного паркування автомобіля

Автономна система лише починає пошуки своїх шанувальників, на відміну від автоматичної, яка їх уже виявила. Нижче будуть описані різновиди системи та їх складові. Говорячи про те, що автомобіль самостійно може припаркуватися, на сьогоднішній день нікого не здивуєш. Найчастіше вона відома як автоматичне паркування автомобіля. Прогрес не стоїть на місці і компанія BMW зробила крок вперед, випереджаючи всіх, запропонувавши автономну систему паркування Remote Valet Parking Assistant. По-друге, це дистанційний помічник паркування.

Однак у основі таких прогресуючих систем лежать найстаріші, робочі зразки. В даному випадку це автоматична система паркування, яка часто зустрічається на багатьох сучасних автомобілях, навіть не нових. Система

автоматичного паркування (друге найменування інтелектуальна система допомоги при паркуванні) є активною паркувальною системою, оскільки забезпечує паркування автомобіля в автоматичному або автоматизованому (виконуються окремі функції) режимі.

Ця система може використовуватися при паралельному або перпендикулярному паркуванні. Найчастіше поширюються системи з паралельним паркуванням. За основу паркування береться швидкість руху автомобіля та кут повороту колеса. У різних компаніях система називається по-різному:

Volkswagen - Park Assist, PAV;

Lexus або Toyota - IPAS;

Opel - Advanced Park Assist;

Ford або Mercedes-Benz - APA;

BMW - RPA System.

Головними складовими є ультразвукові датчики, далі йдуть вимикачі, блок управління та інші виконавчі пристрої машини. Важливо, що без додаткових систем автоматичне паркування не зможе працювати.

Ультразвукові датчики в системі аналогічні пасивній системі паркування, але відрізняються більшою дальністю дії, це порядку до 4.5 м. Як правило, встановлюється 12 ультразвукових датчиків, чотири збоку автомобіля, чотири спереду і чотири ззаду автомобіля.

У дію система наводиться примусово, увімкненням кнопки на панелі приладів або керма керування, коли водій планує припаркуватися. Поле активації системи електронний блок починає приймати сигнал від ультразвукових датчиків, перетворюючи їх на керуючі функції для виконавчих пристроїв. Як такі пристрої можна назвати управління двигуном, електропідсилювач керма, автоматична або механічна кпп і система курсової стійкості.

Залежно від виробника, інформація може виводитися на дисплей як допоміжна для водія і паралельно в автоматичному режимі керувати автомобілем, якщо водій дав дозвіл. Принцип пошуку місця для паркування. Як уже говорилося, пошук потрібного місця для паркування здійснюється за допомогою датчиків ультразвуку. З боків розташовані чотири такі датчики по дві на кожну сторону.

Швидкість руху автомобіля між рядами припаркованих автомобілів досягає 20 км/год для поперечного паркування, 40 км/год для паралельного паркування. Датчики сканують відстань між машинами та їхнє положення щодо вашого автомобіля.

Автоматичне паркування автомобіля.

Цей процес паркування можна розділити на два види:

за допомогою інструкцій під керуванням водія;

автоматичне паркування без участі водія.

У першому способі для водія виводитимуться підказки, візуальні та тестові інструкції на інформаційний дисплей. На основі підказок та траєкторії водій приймає рішення, як краще припаркуватися. Як правило, це рекомендації з рульового колеса, який кут повороту краще вибрати і напрямок руху.

Другий спосіб відрізняється, у порівнянні з попереднім, водій не бере участі в процесі паркування, а всі функції виконуються в автоматичному режимі. Замість водія кермо повертатиметься самостійно у потрібному напрямку, так само як і натискання педалі газу. З метою безпеки автоматичну систему паркування у будь-який час можна перевести в режим ручного паркування.

Розвиток не стоїть на місці, і допоміжні системи автомобіля також прогресують. Як уже говорилося, автоматична система може працювати, коли водій сидить та спостерігає за процесом роботи паркування автомобіля. Система автономного паркування автомобіля складається з електронного блоку управління, пристроїв виконання команд та вхідних пристроїв. До приладів для

зчитування або вхідних відносять лазерні датчики (лідери) та пульт дистанційного керування.

В основі лежить система автоматичного паркування автомобіля та попередження зіткнення по всьому периметру машини. Завдяки таким складникам та технології значно економиться час, найкраще система показала себе у багатоповерхових паркінгах. Крім того, система може помістити автомобіль в простір обмежений 20 см. з кожного боку машини, що дуже економить простір. Принцип роботи автономної системи Активація системи автономного паркування автомобіля починається з натискання кнопки на пульті дистанційного керування (Рисунок 1.5).

Рисунок 1.5 – Пульт дистанційного керування Smart Watch

Найголовнішою відмінністю такої системи від попереднього покоління є самостійне паркування без участі водія. Також автомобіль може самостійно висуватися до місця посадки водія та пасажирів. Принцип роботи та послідовність дій системи такі. Лідери, знявши дані, пересилають їх у електронний блок управління. Програма, зашита в блок, пропускає отриману

інформацію через метод оцінки ситуації, перешкод, регулювання швидкості, навігації та маршрут пошуку місця.

Завдяки використанню безлічі систем автомобіля, блок управління сам може вирішити, наскільки розігнати або призупинити машину. Одним із недоліків можна вважати те, що в блок прошивається цифрова карта (план) будівлі паркінгу, при цьому система не використовує GPS карти та сигнал. Найчастіше це робиться для закритих паркінгів, де сигнал просто не може пройти. Але для відкритих паркувань прошивати нічого не треба.

В результаті електронний блок управління формує сигнали для керуючих систем, які передаються на ці системи автомобіля: управління двигуном, коробкою передач, рульовим колесом та інші. Величезним плюсом системи є те, що вона не вартує великих грошей, а орієнтація в паркінгу не займе багато часу. Крім того, водій не зобов'язаний сидіти та стежити за автомобілем до останнього, доки автомобіль не зупиниться. Як результат, багато виробників встановлюватимуть автономну систему паркування на нові серійні автомобілі замість автоматичного паркування.

1.3 Проблеми автопілота

Автомобіль Tesla, що рухається в автоматичному режимі, знову скоїв дорожньо-транспортну пригоду (ДТП). Кросовер Model X врізався в огорожу праворуч від дороги, після чого відлетів і вдарився в бетонний відбійник, що розділяє смуги зустрічного напрямку. Машина перекинулася, але жертв вдалося уникнути. За словами водія, на момент ДТП автомобіль знаходився в режимі автопілота. За кілька тижнів раніше автомобіль Model S під керуванням автопілота скоїв зіткнення з причепом фури, що призвело до смерті водія.

Інформація про аварії з автопілотованими Tesla приходить все частіше, а компанія не втомлюється пояснювати, що електроніка в машинах є допоміжною, а відповідальність за можливі події у будь-якому разі лежить на водії. Тим не

менш, інформаційне тло навколо так званого автопілота, так само як і заяви Ілона Маска, створюють у багатьох оманливе враження того, що на електроніку Tesla вже можна повністю покластися. Однак можливості автомобіля на цьому етапі не дозволяють використовувати його в безпілотному режимі.

Виділяють три практичні складнощі, з якими стикаються автопілоти зараз і які не виглядають найближчими роками. Перша проблема – датчики. Основне загальмовування формування автопілотів – не методи, а погане уявлення автомобілем, що знаходиться навколо місця. Даних із сучасних вимірювачів категорично мало і вдосконалення пристосування прийняття рішень (нейросети тощо) трохи змінює. Скануючі лідери (темні відра, які стають в основному прототипом, у тому числі Google Car) дуже дорогі з метою масових автомобілів і не надають потрібної повноти даних, з метою рішучого переміщення.

Проста ілюстрація: AI перемагає пілота на симуляторі повітряного бою в тій ситуації, коли машина має повну інформацію про те, що відбувається. Комп'ютер проїде вас у будь-якому симуляторі. На заваді цього не відбудеться.

І комп'ютерні полігони для автономного транспорту, які навчаються на синтетичній інформації, нічого серйозно не змінять. Є думка, що за існуючих датчиків (камери, радари, лідери тощо) повністю вирішити проблему самоврядування взагалі не можна. Так що вдосконалювати можливості сучасних датчиків, вигадувати нові для правильної «оцифровки» місця в динаміці.

«На сьогоднішній день одна з проблем навігаційних пристроїв – це те, що вони не ведуть користувача смугами. Ця проблема збільшує час у дорозі, пробки та аварійність. Нещодавно google maps почали відображати розмітку дороги перед поворотом, що вже добрий результат, але й тут можна багато покращити. Карти не знають, на якій смузі зараз знаходиться машина, засобами gps дізнатися це проблематично, у gps занадто велика похибка для цього »[5]. Якби ми знали поточну смугу, то знали б швидкість руху смугами і могли б задовго підказувати користувачеві в явному вигляді, на яку смугу і коли йому краще перебудуватися (Рисунок 1.6)

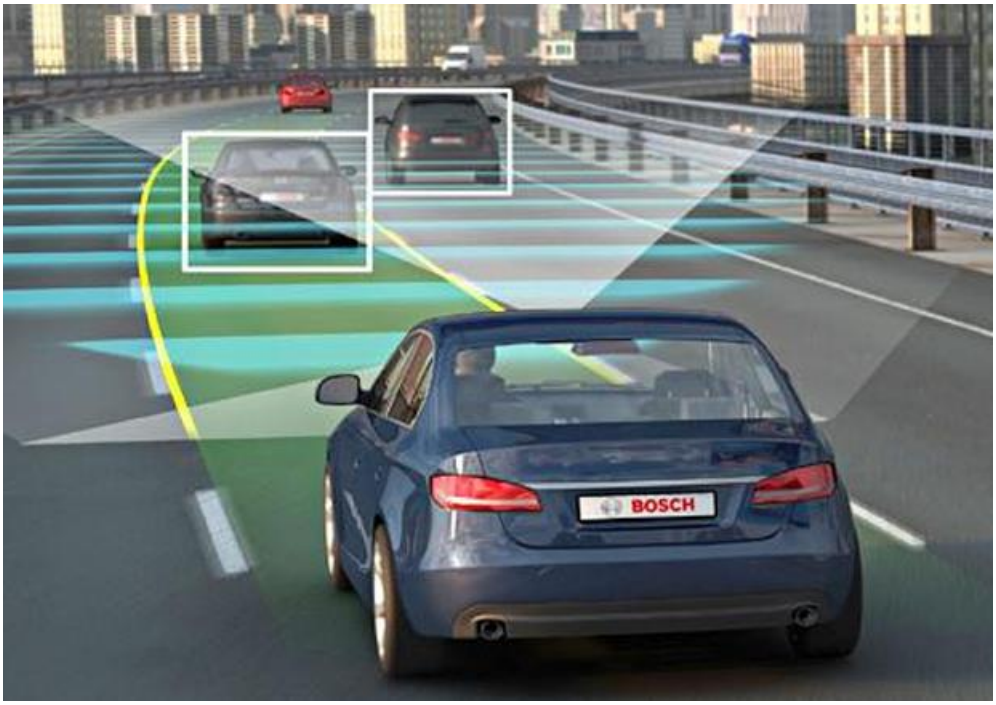

Рисунок 1.6 – Датчики автопілотів.

Чимось схоже на питання з акумуляторами. Поки ємність акумуляторів не вдасться підвищити в рази, а краще на порядок, електроніка, електрокари, що носиться, і багато інших тупцюють на місці. Друга проблема – не універсальність. Всі знайомі із Законом Парето, тож в автопілотах він теж виконується. 80% дорожніх ситуацій машини розуміють, 20%, що залишилися, все псують. «Це не означає, що автопілоти можуть їхати 80% доріг у світі. Це означає, що у межах будь-якої довільно взятої поїздки 80% ситуацій не викличуть проблем, 20% – викличуть. Небезпечна пропорція»[5].

Ці завдання і виявляться найпростішими. Автобанний автопілот, автопаркування, заїзд у гараж – не без помарок, але працює. Вченим залишилося найскладніше: проїзд перетинів, перебудови у жвавому потоці та ще кілька десятків подібних сценаріїв. З деякими кейсами не дуже зрозуміло, що робити. Наприклад, автомобілю складно заздалегідь зрозуміти, чи проїде він між двома перешкодами, що близько розташовані. І поки неможливо зрозуміти свій

динамічний коридор (тобто межі місця, які автомобіль займає в динаміці), а без цього неможливо коректно прорахувати дорожню ситуацію.

Існуючі автопілоти настільки не універсальні, що вимагають кожен дорожній сценарій описувати окремо. Офіційна позиція така: система запобігання зіткненню не розуміє приїзду автомобіля зі зустрічної смуги, це буде забезпечено до 2018 року. «Для вирішення цієї проблеми клієнту, як мінімум, не варто вірити ефектним демонстраціям на автострадах. Щодо гравців, то вони поки не розуміють, який шлях вірний. Звідси нейромережі, ручна розмітка сценаріїв (пишуть, що в Mobileye цим займаються 600 осіб, а скоро буде понад 1000), віртуальні полігони тощо» [5].

Проблема третя, ринкова. Автопілота має бути функціональною, потрібно стати дешевою. Автомобільної навігації понад двадцять років. Але за ціни системи в пару-трійку тисяч доларів вона досі ставиться менш ніж на 20% усіх нових автомобілів. Самоврядний функціонал ще дорожчий. «За лідера Velodyne 70 тисяч доларів чи 30 тисяч за кілька лідерів Sick це дуже багато. Якщо орієнтуватися на серійні автомобілі, advanced driver assistance systems, скорочено ADAS, (які ще автопілоти) додають до ціни 7-12 тисяч доларів» [6].

Дослідження Boston Consulting Group каже, що витратити на автопілот понад 5 тисяч доларів готово лише 17% покупців (американських покупців). Тобто, щоб завоювати хоча б 15 відсотків ринку, системі автопілотування потрібно стати, грубо кажучи, на порядок розумнішим і вдвічі дешевшим, ніж зараз. Це не виглядає близькою перспективою. Так що помічників з елементами автопілотування (автопарковка, авторух у пробці, рух по трасі з розміткою або навіть конвойні режими) у найближчі пару-трійку років буде дедалі більше. Автовиробники здешевлять та масштабують ці функції. Toyota пропонує вже свій найпростіший Safety Sense лише за 500-1000 доларів.

Адже в складних ситуаціях (тобто 20 відсотків поїздки) управління на довгі роки залишиться на людині. «Звикати до абревіатури ADAS, яка у найближчі п'ять років стане такою ж звичною, як стали ABS та ESP. Після 2020 року слід

чекати активного держрегулювання у цій галузі, системи активної допомоги стануть обов'язковими у Європі та США»[5]. І розуміють, що повноцінного автопілота поки що на ринку немає, і взагалі у них наразі непереборні проблеми.

Виявлено наступні проблеми:

Датчики. Головне уповільнення розвитку автопілотів не є алгоритмом, а поганим розумінням машиною навколишнього простору. Інформація з сучасних датчиків категорично недостатня та вдосконалення механізму прийняття рішень (нейросети тощо) мало що змінює. Лідери, що сканують (чорні відра, що стоять на більшості прототипів, включаючи Google Car), позамежні дороги для серійних автомобілів і не дають необхідної для впевненого руху повноти інформації.

Не універсальність. Аналогічно до Закону Парето, в автопілотах він теж виконується. 80% дорожніх ситуацій машини розуміють, 20%, що залишилися, все псують.

Ринкова. Для автопілота важливо бути не тільки функціональним, та й дешевим. Автомобільну навігацію зробили більш аніж 20 лет назад. Але при ціні системи у дві-три тисячі доларів вона по цей день ставиться менше чим на 20% всіх нових автомобілях. Само керуючий функціонал ще дорожче.

Метод ідентифікації об'єктів у реальному часі

Виявлення об'єктів – це процес пошуку екземплярів реальних об'єктів, таких як автомобіль, велосипед, телевізор, квіти та люди, на нерухомих Рисунокх або відео. Він дозволяє розпізнавати, локалізувати та виявляти кілька об'єктів у зображенні, що дає нам набагато краще розуміння Рисунок загалом. Він зазвичай

використовується в таких програмах, як пошук зображень, безпека, спостереження та сучасні системи допомоги водієві (ADAS).

Виявлення об'єктів може бути виконано декількома способами:

Виявлення об'єктів на основі функцій

Віола Джонс Виявлення об'єктів

Класифікації SVM із функціями HOG

Виявлення об'єктів глибокого навчання

Розглянемо застосування різними додатками у галузі.

Розпізнавання осіб:

Група дослідників з Facebook розробила систему розпізнавання обличь із глибоким навчанням під назвою «DeepFace», яка дуже ефективно ідентифікує людські особи на цифровому зображенні. Google використовує власну систему розпізнавання обличь у Google Фотографії, яка автоматично розділяє всі фотографії залежно від людини на зображенні. У розпізнаванні осіб задіяні різні складові, такі як очі, ніс, рот та брови.[7]

Рисунок 1.7 – Розпізнавання осіб

Підрахунок людей:

Виявлення об'єктів також можна використовувати для підрахунку відвідувачів, які використовуються для аналізу роботи магазинів або статистики

натовпу під час фестивалів. Це, як правило, ускладняються, тому що люди скоро виходять із кадру.

Це дуже важлива програма, так як під час збору натовпу цю функцію можна використовувати для кількох цілей.[7]

Рисунок 1.8 – Підрахунок людей

Перевірка промислової якості:

Виявлення об'єктів також використовується у промислових процесах для ідентифікації продуктів. Виявлення конкретного об'єкта за допомогою візуального огляду – це основне завдання, яке задіяне у багатьох промислових процесах, таких як сортування, управління запасами, обробка, управління якістю, упаковка тощо.

Управління запасами то, можливо дуже складним, оскільки предмети важко відстежувати як реального часу. Автоматичний підрахунок та локалізація об'єктів дозволяє підвищити точність інвентаризації. [7]

Рисунок 1.9 – Перевірка промислової якості

Самостійне керування автомобілем:

Безпілотні автомобілі – це майбутнє, у цьому немає жодних сумнівів. Але робота, що стоїть за ним, дуже складна, оскільки вона поєднує різні способи сприйняття навколишнього середовища, включаючи радар, лазерне світло, GPS, одометрію і комп'ютерний зір.

Удосконалені системи управління інтерпретують сенсорну інформацію для визначення відповідних шляхів навігації, а також перешкод, і як датчик Рисунок виявляє будь-які ознаки живої істоти на своєму шляху, він автоматично зупиняється. Це відбувається дуже швидко і є величезним кроком до створення безпілотних автомобілів.[7]

Рисунок 1.10 – Самостійне керування автомобілем

Безпека:

Виявлення об'єктів грає дуже важливу роль у безпеці. Будь то ідентифікатор обличчя Apple або скан сітківки ока, що використовується у всіх науково-фантастичних фільмах.

Він також використовується урядом для доступу до каналу безпеки та зіставлення його з існуючою базою даних, щоб знайти злочинців або виявити автомобіль грабіжників.[7]

Рисунок 1.11 – Безпека

1.5 Метод ідентифікації відстані

RFID (Radio Frequency IDentification, радіочастотна ідентифікація) - спосіб автоматичної ідентифікації об'єктів, в якому за допомогою радіосигналів зчитуються або записуються дані, що зберігаються в так званих транспондер або RFID-мітках.[10]

Будь-яка RFID-система складається зі зчитувального пристрою (зчитувач, рідер або Інтеррогатор) і транспондера (він же RFID-мітка, іноді також застосовується термін RFID-тег).[8]

За дальністю зчитування RFID-системи можна поділити на системи:

- ближню ідентифікацію (зчитування проводиться на відстані до 20 см);
- ідентифікації середньої дальності (від 20 см до 5 м);
- дальньої ідентифікації (від 5 м до 300 м).

Класифікація RFID-міток. Існує кілька способів систематизації RFID-міток та систем:

- За робочою частотою
- За джерелом живлення
- За типом пам'яті

По виконанню:

- За джерелом живлення

За типом джерела живлення RFID-мітки поділяються на:

- Пасивні
- Активні
- Напівпасивні

За типом використовуваної пам'яті RFID-мітки поділяються на :

RO (Read Only) - дані записуються лише один раз, відразу при виготовленні. Такі позначки придатні лише для ідентифікації. Жодну нову інформацію в них записати не можна, і їх практично неможливо підробити.

WORM (Write Once Read Many) - крім унікального ідентифікатора такі мітки містять блок пам'яті, що одноразово записується, яку в подальшому можна багаторазово читати.

RW (Read and Write) - такі мітки містять ідентифікатор і блок пам'яті для читання/запису інформації. Дані в них можуть бути багаторазово перезаписані.

За робочою частотою:

- Мітки діапазону LF (125-134 кГц)
- Мітки діапазону HF (13,56 МГц)
- Мітки діапазону UHF (860-960 МГц)
- Радіочастотні UHF-мітки ближнього поля

Переваги радіочастотної ідентифікації:

- Можливість перезапису
- Відсутність потреби у прямій видимості
- Велика відстань читання
- Більший обсяг зберігання даних.
- Підтримка читання кількох міток.
- Зчитування даних мітки за будь-якого її розташування.
- Стійкість до дії довкілля.
- Багатоцільове використання.
- Високий рівень безпеки.

Недоліки радіочастотної ідентифікації:

- Працездатність мітки втрачається при приватному механічному пошкодженні.
- Вартість системи вище за вартість системи обліку, заснованої на штрих-кодах.
- Схильність до перешкод у вигляді електромагнітних полів.

- Недовіра користувачів, можливість використання її для збору інформації про людей.
- Недостатня відкритість вироблених стандартів.

Характеристика технологии	RFID	Штрих-код	QR-код
Необходимость в прямой видимости метки	Чтение даже скрытых меток	Чтение без прямой видимости невозможно	Чтение без прямой видимости невозможно
Объём памяти	От 10 до 512 000 байт	До 100 байт	До 3 072 байт
Возможность перезаписи данных и многократного использования метки	Есть	Нет	Нет
Дальность регистрации	До 100 м	До 4 м	До 1 м
Одновременная идентификация нескольких объектов	До 200 меток в секунду	Невозможна	Зависит от считывателя
Устойчивость к воздействиям окружающей среды: механическому, температурному, химическому, влаге	Повышенная прочность и сопротивляемость	Зависит от материала, на который наносится	Зависит от материала, на который наносится
Срок жизни метки	Более 10 лет	Зависит от способа печати и материала, из которого состоит отмечаемый объект	Зависит от способа печати и материала, из которого состоит отмечаемый объект
Безопасность и защита от подделки	Подделать возможно	Подделать легко	Подделать легко
Работа при повреждении метки	Невозможна	Затруднена	Затруднена
Идентификация движущихся объектов	Да	Затруднена	Затруднена
Подверженность помехам в виде электромагнитных полей	Есть	Нет	Нет
Идентификация металлических объектов	Возможна	Возможна	Возможна
Использование как стационарных, так и ручных терминалов для идентификации	Да	Да	Да
Возможность введения в тело человека или животного	Возможна	Затруднена	Затруднена
Габаритные характеристики	Средние и малые	Малые	Малые
Стоимость	Средняя и высокая	Низкая	Низкая

Таблиця 1.1 – Порівняння з іншими універсальними ідентифікаторами

1.6 Висновки за розділом 1

Виходячи з технічного завдання і результатів аналізу подібних рішень в роботі буде розроблена інтегрована система для “розумного паркування” автомобіля. Процесс розробки моделі складатиметься з трьох частин:

- Розробка методу розпізнавання об'єктів у реальному часі;
- Розробка методу ідентифікації відстані;
- Реалізація само паркування стенду на Raspberry Pi 4.

ПОБУДОВА МЕТОДІВ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ

2.1 Побудова методу розпізнавання об'єктів у реальному часі

Реалізація роботи системи розпізнавання об’єктів у реальному часі було розроблено на основі методу YOLO.[9]

Позначимо, що розмірами зображення є W,H. Візьмемо сітку розміром S×S, тоді розмір комірки буде $\mathrm{W_c} = {\mathrm{W}}/{\mathrm{S}}, \mathrm{H_c} = {\mathrm{H}}/{\mathrm{S}}$. Центр нашого об’єкту (в системі координат: (C_x', C_y')) потрапляє до комірки (3:2), розміри об'єкта: (W′,H′). Центр об'єкта нам потрібен у координатах від лівого верхнього кута комірки, а це буде:

$$C_x = C_x^{`} - 3 \cdot W_c,$$
$$C_y = C_y^{`} - 2 \cdot H_c, \qquad (2.1)$$

Якщо об'єкт на зображенні один, то всі комірки крім однієї (3:2) будуть утримувати нулі. А для комірки, де центр нашої ворони, замість нулів у тензорі буде:

1. П’ятірка $\langle \hat{x}, \hat{y}, \hat{w}, \hat{h}, 1 \rangle$:

$$\hat{x} = {C_x}/{W_c},$$
$$\hat{y} = {C_y}/{H_c},$$
$$\hat{w} = {W^{`}}/{W},$$
$$\hat{h} = {H^{`}}/{H} \qquad (2.2)$$

2. Вектор ймовірностей для класів: $\hat{p}$=(0,…,0,1,0,…,0) - з одиницею дома класу птиці і нулями інших.

Штрафна функція логічним чином розбивається на суму штрафів по всіх комірках зображення:

$$\mathcal{L} = \sum_{i=1}^{S \cdot S} \mathcal{L}(i) \qquad (2.3)$$

Осередки у нас двох типів: які містять і не містять центр об'єкта, і штрафувати їх треба по-різному. Формалізуємо цю відмінність і введемо індикаторну функцію:

$$x_i = \begin{cases} 1, & i - \text{я комірка, що містить об}'\text{єкт} \\ 0, & i - \text{я комірка, що не містить об}'\text{єкт} \end{cases}$$

Де i=1,…, (S·S).

Для комірок у яких є об'єкт, позначимо його прямокутником:

$$\widehat{R_i} = \left(\hat{x}_i, \hat{y}_i, \widehat{w_i}, \widehat{h_i}\right) \tag{2.4}$$

При цьому для кожної комірки у нас є провісники, кожен з яких видає свій прямокутник. Позначимо прямокутник від j-го провісника в i-ї комірки:

$$R_{ij} = \left(x_{ij}, y_{ij}, w_{ij}, h_{ij}\right) \tag{2.5}$$

Якщо комірка містить центр об'єкта, то мережа повинна правильно передбачити клас цього об'єкта, тому першим складником у штрафній функції комірки буде штраф за класифікацію:

$$\mathcal{L}_{class}(i) = x_i \cdot \sum_{cl \in classes} (p_i(cl) - \widehat{p_i}(cl))^2 \tag{2.6}$$

Де χi - залишає це доданок тільки для комірок, де є об'єкт.

pi(cl),cl=1,…,C - передбачена ймовірність приналежності об'єкта класу cl.

$\widehat{p_i}$(cl) - одиниця для дійсного класу в який розмічений об'єкт у комірці, і нуль всім інших класів.

Якщо комірка містить центр об'єкта, то "кращий" провісник (той у якого максимальний IoU передбаченого прямокутника з розміченим) повинен добре

передбачити координати центру та розміри прямокутника, решта провісників штрафувати не будемо:

$$\mathcal{L}_{coord}(i) = \sum_{j=1}^{B} x_{ij} \cdot \mathcal{L}_{coord}(i,j) \tag{2.7}$$

Штраф кращому провіснику задається як:

$$\mathcal{L}_{coord}(i,j) = \left(x_{ij} - \widehat{x_{\imath}}\right)^2 + \left(y_{ij} - \widehat{y_{\imath}}\right)^2 + \left(\sqrt{w_{ji}} - \sqrt{\widehat{w_{\imath}}}\right)^2 + \left(\sqrt{h_{ji}} - \sqrt{\widehat{h_{\imath}}}\right)^2 \tag{2.8}$$

Оскільки χ_{ij} не нуль, тільки для комірок, які містять об'єкт, $L_{coord}(i)$ ненульова, також тільки для таких комірок. Також квадратне коріння в розмірах введено, щоб однакові помилки розмірів для маленьких і великих прямокутників штрафувалися по-різному

За впевненість будемо штрафувати по-різному у випадку, якщо в осередку є центр об'єкта і якщо ні. Якщо є, штраф вважаємо як:

$$\mathcal{L}_{conf}(i) = \sum_{j=1}^{B} x_{ij} \cdot (C_{ij} - \widehat{C_{\imath j}})^2 \tag{2.9}$$

Де C_{ij} - це впевненість, яку видає мережа для і-ої комірки j-м провісником, а $\widehat{C_{\imath j}}$- це IoU між реальним прямокутником і тим, що видав провісник.

Якщо в осередку немає центру об'єкта, то штрафуємо:

$$\mathcal{L}_{noobj}(i) = (1 - x_i) \cdot \sum_{j=1}^{B} \left(C_{ij}\right)^2 \tag{2.10}$$

Об'єднаємо всі штрафи в одну штрафну функцію для комірок:

$$\mathcal{L}(i) = \mathcal{L}_{class}(i) + \lambda_{coord} \cdot \mathcal{L}_{coord}(i) + \mathcal{L}_{conf}(i) + \lambda_{noobj} \cdot \mathcal{L}_{noobj}(i) \qquad (2.11)$$

Щоб зробити детектор об'єктів у реальному часі, треба: [11]

- Отримати доступ до веб-камери/відео потоку.
- Застосувати розпізнавання об'єкта кожного кадру.

Щоб подивитися, як це робиться, відкриємо новий файл, назвемо real_time_object_detection.py і вставте наступний код, що зображений на рисунку 2.1.

```
# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import time
import cv2
```

Рисунок 2.1 – Фрагмент коду

Пишемо код для роботи з командним рядком.

Далі аналізуємо аргументи командного рядка (рисунок 2.2).

```
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
        help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
        help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.2,
        help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

Рисунок 2.2 – Фрагмент коду

--prototxt : Шлях до файлу prototxt Caffe.

--model : Шлях до попередньо підготовленої моделі.

--confidence : Мінімальний поріг валідності (подібності) для розпізнавання об'єкта (за замовчуванням – 20%).

Додаємо основні об'єкти.

Потім ініціалізуємо перелік класів. (рисунок 2.3).[14]

```
# initialize the list of class labels MobileNet SSD was trained to
# detect, then generate a set of bounding box colors for each class
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
        "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
        "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
        "sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
```

Рисунок 2.3 – Фрагмент коду

Ініціалізуємо мітки CLASS та відповідні випадкові кольори.

Тепер завантажимо модель та налаштуємо наш відео потік (рисунок 2.4).

```
# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# initialize the video stream, allow the cammera sensor to warmup,
# and initialize the FPS counter
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
fps = FPS().start()
```

Рисунок 2.4 – Фрагмент коду

Потім ініціалізуємо відео потік (це може бути відео або веб-камера). Спочатку запускаємо VideoStream, потім чекаємо, поки камера увімкнеться, і починаємо відлік кадрів за секунду. Класи VideoStream та FPS є частиною пакету imutils.

Пишемо код для роботи із кадрами.

Тепер проходимо по кожному кадру (рисунок 2.5).[12]

```
while True:
        # grab the frame from the threaded video stream and resize it
        # to have a maximum width of 400 pixels
        frame = vs.read()
        frame = imutils.resize(frame, width=400)

        # grab the frame dimensions and convert it to a blob
        (h, w) = frame.shape[:2]
        blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
                0.007843, (300, 300), 127.5)

        # pass the blob through the network and obtain the detections and
        # predictions
        net.setInput(blob)
        detections = net.forward()
```

Рисунок 2.5 – Фрагмент коду

Перше – зчитуємо кадр із потоку, потім замінюємо його розмір.

Оскільки трохи згодом знадобиться ширина та висота, отримаємо зараз. Потім слід перетворення кадру на blob з модулем dnn. Тепер складне: ми встановлюємо blob як вхідні дані в нашу нейромережу та передаємо ці дані через net, яка виявляє наші предмети. "Фільтруємо" об'єкти.

На даний момент ми знайшли об'єкти у відео потоці. Тепер настав час подивитися на значення валідності і вирішити, що повинні амалювати квадрат навколо об'єкта і повісити лейбл (рисунок 2.6).[13]

```
# loop over the detections
for i in np.arange(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated with
        # the prediction
        confidence = detections[0, 0, i, 2]

        # filter out weak detections by ensuring the `confidence` is
        # greater than the minimum confidence
        if confidence > args["confidence"]:
                # extract the index of the class label from the
                # `detections`, then compute the (x, y)-coordinates of
                # the bounding box for the object
                idx = int(detections[0, 0, i, 1])
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")
           # draw the prediction on the frame
           label = "{}: {:.2f}%".format(CLASSES[idx],
                   confidence * 100)
           cv2.rectangle(frame, (startX, startY), (endX, endY),
                   COLORS[idx], 2)
           y = startY - 15 if startY - 15 > 15 else startY + 15
           cv2.putText(frame, label, (startX, y),
                   cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
```

Рисунок 2.6 – Фрагмент коду

Починаємо проходити циклами через наші detections, пам'ятаючи, що кілька об'єктів можна сприйняти як єдине Рисунок. Також робимо перевірку на валідність (тобто ймовірність) для кожного виявлення. Якщо валідність досить

велика (тобто вище заданого порогу), відображаємо пророцтво в терміналі, а також малюємо на потоці пророцтво (обводимо об'єкт у кольоровий прямокутник і вішаємо лейбл).

Давайте розберемо по рядках:

Проходимо по detections, отримуємо значення валідності).

Якщо значення валідності вище за заданий поріг, витягуємо індекс лейбла в класі і вираховуємо координати рамки навколо виявленого об'єкта .

Потім витягаємо (x;y)-координати рамки, які будемо використовувати для відображення прямокутника та тексту.

Робимо текстовий лейбл, що містить ім'я з CLASS та значення валідності.

Також, малюємо кольоровий прямокутник навколо об'єкта, використовуючи кольори класу і раніше вилучені (x; y)-координати.

В цілому, потрібно, щоб лейбл розташовувався над кольоровим прямокутником, однак може виникнути така ситуація, що зверху буде недостатньо місця, тому в таких випадках виводимо лейбл під верхньою стороною прямокутника.

Накладаємо кольоровий текст і рамку на кадр, використовуючи значення 'y', яке щойно вирахували.

2.2 Побудова методу ідентифікації відстані

Ультразвуковий датчик HC-SR04, який будемо використовувати в цьому посібнику для Raspberry Pi, має чотири контакти: заземлення (GND), вихід ехо-імпульсу (ECHO), тригерний імпульсний вхід (TRIG) і живлення 5 В (Vcc). Живимо модуль за допомогою Vcc, заземлюємо його за допомогою GND і використовуємо Raspberry Pi для відправки вхідного сигналу на TRIG, який

запускає датчик для відправлення ультразвукового імпульсу. Пульсові хвилі відбиваються від будь-яких прилеглих об’єктів, а деякі відбиваються назад до датчика. Датчик виявляє ці зворотні хвилі і вимірює час між тригером і зворотним імпульсом, а потім посилає сигнал 5 В на контакт ECHO.[15]

ECHO буде «низьким» (0 В), доки датчик не спрацює, коли він отримає ехо-імпульс. Після визначення зворотного імпульсу ECHO встановлюється «високий» (5 В) на тривалість цього імпульсу. Тривалість імпульсу – це повний час між датчиком, який видає ультразвуковий імпульс, і зворотним імпульсом, виявленим приймачем датчика. Тому скрипт Python повинен виміряти тривалість імпульсу, а потім обчислити відстань від цього.

ВАЖЛИВО. Вихідний сигнал датчика (ECHO) на HC-SR04 розрахований на 5 В. Однак вхідний контакт на Raspberry Pi GPIO розрахований на 3,3 В. Надсилання сигналу 5 В у цей незахищений вхідний порт 3,3 В може пошкодити ваші контакти GPIO. Потрібно буде використовувати невелику схему подільника напруги, що складається з двох резисторів, щоб знизити вихідну напругу датчика до того, що може впоратися наш Raspberry Pi.

Дільники напруги

Дільник напруги складається з двох резисторів (R1 і R2), послідовно підключених до вхідної напруги (Vin), яку потрібно зменшити до нашої вихідної напруги (Vout). У нашій схемі Vin буде ECHO, який потрібно зменшити з 5 В до нашого Vout 3,3 В. Схема електрична принципова зображена на рисунку 2.6.

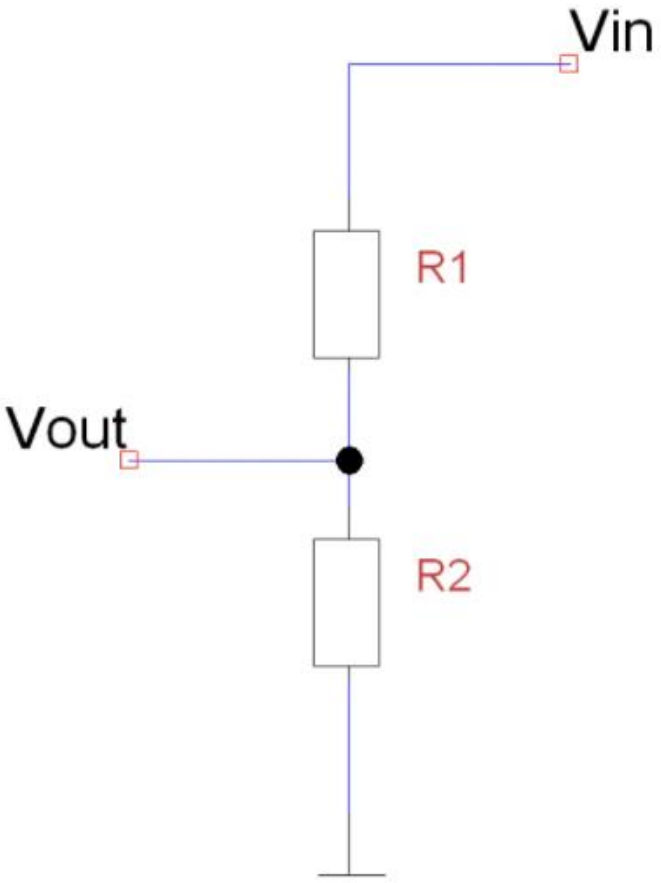

Рисунок 2.6 – Дільники напруги

Наступну схему та просте рівняння можна застосувати до багатьох застосувань, де потрібно зменшити напругу.

$$V_{out} = V_{in} * \frac{R2}{R1 + R2}$$

$$\frac{V_{out}}{V_{in}} = \frac{R2}{R1 + R2} \qquad (2.12)$$

Якщо розраховувати використовуємі значення, це буде наступне:

Отже, будемо використовувати 1kΩ для R1 і резистор 2kΩ як R2.

Схема підключення зображена на рисунку 2.7.

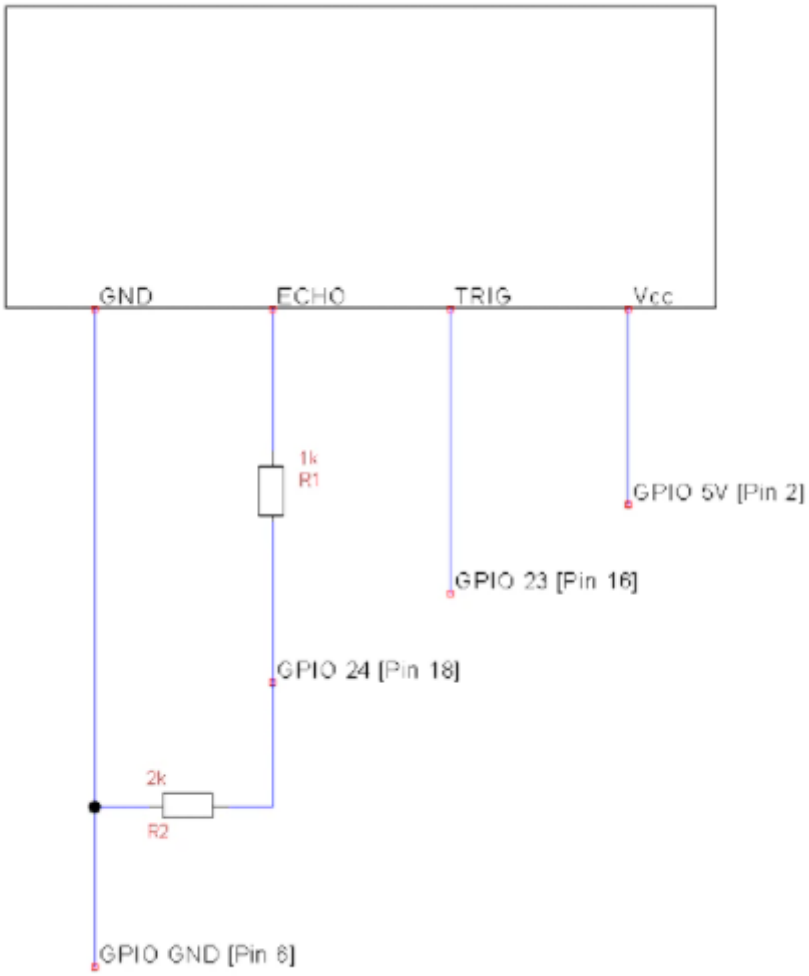

Рисунок 2.7 – Схема підключення

Для цього проекту буде використовуватися чотири контакти Raspberry Pi: GPIO 5V [Pin 2]; Vcc (живлення 5 В), GPIO GND [контакт 6]; GND (0В заземлення), GPIO 23 [контакт 16]; TRIG (вихід GPIO) і GPIO 24 [контакт 18]; ECHO (вхід GPIO)

Зібрана схема має наступний вигляд (рисунок 2.8):

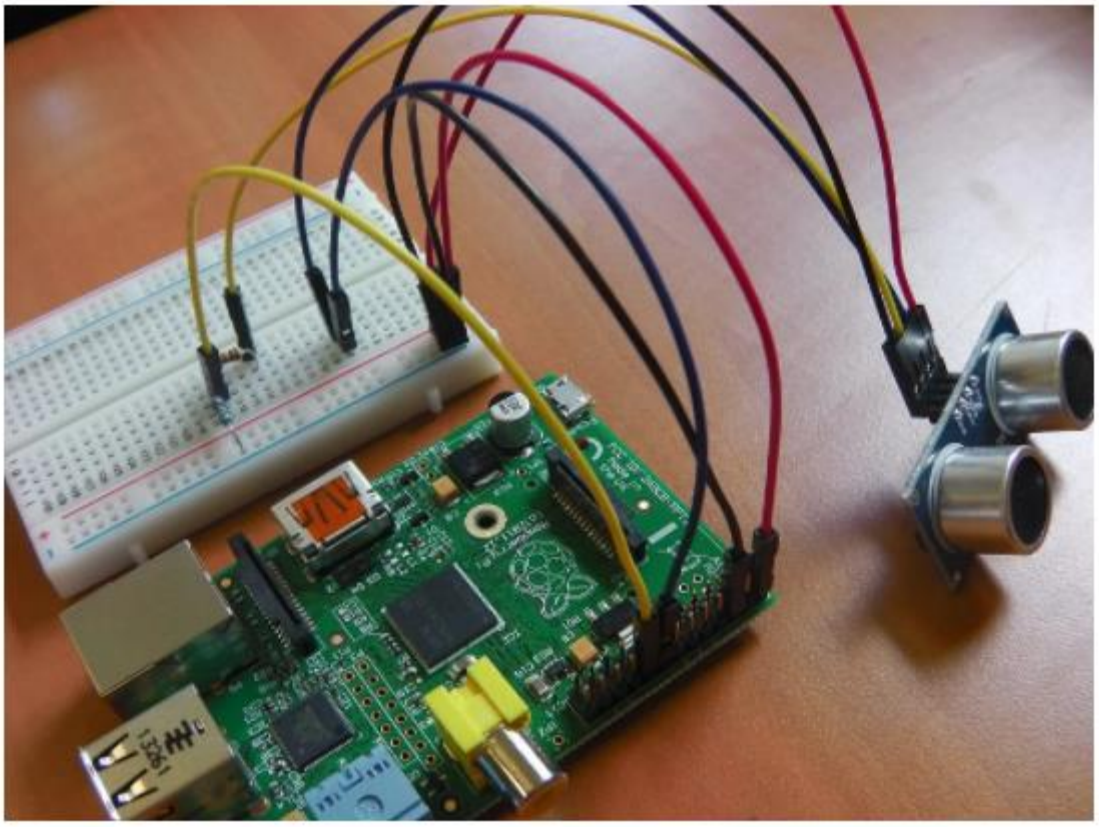

Рисунок 2.8 – Зібрана схема

Тепер, коли підключили ультразвуковий датчик до Pi, потрібно запрограмувати Python скрипт для визначення відстані.

Щоб зрахувати дані з далекоміра HC-SR04, використовується бібліотека для роботи з GPIO – pigpio.

Для роботи з pigpio необхідно запустити відповідний демон:

sudo systemctl start pigpiod.service

Також включаємо автоматичний запуск pigpio під час старту системи:

sudo systemctl enable pigpiod.service

Таким чином стає можливою взаємодія з демоном Python:

```
import pigpio
pi = pigpio.pi()
```

Програма для читання даних з HC-SR04:

```
import time
import threading
import pigpio

TRIG = 23  # пин, к которому подключен контакт Trig дальномера
ECHO = 24  # пин, к которому подключен контакт Echo дальномера

pi = pigpio.pi()
done = threading.Event()

def rise(gpio, level, tick):
    global high
    high = tick

def fall(gpio, level, tick):
    global low
    low = tick - high
    done.set()

def read_distance():
    global low
    done.clear()
    pi.gpio_trigger(TRIG, 50, 1)
    if done.wait(timeout=5):
        return low / 58.0 / 100.0
```

```
pi.set_mode(TRIG, pigpio.OUTPUT)
pi.set_mode(ECHO, pigpio.INPUT)
pi.callback(ECHO, pigpio.RISING_EDGE, rise)
pi.callback(ECHO, pigpio.FALLING_EDGE, fall)

while True:
    # Читаем дистанцию:
    print(read_distance())
```

Для фільтрації (згладжування) даних та видалення викидів використовується фільтр Калмана або простіший медіанний фільтр. Розглянемо реалізацію медіанної фільтрації:

```
import collections
import numpy

# ...

history = collections.deque(maxlen=10)  # 10 - количество сэмплов для усреднения

def read_distance_filtered():
    history.append(read_distance())
    return numpy.median(history)

while True:
    print(read_distance_filtered())
```

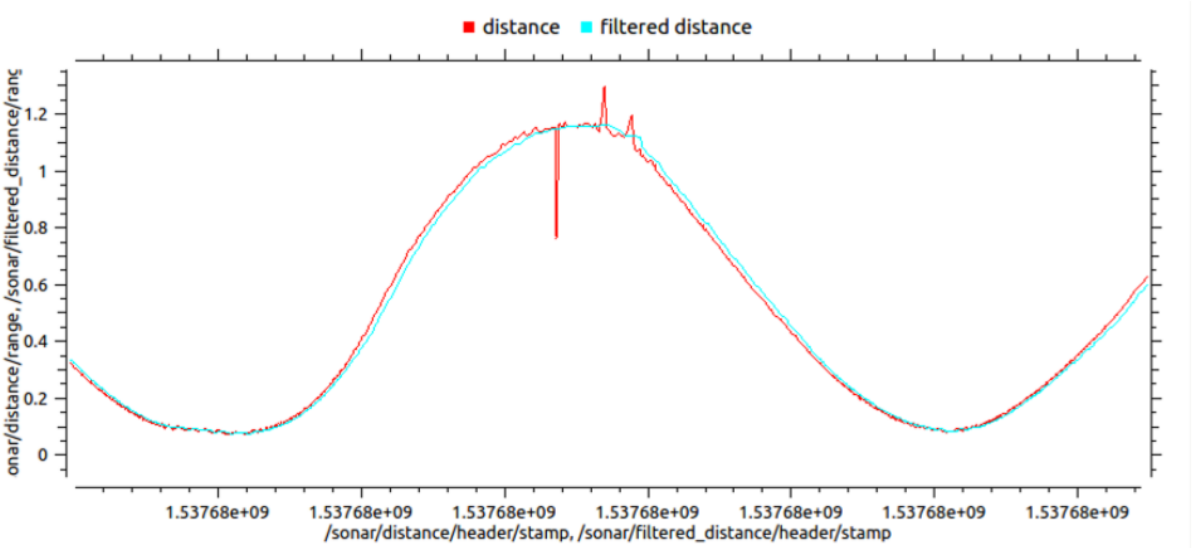

Рисунок 2.9 – Графік вихідних та відфільтрованих даних

2.3 Висновки за розділом 2

У даному розділі було вибрано технології, системи та методи для реалізації підвищення ефективності, а саме точності автоматичного паркування. Буде інтегровано дві системи в єдину. Стендове авто буде зібрано на основі Raspberry Pi. Автоматичний паркінг буде реалізовано за допомогою єдиного результату вимірювань з камери та датчику відстані. Після отримання результату середовища, оцінки об’єктів та відстані до них - буде здійснене автоматичне паркування.

ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

3.1 Тестування методу розпізнавання об'єктів у реальному часі

Під’єднавши модуль камери до Raspberry Pi запускаємо її.

Переходимо до головного меню та відкриваємо інструмент налаштування Raspberry Pi (рисунок 3.1)

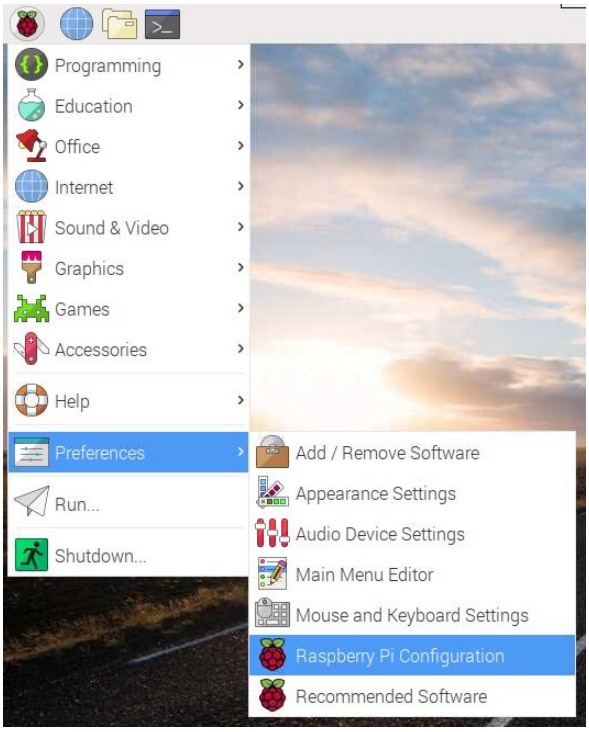

Рисунок 3.1 - Інструмент налаштування Raspberry Pi

Вибираємо вкладку Інтерфейси та переконуємося, що камера включена(рисунок 3.2):

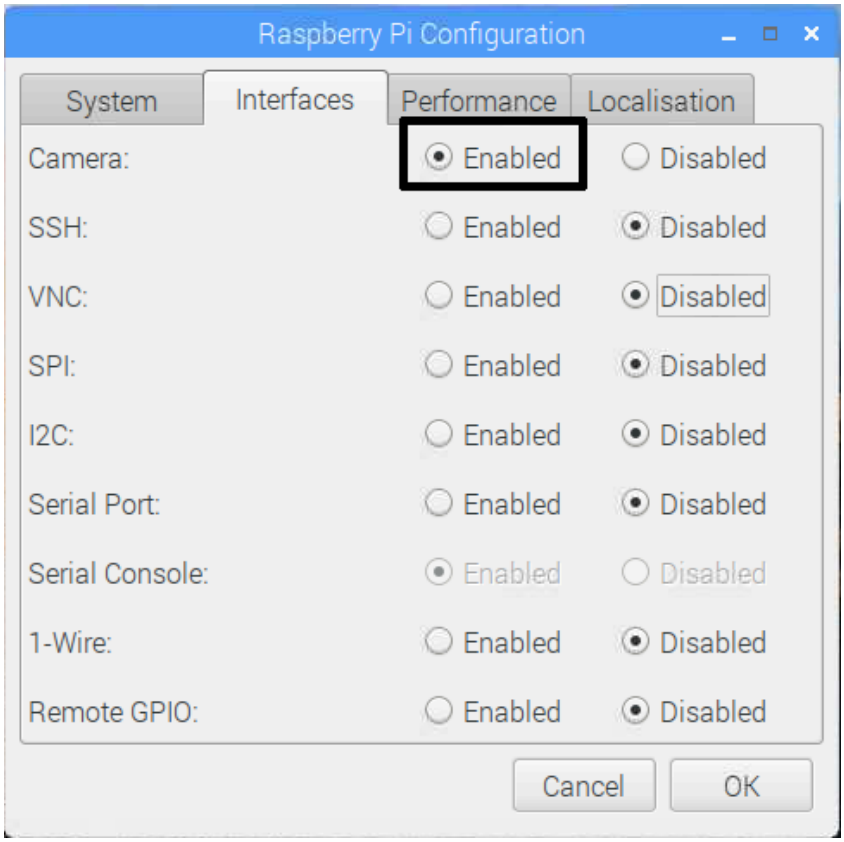

Рисунок 3.2 – Меню Інтерфейса

Перезавантажуємо Raspberry Pi.

Тепер модуль камери підключений та програмне забезпечення увімкнено.

Введемо наступну команду, щоб зробити знімок та зберегти його на робочому столі:

raspistill -o Desktop/image.jpg

Коли команда запускається, можна побачити попередній перегляд камери відкритим протягом п'яти секунд, перш ніж буде зроблено нерухоме Рисунок.

Додаючи різні параметри, можна встановити розмір і зовнішній вигляд Рисунок, отриманого raspistill командою.

Додамо –h та –w щоб змінити висоту та ширину Рисунок:

raspistill -o Desktop/image-small.jpg -w 640 -h 480

Тепер запишємо відео за допомогою модуля камери, використовуючи наступну raspivid команду:

raspivid -o Desktop/video.h264

Відкрити його можно у VLC Media Player.

Бібліотека Python picamera дозволяє керувати модулем камери.

Відкриємо редактор Python - Thonny PythonIDE(рисунок 3.3):

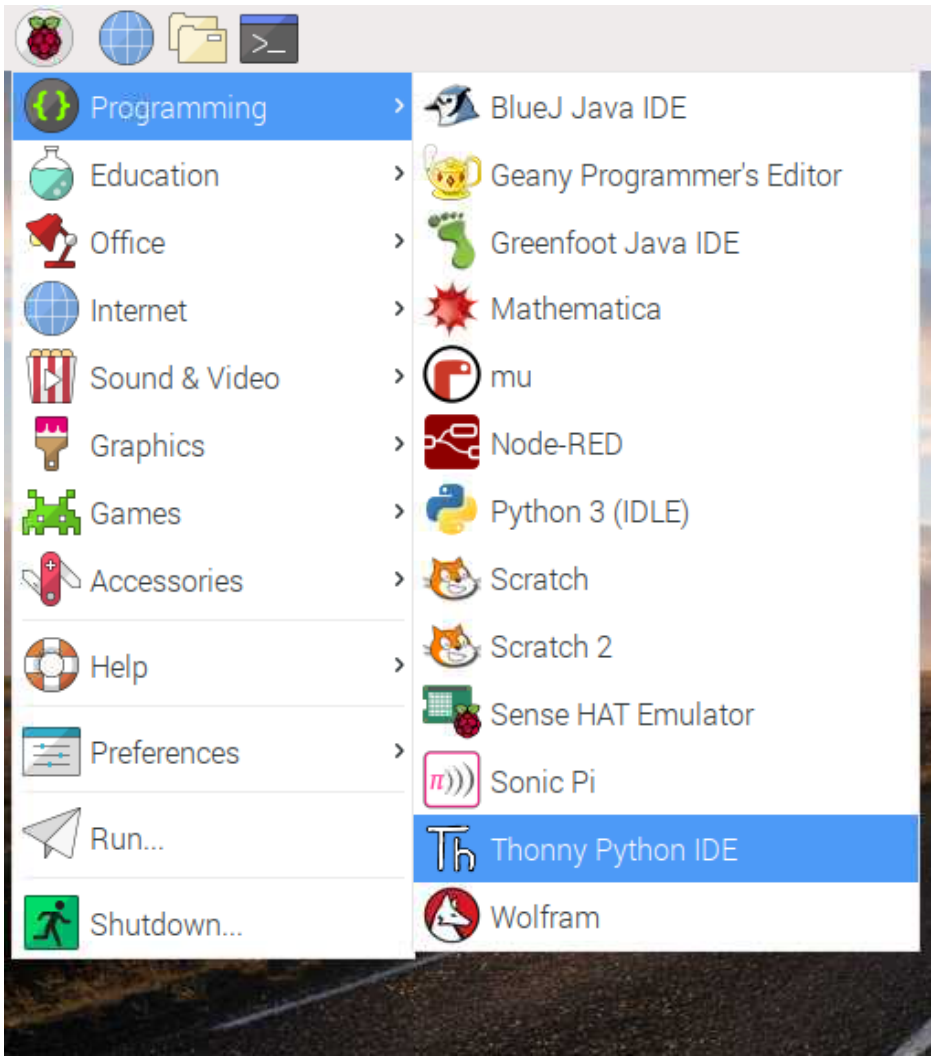

Рисунок 3.3 – Редактор Python

Відкриємо новий файл та збережемо його як camera.py.

Примітка: Важливо ніколи не зберігати файл у форматі picamera.py.

Введемо наступний код:

```
from picamera import PiCamera
from time import sleep

camera = PiCamera()

camera.start_preview()
sleep(5)
camera.stop_preview()
```

Збережемо та запустимо нашу програму. Попередній перегляд камери повинен відображатись протягом п'яти секунд, а потім знову закриватися.

Якщо наш попередній перегляд перевернутий, ми можемо повернути його на 180 градусів за допомогою наступного коду:

```
camera = PiCamera()
camera.rotation = 180
```

Можна повернути Рисунок на 90, 180 або 270 градусів. Щоб скинути Рисунок, встановимо rotationв 0градусах.

Зробимо прев'ю камери прозорою. встановив alpha рівень:

```
camera.start_preview(alpha=200)
```

Значення alpha може бути будь-яке число між 0 та 255.

Тепер використовуйте модуль камери та Python, щоб зробити кілька фотографій.

Додаємо camera.capture()

```
camera.start_preview()
sleep(5)
camera.capture('/home/pi/Desktop/image.jpg')
camera.stop_preview()
```

Важливо sleep зачекати не менше двох секунд, перш ніж зробити знімок, тому що це дає датчику камери час для визначення рівня освітленості.

Наше нове Рисунок має бути збережене на робочому столі.

Тепер додамо петлю, щоб зробити п'ять знімків поспіль:

```
camera.start_preview()
for i in range(5):
sleep(5)
camera.capture('/home/pi/Desktop/image%s.jpg' % i)
camera.stop_preview()
```

Змінна “I” підраховує, скільки разів цикл був виконаний, від 0 до 4. Таким чином, Рисунок зберігаються як image0.jpg, image1.jpg і так далі.

Введемо код знову та утримуємо модуль камери на місці.

Камера повинна робити знімок кожні п'ять секунд. Після того, як буде зроблено п'ятий знімок, перегляд закриється.

Для записання відео змінемо свій код, щоб видалити capture(), а замість цього треба додати start_recording()istop_recording()

Тепер код має виглядати так:

```
camera.start_preview()
camera.start_recording('/home/pi/Desktop/video.h264')
sleep(5)
camera.stop_recording()
camera.stop_preview()
```

Raspberry Pi має відкрити попередній перегляд, записати 5 секунд відео, а потім закрити попередній перегляд.

Python надає низку ефектів та конфігурацій для зміни зовнішнього вигляду ваших зображень.

Деякі настройки впливають лише на перегляд, але не на захоплене Рисунок, деякі впливають тільки на захоплене Рисунок, а багато інших впливають на обидва.

Встановимо роздільну здатність Рисунок
Можна змінити resolution Рисунок, яке робить модуль камери.

За замовчуванням роздільна здатність Рисунок відповідає роздільній здатності монітора. Максимальна роздільна здатність становить 2592×1944 для фотографій та 1920×1080 для відеозапису.

Для того щоб встановити максимальне значення відповідності і зробити знімок введемо наступний код.

Також потрібно встановити частоту кадрів, щоб включити цю максимальну роздільну здатність.

```
camera.resolution = (2592, 1944)
camera.framerate = 15
camera.start_preview()
sleep(5)
camera.capture('/home/pi/Desktop/max.jpg')
camera.stop_preview()
```

Додамо текст до нашого Рисунок

Можна додати текст до Рисунок за допомогою команди annotate_text.

Запустимо цей код, щоб спробувати:

```
camera.start_preview()
camera.annotate_text = "Hello world!"
sleep(5)
camera.capture('/home/pi/Desktop/text.jpg')
camera.stop_preview()
```

Встановити розмір тексту можна за допомогою наступного коду

```
camera.annotate_text_size = 50
```

Можна встановити будь-який розмір тексту від 6 до 160. Стандартний розмір 32.

Можна також змінити колір тексту.

Насамперед, додамо Color у свій import терміну вгорі програми:

```
from picamera import PiCamera, Color
```

Потім під import рядком змінимо решту коду, щоб він виглядав так:

```
camera.start_preview()
camera.annotate_background = Color('blue')
```

```
camera.annotate_foreground = Color('yellow')
camera.annotate_text = " Hello world "
sleep(5)
camera.stop_preview()
```

Введемо наступний код

```
git clone https://github.com/pjreddie/darknet
cd darknet
make
```

Підтягнемо файл попередньо тренованої ваги

```
wget https://pjreddie.com/media/files/yolov3.weights
```

Запускаємо детектор

```
./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```

Отриманий результат виглядає наступним чином(рисунок 3.4).

Рисунок 3.4 – Ідентифікований результат

3.2 Тестування методу ідентифікації відстані

Спочатку імпортується бібліотека Python GPIO, потів бібліотеку часу (щоб Рі чекав між кроками) та встановлюється нумерацію контактів GPIO.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
```

Потім потрібно назвати вхідні та вихідні контакти, щоб була можливість посилатися на них пізніше у нашому коді Python. Назвемо вихідний контакт (запуск датчика) GPIO 23 [контакт 16] як TRIG, а вхідний контакт (який зчитує зворотний сигнал від датчика) GPIO 24 [контакт 18] як ECHO.

```
    TRIG = 23
ECHO = 24
```

Потім надрукуємо повідомлення, щоб повідомити користувачеві про вимірювання відстані. . . .

```
print "Distance Measurement In Progress"
Встановлюємо два порти GPIO у вигляді входів або виходів,
як визначено раніше.
GPIO.setup(TRIG,GPIO.OUT)
GPIO.setup(ECHO,GPIO.IN)
```

Потім переконаємося, що тригерний штифт встановлений на низький рівень та дайте датчику час стабілізації.

```
GPIO.output(TRIG, False)
```

```
print "Waiting For Sensor To Settle"
time.sleep(2)
```

Тепер робиться обчислення різниці між двома записаними мітками часу, а отже, і тривалість імпульсу (pulse_duration).

```
pulse_duration = pulse_end - pulse_start
```

Швидкість звуку змінюється, залежно від того, через яке середовище він проходить, а також від температури цього середовища. Однак деякі спритні фізики розрахували швидкість звуку на рівні моря, тому береться базова лінія за 343 м/с.

Також потрібно розділити час на два, тому що те, що разраховані показники, насправді є часом, необхідним ультразвуковому імпульсу, щоб пройти відстань до об’єкта і назад. Можно спростити розрахунок, який буде завершено в скрипті Python наступним чином:

Інший розрахунок chan plough у коді Python:

```
distance = pulse_duration x 17150
```

Потім друкуємо відстань. Наведена нижче команда надрукує слово «Відстань:», за яким слідує змінна відстані, а потім одиниця вимірювання «см».

```
print "Distance:",distance,"cm"
```

В кінці очистіть GPIO, щоб переконатися, що всі входи/виходи скинуто

GPIO.cleanup()

Результат виковання програми зображено на рисунку 3.5.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)

TRIG = 23
ECHO = 24

print "Distance Measurement In Progress"

GPIO.setup(TRIG,GPIO.OUT)
GPIO.setup(ECHO,GPIO.IN)

GPIO.output(TRIG, False)
print "Waiting For Sensor To Settle"
time.sleep(2)

GPIO.output(TRIG, True)
time.sleep(0.00001)
GPIO.output(TRIG, False)

while GPIO.input(ECHO)==0:
  pulse_start = time.time()

while GPIO.input(ECHO)==1:
  pulse_end = time.time()

pulse_duration = pulse_end - pulse_start

distance = pulse_duration * 17150

distance = round(distance, 2)

print "Distance:",distance,"cm"

GPIO.cleanup()
pi@raspberrypi ~ $ sudo python range_sensor.py
Distance Measurement In Progress
Waiting For Sensor To Settle
Distance: 12.52 cm
pi@raspberrypi ~ $
```

Рисунок 3.5 – Результат

3.3 Тестування інтегрованої системи

Для перевірки працездатності розробленої системи було створено модель на базі Raspberry Pi 4 [11] (рисунок 3.6).

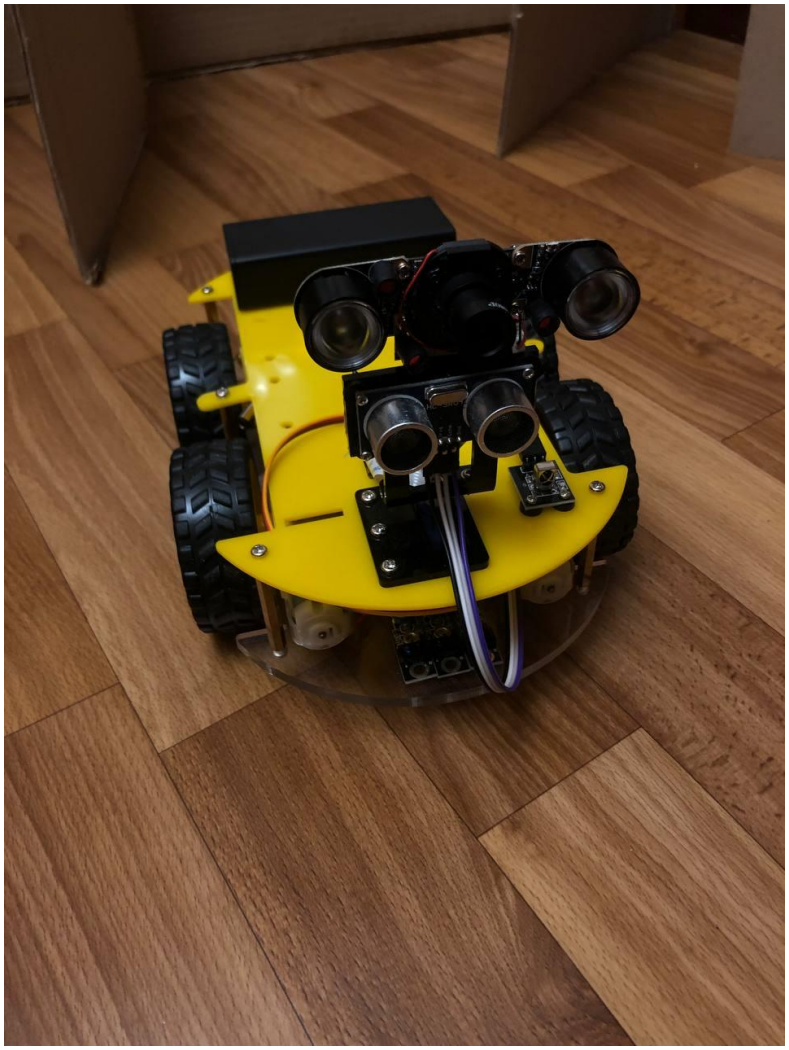

Рисунок 3.6 - Модель для тестування (вид з переду)

УЗ модуль та модуль камери були закріплені на одному й тому ж серво приводі для утримання їх в однакових умовах(рисунок 3.7).

Рисунок 3.7 - Модель для тестування (вид зі сторони)

Також було створено зону для тестування з об’єктами які повинні розпізнаватись як перешкоди та різними ділянками для ймовірного паркування (рисунок 3.8).

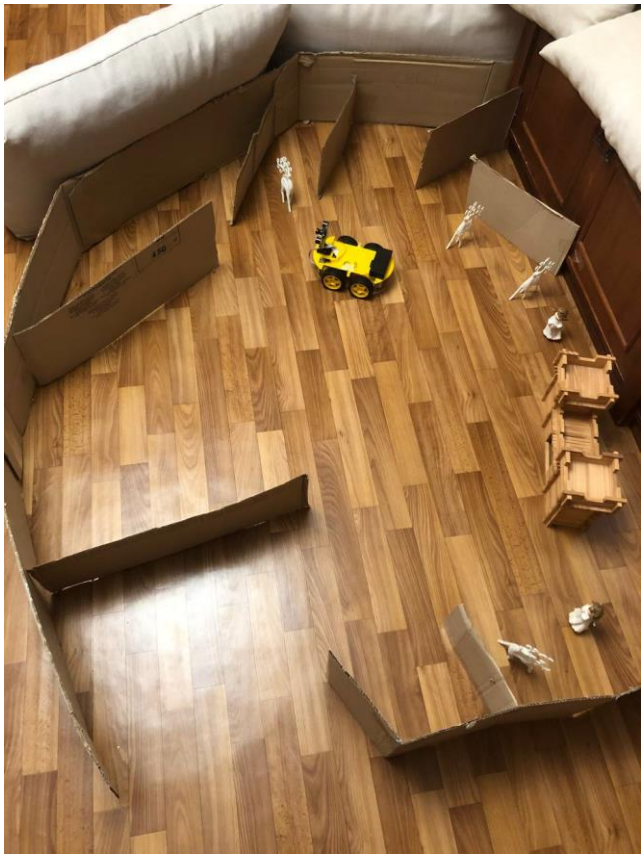

Рисунок 3.8 - Зона для тестування

Модель має самостійно проаналізувати можливі вільні місця спираючись на навколишнє середовище та завдяки модулів самостійно запаркуватися на вільну створену ділянку. Виглядає це наступним чином(рисунок 3.9).

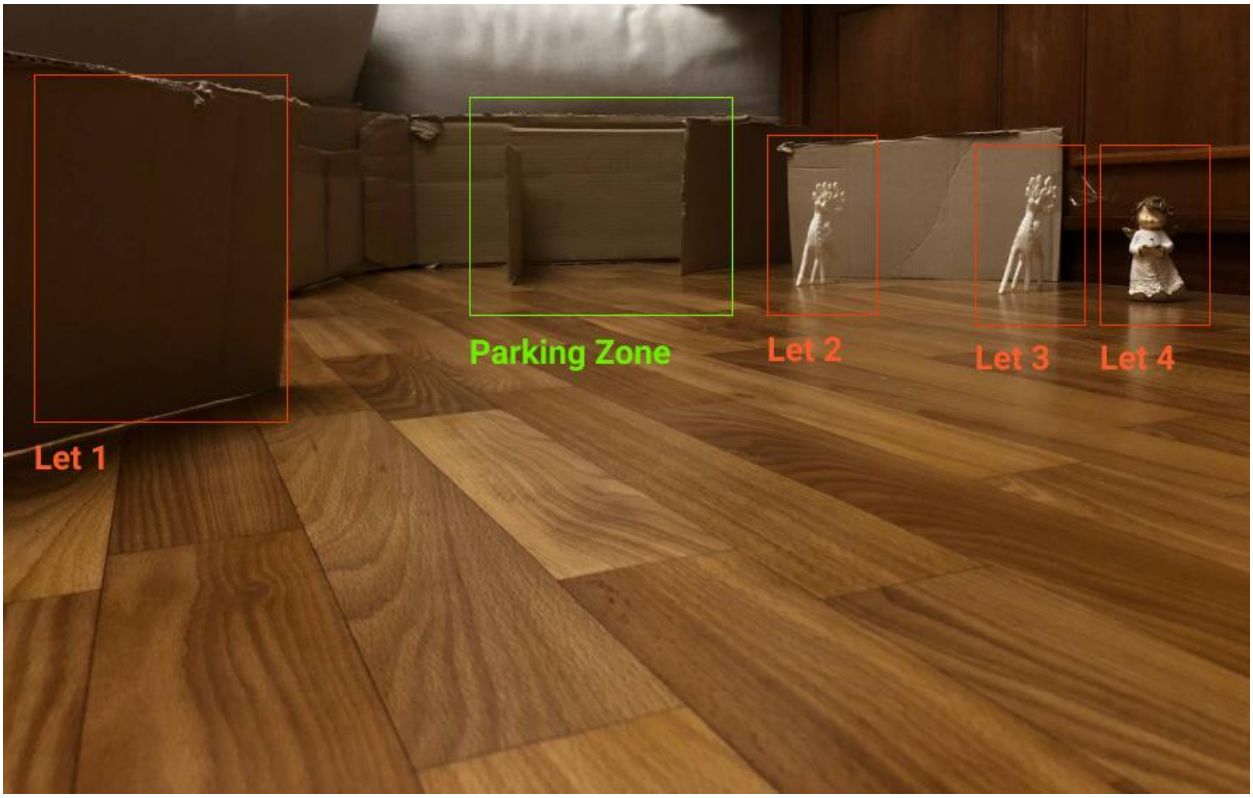

Рисунок 3.9 - Показники з камери

Заїхавши на зону паркування створена модель спочатку аналізує що вона бачить перед собою та виділити для себе місце де вона зможе запаркуватися.

На фото можна побачити розділення на “Let 1”, “Let 2 ”, “Let 3” та “Parking Zone”. Знайшовши для себе місце вона під’їжджає ближче при цьому постійно аналізуючи навколишнє середовище (рисунок 3.10).

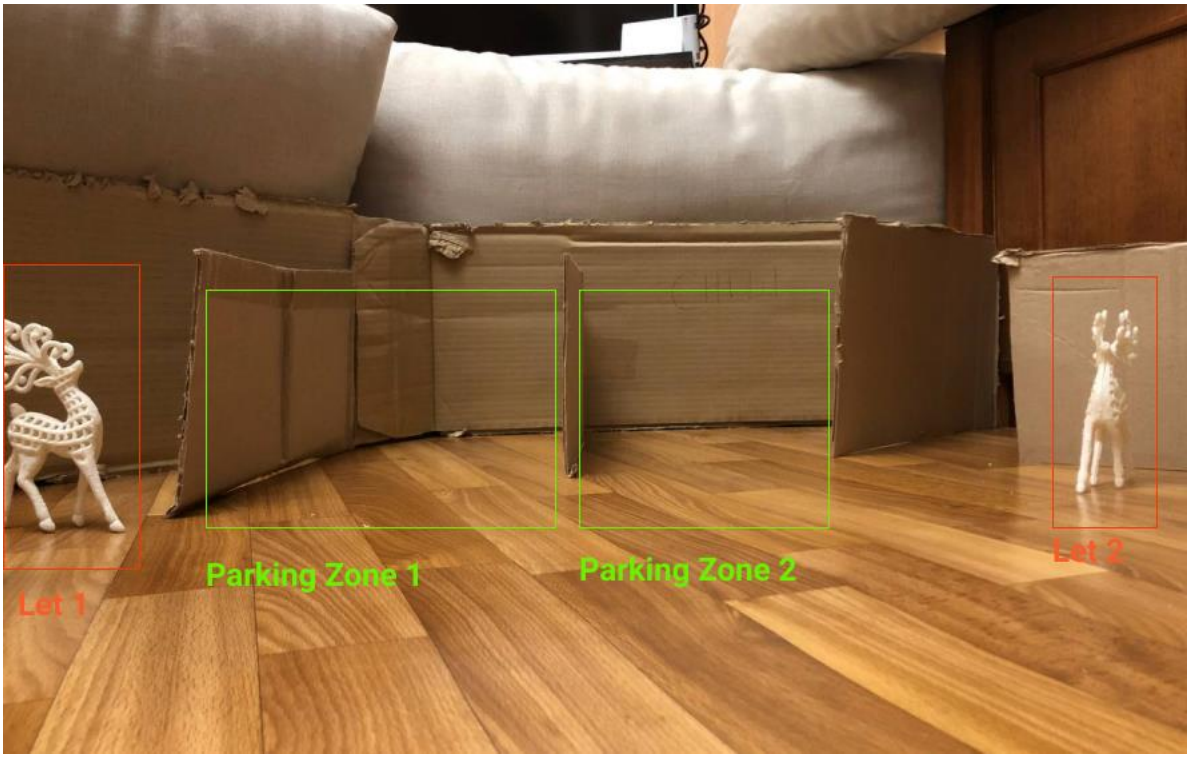

Рисунок 3.10 - Показники з камери

З початку була тільки одне місце але під’їхавши ближче, перегородка зникла з об’эктиву та з’явилась нова ділянка. Після чого модель починає паркуватися в найближче для неї місце (рисунок 3.11).

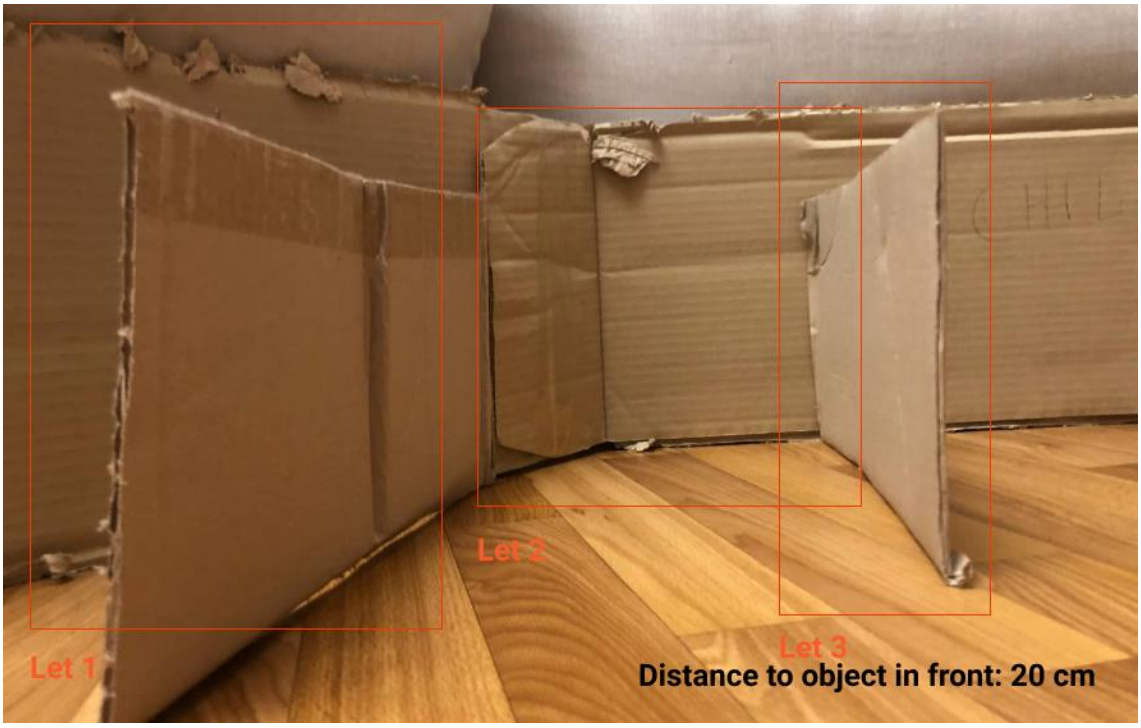

Рисунок 3.11- Інтегровані показники з камери

В цей час УЗ датчик починає передавати дані відстані до предмету попереду. Було би важко це зробити спираючись лише на камеру. Далі завдяки датчику УЗ модель потрохи під’їжджає на обране місце (рисунок 3.12).

Рисунок 3.12 - Інтегровані показники з камери

Якщо зненацька вискочить якийсь об’єкт – він буде розцінений як перешкода та камера почне шукати нове вільне місце (рисунок 3.13).

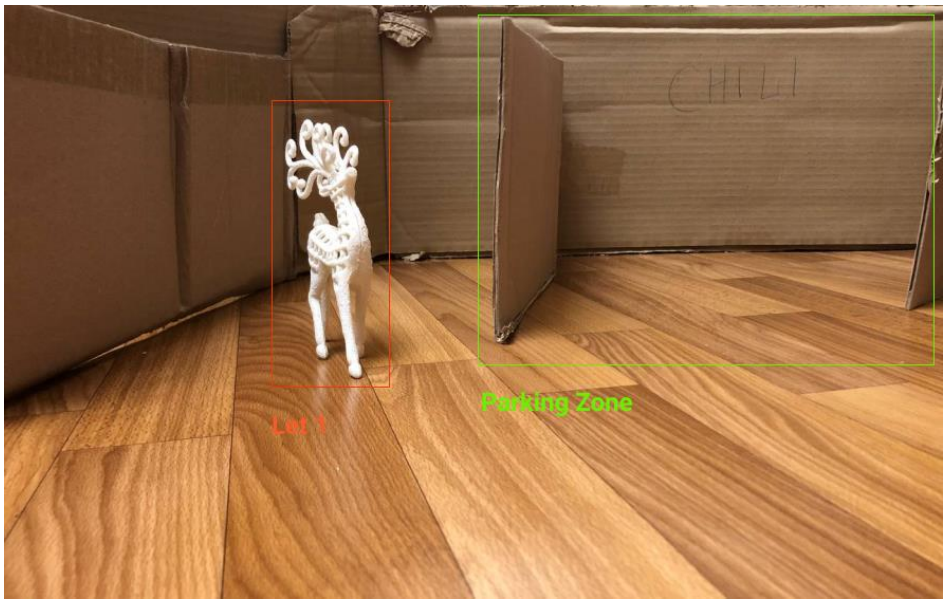

Рисунок 3.13 - Показники з камери

Приклад аналізування іншої обраної зони (рисунок 3.14).

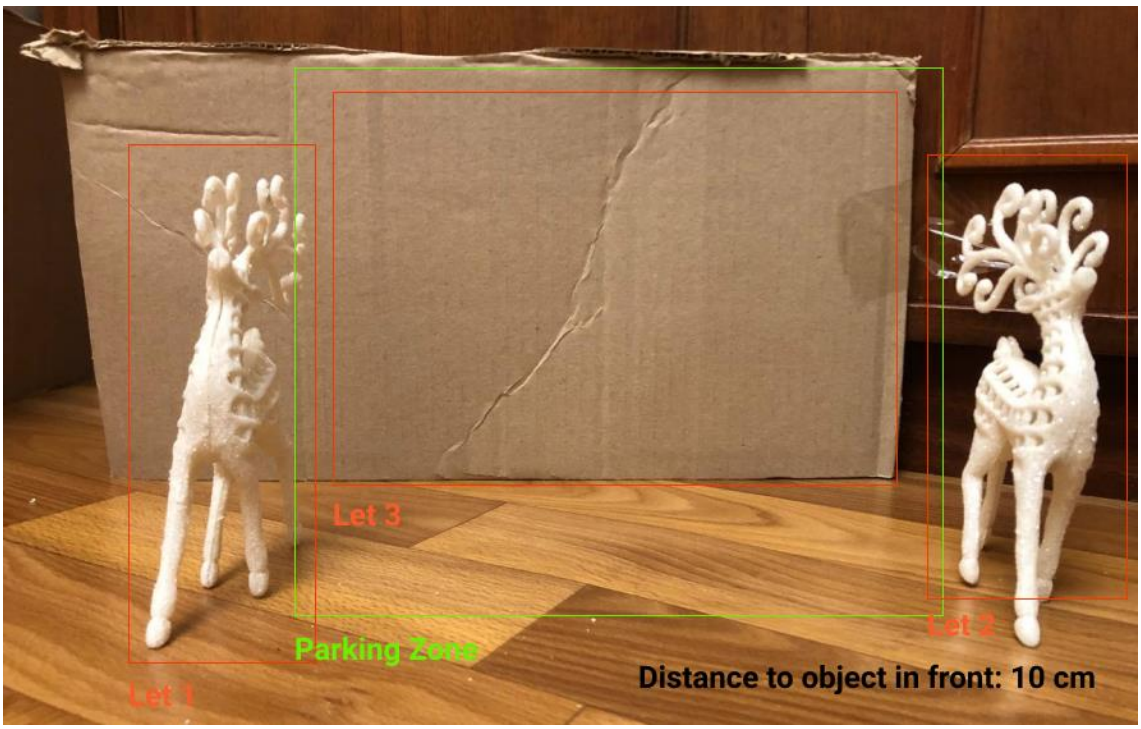

Рисунок 3.14 - Інтегровані показники з камери

3.4 Висновки за розділом 3

За результатом розпаралелювання процесів дало відмінні результати так як паркування лише за допомогою одного з модулів не давало високих результатів. Паркування по камері було ефективно у 5 з 10 протестованих ситуацій бо вона не завжди встигала реагувати на нові об’єкти по перед себе.

Паркування по модулю УЗ було дуже довгим через самостійний пошук місця, а саме “обкатку” всієї зони. Воно показало себе більш ефективно, 8 з 10 успішних протестованих ситуацій. Але витрачений час є головною проблемою цього методу. При використані двох модулів водночас було виявлено найбільшого успіху, 9 з 10 протестованих ситуацій. На один невдалий тест сприяло додавання додаткових факторів.

ВИСНОВКИ

В даній роботі було проаналізовано та виділено два основних методу паркування які ідеально працюють разом. Було проведено моделювання та тестування створеної моделі яка відмінно впоралась з поставленою задачею в контрольованих умовах.

Процес розробки моделі складався з трьох частин:

- Розробка методу розпізнавання об'єктів у реальному часі;
- Розробка методу ідентифікації відстані;
- Реалізація само паркування стенду на Raspberry Pi 4.

Було вибрано технології, системи та методи для реалізації підвищення ефективності, а саме точності автоматичного паркування. Було інтегровано дві системи в єдину. Стендове авто було зібрано на основі Raspberry Pi. Автоматичний паркінг було реалізовано за допомогою єдиного результату вимірювань з камери та датчику відстані. Після отримання результату середовища, оцінки об’єктів та відстані до них - було здійснене автоматичне паркування.

За результатами дослідження паркування по камері було ефективно у 5 з 10 протестованих а паркування по модулю УЗ показало себе більш ефективно, 8 з 10 успішних протестованих ситуацій. Але витрачений час є головною проблемою цього методу. При використані двох модулів водночас було виявлено найбільшого успіху, 9 з 10 протестованих ситуацій. На один невдалий тест сприяло додавання додаткових факторів.

СПИСОК ВИКОРИСТАННОЇ ЛІТЕРАТУРИ

1. Аизерман М.А. Автоматика переключения передач[Текст].- М.: Машгиз, 1948.-140 с.

2. Акимов Н.Н. Резисторы, конденсаторы, трансформаторы, дроссели, коммутационные устройства РЭА / Минск, 1994 г., стр. 84-97 [Електронний ресурс] – Режим доступу до ресурса: http://www.minstroyrf.ru/trades/gradostroitelnayadeyatelnost-i-arhitektura/14/

3. Аксенов И.Б. Конспект лекций по дисциплине «Тепло и массообмен в РЭА» [Електронний ресурс] – Режим доступу до ресурса: http://fastmb.ru/auto_shem/1169sistema-avtonomnoy-parkovki-avtomobilya.html

4. Балабин И.В.Испытания автомобилей [Електронний ресурс] – Режим доступу до ресурса:

http://www.belsis.ru/news/popularnaja_elektronika/volvo-selfparking.html

5. Гришкевич А.И. Автомобили-роботы испытания, 1991.-187с [Текст].

6. Дрю Д. Теория транспортных потоков и управление ими. М.: Транспорт, 1972 [Електронний ресурс] – Режим доступу до ресурса: http://howcarworks.ru

7. Виявлення об'єктів у реальному часі [Електронний ресурс] – Режим доступу до ресурса: https://ichi.pro/ru/ucebnik-po-obnaruzeniu-ob-ektov-v-tensorflow-obnaruzenie-ob-ektov-v-real-nom-vremeni-155483233532636

8. RFID [Електронний ресурс] – Режим доступу до ресурса: https://ru.wikipedia.org/wiki/RFID

9. Підручник з виявлення об'єктів у YOLO – виявлення об'єктів у реальному часі [Електронний ресурс] – Режим доступу до ресурса: https://vbystricky.github.io/2020/08/yolo_ssd_etc.html#yolov2

10. Sonar [Електронний ресурс] – Режим доступу до ресурса:

https://gist.github.com/okalachev/feb2d7235f5c9636802c3cda43add253

11. Pi Car [Електронний ресурс] – Режим доступу до ресурса: https://www.hackster.io/bestd25/pi-car-016e66
12. Python: розпізнавання об'єктів у реальному часі [Електронний ресурс] – Режим доступу до ресурса: https://proglib.io/p/real-time-object-detection#part1
13. Класифікація об'єктів у режимі реального часу [Електронний ресурс] – Режим доступу до ресурса: https://www.dataart.com.ua/news/klassifikafiya-ob-ektov-v-rezhime-realnogo-vremeni/
14. Camera module [Електронний ресурс] – Режим доступу до ресурса: https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/6
15. Робота з ультразвуковим далекоміром [Електронний ресурс] – Режим доступу до ресурса: https://clover.coex.tech/ru/sonar.html

ДОДАТОК А

Лістинг програми

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
import time
import threading
import pigpio
import cv2
import collections
import numpy
import workprocces
import RPi.GPIO as gpio
import time
import sys
import Tkinter as tk
from sensor import distance

from threading import Thread
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
```

```
from utils import label_map_util
from utils import visualization_utils as vis_util

sys.path.append("..")

TRIG = 23
ECHO = 24

pi = pigpio.pi()
done = threading.Event()
cap = cv2.VideoCapture(0)


class ThreadSonar(Thread):

    def rise(gpio, level, tick):
        global high
        high = tick

    def fall(gpio, level, tick):
        global low
        low = tick - high
        done.set()

    def read_distance():
        global low
        done.clear()
        pi.gpio_trigger(TRIG, 50, 1)
        if done.wait(timeout=5):
```

```
            return low / 58.0 / 100.0

        pi.set_mode(TRIG, pigpio.OUTPUT)
        pi.set_mode(ECHO, pigpio.INPUT)
        pi.callback(ECHO, pigpio.RISING_EDGE, rise)
        pi.callback(ECHO, pigpio.FALLING_EDGE, fall)


    def read_distance_filtered():
        history = collections.deque(maxlen=10)
        history.append(read_distance())
        return numpy.median(history)


class ThreadCamera(Thread):
    MODEL_NAME = 'ssd_mobilenet_v1_coco_11_06_2017'
    MODEL_FILE = MODEL_NAME + '.tar.gz'
    PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'
    PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')

    NUM_CLASSES = 90

    def init():
        gpio.setmode(gpio.BOARD)
        gpio.setup(7, gpio.OUT)
        gpio.setup(11, gpio.OUT)
        gpio.setup(13, gpio.OUT)
        gpio.setup(15, gpio.OUT)
```

```
 def reverse(tf):
     gpio.output(7, False)
     gpio.output(11, True)
     gpio.output(13, False)
     gpio.output(15, True)
     time.sleep(tf)

 def forward(tf):
     gpio.output(7, True)
     gpio.output(11, False)
     gpio.output(13, True)
     gpio.output(15, False)
     time.sleep(tf)

def turn_right(tf):
    gpio.output(7, True)
    gpio.output(11, False)
    gpio.output(13, False)
    gpio.output(15, True)
    time.sleep(tf)

def turn_left(tf):
    gpio.output(7, False)
    gpio.output(11, True)
    gpio.output(13, True)
    gpio.output(15, False)
    time.sleep(tf)

def stop(tf):
```

```
    gpio.output(7, False)
    gpio.output(11, False)
    gpio.output(13, False)
    gpio.output(15, False)
    time.sleep(tf)
    gpio.cleanup()

def move_input(event):
    init()
    print "Move:", event.char
    sleep_time = 0.060

 if “move_forward” == receiveData():
     forward(sleep_time)
 elif “move_back” == receiveData():
     reverse(sleep_time)
 elif “move_left” == receiveData():
     turn_left(sleep_time)
 elif “move_right” == receiveData():
     turn_right(sleep_time)
 elif “move_stop” == receiveData():
     stop(sleep_time)
 else:
     pass

 cur_dis = distance("cm")
 print("Distance:", cur_dis)

 if cur_dis <15:
```

```
            init()
            reverse(0.5)

        command.mainloop()
    gpio.cleanup()

    opener = urllib.request.URLopener()
    opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
    tar_file = tarfile.open(MODEL_FILE)
    for file in tar_file.getmembers():
      file_name = os.path.basename(file.name)
      if 'frozen_inference_graph.pb' in file_name:
        tar_file.extract(file, os.getcwd())

    detection_graph = tf.Graph()
    with detection_graph.as_default():
      od_graph_def = tf.GraphDef()
      with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

    label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
    categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
    category_index = label_map_util.create_category_index(categories)

    with detection_graph.as_default():
      with tf.Session(graph=detection_graph) as sess:
```

```
while True:
ret, image_np = cap.read()
# Expand dimensions since the model expects images to have shape: [1, None, None, 3]
image_np_expanded = np.expand_dims(image_np, axis=0)
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
# Each box represents a part of the image where a particular object was detected.
boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
# Each score represent how level of confidence for each of the objects.
# Score is shown on the result image, together with the class label.
scores = detection_graph.get_tensor_by_name('detection_scores:0')
classes = detection_graph.get_tensor_by_name('detection_classes:0')
num_detections = detection_graph.get_tensor_by_name('num_detections:0')
# Actual detection.
(boxes, scores, classes, num_detections) = sess.run(
  [boxes, scores, classes, num_detections],
  feed_dict={image_tensor: image_np_expanded})
# Visualization of the results of a detection.
vis_util.visualize_boxes_and_labels_on_image_array(
    image_np,
    np.squeeze(boxes),
    np.squeeze(classes).astype(np.int32),
    np.squeeze(scores),
    category_index,
    use_normalized_coordinates=True,
    line_thickness=8)

cv2.imshow('object detection', cv2.resize(image_np, (800,600)))
```

```
    if cv2.waitKey(25) 0xFF == ord('q'):
      cv2.destroyAllWindows()
      break


while True:
    th1 = Thread(target=ThreadSonar, args=())
    th1.start()
    th1.join()
    th2 = Thread(target=ThreadCamera, args=())
    th2.start()
    th2.join()
```