МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є.
"_____"_____2021 р.

ДИПЛОМНИЙ ПРОЄКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Модуль розпізнавання жестів рук для керування квадрокоптером

Виконавець: Левчук В.В.

Керівник: к.т.н. доц. Глазок О.М.

Консультанти з окремих розділів пояснювальної записки:

Нормоконтролер: Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Інститут комп'ютерних інформаційних технологій

Кафедра комп'ютеризованих систем управління

Освітньо-кваліфікаційний рівень магістр

Спеціальність 8.05010202 "Системне програмування"

ЗАТВЕРЖУЮ
Завідувач кафедри

_____ Литвиненко О.Є.

«___» _____2014 р.

З А В Д А Н Н Я

НА ВИКОНАННЯ ДИПЛОМНОЇ РОБОТИ СТУДЕНТА

Кашкевич І-Ф. Ф.

(прізвище, ім'я, по батькові)

1.Тема проєкту (роботи): "Методика використання платформи *.NET* при створенні тригерів бази даних "

затверджена наказом ректора від " 17 " січня 2014 року № 51/ст.

2.Термін виконання проєкту (роботи): з 10.03.2014 до 22.06.2014

3. Вихідні дані до проекту (роботи): постановка задачі до виконання роботи, мова програмування *C#*, середовище програмування *MS Visual Studio 2010*, СУБД *MS SQL Server*

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

1) аналіз приципів створення і використання тригерів баз даних;

2) можливості використання тригерів баз даних;

3) розробка методики створення тригерів таблиць бази даних з використанням платформи *.NET*.

5. Перелік обов'язкового графічного матеріалу:

1) основне вікно програми;

2) вікно виконання сценаріїв;

3) структура таблиць бази даних;

4) схема алгоритму виконання тригерів в програмі;

5) порівняльна таблиця середнього часу обробки тригерів.

6. КАЛЕНДАРНИЙ ПЛАН

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1	Ознайомитись з постановкою задачі дипломного проєктування	17.03.14	виконано
2	Вивчити спеціальну літературу і технічну документацію	20.03.14	виконано
3	Проаналізувати принципи створення тригерів при використанні платформи *.NET*	29.03.14	виконано
4	Написати розділ 1.	10.04.14	виконано
5	Проаналізувати можливості платформи *.NET* в роботі з тригерами	22.04.14	виконано
6	Написати розділ 2.	29.04.14	виконано
7	Провести розробку методики створення тригерів таблиць БД	10.05.14	виконано
8	Написати розділ 3.	15.05.14	виконано
9	Оформити пояснювальну записку	05.06.14	виконано
10	Підготувати графічний демонстраційний матеріал та доповідь	16.06.14	виконано

7. Дата видачі завдання 04.02.2021

Керівник дипломної роботи Глазок О.М.
(підпис)

Завдання прийняв до виконання Кашкевич І-Ф. Ф.
(підпис студента)

ПОЯСНЮВАЛЬНА ЗАПИСКА

Пояснювальна записка до дипломної роботи “Модуль розпізнавання жестів рук для керування квадрокоптером”: 95 с., 14 рис., 7 табл., 27 літературних джерел, 4 додатки.

ШТУЧНИЙ ІНТЕЛЕКТ, ГЛИБОКЕ НАВЧАННЯ, ОДНОПЛАТНА ЕОМ, *Raspberry Pi, MediaPipe*, ЖЕСТ, КВАДРОКОПТЕР

Об’єкт дипломного проєктування – розпізнавання жестів рук на основі нейромережевим алгоритмам.

Предмет дипломного проєктування – модуль розпізнавання жестів рук для керування квадрокоптером.

Мета дипломної роботи – створення модуля на основі одноплатної ЕОМ для розпізнавання жестів рук з використанням алгоритмів штучного інтелекту з подальшою інтерпретацією результатів роботи модуля у команди для керування квадрокоптера.

Метод проєктування – застосування засобів розпізнавання з поєднанням алгоритмів глибокого навчання та використання апаратних можливостей одноплатних ЕОМ.

Галузь застосування – дистанційне жестикуляційне керування динамічних систем.

Значущість роботи та висновки – модуль жестикуляційного керування забезпечує просте та інтуїтивне керування рухомими системами, такими як квадрокоптер.

Прогнози та припущення щодо розвитку об’єкта дослідження – створення робочого зразка модуля, інтеграція зразка у квадрокоптерта використання його для керування. Подальший розвиток системи передбачає додавання нового функціоналу у модуль, зокрема: розпізнавання та ідентифікація обличчя власника або кола персон, розширення спектру жестів, які підлягають розпізнаванню, додовання можливості розпізнавання звукових команд з подальшим виконання відповідних, до розпізнаних сигналів, завдань.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ЕОМ – Електронна обчислювальна машина
ШІ – Штучний інтелект
ПЗ – Програмне забезпечення
IoT - Internet of Things, Інтернет речі
ОПЗ – Оперативний запам'ятовуючий пристрій
ПЗЗ – Пристрій із зарядним зв'язком
CCD – Charge-coupled device,
КМОН – Комплементарна структура метал-оксид-напівпровідник
CMOS – Complementary-symmetry/metal-oxide semiconductor
Mn – Мегапіксель
GIF – Graphics Interchange Format
BMP – Bitmap Picture
JPEG – Joint Photographic Experts Group
PNG – Portable Network Graphics
RGB – Red, Green, Blue
ЗПП – Зворотнє поширення помилки
ЦАП – Цифро-аналоговий перетворювач
LCD - Liquid crystal display
HSV - Hue, Saturation, Value
GPIO - General-purpose input/output
USB - Universal Serial Bus
HDML - Handheld Device Markup Language
DSI - Display Serial Interface
CSI - Camera Serial Interface
PWM – Pulse-width modulation
ШІМ – Широтно-імпульсна модуляція
JSON - JavaScript Object Notation

ВСТУП

На сьогоднішній день, дедалі обширним та тісним стає вплив електронних систем на повсякденне життя людей. Ріст попиту людей на задоволення своїх потреб як у побуті, так у вирішенні низки проблем економічного та логістичного характеру безпрецедентно спонукає до розвитку науково-технічної галузі інтерактивних систем людино-комп'ютерної взаємодії. Виникає дедалі більший запит на системи, які забезпечують керування людиною різноманітних комплексів, і вимагають дедалі більш простого інтерфейсу для взаємодії з цими комплексами. Можна виділити кілька причин для спрощення систем інтерфейсу керування людино-машинних комплексів, а саме: легкість в освоєні персоналом простого та зручного інтерфейсу, зменшення витрат на навчання персоналу для керування комплексом, інтуїтивно зрозумілий інтерфейс зменшує ризик виникнення помилок під час експлуатації. Таким чином, простий та інтуїтивно зрозумілий інтерфейс надає значно кращі можливості для керування, що у свою чергу, має ліпший вплив на економічні витрати, зокрема під час освоєння та експлуатації подібних систем.

Однією з таких систем інтерфейсу керування є керування за допомогою жестикуляції рук людини, що у свою чергу складаються з інтерактивної системи розпізнавання жестів. Інтерфейси керування, які полягають у керуванні за допомогою жестикуляції рук починають лише впроваджуватися у життєві процеси людини, але дедалі більше досліджень здійснюють у цій області. Зокрема, як зазначалося раніше, дана система дозволяє значно спростити керування комплексом, звівши це до інтуїтивно зрозумілого рівня. Також дані системи надають можливість виключити фізичний контакт оператора з комплексом, що досить актуально, зважаючи на епідеміологічний стан у світі.

Керування жестикуляційним способом значно краще відображається на здатності людини виконувати кілька операцій одночасно – здатність людини здійснювати розподіл уваги і здійснювати кілька видів діяльності.

Ключовим аспектом у виконанні одночасно кількох операцій є увага. Увага – це така форма психологічної діяльності людини, що виявляється у спроможності свідомості зосереджуватися на певних об’єктах, які мають те чи інше значення для людини у певний момент часу. Також, одна з умов виконання такої «багатозадачності», полягає у тому, що одна з таких операцій буде виражена у вигляді міміки кінцівок рук. Звісно, людина здатна виконувати різноманітні дії, які людина може виконувати одночасно, але найбільш успішно вона виконує одночасно дві дії, одна з яких – фізична робота, наприклад моторика рук.

Зокрема можна навести приклад подібних ситуацій, коли людина успішно та ефективно виконує кілька дій одночасно. Наприклад вона здатна в один і той же час мити посуд і вести розмову із співрозмовником на кухні, спілкуватися по телефону па в’язати світер, керувати машиною та спілкуватися з іншою людиною по голосовому зв’язку через гарнітуру. Людина не використовує великих розумових зусиль задля виконання привичних та інтуїтивних задач, наприклад, розглядаючи у даному випадку миття посуду. Вона без перешкод здатна вести діалог на різноманітні теми, що одночасно не несе ніякого негативного впливу на виконання другої задачі (миття посуду).

Окрім спілкування, людина здатна і на інші інтелектуальні задачі, одночасно з виконанням задач із залученням моторики рук. Завдяки довгому шляху еволюції людина набула навичок ефективно здійснювати розподіл уваги між виконанням розумової роботи та фізичної праці рук і переключення уваги з однієї діяльності на іншу стає непомітним для людини. Хоча варто зауважити, що на багатозадачність людини впливає безліч чинників, зокрема фізичний і психологічний стан людини і навколишнє середовище: індивідум не здатний якісно виконувати навіть одну задачу, у разі, якщо є певні чинники, котрі будуть негативно впливати на зосередженість (концентрацію) – інженер не здатний вести розрахунки для проєктування каркасу будівлі, якщо на вулиці будуть здійснюватися будівельні роботи, які супроводжуються гучними та різкими звуками.

Одним із ключових аспектів формування стійкої багатозадачності людини є стійкість уваги, що можна розтлумачити як здатність протягом тривалого часу утримуватися на якомусь об’єкті, дії. Розглядаючи дане визначення у контексті багатозадачності, можна сказати, що це здатність людини приділяти достатню кількість уваги протягом певного періоду часу для кожної роботи, яка виконується у даний момент, за який людина встигає виконати певний мінімальний обсяг робіт, при якому кожна з виконуваних дій людиною буде здійснюватися без проблем чи затримок, і не перешкоджатиме виконанню іншої дії, яку здійснює людина у цей час.

Одночасно, така успішна діяльність людини у плані багатозадачності втрачає ефективність, якщо дії, які виконує людина, будуть носити виключно характер розумової праці. Значно важкіше, а більшості випадках – неможливо, здійснювати одночасно кілька інтелектуальних задач – складно проводити відразу кілька обчислень, проводити постановку різних задач кільком групам людей і т.п.. В такому випадку відразу проявляється зниження ефективності виконання кожної з робіт, що може нести за собою вагомі наслідки, в залежності від того, наскільки критичною може бути помилка в тій чи іншій діяльності.

Ознайомившись з особливостями та проблематикою багатозадачності людини, можна зробити висновок, що системи керування жестами – ефективне рішення для керування різноманітними системами. Якщо розглядати можливі області застосування подібних інтерактивних систем для керування, перш за все впадає у вічі системи, що впроваджені у мобільних телефонах та мобільних додатках. Зокрема, у них впроваджена можливість захоплення та відстеження рук, розпізнавання та класифікація жестів рук, що надає зможу виконувати широкий спектр дій, як наприклад: здійснення знімку екрана, початок та закінчення відеозапису екрана телефона, розблокування пристрою, збільшення/зменшення гучності і т.п..

Також чудовим прикладом може слугувати інтегрована система керування жестами у бортових комп’ютерах сучасних автомобілів. Наприклад, застосування

жестів передбачено у керування мультимедією автомобіля. Прості жести, такі як "вказати" або "змахнути" надають можливість викликати певні функції - наприклад, прийняти або відхилити дзвінок. Також передбачена регулювання гучність аудіосистеми завдяки обертовим рухом вказівного пальця. Здійснення планування маршруту в навігаційній системі також здійснюється за допомогою жестів рук, надана можливість за допомогою жестикуляції керувати навігацією в меню. Система управління жестами чудово доповнює канонічний інтерфейс керування мультимедією, такий як кнопковий.

Варто зауважити, що набуло поширення керування за допомогою жестикуляції і в системах «розумний дім». Набір як простих, так і більш складних, комбінованих жестів, відкриває перед користувачем великі можливості для зручного проживання у власній оселі. Таким чином, лише здійснивши кілька простих рухів руками, система надасть можливість керування освітлення, мульмедією будинку, побудовими приладами.

Одним з можливих рішень у застосуванні інтерактивних систем керування жестами, на думку автора, є керування літальними апаратами – мультикоптерами. Натхненний швидким та буремним розвитком коптеробудування та появою нових алгоритмів, методик та інструментів у галузях комп’ютерного зору та глибокого навчання (deep learning), автор розробив власний модуль розпізнавання жестів рук для керування квадрокоптером. Даний вид керування дозволить розширити можливості по управлінню літальними апаратами даного типу, а не лише застосовувати загальноприйняті пульти керування на радіоуправлінні.

Таким чином, інтерактивна система розпізнавання жестів – це складний але необхідний та зручний комплекс, який розкриває нову можливості для керування різноманітними комплексами. Системи такого типу інтуїтивно зрозумілі, завдяки тому, що керування здійснюється жестикуляцією рук оператора. Сам процес розпізнавання жестів рук ділиться на кілька етапів, протягом яких система отримує інформацію, аналізує її, обробляє і на основі неї здійснює висновок щодо жестикуляцій. Інтерактивні системи розпізнавання жестів є складним, але у той же час, надзвичайно вагомим знаряддям людино-машинного інтерфейсу.

РОЗДІЛ 1

АНАЛІЗ ЗАСОБІВ, МЕТОДІВ ТА АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ, ЯКІ ЗАСТОСОВУЮТЬ ПРИ СТВОРЕННІ ІНТЕРАКТИВНИХ СИСТЕМ КЕРУВАННЯ ЖЕСТАМИ

1.1. Вступ до розділу

Буремний розвиток технологічного прогресу людства дедалі більш сильніше розпочинає впливати на повсякдення людей. Зокрема, завдяки революції в методологіях розробки систем штучного інтелекту та розвитку архітектури обчислювальних машин, стало можливо проєктування, розробка та впровадження різноманітних систем, які можна інтегрувати у побут. Дані системи представляють собою складний комплекс поєднань найрізноманітніших технологій та процесів, тісно інтегрованих та пов’язаних між собою, які призначені вирішувати задачі, донині вирішення яких було неможливе, не застосувавши окремо виділених ресурсів, або потребувало зусиль, невиправдних як з економічної точки зору, так з точки зору затраченого на це часу. Однією з «гілок» у системах даного типу є інтерактивні системи керування жестами, які можуть бути представлені як модулями, вбудовані в більш складні програмно-апаратні комплекси, так і окремими самостійними системами, на плечі яких покладено завдання по вирішенню задач, які потребують розпізнання та обробки абстрактних типів даних, наприклад жестів людських рук, безпосередня репрезентація яких у звичний, для обробки комп’ютерами, двійковий вигляд, не є можливою, і подальше конвертування обробленої інформації у сигнали, які подаються на обробку системам, які працюють із жестикуляцією операторів.

Надалі буде більш детально описано, що собою представляють дані системи, які зразки даних систем існують та як саме працюють модулі керування жестами у таких системах. Також буде оглянуто теоретичне підґрунтя інтерактивних системи розпізнавання жестів, загальна схема та принцип роботи подібних структру.

1.2. Огляд загальної схеми роботи інтерактивних систем розпізнавання жестів, теорія і приклади

«Що таке інтерактивна система керування жестами?» - задасть доволі слушне питання пересічна людина. Інша людина могла б припустити і дати відповідь на це питання наступним чином: «Це система, яка здатна розрізняти певні жести оператора, і на основі них робити ті чи інші дії». І насправді це буде цілком вичерпна відповідь на дану квестію. Якщо більш детально розглянути дане питання, то під визначенням «інтерактивна система керування жестами» розуміється система, яка здатна виявляти та розрізняти серед стохастичного набору рухів людини набір конкретних жестів, проводити аналіз даних жестів та інтерпретувати їх у вибірку команд або іншого вигляду інформацію, яка може використовуватися я к самою системою так і іншими системами чи комплексами.

Розглянемо загальну структуру таких систем і її основні частини (рис. 1.2.1.)

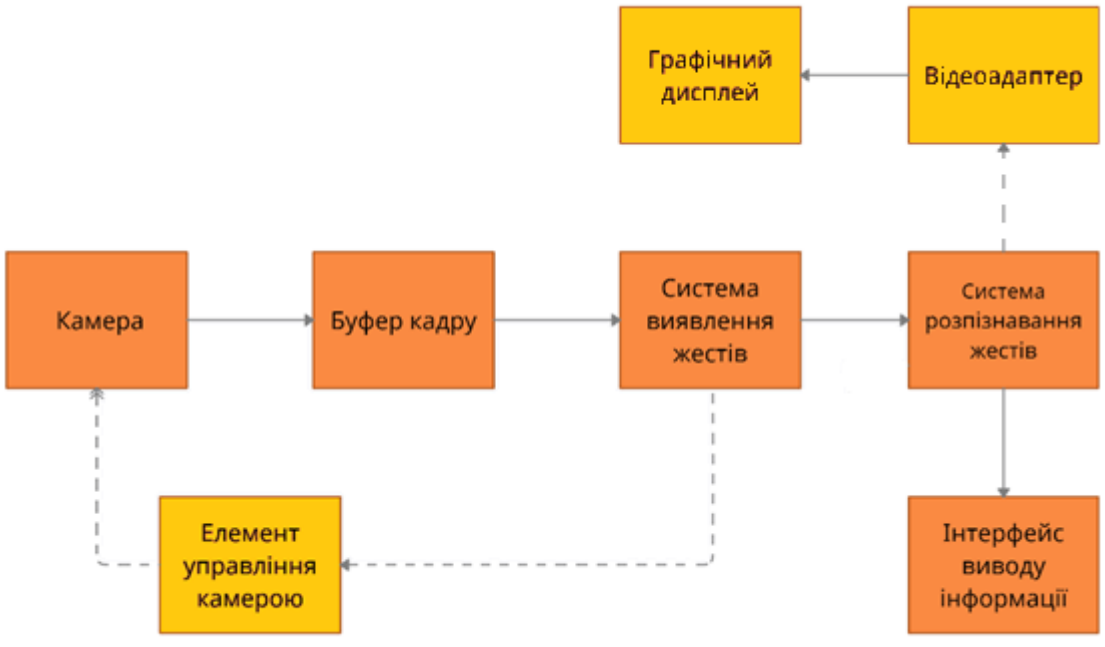


Рис. 1.2.1. – Загальна схема інтерактивної системи керування жестами

У даній системі наявні наступні основні вузли:

- Камера
- Буфер кадру
- Система виявлення жестів
- Система розпізнавання жестів
- Інтерфейс виведення інформації
- Відеоадаптер та Графічний дисплей
- Елемент управління камерою

Також присутній необов'язковий модуль «Елемент управління камерою».

Розглянемо більш детально кожен із наведених пунктів, розберемо роль даних елементів у цілісній системі і взаємодію між даними блоками.

Камера. Камера є основним елементом, який забезпечує систему інформацією у вигляді кадрів відеопотоку або окремих зображень. Кадри зображення є основною інформацією, якою оперує система, обробляючи і витягуючи необхідну інформацію (аналізуючи виявлений жест). На даний момент, ринок наповнений різноманітною продукцією, що дозволяє вибрати цілком бездоганний варіант камери, який буде задовольняти як якістю роботи, надійністю, так і влаштовувати ціновою категорією, тобто відповідати критерію «ціна-якіть».

Необхідно описати основні характеристики, за якими обирають камери, для систем керування подібного типу. Серед основних характеристик камер, можна виділити:

- Тип матриці
- Розмір матриці
- Фокусна відстань та кут огляду
- Роздільна здатність
- Додаткові можливості

Більшість камер володіють одним із двох типів матриць: ПЗЗ-матрицю (англ. CCD) або КМОН-матрицю (англ. CMOS). ПЗЗ-матриця – це така спеціалізована аналогова інтегральна мікросхема (електронна схема, яка розроблена у вигляді

напівпровідникового кристалу і здатна здійснювати задану функцію), яка складається з набору світлочутливих фотодіодів (елементи, які мають властивість сприймати падаюче світло і конвертувати його у електричний заряд), яка створена на основі кремнію, що використовує технологію ПЗЗ – приладів із зарядним зв'язком. У свою чергу, КМОН-матриця – це матриця, створена на основі технології КМОН. КМОН-технологія – це спосіб виготовлення матриці , використовуючи польові транзистори з ізольованим затвором і каналами різного степення провідності. Основна відмінність цих двох матриць полягає у тому, у який спосіб зчитується інформація із матриць. У ПЗЗ-матриць зчитування інформації відбувається послідовно, на відмінну від КМОН-матриць, де зчитування відбувається у довільний спосіб. Через дані особливості, у камерах на основі ПЗЗ-матриці, неможливо здійснити знімок кадру, у разі, якщо попередній кадр не був сформований повністю. Переваги ПЗЗ полягають у більш якіснішому зображені, ніж у КМОН, проте з недоліків можна виділити більшу вартість через складність технології, що у свою чергу збільшує енерговитрати. Якщо розглянути переваги КМОН, то відразу варто зазначити, що через можливість довільного отримання інформації з комірки матриці, зменшується інтервал між створенням кадрів. Також КМОН-матриці дешевші у виробництві і споживають менше електроенергії. З недоліків варто зазначити, що фотодіод займає значно меншу площу у комірці на елементі матриці, ніж аналогічно в ПЗЗ, що зменшує світлочутливість камери.

Розмір матриці є важливим пунктом при виборі камери. Фізичний розмір матриці визначає якість як самої матриці, так якість формування зображення. Проте, насправді якість зображення залежить не від самого розміру матриці, а саме від розміру окремого розміру елемента матриці – комірки. Збільшення розміру даної комірки призводить до покращення характеристик якості кадру відео чи зображення. Позначають розмір матриці зазвичай у дюймах, а саме через дріб, наприклад 1/3". Зазвичай, розмір у міліметрах не наводиться, проте можна переглянути таблицю відповідності (табл. 1.2.1.), яка відображає залежність дюйми-міліметри.

Таблиця 1.2.1

Таблиця приведення фізичного розміру матриці камери(дюйми у міліметри)

Дюйми (1 / дюйм)	Міліметри (ширина × висота)
1/3.6"	4,000 × 3,000
1/3.2"	4,536 × 3.416
1/3"	4,800 × 3,600
1/2.7"	5,371 × 4,035
1/2.5"	5,760 × 4,290
1/2.3"	6,160 × 4,620
1/2"	6,400 × 4,800
1/1.8"	7,176 × 5,319
1/1.7"	7,600 × 5,700
1/1.63"	8,311 × 6,233
2/3"	8,800 × 6,600
1"	12,800 × 9,600

Фокусна відстань та кут огляду – два взаємопов’язані параметри, які визначають дальність та кут огляду камери. Розглядаючи дані параметри, можна сказати, що зі зменшенням фокусної відстані зростає кут огляду, і зменшується кількість деталей на кадрі, які є можливість оглянути, і навпаки, зі збільшенням фокусної відстані, кут огляду зменшується, що призводить до більшої деталізації зображення, але веде за собою зменшення площі спостереження. У таблиці 1.2.2. наведена порівняльна характеристика залежності фокусної відстані від кута огляду та відстані, на якій можна спостерігати об’єкт, залежно від вищезазначених параметрів.

Таблиця 1.2.2.

Таблиця відношення фокусної відстані та кута огляду до відстані до об’єкта

Фокусна відстань (мм)	2,8	3,6	6	8	12	16
Кут огляду	86	72	48	30	25	17

(градуси, ˚)						
Відстань до об’єкта (м)	0-5	0-6	5-10	10-20	25-35	35-50

Параметр роздільної здатності камери вважають самим важливим серед іншим і, у більшості випадків, саме на нього звертають найбільшу увагу при виборі камери. Хоча роздільна здатність грає велику роль, проте стверджувати, що це найбільш важливий параметр не можна, оскільки при одній і тій самій роздільній здатності, якість фото може сильно варіюватися, залежно, зокрема, від типу матриці камери. Роздільна здатність вимірюється у мегапікселях (Мп). Один мегапіксель уміщає у собі один мільйон пікселів. Кількість мегапікселів визначає розмір зображення і, заодно, впливає на якість сформованого кадру. У таблиці 1.2.3. наведена залежність роздільної здатності від кількості мегапікселей.

Таблиця 1.2.3

Залежність роздільної здатності від кількості мегапікселей

Мегапікселі, Мп	Роздільна здатність, пікселі
0.3	640 × 480
0.8	1024 × 768
2	1600 × 1200
3	2048 × 1536
4	2272 × 1704
5	2560 × 1920
6	3008 × 2000
7	3072 × 2304
8	3456 × 2304
10	3872 × 2592
12	4288 × 2848

Камери можуть володіти додатковими можливостями, які збільшують спектр застосувань камери або змінюють параметри камери задля збільшення якості змімків або кадрів відеопотоку. Наприклад ,камера може мати можливість монтувати на неї додаткові об’єктиви, містити інфрачервоний фільтр, що дозволить проводити зйомку у нічний час доби, мати додаткову підсвідку або спалах, мікроконтролери, які будуть регулювати роботу камери.

Варто розглянути один цікавий представник камер, розроблений саме для використання у технологіях комп’ютерного зору, а саме модуль StereoPi (рис. 1.2.1.) на базі Raspberry Pi Compute Module.

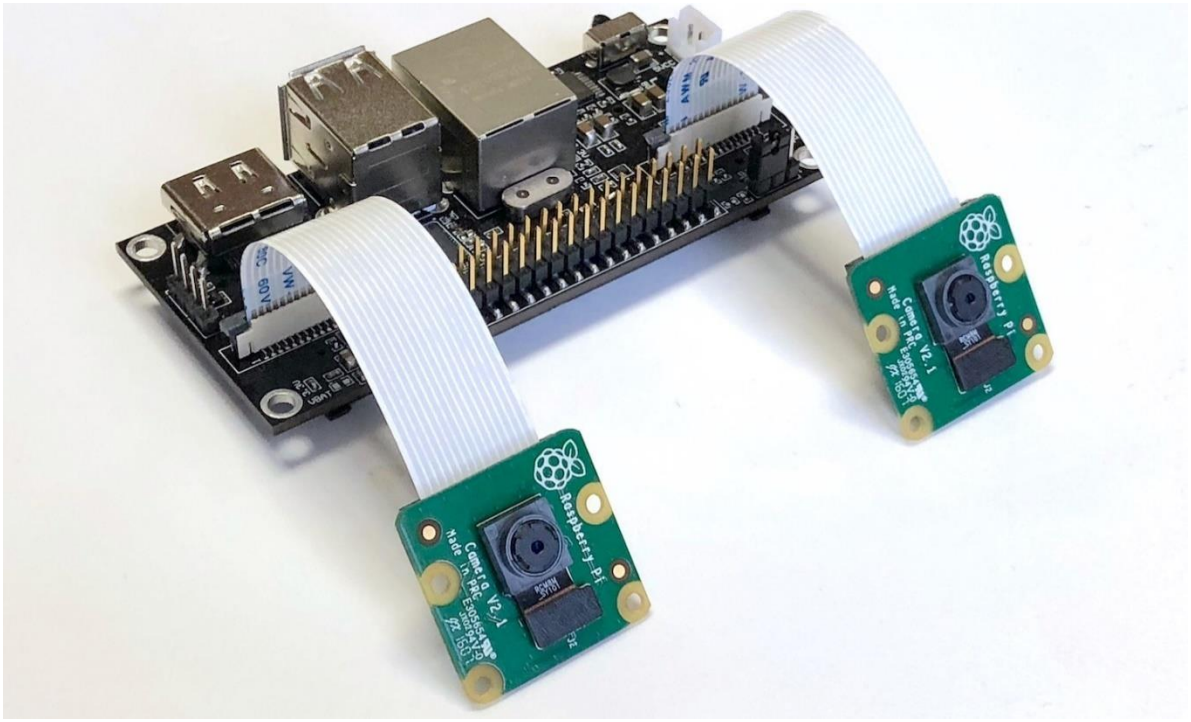


Рис. 1.2.1. – Модуль StereoPi

Даний модуль забезпечує можливіть одночасного синхронного отримання двох зображень, що є незамінним у продвинотому використанні комп'юторного зору. Зокрема, завдяки двом камерам, які надають зображення, зроблені в один і той самий момент часу, можна по фото обрахувати просторову відстань між об'єктом та камерами. Також сам модуль надає перевагу монтувати різні камери, які у свою чергу, можуть мати додаткові можливості, у вигляді підсвідки, інфрачервоного фільтра, об'єктивів і т.п..

Буфер кадру. Буфер кадру слугує місцем, куди поміщається зображення, отримане у ході фото або відеозйомки.

Зображення з камери може зберігатися у різноманітних форматах. Вибір конкетного формату для зберігання кадру зображення залежить від декількох факторів, якість зображення. Наскільки є вагомим чинником якісне представлення кадру зображення для системи, що проєктується?. Чим більш якісне зображення, тим системі керування жестами знадобиться здійснити більше роботи над зображенням, протягом всього робочого циклу – формування кадру, попередня обробка кадру, опрацювання кадру системою виявлення жестів. Це у свою чергу може сильно відобразитися на продуктивності окремих блоків, що впливатиме на всю систему в цілому. Також від якості фото залежить і розмір кожного конкретного зображення. У разі, якщо система співпрацює з відеопотоком, або великою кількістю світлин, може виявитись необхідність у зберіганні даного матеріалу, як у буфері системі так на твердому накопичувачі. Таким чином, від формату зображення напряму може залежати робота всієї системи і необхідно підібрати такий формат, який забезпечував би належний рівень роботи, при задовільній швидкості роботи системи та надавав високий відсоток правильно обробленої інформації.

Наразі, найбільш популярними форматами зображень є:

- GIF
- BMP
- JPEG
- PNG

Формат GIF. Вимовляється як «гіф» а абревіатура розшифровується як Graphics Interchange Format. Даний формат є одним з найстаріших, але в той же час, одним з найбільш популярних форматів. Має можливість зберігати дані з відсутністю втрати якості. Він не залежить від апаратних характеристик обчислювальної машини, що дозволяє використовувати його на різноманітних платформах. Проте головною проблемою даного формату є лише підтримка 256 кольорів, проте це забезпечує невелику вагу зображення. Також формат може зберігати кілька кадрів в одному файлі і час, протягом якого має бути відображений кожен кадр.

Формат BMP. Один із найстаріших форматів, разом з GIF. Перш за все, даний формат розроблявся для операційних систем Windows та OS/2 і може зберігати лише один кадр, проте на кожен піксель в різних файлах може припадати

різна кількість біт, що формує глубину кольору. Наразі доступні бітності 1,2,4,8,16,32,48, 64, що забезпечує широку кольорову гаму. Проте, вже не має такого широкого поширення, на відмінну від наступних форматів.

Формат JPEG. Формат, що може йменувати себе як один із найбільш популярних і поширених форматів. Вимовляється як «джейпег». Даний формат пропонує «золоту середину» між якістю та розміром картинки. Дозволяє проводити зжимання як з втратами так і без. Лінійний розмір зображення не може перевищувати 65535 × 65535 пікселів. Формат прекрасно підходить для зжимання світлин, що містять різноманітні об’єкти з плавним переходом яркості і кольору. Проте у режимі стиснення є велика ймовірність появи різноманітних шумів та та певного роду артефактів (спотворення частини зображення) , що може негативно вплинути на роботу систем, які здійснюють аналіз зображень.

Формат PNG. Нарешті, найбільш поширений формат, який застосовують для передачі зображень, у яких не має відбуватися втрат при зжимані. Сам формат прийшов на заміну відомому GIF, з більшим ареалом можливостей. Зокрема, може підтримувати різну глибину зображень, 16, 24 та 48 біт, що забезпечує невичерпну палітру кольорів, у порівнянні з GIF, який підтримує кольоровий спектр у межах 256 кольорів (2^8). Колір подається у форматі RGB, включаючи підтримку альфа-каналу. Проте, даний формат не підтримує анімацію, яка наявна у його «попередника» - GIF. Також, враховуючи наявний спектр кольорів, PNG-формат зазвичай має більшу вагу файла, ніж будь-який інший формат.

Виходячи з вищевказаної інформації, можна зробити висновок, що кожен із форматів надає користувачу свої переваги, але одночасно має і певні недоліки. Вибираючи формат, у якому будуть зберігатися кадри відеопотоку, необхідно брати до уваги можливості інтерактивної системи керування жестами, розробка якої проводиться у даний момент. Якщо апаратні чи програмні можливості не дозволяють використовувати надто «важкі» зображення, бажано прибігти до GIF або JPEG, інакше, якщо система сильно залежить від якості кадрів, необхідно застосовувати PNG.

Система виявлення жестів. Отримавши інформацію з камери і провівши збереження кадру в одному із форматів, отримали зображення, яке слугуватиме одиницею інформації у інтерактивній системі розпізнавання жестів. Дане зображення необхідно попередньо обробити спеціальними засобами, після чого,

можна процедуру виявлення жесту.

Варто розглянути, що собою представляють жести. Жест – це така дія або рух людини, що забезпечує передачу інформації чи емоції без застосування мовлення. Жестом можна назвати будь-яку послідовність рухів людини, які можуть бути передати певну інформацію, або нести певний сенс. Людина використовує різноманітні жести, такі як: маніпулятивні, комунікаційні, емоційні, ілюстративні жести.

Маніпулятивні жести передбачають за собою надання сигналу на здійснення якоїсь дії, здійснення впливу на об’єкт або на інший суб’єкт.

Комунікаційна жестикуляція є однією з найбільш поширеною ланкою як у невербальному так і у словесному спілкуванні і застосовується, щоб підкреслити та надати конкретики у висловлені думок, або повністю надавати інформацію співбесіднику у разі відсутності словесного діалогу.

Емоційні жести є важливою нішею у житті, через те, що завдяки їм, люди

здатні легко передавати різноманітні емоційні стани, як радість, дискомфорт, щастя, біль, смуток, відчайдушність, байдужість і т.п..

Ілюстративні жести передбачають за собою демонстрацію певної інформації, яку можна також описати словами. Тобто дані жести можуть виступати як самостійним висловленням, так і засобом, який допомагає описувати об’єкти чи явища під час спілкування.

Різноманітні системи розпізнавання жестів здатні опрацьовувати багато типів жестів, залежно від задач, які вони вирішують. Якщо ця система призначена для керування, будуть оброблятися маніпулятивні жести, якщо система спрямована на отримання якоїсь описової інформації, швидше за все, вона буде сприймати жести саме ілюстративного типу. Також можуть бути дактилогічні системи, які забезпечують розпізнавання жестів мовлення , наприклад, що може бути корисним для людей з обмеженими можливостями.

Тепер варто зрозуміти, чим являється жест, у системі розпізнавання і яких форм він набуває.

Жест є основоположною одиницею інформації. Як було сказано раніше, перед початком розпізнавання та класифікації жеста, необхідно певним чином обробити кадр зображення, щоб на ньому виявити сам жест. Лише після того, як кадр був оброблений, можна отримати кадр із розпізнаним силуетом руки і розглядати такий кадр як одиницю інформації, яку надалі можна використовувати як для налагодження роботи розпізнавання і класифікації жестів, так і для безпосереднього розпізнавання логіки жеста. Засоби і методи, які використовують для виявлення жеста на кадрі зображення, буде детально розказано у пункті 1.3.

У більш ранніх технологіях по розпізнаванню і керуванню, використовувалися спеціальні маркери на кінчиках пальцях (рис. 1.2.1.)

Рис. 1.2.1. – Рука із кольоровими маркерами

що дозволяло відслідковувати ключові точки на руці. Зазвичай, ці маркери представляли собою певні невеликі елементи визначеної кольорової гамми. Таким чином, система, аналізуючи зображення, отримані з камери, аналізувала, за допомогою деякого алгоритму, зображення на наявність певних «кольорових точок», і у разі, якщо колір даних точок входить у визначений діапазон відтінків, дана точка ідентифікувалася як ключовою. На кадрі зображення визначалася вибірка таких точок, координати яких заносилися до двомірної матриці, яка тепер цифровою одиницею інформації, готовою до обробки засобами розпізнання жестів. Система аналізувала матрицю і у разі успіху ідентифікувала ці точки як жест. На жаль, такий підхід ускладнював експлуатацію системи користувачем (зокрема через ймовірність попадання шумів, які можуть ідентифікуватися як ключові точки пальців).

На сьогоднішній день, методологія розпізнавання жестів лежить на основі комп'ютерного зору. Жест може представлятися структурою різноманітних типів, але для кожної системи це буде строго визначена еталонна модель. Модель може набувати вигляду однотонних силуетів (рис. 1.2.2)

Рис. 1.2.2. – Силует кисті руки

Проте, така структура не використовується безпосередньо при розпізнаванні образа руки через складність і велику витрату ресурсів, при обробці такого кадру і ідентифікації на кадрі жесту руки. Зазвичай силует обробляють спецзасобами, які забезпечують отримання контурів руки (рис. 1.2.3.)

Рис. 1.2.4. – Контури руки

Такий спосіб представлення інформації більш кращий, у порівнянні з усім силуетом руки, через те, що системі необхідно опрацьовувати значно меншу кількіть елементів (тобто - контури). Хоча, для повноцінної роботи системи розпізнавання, цього може бути недостатньо, тому що, можуть бути наявні жести, у яких не будуть виявлені контури усіх пальців.

Також наявний спосіб розпізнавання жестів на основі скелетної моделі кисті руки (рис. 1.2.5.).

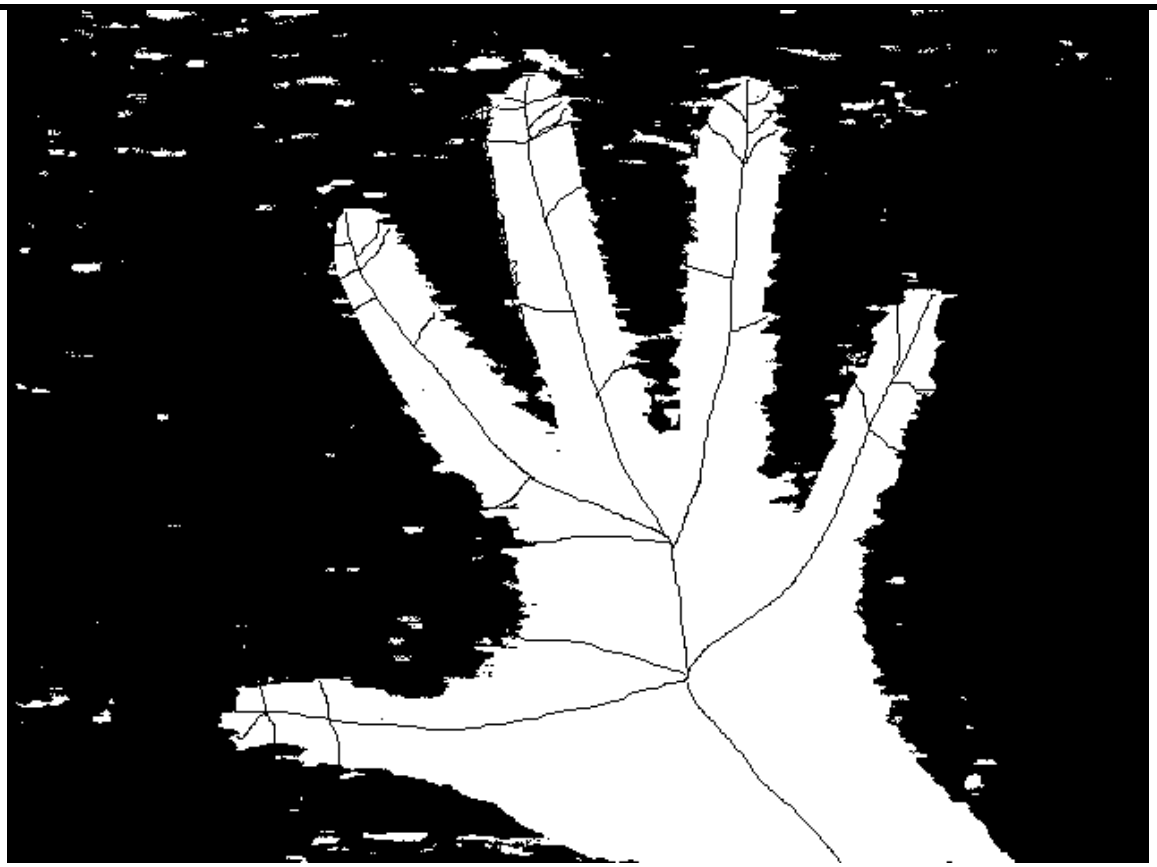


Рис. 1.2.5 – Скелетна модель кисті руки

Він представляє симбіоз попередніх методик отримання жесту руки, з виділенням ключових точок (точок ламаної) у контурах та обчислення центру мас та побудови скелету по цим даним.

Після обробки та виявлення контурів жеста, зображення передається безпосередньо у систему виявлення жестів, яка може бути представленням наборів алгоритмів, зокрема і алгоритмів штучного інтелекту. На виході отримуємо результат, який може набувати різноманітних форм, як масивом точок, двомірною, трьохмірною матрицями, матрицями n-мірності. Також кожен елемент матриці може представляти собою структуру з кількох елементів, наприклад координат x та y, що відображають координати положення ключової точки на зображенні та координати z, яка позначає глибину точки – тобто відображає віддаленість поточної точки від певної конкретної заданої еталонної точки.

Система розпізнавання жестів. Розпізнавання – це процес, що здійснює групування та оброблення просторово-часової інформації, отриманої на попередніх етапах і здійснення остаточного висновку щодо того, чи є розпізнана комбінація рухів елементів рук жестом, і якщо є, то до якого набору жестів її віднести. Таким чином, здійснюється інтерпретація виявлених особливостей

розміщення рук і яке смислове навантаження несе даний жест. Для вирішення цього питання вибирають алгоритми штучного інтелекту, саме штучні нейронні мережі, також відомі як алгоритми машинного навчання. Проте, перед початком роботи такої системи, необхідно провести її навчання. Під навчанням розуміють надання системі певного об'єму однотипних даних із мітками, що дає змогу отримати систему із стійкою здатністю вірно реагувати на дані, які їй надають і на основі цих даних, робити той чи інший висновок – надавати відповідь. Об'єм даних, який необхідний для досягнення повного навчання нейромережі є індивідуальним і залежить від типу вибраного алгоритму і конфігурацій, заданих у даному алгоритму. Для розпізнавання переважно застосовують кілька систем, серед яких варто виділити: нейромережа з зворотнім поширенням помилки (ЗПП), мережа Хопфілда, мережа Хемінга.

Нейромережа З.П.П. є класичним представленням нейромереж, і вважається однією з основоположниць у світі штучного інтелекту. Вона надає кращі результати у розпізнаванні жестів, проте вимагає значно більшого часу на навчання та здійснення аналізу, ніж мережі Хопфілда та Хемінга, які у свою чергу здатні швидко навчатися, але їх ефективність менша. Також варто зауважити, що мережа Хопфілда може застосовуватися до обмеженої кількості образів. Проте кожна мережа надає можливість здійснювати розпізнавання жестів і здійснювати їх класифікацію.

На кінець, нейромережа будь-якого зразка, у кінці роботи надає відповідь, що саме за жест було виявленно. Після цього необхідно провести класифікацію жеста, щоб встановити, яку саме інформацію необхідно передати на вихід через інтерфейс виведення інформації, що є фінальним акордом роботи інтерактивної системи керування жестами. Також, на основі отриманого результату, система може передавати певну графічну інформацію, яка може відображатися на графічному екрані, за необхідності.

Інтерфейс виведення інформації. Фінальним елементом системи по керування жестами є інтерфейс, через який система здійснює надсилання інформації отримувачу. Отримувачем може слугувати будь-яка структура чи комплекс, який потребує даних від даної інтерактивної системи. На етапі розробки необхідно визначити, яку саме інформацію має надавати мережа. Що саме розуміється під цим? Перш за все, розробник повинен визначити, який тип

даних система повинна повертати: чи це буде «класичний» тип даних, наприклад цілочисельне значення, певний символ або значення з плаваючою комою, або користувацький тип даних, такий як структура або клас. Не обов'язково система має повертати цифровий тип даних, це також може бути аналоговий сигнал, як наприклад радіочастота. У такому випадку, розробник має передбачити реалізацію ЦАП у системі на даному етапі задля можливості роботи з аналоговими типами даних.

Відеоадаптер та Графічний дисплей. Даний елемент не є обов'язковим у системі керування жестами і може слугувати як додаткова можливість системи. Під час роботи системи, буває зручно мати можливість відразу переглядати певні результати її роботи, наприклад під час відладки системи можна виводити повідомлення про помилку або певні графічні елементи на екран, або відображати зображення відеокамери, на яке накладено певні спецефекти. В залежності від результату роботи всієї системи (як приклад – підсвічувати ключові точки кисті руки на зображенні).

Вивід графічної інформації розподіляється на два етапи – передача інформації від системи керування жестами у відеоадаптер і передача зображення від відеоадаптера на графічний дисплей. Під відеоадаптером розуміється такий програмний блок, який забезпечує отримання інформації від системи розпізнавання жестів (зображення і результату, отриманого в процесі обробки цього зображення системою), її аналіз, обробка початкового кадру, отриманого від камери, і накладення на нього графічних елементів та передача зображення на вивід графічному дисплею. Проте, не обов'язково має передаватися зображення - не всі дисплеї здатні відображати повноцінну графіку. Деякі дисплеї типу LCD можуть відображати лише просту графіку або текст, чого цілком вистачає для відладки системи.

Елемент управління камерою. Ще один додатковий модуль, який не є обов'язковим. Залежно від реалізації системи, камера може бути як статично закріпленою, тобто нерухомою, так такою, яка має здатність обертатися. У випадку, якщо камера має бути рухомою, необхідно передбачити елемент, що надасть таку можливість. Для цього чудовим рішенням буде альт-азимутальна установка (рис. 1.2.1).

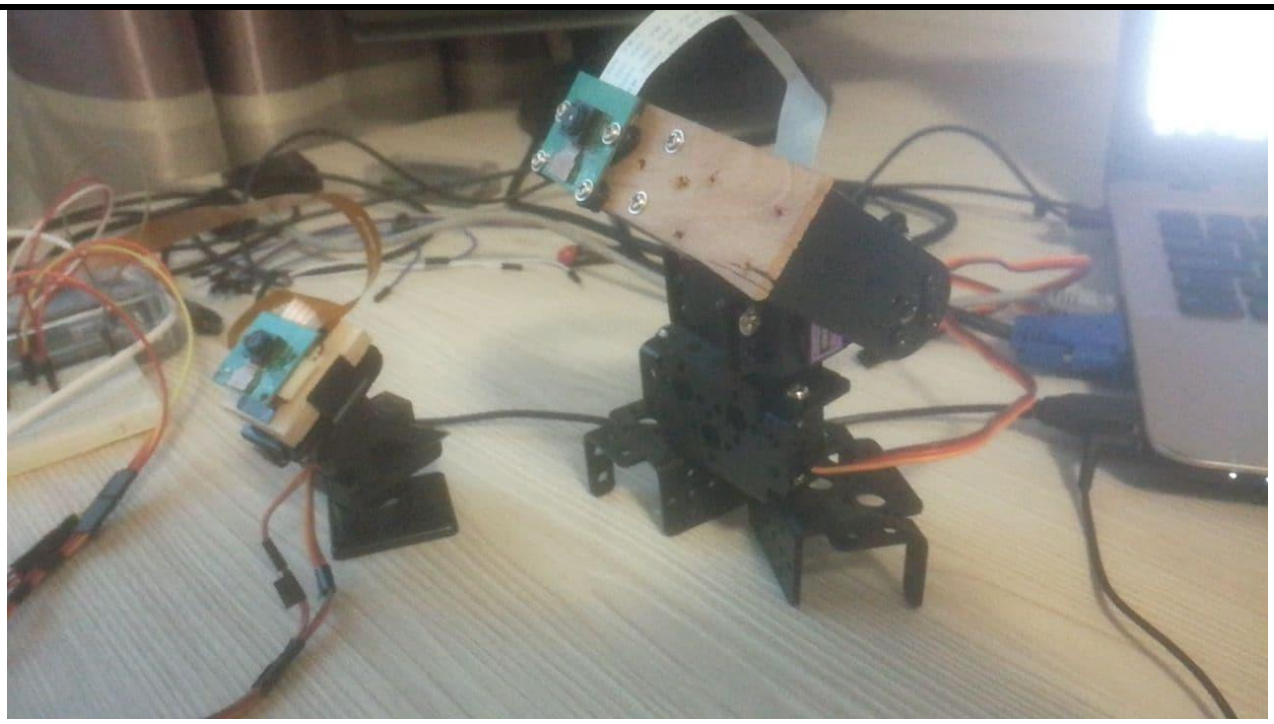


Рис. 1.2.1 – Альт-азимутальні установки із закріпленими камерами

Альт-азимутальна установка – це такий пристрій, що має вертикальну та горизонтальну осі обертання. Обертання може здійснюватися як вручну так і за допомогою спецзасобів, наприклад сервоприводів (рис. 1.2.2.).

Рис. 1.2.2. – Сервопривід марки SG 90

Залежно від марки, сервопривід зазвичай може обертатися або на 180˚ або на 360˚, що буде визначати можливості обертання камери. Керування сервоприводами здійснює система керування жестами на етапі виявлення жеста на зображенні. Залежно від положення кисті руки на зображенні, система надає відповідні сигнали, які скеровують сервоприводи, що скеровує альт-азимутальну установку з камерою у вірному напрямку.

Розглянемо приклади використання інтерактивних систем розпізнавання жестів у різноманітних проєктах.

Наразі поширене явище створення пристроїв, відомих як інтернет речі (англ. Internet of Things, IoT). Даний термін вперше був уведений британським інженером

Кевіном Ештоном, який прагнув описати систему, у якій різноманітні об’єкти здатні бути пов’язаними між собою, мають доступ до виходу у мережу Інтернет (або в будь-яку іншу мережу, зокрема локальну), різноманітними датчиками та сенсорами. Взаємозв’язок і спільна робота над спільними задачами роблять IoT потужними засобами для виконання різноманітного кола задач, з якими людина стискається протягом всього циклу дня. Наприклад, найпоширенішим завданням, яке виконують IoT є організація роботи побутових пристроїв у будинку – система, відома як «розумний дім». Система «Розумний дім» - це система, яка складається з певного набору пристроїв, які розташовуються у будинку, або поряд будинку, призначених виконувати набір дій з списку повсякденних завдань, без участі людини. «Розумний дім» може мати набір різноманітних датчиків – датчики освітлення, руху, вологості і т.п.. Одним із таких датчиків може бути інтерактивний модуль керування жестами. Система, яка буде захоплювати та відстежувати рухи рук господаря та розпізнавати набір жестів, що дозволить керувати найрізноманітнішою технікою у домі, як збільшення гучності на медіаплеєрі, керування освітленням або увімкненням побутових пристроїв лише одним рухом руки.

Якщо розглядати можливі області застосування подібних інтерактивних систем для керування, варто згадати системи, що впроваджені у мобільних телефонах та мобільних додатках. Зокрема, у них надана можливість захоплення та відстеження рук, розпізнавання та класифікація жестів рук, що надає зможу виконувати широкий спектр дій, як наприклад: здійснення знімку екрана, початок та закінчення відеозапису екрана телефона, розблокування пристрою, збільшення/зменшення гучності і т.п..

Також чудовим прикладом може слугувати інтегрована система керування жестами у бортових комп’ютерах сучасних автомобілів. Наприклад, застосування жестів передбачено у керування мультимедією автомобіля. Прості жести, такі як "вказати" або "змахнути" надають можливість викликати певні функції - наприклад, прийняти або відхилити дзвінок. Також передбачена регулювання гучність аудіосистеми завдяки обертовим рухом вказівного пальця. Здійснення планування маршруту в навігаційній системі також здійснюється за допомогою жестів рук, надана можливість за допомогою жестикуляції керувати навігацією в меню. Система управління жестами чудово доповнює канонічний інтерфейс

керування мультимедією, такий як кнопковий.

На завершення, можна зазначити перспективний напрямок застосування систем такого виду, як наприклад у виробництві, зокрема при керування різною апаратурою, як на кшталт крани чи маніпулятори. Інтуїтивно зрозумілі жести нададуть можливість значно спростити роботу та проводити навчання нового персоналу швидше. Просте такі системи лише існують у вигляді ескізів та планів, проте, враховуючи тенденцію технологічного розвитку людства за останній час, проєктування, розробка та впровадження таких систем є питанням часу.

1.3. Теорія розпізнавання образів

Виявлення жестів є окремим складним завданням, яке передбачає застосування цілого набору різноманітних операцій та алгоритмів.

Що собою представляє теорія розпізнавання образів? Це набір теоретичних основ та методик ідентифікації та класифікації предметів, процесів, явищ, за якими закріплена певна вибірка певних специфічних ознак, які допомагають точно ідентифікувати сутність із сукупності інших сутностей. Варто дати визначення, що саме розуміється під терміном «об’єкт». У теорії розпізнавання образів, об’єктом можна назвати виокремлену частину Всесвіту, що має певну форму та вид, за допомогою чого можна виділити набір ознак, притаманних саме цьому об’єкту. Таких ознак може бути вельми велика кількість, тому важливо виокремити ті, які найбіль точно можуть допомогти однозначно класифікувати об’єкт. Відібравши вибірку ознак, необхідно вказати процедуру, яка описуватиме змінну кожної ознак. Наприклад, переважна більшість рослин мають зелене забарвлення, і зелений колір – одна із ключових ознак рослин. Також ключовою ознакою рослин є наявність листків, які у свою чергу мають зелене забарвлення. Проте, при настанні осені, колір листків змінюється від зеленого до помаранчевого, червоного, а через деякий час рослини взагалі позбуваються листків. Проте це не заважає нам ідентифікувати дані об’єкти як рослини. Отже колір та листки – це ознаки, а змінна кольору та наявність листків є варіацією цих ознак. Таким чином, кожний об’єкт – це виокремлена сутність серед інших сутностей із набором ознак, які однозначного ідентифікують об’єкт, а для кожної ознаки відоме правило, за яким ознака може варіюватися, тобто набувати різних значень. Тепер, давши опис

такому поняттю як об’єкт, можна говорити про розпізнавання об’єктів.

На сьогоднішній день, задача розпізнавання залишається доволі складним завданням, як у теоретичному розумінні, так і у практичному втілені. При формуванні теоретичної бази, розглядають два напрямки. Перший спосіб – спостереження та аналіз зорових можливостей живих істот, дослідження здібностей до розпізнавання, побудова пояснення, як саме здійснюється зорове розпізнавання об’єктів та явищ живими істотами і на основі отриманих знань. Другий підхід – розвиток і дослідження математичної моделі, яка описує роботу розпізнавання сутностей, розробка окремих пристроїв, які спеціалізуються на проведені роботи подібного плану. Обидва способи надають широкий спектр можливостей по розробці і вдосконалень таких систем. Наприклад, камера, яка слугує ключовим елементом у системах розпізнавання, є прототипом людського ока.

Тепер пункти, які дозволять описати систему виявлення образів (жестів):

1) Перш за все, необхідно вказати, який саме об’єкт, або вибірка об’єктів буде підпадати під розпізнавання, і виділити основні ознаки в цих об’єктів.

2) Побудувати математичну модель, яка описуватиме той чи інший об’єкт.

3) Вибрати необхідний алгоритм, або набір алгоритмів, які здійснюватимуть виявлення об’єктів.

4) Забезпечити необхідні перетворення початкових даних (наприклад кадру зображення, на якому припускається наявність об’єкта) для успішного опрацювання цих даних вибраним алгоритмом.

Розглядаючи розпізнавання об’єктів у контексті розпізнавання жестів руки, очевидно, що об’єктом розпізнавання слугуватиме кисть руки. З фізіологічної точки зору, це елемент кінцівки людської руки. Ознаки руки – наявність пальців, спецефічна форма долоні (силует, що нагадує коло, овал) та заданий спектр кольорів людської шкіри. За даними ознаками, людина здатна без проблем

ідентифікувати кисть руки серед набору інших об’єкті і провести цю ідентифікацію надзвичайно точно.

Тепер необхідно провести побудову математичної моделі кисті руки. Математична модель – це певні співвідношення, задані через певні математичні правила, які описують досліджуваний об’єкт. Якщо описувати математичну модуль кисті руки, то найкрашим способом буде подача її у вигляді структури, яка складається з набору менших підструктур – точок, які описуються кількома параметрами: координатами положення точки у площині XY та змінної Z, яка позначає віддаленість точки від місця спостереження (камери). Також точку Z можна розглядати як третю координату, і таким чином маємо набір точок, що описують просторове розміщення ключових точок кисті руки (у площині XYZ). Підсумовоючи, можна створити структуру, яка складається з 21 ключової точки кисті руки (рис. 1.3.1.), а кожна точка, у свою чергу, описується трьома змінними – просторовими координатами.

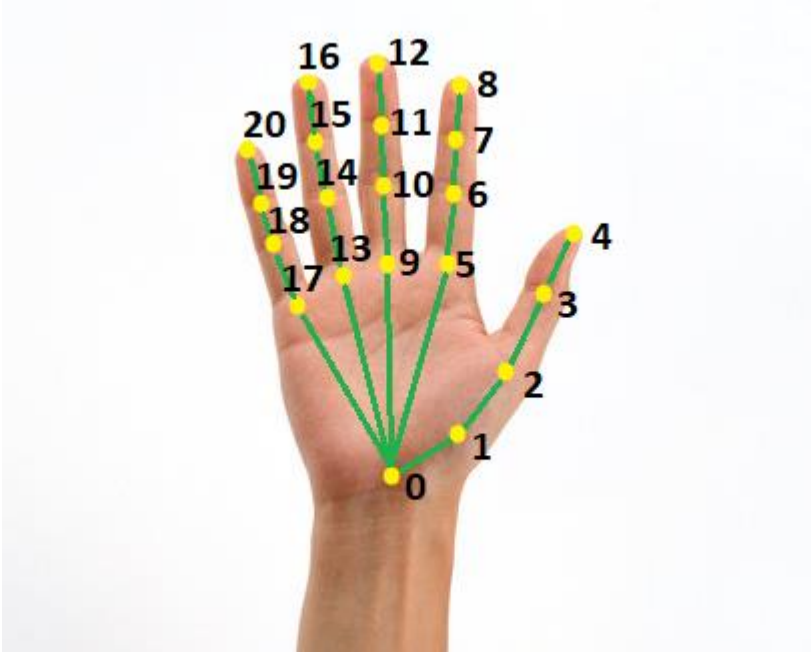


Рис. 1.3.1. – Набір ключових точок руки

Як видно, дані 21 точки – це місця де містяться суглоби, або однозначно охарактеризовують конкретну ділянку об’єкту (кінчики пальців).

Наступним етапом в побудові системи виявлення жестів, є побудова математичного алгоритму, призначеного вирішити проблему виявлення жеста. Одночасно це є одним із найбільш важких моментів у всій системі виявлення об’єктів, але і найбільш «широким» - є вельми великий набір математичних структур, які здатні вирішити дану проблему. Надалі буде наведено деякі із них. Вони являють одні з найбільш ранніх варіантів алгоритмів, запропонованих вирішити задачу розпізнавання, проте зараз вони слугують основою для інших, більш продвинутих алгоритмів, або застосовуються разом з іншими алгоритмами, зокрема з моделями штучного інтелекту.

Виявлення об’єкта зазвичай є процесом «порівняння» по певному зразку із частинами зображення на якому необхідно знайти подібний об’єкт. Проте, різні алгоритми передбачають різну інтерпретацію «зразка» у такий вигляд, яким його зручно надавати системі для проведення роботи розпізнавання. Як було сказано раніше, об’єкт має певні характеристики, які його однозначно ідентифікують: колір, форма, розташування у просторі, певні спеціальні характеристики (наприклад, пальці на долоні однозначно ідентифікують об’єкт як кисть руки) і т.п.. Зважаючи на ці фактори, об’єкти підпадають під класифікацію. Розглядаючи відеопотік, ставиться задача виділити об’єкти певного класу, а не проводити ідентифікацію всіх сутностей на кадрі.

Таким чином, можна таку класифікацію об’єктів:

- Класифікація по формі
- Класифікація по просторовому розміщенню
- Класифікація по кольору

Класифікацію по формі можна розділити на два напрямки: по заданому шаблону та по аналітичній заданій формулі.

Пошук по заданому шаблону можна вважати еталонним. У такому разі, шаблон має набувати форми, подібної до форми об’єкта, який має знаходитися на фото. Такий шаблон, із необхідно заданою формою, переміщається по зображенню, одночасно проводиться розрахунок положення шаблону на кадрі по певній заданій формулі, і у разі, якщо значення результату є вищим якогось

шуканий об’єкт. На рис. 1.3.1 можна побачити шаблон певного порогового значення – у зоні шаблона швидше за все знаходиться «сердечка», який захоплює частину умовного зображення.

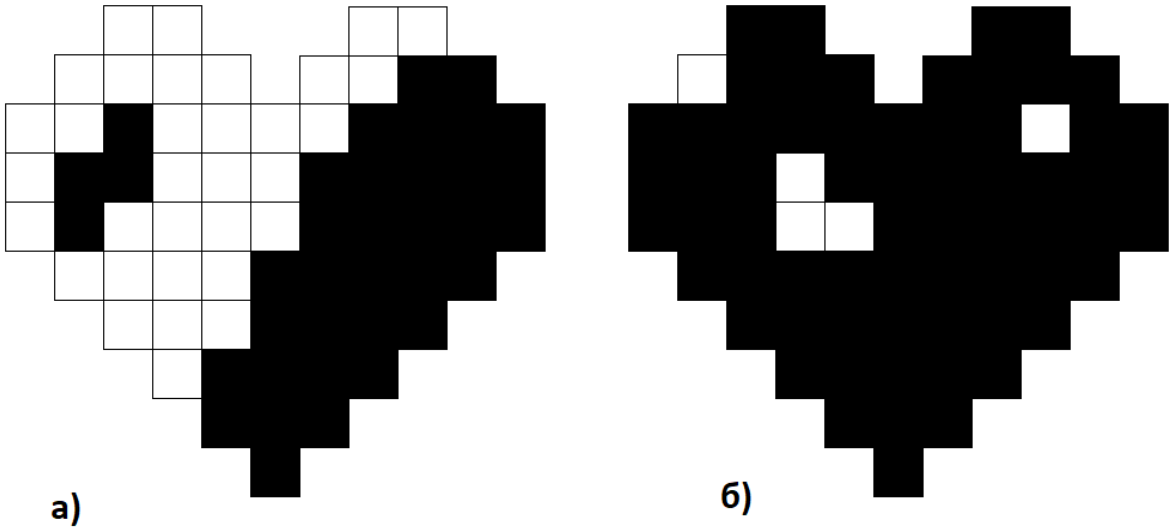


Рис. 1.3.1. – Демонстрація роботи порогової функції

Шаблон вміщає у собі 72 пікселя. На кожному етапі підраховується кількість чорних пікселей, і у разі, якщо їх кількість буде більша за певне задане значення (наприклад більше за 90% від загальної кількості пікселей у шаблоні, а саме 65). Таким чином, на розглядаючи варіант а) на рисунку 1.3.1 порогова функція не спрацює, тому що сума чорних пікселей складає лише 37, що менше за 65. У варіанті б) таких пікселей 67, що більше ніж 65 пікселей, таким чином, можна стверджувати, що був виявлений необхідний об’єкт.

Пошук по аналітичній заданій формулі полягає у застосуванні певної математичної формули, яка вміщає у собі опрацювання певної кількості точок на зображенні. Після опрацювання даного набору точок, фігура, задана математичної формулою, просто здвигається по координатним осям і обрахунок проводиться заново. Під час опрацювань точок функції, відбувається розгляд характеристики кожної точки, наприкад переглядається інтенсивність яскравості точок, Наприклад, маємо формулу кола:

$$x^2 + y^2 = R^2,$$

де, x та y – координати центра кола,

R – діаметр кола.

Нехай кожне значення точки є пікселем з кольором, що відповідає градієнту сірого (значення від 0 до 255). Для того, щоб об’єкт був ідентифікований, припустимо що 85% від усіх точок кола, повинні мати значення від 100 до 150. Таким чином, здійснюючи обчислення математичної функції, здвигаючи фігуру по координатним осям, і перевірку на дійсність заданої умови для кожної точки, можна проводити ідентифікацію об’єкта.

Засоби статичного аналізу допомагають розробникам, спеціалістам з тестування і співробітникам забезпечення якості в пошуку дефектів до початку динамічного тестування. Основні цілі включають:

Класифікація по просторовому розміщенню. Такоє, важливим методиос пошуку є пошук по просторовому розміщеню. При здійсненні пошуку об’єкта в області кадр, здійснюється припущення, що шукані області представляють собою довгасті, витягнуті у певному напрямку набір пікселів, які мають схожу яскравість. Так само застосовується безліч інших припущень щодо взаємного розташування об'єктів. Наприклад, об’єкт може містити певні мітки, які вирізнятимуть об’єкт, або якісь інші деталі, виявивши ці деталі, можна легко здійснити класифікацію шуканих об’єктів. Тобто, якщо існує метод, який забезпечує виділення в кадрі, наприклад, очей людини або носа, то, використовуючи ці деталі, можна зробити припущення, де знаходяться інші елементи об’єкта і сам об’єкт.

Класифікація по кольору є одним із основоположних методів по виявленню об’єктів. Зазвичай використовується разом з іншими методами для отримання максимального ефекту. Багато об’єктів підлягають ідентифікації через аналіз їх кольворої гами – вони або володіють сталим кольоровим забарвленням, або мати кольорову гаму у заданому діапазоні. Завдяки наявності великої кількості способів представлення кольорів у зображені (Наприклад всім відома RGB-система, або HSV), появляється можливість проводити глибокий і точний аналіз зображення. Для роботи подібної системи необхідно вибрати кольорову модель і задати необхідні діапазони для координат кольора, який шукається. Наприклад кольорова модель HSV (Hue, Saturation, Value). Hue (тон), Saturation (насиченість) , Value (яскравість) можуть приймати значення від 0 до 100 (залежно від потреб, можуть приводитися до довільних діапазонів).

Задаючи конкретні діапазони для кожної змінної, можна буде досягти ефекту, коли система зможе однозначно ідентифікувати заданий колір на зображені і вказати ймовірне місце розташування об’єкта на зображені.

Вищеперелічені способи, наразі, використовуються як один із основних елементів при виявленні об’єктів. Зазвичай їх поєднують між собою і застосовують разом із іншими математичними функціями по обробці зображень, як наприклад виявлення контурів, або розсіювання фото.

Всі ці перетворення здійснюють лише половину роботи по виявленню об’єктів. Надалі, необхідно створити спеціальну математичну функцію, яка зважаючи на всі ці критерії (як наприклад, розмір, форма, колір, розміщення об’єкта), могла проводити автоматичне розпізнавання об’єкту. Для цього використовують алгоритми штучного інтелекту, зокрема нейронні мережі. Більше про нейронні мережі буде висвітлено у підрозділі 1.4.

На кінець, потрібно забезпечити необхідні перетворення вхідних даних, які будуть надаватися системі виявленню об’єкті. Наприклад, вирішити, у якому формати має бути зображення, вирішити, до якої кольорової моделі варто привести зображення, здійснити процедуру по знешкодженню шумів і т.п..

На завершення, можна зробити висновок, що виявлення об’єктів це складна процедура, яка розбивається на кілька етапів. Вирішивши питання по реалізації цих етапів, а саме: вибору моделі об’єкта, який відстежується, побудова математичної моделі цього об’єкта, вибір алгоритмів по реалізації виявлення, здійснення підготовки вхідних даних, можна створити цілком працездатну систему.

1.4. Концепція та теоретичне представлення нейронних мереж

Як варто трактувати поняття «Штучний інтелект»? Даний термін є обширним і має кілька трактувань. Одне з них, полягає в тому, що штучним інтелектом можна назвати таку систему, яка має здатність виконувати творчі функції, які вважаються притаманні лише людині. Таким чином, штучний інтелект можна розглядати як спробу імітації людського розуму або створення подібного аналога. Проте, це є одним із загальних трактувань. Насправді, під

штучним інтелектом розуміється не тільки спроба відтворити людський інтелект, свідомість, а набір алгоритмів, які мають можливість виконувати функції, які якісно виконувати здатна лише людина. Наприклад, розпізнавання зорових образів. Для людине не є вельми важко розпізнати той чи інший об’єкт, навіть за умови, що раніше людина не стискалася з ним. Кожен з нас, при виявленні незнайомої квітки, може точно сказати, що даний об’єкт – це рослина, і відноситься до квіток. Завдяки тому факту, що людина має можливість однозначно ідентифікувати ключові елементи рослини, такі як стебло, листки, бутон, вона може встановити, що даний об’єкт – квітка. На жаль, для обчислювальних систем це є важкою задачею і донині. Для вирішення задач подібного типу звертаються до цього напрямку комп’ютерних наук, а саме до нейронних мереж.

Штучні нейронні мережі – це такі системи, які є прототипами справжніх біологічних нейромереж. Батьком нейронних мереж по праву вважається Френк Розенблат, який описав у своїй книзі «Принципи нейродинаміки: перцептрони і теорія механізма мозку» загальний принцип роботи мозку людини, увів і описав прототип біологічного нейрону – перцептрон, і надав розгорнуте поняття роботи набору таких перцептронів – нейромережі. Його праця надовго заклала фундамент у принципах формування штучного інтелекту, як такого. Надалі буде наданий загальний опис для одного з основоположних алгоритмів нейронних мереж – алгоритму зворотного поширення помилки (ЗПП), який використовувався зокрема у реалізації цієї дипломної роботи. Детальний опис роботи цього алгоритму буде наведено у Розділі 2. Також буде данний огляд ще кількох алгоритмів, які також виконують роботу у розпізнанні об’єктів (жестів) і логіки, яку вони несуть.

Алгоритм зворотнього поширення помилки – алгоритм реалізації штучної нейронної мережі на основі перцептронів за Розенбладом. Розглянемо загальну будову такої мережі (рис. 1.4.1).

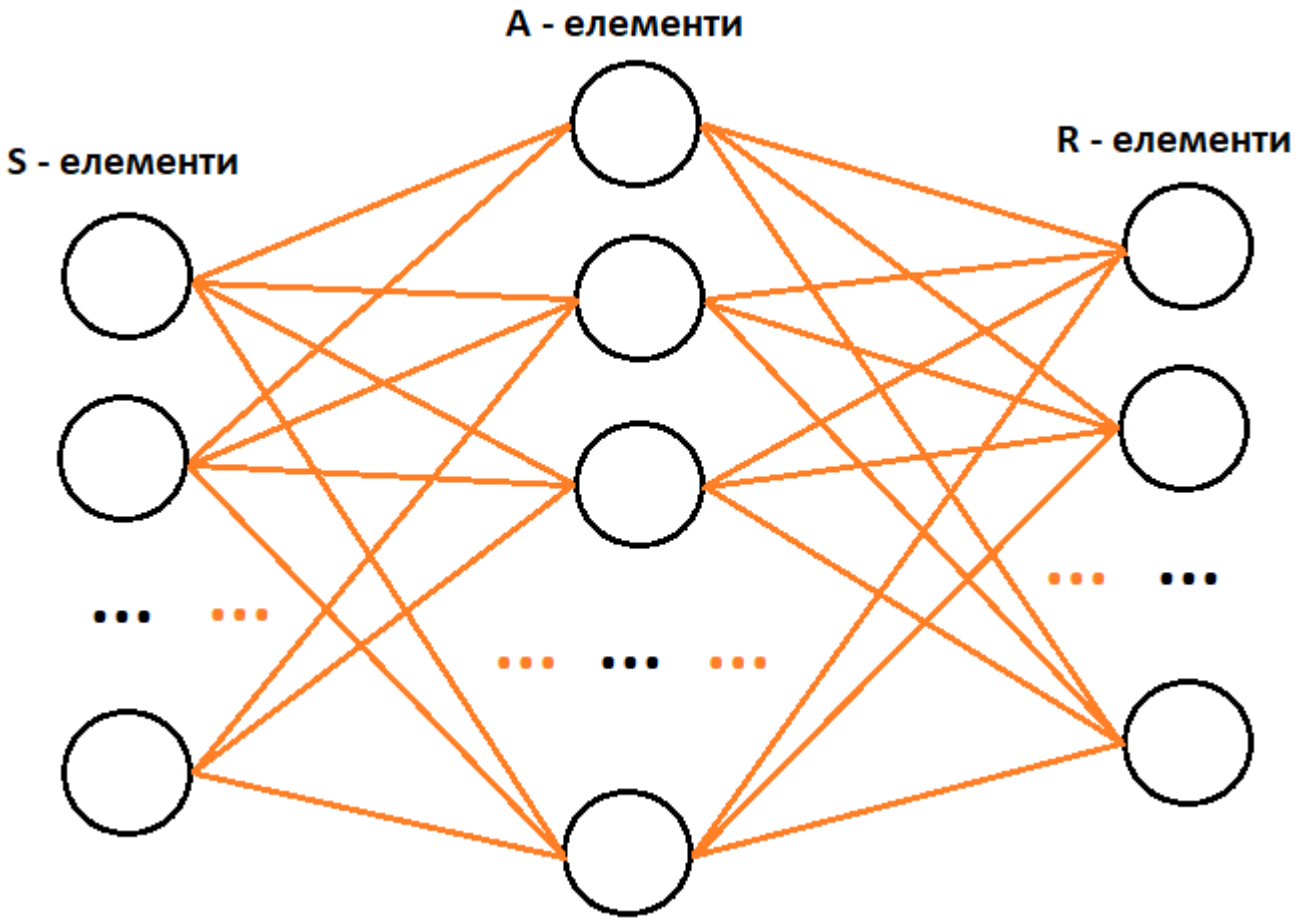


Рис. 1.4.1 – Загальна схема нейронної мережі

Примітка: у своїй книзі, Розенблат розглядає структуру зображену на рисунку 1.4.1. як окремий персептрон і під всією нейронною системою, яка має один прихований шар (А-елементи), розуміє саме персептрон. Із розвитком галузі штучного інтелекту, під поняття «персептрон» будуть розуміти окремий вузел з вхідними і вихідними зв’язками з цієї нейромережі. Надалі під персептроном буде розумітися саме окремий вузел, а під нейронною мережею – поєднанян цих вузлів.

Нейронна мережа за Розенблатом, складається із певної вибірки перцептронів, які організовані у структури – шари. Як бачимо, нейромережа, розроблена за алгоритмом ЗПП, складається з кількох різних шарів персептронів, названих S, A, та R елементи (або вхідний, прихований та вихідний шари відповідно). Кількість прихованих шарів може бути довільною, як і кільцість перцептронів. Між кожним перцептроном у шарі є зв'язок, який сполучає його з перцептронами наступного шару. Через дані зв’язки відбувається обмін, обробка і ,певним чином, зберігання інформації. Самі вузли мережі, персептрони, зберігають значення обчислень з попередніх шарів.

Варто розглягути, чим взагалі є перцептрон. На рисунку 1.4.2. зображений перцептрон із зв’язками.

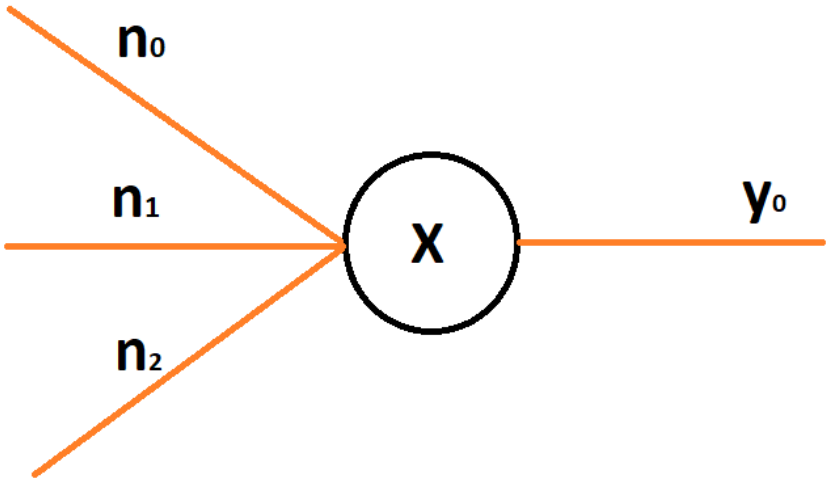


Рис. 1.4.2. - Перцептрон

Даний перцептрон має кілька взідних зв'язків ($n_0 – n_2$) та один вихідний (y_0), хоча кількість зв'язків може бути різною, і саму основу (x). Як зазначалося раніше, перцептрон є прототипом реального біологічного нейрона, який працює завдяки збудженню зв'язків – дендритів, і нейрон, в залежності, наскільки активні дендрити, збуджує або не збуджує вихідний відросток – аксон. Робота перцептрона є аналогічною. Перцептрон збирає поточну інформацію із вхідних зв'язків (дендритів), пропускає її через певну порогову функцію і передає оброблену інформацію на вихідний зв'язок (аксон). Таким чином, інформація проходить по всій нейромережі – від вхідного шару, проходить через приховані шари і оброблена інформація виводиться на вихідний шар. Після цього, отриманий результат аналізується, і у разі помилки – значення ваг персептронів корегується. Проводячи велику кількість ітерацій цих дій, нейромережа проходить етап навчання, що надалі дозволяє їй виконувати завдання із великою точністю. Таким чином, подібну систему можна успішно навчати і покладатися на неї у виконані різноманітної роботи, зокрема у розпізнанні образів. Надаючи системі певні зображення (або іншу інформацію, що характеризують однозначно об'єкт) із мітками, можна навчити розрізняти і класифікувати різноманітні об'єкти.

Варто також сказати про деякі інші нейронні мережі, які використовують у розпізнаванні образів. Для зорового розпізнання елементі застосовують мережі Хопфілда та Хемінга, яка є розвитком першої. Ці мережі також використовують персептрони, проте мають дещо інше розташування персептронів і складаються з

єдиного шару персептронів. Переглянути структуру такої нейромережі можна на рисунку 1.4.3..

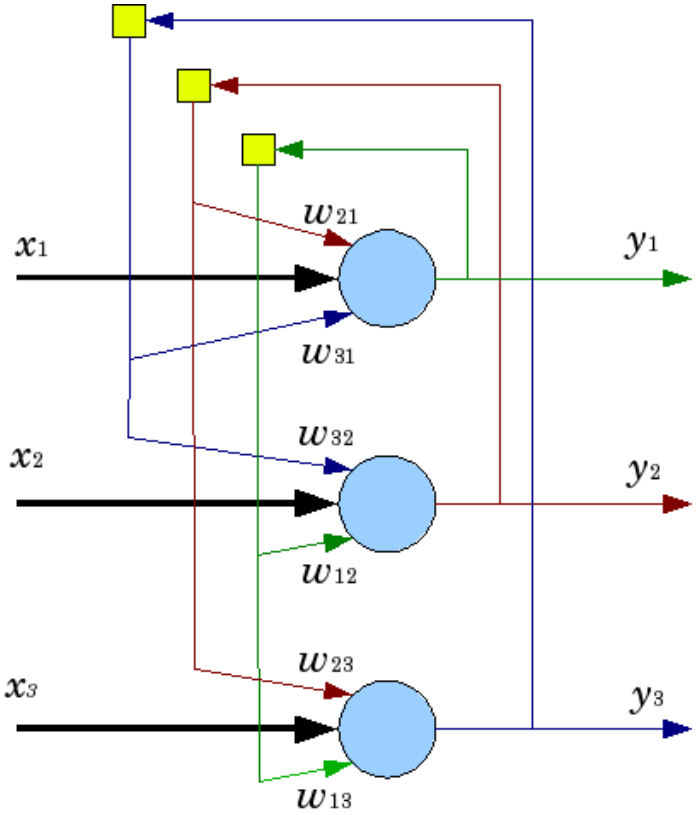


Рис. 1.4.3. – Схема мережі Хопфілда

Можна побачити, що у нейромережі цього виду, розташування та з'єднання персептронів відрізняється від нейромережі Розенблата. Алгоритм навчання мережі Хопфілда значно відрізняється алгоритму навчання за методом ЗПП. Відмінність полягає в тому, що замість послідовного наближення до потрібного стану з обчисленням помилок, все коефіцієнти матриці розраховуються за однією формулою, за один цикл, після чого мережа відразу готова до роботи. Перевага даної нейромережі є у швидкості навчання, проте недоліком є нездатність мережі бути навченою розпізнавати велику кількість образів.

Підбиваючи підсумок, бачимо, що нейронні мережі є прототипами справжніх біологічних нервових систем. Основуючись на перцептронах – прототипах нейронів, дані мережі здатні проводити вирішення складних задач, зокрема, розпізнавання образів та їх класифікація.

1.5. Застосування одноплатних комп'ютерів у модулях керування жестами.

При створенні інтерактивних систем керування жестами, необхідно вирішити питання апаратної частини, тобто на якій електронно-обчислювальній машині буде виконуватися програмний код.

Найбільшого поширення набуло використання одноплатних ЕОМ через задовільні характеристики та компактність. Одноплатний ЕОМ – це комплекс технічних засобів, елементи, які призначенні для опрацювання інформації в процесі вирішення обчислювальних задач, розміщенні на одній платі. Зазвичай такий пристрій включає у собі всі елементи сучасного комп'ютера з архітектурою Фон Неймана – мікропроцесор, енергонезалежна пам'ять, ОЗП, система вводу-виводу інформації і інші елементи.

Останнім часом, появилася величезна кількість одноплатних комп'ютерів, що дозволяють реалізувати різноманітні компактні та автономні системи керування жестами. Виробники пропонують різноманітну кількість таких пристроїв, як наприклад: Raspberry Pi, Banana Pi, Orange Pi, NanoPi Neo Air, Libre Computer Renegade і інші. У таблиці 1.5.1 наведена порівняльна характеристика даних ЕОМ.

Таблиця 1.5.1

Порівняльна таблиця одноплатних ЕОМ

ЕОМ	Ціна	Процесор	Ядра	ОЗП	Пам'ять
Raspberry Pi 3B	35 $	Broadcom BCM2837	4x A53, 1.2 Ghz	1 Gb	MicroSD
Banana Pi M3	74 $	Allwinner A83T	8x A7, 2GHz	2 Gb	8GB eMMC, MicroSD
Orange Pi Prime	30 $	Allwinner H5	4x A53	2 Gb	MicroSD
NanoPi Neo Air	20 $	Allwinner H3	4x A7, 1.2GHz	512 Mb	8GB eMMC
Libre Computer Renegade	50 $	ARM Cortex	4x A53	2 Gb	MicroSD

В залежності від бюджету та апаратних потреб, ринок відкриває широкий набір різноманітних обчислювальних пристроїв з різним апаратним наповненням. Також дані комп’ютери підтримують операційні UNIX-системи, що дозволяє без проблем створювати та запускати у дію програмне забезпечення, а наявність різноманітних портів, надасть можливість підключати різноманітну переферію і модулі, що забезпечує певну «маневреність» при реалізації власних проєктів.

1.6. Висновки до розділу

Під час написання першого розділу, було наведено пояснення, що саме розуміється під терміном «інтерактивна система керування жестами». Описано загальну структуру такої системи, наведені основні і додаткові функціональні блоки. Кожен блок був детально описаний, наведено апаратні засоби, які потребують деякі з блоків, надані алгоритми роботи та методики, які використовуються при проєктуванні кожного з блоків. Також наведені відомості з теорії розпізнавання образів, наведені відомості про загальний принцип виявлення об’єктів та поясненно, якими інструментами та методами досягається результат такої роботи. Викладені загальні положення про нейронні мережі, наведені поширені алгоритми, які використовуються у системах керування жестами, які також застосовуються під час виявлення об’єктів. На кінець був проведений огляд апаратури, на якій, зазвичай, працюють системи розпізнавання жестів, а саме одноплатні комп’ютери.

Таким чином, у даном розділі була наведена загальна теорія по принципу роботи інтерактивних систем керування жестами.

РОЗДІЛ 2

ПРОЕКТУВАННЯ МОДУЛЯ РОЗПІЗНАВАНЯН ЖЕСТІВ РУК ДЛЯ КЕРУВАННЯ КВАДРОКОПТЕРОМ

2.1. Опис апаратних елементів, які застосовувалися під час проектування модуля.

2.1.1. Одноплатний комп’ютер.

Основою для виконання програмного коду та керування елементами модуля, був обраний одноплатна ЕОМ Raspberry Pi 3B+ (рис. 2.1.1.1.).

Рис. 2.1.1.1. – Raspberry Pi 3B+

Даний одноплатний комп’ютер відповідає усім необхідним вимогам, які потрібні для успішного проектування та створення автономного модуля, а саме:

- Компактність
- Задовільні апаратні можливості.
- Наявність інтерфейсу вводу-виводу загального призначення (GPIO)
- Мале споживання електроенергії
- Архітектура комп’ютера підтримує роботу UNIX-подібних операційних систем

Компактність. Розміри даного одноплатного комп’ютера наступні:

- довжина: 85 мм
- ширина: 56 мм
- висота: 17 мм

Задовільні апаратні можливості. Апаратні характеристики даного пристрою задовольняють потреби, які виникають при роботі програмного забезпечення.

Характеристики Raspberry Pi 3B+:

- Чіп: Broadcom BCM2837B0
- Процесор: 64-бітний 4-ядерний ARMv8 Cortex-A53 з тактовою частотою 1,4 ГГц
- Відеопроцесор: VideoCore IV 3D
- ОПЗ: 1 Гб LPDDR2 (900 МГц)
- Роз'єми: HDMI, 3,5 мм jack, USB 2.0 4 шт., роз'єм для microSD, 10/100 Ethernet, порт для дисплея DSI, потр для камери CSI-2, GPIO 40 pins,
- Безпроводні інтерфейси: WiFi 2,4 ГГц і 5 ГГц IEEE 802.11.b/g/n/ac 2,4 ГГц, Bluetooth: 4.2 Classic.

Вважливим моментом є наявність інтерфейс вводу-вивода загального призначення (GPIO). Даний інтерфейс забезпечую можливість підключення до ЕОМ різноманітних модулей і їх керування, також надає можливість через піни здійснювати надсилання сигналів.

Даний комп'ютер здатний працювати при малому споживанні електроенергії – достатньо напруги в 5V та сили струму в діапазоні 1.5-2 А.

Операційна система, вибрана для роботи, є Raspbian OS – операційна система, заснована на основі Debian.

2.1.2. Сервоприводи

Сервопривід – це пристрій в системах автоматичного регулювання або дистанційного керування, що за рахунок енергії допоміжного джерела здійснює механічне переміщення регулюючого органу відповідно до отримуваних від системи керування сигналів. Для забезпечення обертання рухомих частин модуля (альт-азимутальна установка), було вибрано сервоприводи Tower Pro MG90S (рис. 2.1.2.1.)

Рис. 2.1.2.1. - Сервопривід

Характеристики:

- Робоча напруга: 4-6 V
- Пусковий момент: 1.8 кг*см
- Кут повороту: 180˚
- Час повороту на 60: (при напрузі 4.8V без навантажень): 0.1 с, (при напрузі 6V без навантажень): 0.08 с
- Розміри: довжина: 32,3 мм , ширина: 33 мм, висота: 12,4 мм

2.1.3. Камера та альт-азимутальна установка.

Одним із основних елементів модуля є камера, яка здійснює відеозахоплення. Враховуючи, що одноплатним ЕОМ є Raspberry Pi, то вибір був здійснений на користь спеціального модуля камери для Raspberry Pi, версії 1.3 (рис. 2.1.3.1)

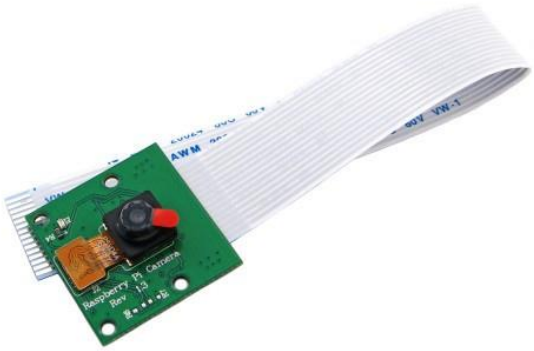


Рис. 2.1.3.1. – Модуль камери для Raspberry Pi, версія 1.3

Характеристики:

- Роздільна здатність матриці: 5 Мп
- Розмір матриці: 1/4", 3,76 × 2,74 мм
- Максимальний розмір зображення: 2592 × 1944 пікселів
- Розміри: довжина: 25 мм , ширина: 24 мм, висота: 9 мм

Задля забезпечення фіксації і обертання камери, було використано альт-азимутальну установку (рис. 2.1.3.2.).

Рис. 2.1.3.2. – Альт-азимутальна установка

Дана установка забезпечує можливість маніпулювання камерою. Обертання відбувається завдяки роботі сервоприводів.

2.1.4. Живлення

Живлення модуля відбувається за допомогою двох джерел.

Для забезпечення роботи одноплатного комп’ютера Raspberry Pi та сервоприводів, використовується два зовнішніх акумулятора (повербанк) Power Bank 3 10000 mAh (рис. 2.1.4.1.).

Рис. 2.1.4.1. – Зовнішній акумулятор

Вихідна напруга: 5V, сила струму: 1.5 A, ємність батареї: 10000 mAh.

На даний момент, у силу економії та відсутності нагальної потреби монтування модуля на квадрокоптер, живлення сервоприводів відбувається через зарядний пристрій з вихідною напругою в 9V (з подальшою конвертацією у 5V) та силою струму 1A.

2.2. Технології, застосовані для обробки зображень, реалізації процеса виявлення жестів та керування альт-азимутальною установкою.

2.2.1. Бібліотека Mediapipe

Mediapipe – це кросплатформенна бібліотека рішень задач комп’ютерного зору, на основі використання бібліотеки OpenCV та алгоритмів штучного інтелекту. OpenCV – бібліотека алгоритмів комп’ютерного зору та алгоритмів для опрацювання зображень та відеопотоків, яка написана на мові C/C++. Алгоритми штучного інтелекту, які використовуються у MediaPipe, забезпечуються завдяки відкритій бібліотеці для машиного навчання TensorFlow, яка розроблена компанією Google. Алгоритми TensorFlow містять інструменти для створення моделей нейроних мереж і їх навчання.

Робота даної бібліотеки організована у вигляді «конвеєра» або «трубопровода» (pipeline) – у систему подаються певні вхідні дані, які проходять через кілька вузлів обробки і користувач отримує на виході результат, залежно від заданих конфігурацій. Таким чином, робочою системою бібліотеки є пайплайн («трубопровід»), який зображується у вигляді графа. Сам граф складається низки елементів, а саме:

- Вершин графа. Їх також називають «калькуляторами». Надалі, вершини графа будуть називатися калькуляторами.
- Ребер графа. Ребрами графа є потоки, які передають інформацію від одного калькулятора до іншого.
- Пакетів. Пакет – це одиниця даних, яка курсує від одного калькулятора до іншого, завдяки потокам, які пов’язують калькулятори.

Також, в роботі графа застосована синхронізація між пакетами, задля забезпечення коректної і стабільної роботи системи.

Розберемо детальніше концепцію даної бібліотеки. Бібліотека є набором графів, які складені з калькуляторів, тому, для успішного використання бібліотеки, необхідно розуміти принцип роботи вершин графа, а саме, як все влаштовано «під капотом».

Кожен з калькуляторів є вузлом графа. Для того, щоб використати калькулятор у графі, необхідно провести певну послідовність дій, а саме:

- Визначити новий підклас – калькулятор, який бажаємо створити
- Реалізувати обов’язкові методи класа
- Визначити вхідні та вихідні потоки
- Провести реєстацію нового калькулятора

Розглянемо більш детальніше кожен із кроків.

Для того, щоб створити калькулятор, необхідно створити клас, дочірній класу CalculatorBase. Даний клас є батьківським для усіх класів, що реалізовують роботу калькуляторів.

Надалі необхідно провести перевизначення,, як мінімум, чотирьох методів: GetContract(), Open(), Process(), Close().

афедра КСУ				НАУ вввввв 000 ПЗ		
Виконав				Обгрунтування підходу до розробки системи	**Літера**	**Аркуш**	**Аркушів**
Керівник					Д	*26*	*61*
Консульт.							
Норм. контр.					СП 4……. 123		
Зав. Каф.							

Робота даної бібліотеки організована у вигляді «конвеєра» або «трубопровода» (pipeline) – у систему подаються певні вхідні дані, які проходять через кілька вузлів обробки і користувач отримує на виході результат, залежно від заданих конфігурацій. Таким чином, робочою системою бібліотеки є пайплайн («трубопровід»), який зображується у вигляді графа. Сам граф складається низки елементів, а саме:

- Вершин графа. Їх також називають «калькуляторами». Надалі, вершини графа будуть називатися калькуляторами.
- Ребер графа. Ребрами графа є потоки, які передають інформацію від одного калькулятора до іншого.
- Пакетів. Пакет – це одиниця даних, яка курсує від одного калькулятора до іншого, завдяки потокам, які пов’язують калькулятори.

Також, в роботі графа застосована синхронізація між пакетами, задля забезпечення коректної і стабільної роботи системи.

Розберемо детальніше концепцію даної бібліотеки. Бібліотека є набором графів, які складені з калькуляторів, які оперують пакетами, отриманих черех потоки, тому, для успішного використання бібліотеки, необхідно розуміти принцип роботи даних елементів, а саме, як все влаштовано «під капотом».

Перш за все, необхідно зрозуміти, що собою представляє одиниця даних, якою оперує калькулятор, а саме – пакет.

Пакет – це структура, яка може містити довільну кількість різноманітних даних. Тип цих даних може бути як користувацьким типом даних (структура, клас), так і одним із типів даних, передбачених у C/C++ (bool, int, char, double і т.п.). Окрім структури із даними, пакет містить часову мітку. Часова мітка показує, в який момент часу даний пакет був створений, що забезпечує синхроніхацію роботи графа.

У роботі графа фігурують потоки вхідного і вихідного типу. Саме через потоки здійснюється обмін інформацією між вершинами графу.

Кожен з калькуляторів є вузлом графа. Для того, щоб використати калькулятор у графі, необхідно провести певну послідовність дій, а саме:

- Визначити новий підклас – калькулятор, який бажаємо створити

- Реалізувати обов’язкові методи класа
- Визначити вхідні та вихідні потоки
- Провести реєстацію нового калькулятора

Розглянемо більш детальніше кожен із кроків.

Для того, щоб створити калькулятор, необхідно створити клас, дочірній класу CalculatorBase. Даний клас є батьківським для усіх класів, що реалізовують роботу калькуляторів.

Надалі необхідно провести перевизначення,, як мінімум, чотирьох методів: GetContract(), Open(), Process(), Close().

Метод GetContract(). Даний метод є конфігураційним і викликається лише один раз на початку роботи графа. У ньому необхідно вказати типи даних, які очікуються з вхідних потоків (якщо такі наявні) і тип даних, які будуть передаватися у вихідні потоки (якщо такі наявні). Як було сказано раніше, на початку роботи графа, вузел викликає дану функцію і звіряє тип даних, які поступають з вхідних/вихідних потоків з очікуваними типами даних, вказаними у даному методі.

Метод Open(). Після ініціалізації графа та виклику метода GetContract(), викликається даний метод. Даний метод проводить сполучення вхідного потоку з вершиною, тобто захоплює потік. Також, на даному етапі можна підключити спеціальні конфігураційні пакети, які є статичними (тобто, всі дані у даному пакеті не змінюються протягом робочого цикла графа). Також, у даному методі є можливіть перенаправити дані з вхідних потоків у вихідні потоки, у разі, якщо не потребується їх обробка.

Метод Process(). Основний метод, який проводить різноманітні операції над даними з пакетів. Даний метод викликається, якщо хоча б з одного вхідного потоку, приходить пакет. Даний метод може розпаралелюватися, тобто може здійснюватися виконання кількох таких методів одночасно. Особливістю роботи даного методу є те, що тут реалізована синхронізація з використанням часових міток. Якщо коротко описати цей процес, то жоден із методів Process() не розпочне виконання до тих пір, поки на відповідні вхідні потоки, які закріплені за одним із конкретних Process() не надійдуть пакети, у яких буде одна і та сама часова мітка. Таким чином досягається синхронізація. У разі успішного виконання цикла, метод має повертати значення absl::OkStatus(), у разі якщо повернуто значення, відмінне

від зазначеного, або в процесі роботи метода виникла помилка, викликається метод Close().

Метод Close(). Даний метод викликається, якщо у вхідних потоках калькулятора не залишилося пакетів, або метод Process() повернув значення, відмінне від absl::OkStatus() чи сталася помилка. У даній функції не має доступу до вхідних потоків (окрім підключених методом Open() спеціальних статичних пакетів), але є доступ до вихідних потоків, завдяки чому можна провести надання певної інформації у вихідні потоки.

Для кожного калькулятора необхідно створити конфігураційний файл, у якому вказуються потоки: як вхідні, так і вихідні. Кожен з потоків повинен мати унікальну числову і/або символьну мітку, яка однозначно ідентифікує його.

Синхронізація у MediaPipe досягається шляхом використання у пакетах часових міток і застосуванні механізму обробки цих міток. Виконання графа MediaPipe децентралізовано: відсутні будь-які глобальні «годинники», які контролювали виконання роботи графа, а, залежно від конфігурацій, різні вузли можуть обробляти дані з різними часовими мітками одночасно. Це забезпечує більш високу пропускну здатність за рахунок конвеєрної обробки. У заголовковому файлі default_input_stream_handler.h прописані налаштування по обробці пакетів, а саме:

- Якщо пакети з однаковою часовою міткою надаються в декількох вхідних потоках, вони завжди будуть оброблятися разом незалежно від порядку їх надходження в реальному часі
- Вхідні пакети проходять опрацювання в строго визначеному порядку – по зростанню часових міток пакетів
- Калькулятор готовий до обробки даних в найкоротші терміни з урахуванням вищевказаних правил

Калькулятор буде здійснювати обробку найновіших пакетів, у яких буде співпадати часова мітка із тією міткою, яка приходить з вхідного потоку.

Ознайомившись з основними елементами, переглянемо детально схему роботи калькулятора. Для наглядності, розглянемо рисунок 2.2.1.1..

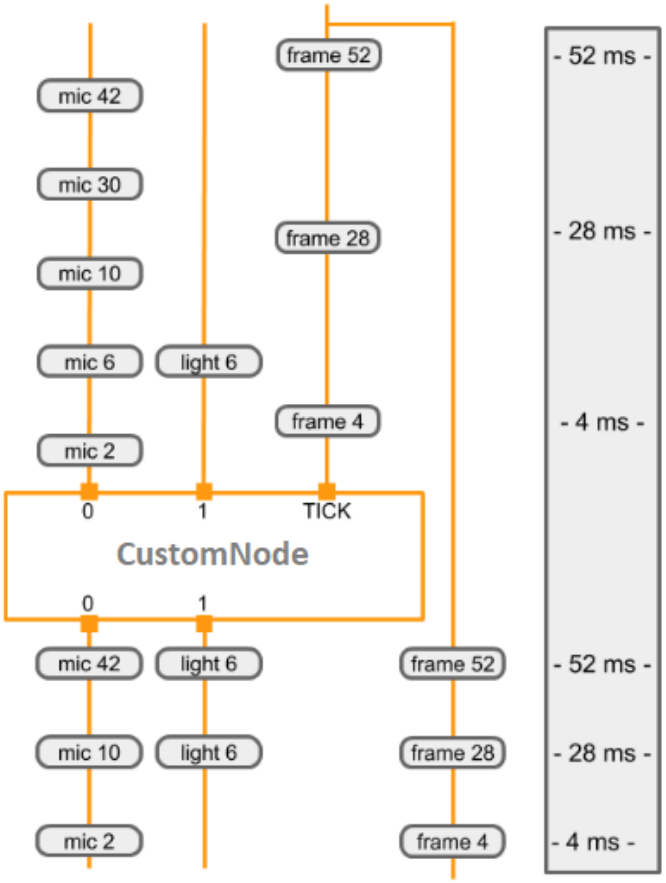


Рис. 2.2.1.1. – Демонстрація схеми роботи калькулятора

Маємо калькулятор, який збирає різноманітні дані і має їх упорядковувати за часовою міткою. З правого крайнього боку є шкала, яка відображає часові мітки. Вони позначають місцезнаходження часових міток. Наявний калькулятор CustomNode, який має 3 вхідних потоки (зліва на право: потім, що надає аудіодані (mic), потік, що надає дані по освітленості (light), та потік, відносно пакетів якого буде здійснюватися синхронізація (frame)). Як бачимо, вхідні потоки містять безліч наборів даних. Проте, завдяки надходження часових інтервалів, відносно яких можна здійснити синхронізацію (frame), відбувається формування пакетів із часовими мітками. Наприклад, якщо глянути на frame4 і його часову мітку 4 ms, то видно, що разом з ним, найновіші пакети, які прийдуть на вузел це mic2 та light6. Таким чином, калькулятор здійснює синхронне відправлення даних трьох пакетів у вихідні потоки. Якщо розглядати frame28, то найновіші пакети, які прийдуть на вхідні потоки є mic10 і light6, який не був оновлений, тому він знову використовується. Знову здійснюється синхронне відправлення пакетів і т.д..

Тепер, ознайомившись з принципом роботи ключових елементів графу, необхідно оглянути всю структуру графа і весь його робочий цикл.

Граф насправді є поєднанням комбінації менших графів, які з'єднані потоками. Дані графи називають підграфами. Це можна уявити конструктор, який будується із безліч невеличких блоків – даних підграфів. У файлі, де проводиться опис графа, через GraphConfignаявна позначають специфікацію, що описує топологію і функціональність графа MediaPipe. GraphConfig містить поля для настройки глобальних параметрів загального графа, наприклад, конфігурацій виконавця графа, кількості потоків і максимального розміру черги вхідних потоків. Головний граф позначається як CalculatorGraphConfig. Тепер необхідно визначити підграфи, які будуть використовуватися у головному графі. Вони позначаються у конфігураційному файлі як Subgraph. В них здійснюється налаштування вхідних потоків, вихідних потоків та вхідних статичних пакетів. Після цього, всі дані вносяться у один конфігураційний файл з розширенням .pbtxt, який зберігає дані про граф.

Таким чином, відбувається створення графа і побудова плану, по якому відбувається виконання методів класів (калькуляторів). Варто зауважити, що граф створюється щоразу заново при необхідності нових обчислень. Наприклад, у графі по виявленню жестів (який буде розглянуто далі), на кожен кадр відеопотоку створюється новий граф.

Під час проектування модуля керування жестами, був застосований граф MediaPipe Hands. Із загальною структурою головного графу можна ознайомитися на рисунку 2.2.1.2..

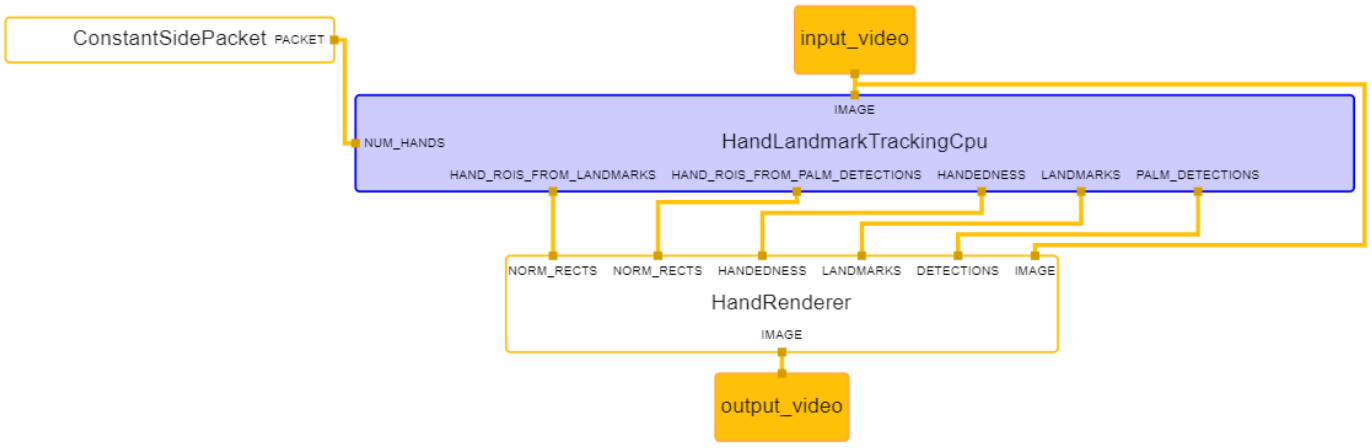


Рис. 2.2.1.2. – Головний граф

На рисунку 2.2.1.3. зображений підграф, що описує процес створення фінального вихідного зображення.

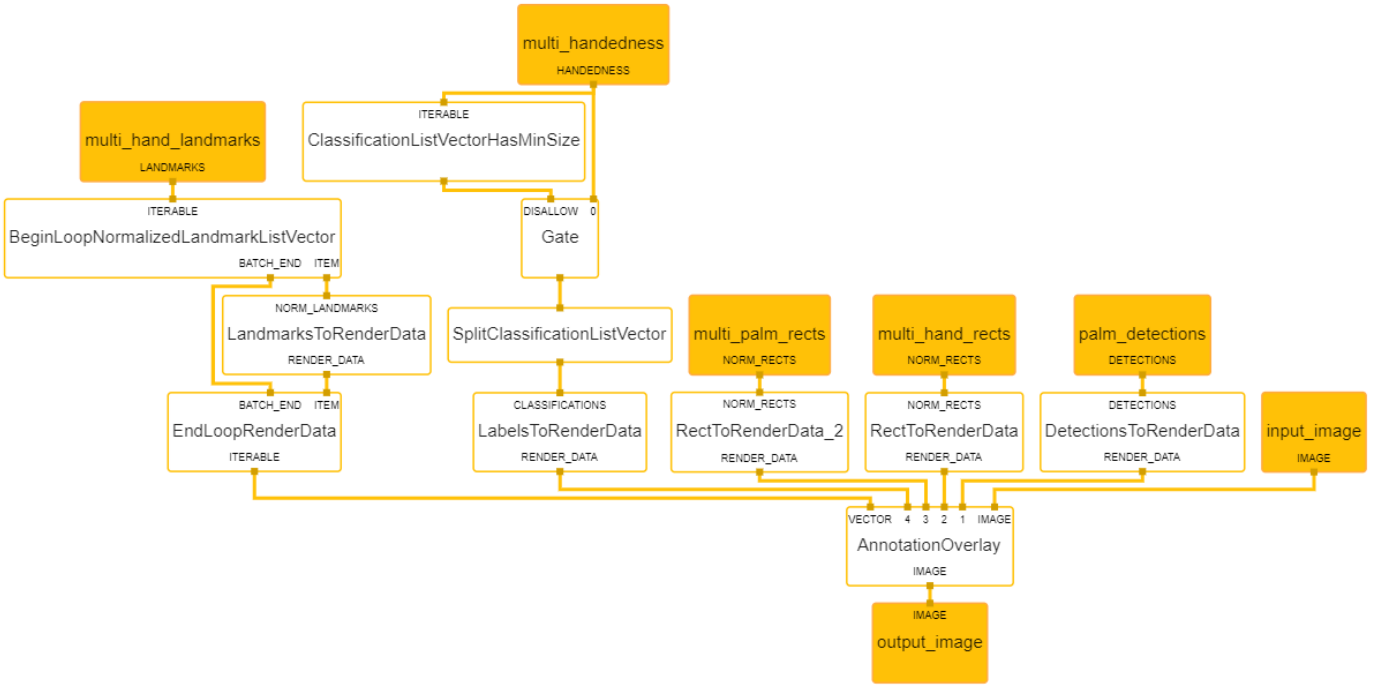


Рис. 2.2.1.3. – Підграф, що описує процес створення фінального вихідного зображення

На рисунку 2.2.1.4. відображено підграф, на якому детально можна проглянути всі салькулятори, які задіяні під час виявлення кисті руки.

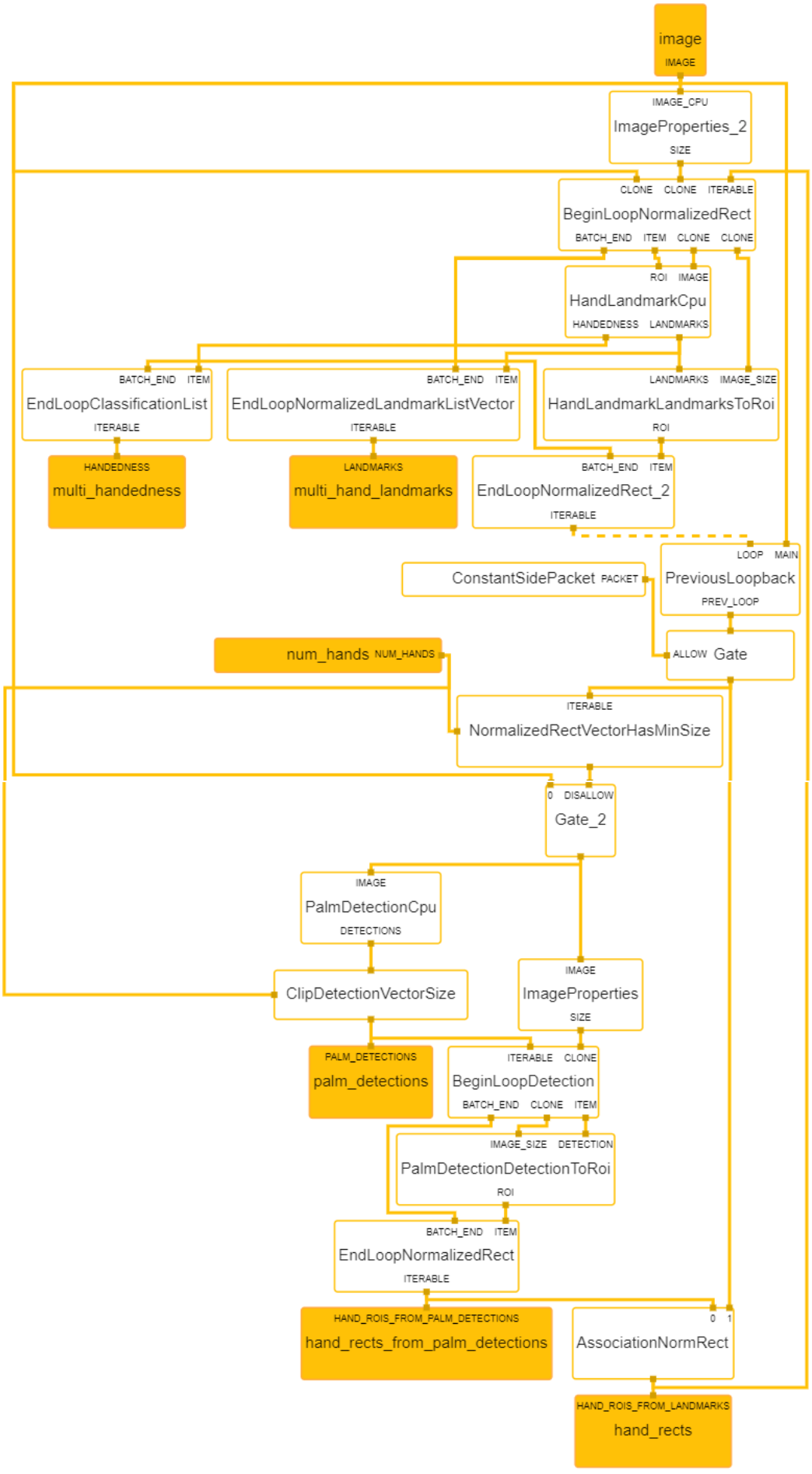


Рис.2.2.1.4. – Підграф виявлення кисті руки

MediaPipe Hands забезпечує рішення точного відстеження рук і пальців. Він використовує засоби OpenCV та алгоритми машинного навчання, щоб надати 21 трьохмірний орієнтир руки з усього одного кадру і позначає область, у якій виявлена долонь. Даний метод забезпечує продуктивність в реальному часі на малопотужних присторях і навіть підтримується можливість проводити виявлення кількох рук одночасно. Даний метод вирішує проблему виявлення кисті руки з використанням різних стратегій. Детектор кисті руки – це нейромережа, котра пройшла навчання на величезній кількості даних, а самі дані підготовленні засобами OpenCV. Таким чином, забезпечується отримання 21 ключової точки на кисті руки (рис. 2.2.1.5).

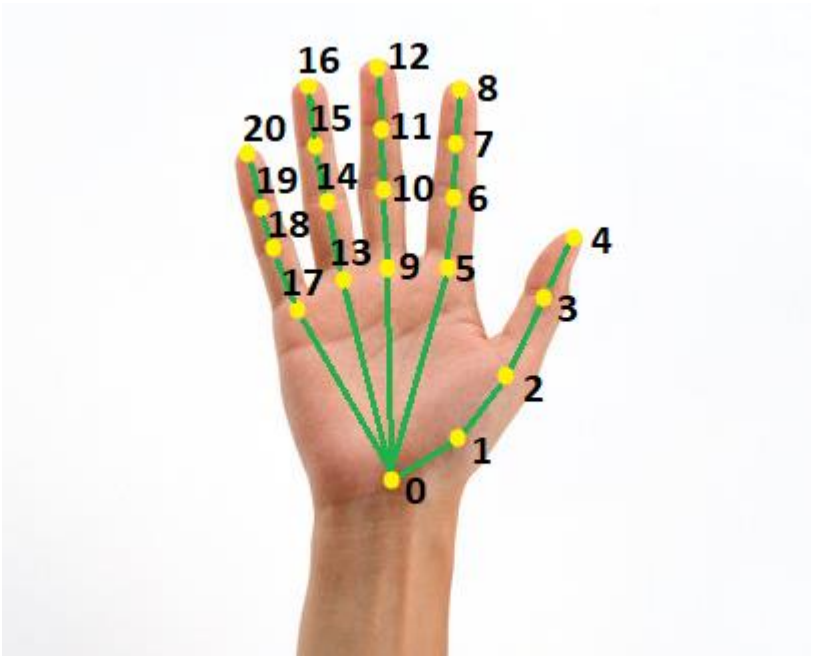


Рис. 2.2.1.5. – Ключові точки долоні, які виявляються MediaPipe Hands

Виявлення відбувається навіть частково видимих рук або у випадку, якщо одна рука перекриває іншу – завдяки регресії, тобто прямого передбачення координат.

Отже, алгоритм отримує у якості вхідних даних кадр відеопотоку, де ймовірно зображена кисть руки, і у разі, якщо кисть присутня – виявляє її і надає масив з 21 точки, кожна точка якої містить три координати, які описують положення точки у трьохмірному просторі.

2.1.1. Бібліотека OpenCV

2.2.2.1. Загальні відомомсті

OpenCV – це бібліотека, яка забезпечує можливість використання алгоритмів комп'ютерного зору і деяких інших алгоритмів по обробці зображень. Дана бібліотека написана на мові C/C++ що надає чудову швидкодійність програмного коду. Вона надає можливість здійснювати інтерпретації зображень, калібровки камери по еталону, ліквідацію шумів на зображені, визначення форми різноманітних об'єктів, здійснювати розділення за кольорами і т.п..

При проектуванні об'єкта дипломної роботи, дана бібліотека використовувалася для вирішення низки задач, також вона застосовується у роботі раніше згаданої бібліотеки MediaPipe. Надалі буде описано вибірку алгоритмів, які були застосовані у проектуванні модуля розпізнавання жестів.

2.2.2.2. Розмивання Гауса

Розмивання Гауса – це алгоритм, який здійснює розмиття зображення. Сам алгоритм був названий у честь німецького математика Карла Фрідріха Гауса.

Розмиття є інструментом обробки зображення. Основною метою застосування є зменшення деталізації зображення, що використовується для багатьох алгоритмів обробки зображень, завдяки зменшенню кількості шумів на зображені. Розмиття здійснюється завдяки застосування функції Гауса:

$$g(x) = ae^{-\frac{(x-b)^2}{2c^2}},$$

де a,b,c – довільні дійсні числа.

На рисунку 2.2.2.2.1. можна побачити один із варіантів роботи даної функції:

Рис. 2.2.2.2.1. – Розмиття за Гаусом

2.2.2.3. Кольорова модель HSV

Кольорова модель HSV (вимовляється як «ейчесві») – математична модель представлення кольорів зображення, у якій застосовується певна кількість кортежів (упорядкований набір фіксованої довжини) чисел. Модель представляється через три значення: Hue (тон), Saturation (насиченість), Value (яскравість). На рисунку 2.2.2.3.1. демонструється просторове розміщення даних координат.

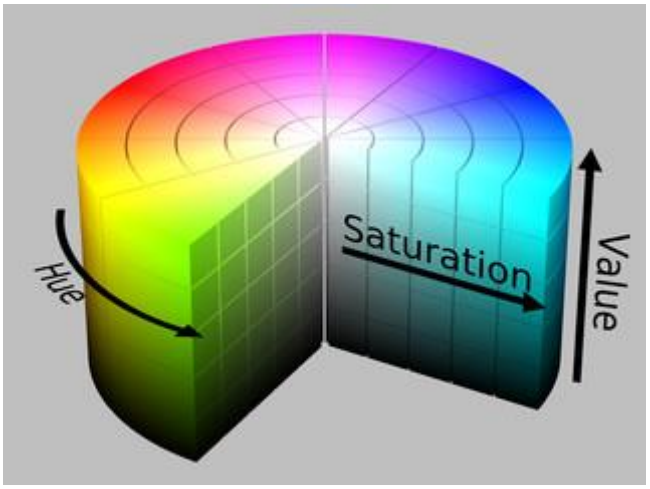


Рис. 2.2.2.3.1. – Просторове розміщення Hue, Saturation, Value.

Зазвичай приймають значення у діапазоні від 0 до 100 (окрім тому, стандартний діапазон якого від 0 до 360), проте, за необхідності, діапазон може бути масштабований до необхідних значень. У бібліотеці OpenCV, діапазони для даних змінних приведені до наступних значень: Hue – від 0 до 179, Saturation – від 0 до 255, Value – від 0 до 255.

Дана кольорова модель використовується для попередньої обробки зображень, зокрема для алгоритмів порогових операцій (алгоритми, які дозволяють виділити один певний кольоровий спектр із зображення). На рисунку 2.2.2.2.3.2. зображенно порівняння зображень, які використовують дві кольорові моделі – RGB та HSV (зліва – RGB, справа - HSV)

Рис. 2.2.2.3.2. – Кольорові моделі – RGB та HSV

2.2.2.4. Порогові операції

Порогові операції – це методи сегментації заданих, через пороги, кольорів, тобто – виділення певного кольору на зображенні. Якщо розглянути рисунок просторового розташування HSV у попередньому розділі (рис. 2.2.2.3.1.), то можна побачити, як даний «конус» розділений на умовні сегменти. Даний сегмент описується парою координат Hue, Saturation, Value («мінімальний поріг» та «максимальний поріг»), які і описують подібні об’ємні сегменти. Даний сегмент і буде містити необхідний колір, який необхідно виділити. Після того, як сегмент задано, задана кольорова палітра на рисунку набуває білого забарвлення, а все інше – чорного.

Зображення стає бінарним (зображення, яке складається лише з двох кольорів – білого і чорного). На рисунку 2.2.2.4.1. продемонстровано приклад виділення жовтого кольору

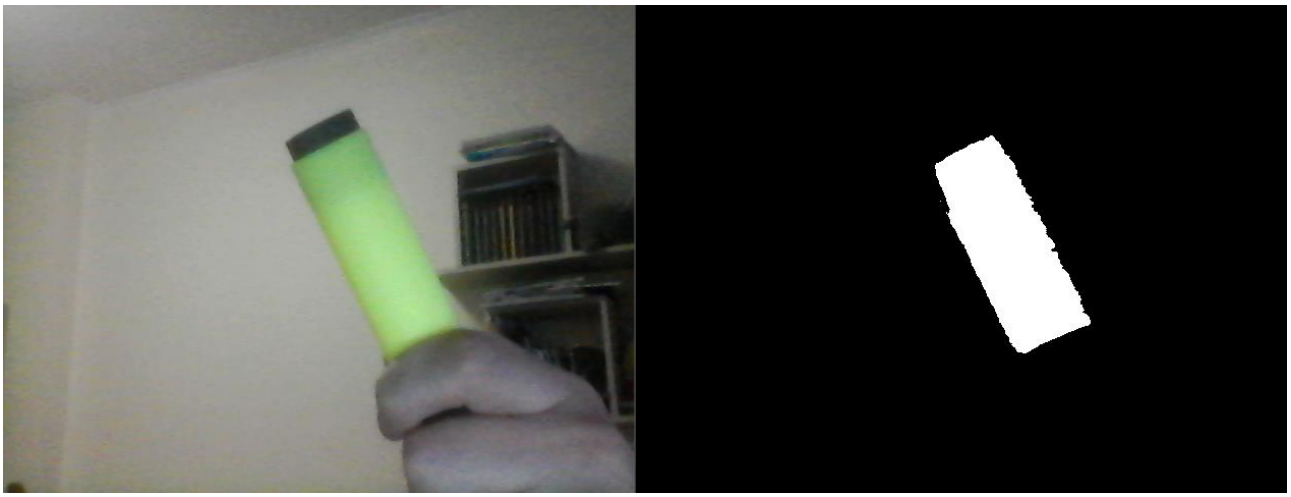


Рис. 2.2.2.4.1. – Виділення жовтого кольору

При здійсненні такої сегментації, значення «мінімального порогу» буде наступним: Hue = 23,Saturation = 89, Value = 112, значення «максимального порогу»: Hue = 54,Saturation = 232, Value = 255. Таким чином, за допомогою даних значень. виділяється сегмент, що забезпечує виокремлення потрібного кольору (кольорового спектру).

2.2.2.5. Алгоритм виявлення контурів

Алгоритм виявлення контурів, що використовується в OpenCV, є алгоритмом японського математика Сатосі Сузукі. Алгоритм відомий як Suzuki85 (англ. Сузукі85). Даний алгоритм забезпечує можливість ієрархічних відносин між границями контурів. Окрім здатності формувати ієрархію контурів, алгоритм здатний вказувати зовнішні та внутрішні границі. На рисунку 2.2.2.5.1. можна побачити різноманітні фігури, а цифрами позначенні контури

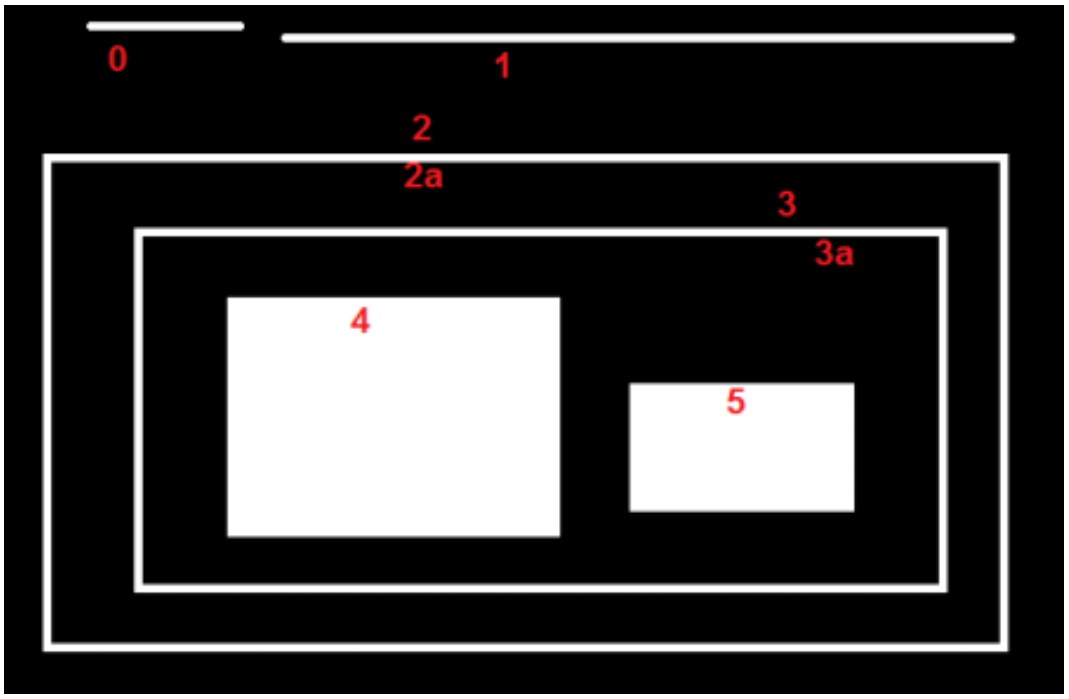


Рис. 2.2.2.5.1. – Набір фігур і позначень контурів

Літера «а» біля цифри означає внутрішній контурами певної фігури. Також, контури 0,1 та 2 називаються крайніми (зовнішні) (відносно великого прямокутника, який описується контурами 2 та 2а), дані контури знаходяться на одному рівні ієрархії, а всі інші - внутрішні. Завдяки такому виокремленню, є можливість додаткових операцій над контурами, наприклад ігнорування контурів за певними властивостями (наприклад, якщо контури зовнішні).

Загалом, алгоритм працює на основі аналізу пікселів і сусідніх пікселей, до пікселя який наразі розглядається. Обробка пікселей відбувається зверху-вниз і зліва на право. На бінарному зображенні всі чорні пікселі позначаються умовним нулем, білі – одиницею. У ході опрацювання пікселей, можливі два варіанта, які можуть підсказати, яка саме границя наразі розглядається – зовнішня чи внутрішня (рис. 2.2.2.5.2)

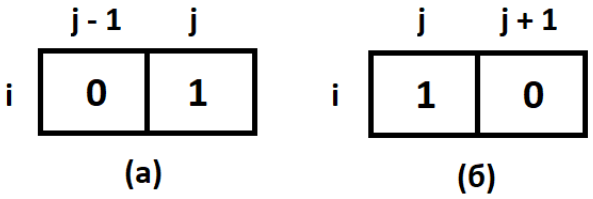


Рис. 2.2.2.5.2. – Визначення границі

Якщо перший піксель – нульовий, а наспупний є відмінним від нуля, то це повідомляє про виявлення контуру. Якщо поточний піксель позначений як ненульовий а наступний є нульовим, то розглядаємо це як закінчення контуру. Після того, як була досягнута дана ситуація, іде сканування сусідніх пікселів, спочатку за годинниковою стрілкою, потім – проти годинникової. Перевіривши ще низку умов, встановлюємо номери для кожного з контурів. Також необхідно мати змінну-лічильник, яка буде відслідковувати поточну кількість контурів. Таким чином, можна позначити пікселі, які відповідають одним і тим самим умовам, однаковими значеннями. Пікселі, що мають однакові значення, формують окремий контур.

2.2.2.6. Алгоритм Рамера-Дугласа-Пекера

При роботі алгоритму по виявленню контурів (наприклад Сузукі85) може бути ситуація, коли алгоритм виявить величезну кількість контурів, кожен з контурів буде доволі коротким, а самі контури утворять ламану, яка містить безліч точок (рис. 2.2.2.6.1.)

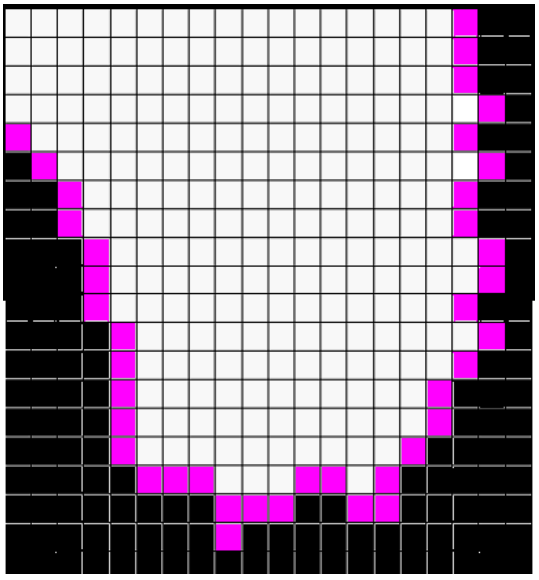


Рис. 2.2.2.6.1 – Контур фігури (фіолетовим)

Наявність величезної кількість точок може нашкодити продуктивності програми, внаслідок збільшення часу на опрацювання усіх точок контуру. Для того, щоб зменшити кількість точок, майже не видозмінюючи контур, застосовують алгоритм Рамера-Дугласа-Пекера.

Даний алгоритм здійснює апроксимацію ламаної до ламаної з меншою кількістю точок. Наприклад, якщо застосувати даний алгоритм на контурі з рисунку 2.2.2.6.2, то отримаємо більш гладкішу ламану, зображену на рисунку 2.2.2.6.2.

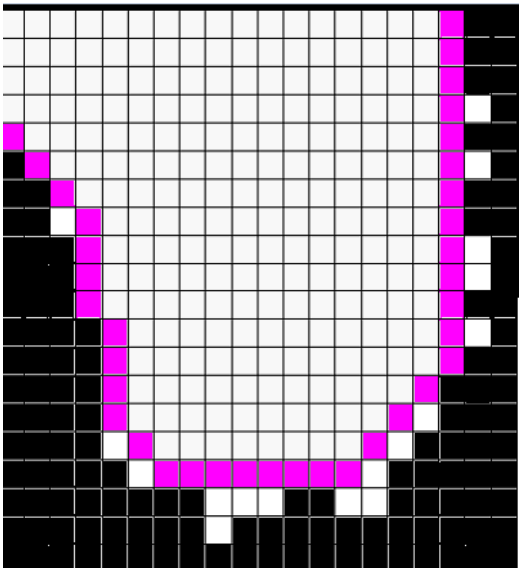


Рис. 2.2.2.6.2. – Апроксимований контур

Рамера-Дугласа-Пекера видно, що даний контур є ламаною з меншою кількістю точок переломів, що спрощує подальші оперції над контуром. Сам алгоритм доволі простий. Наша ламана, яку потрібно апроксимувати, розглядається як упорядкована вибірка з точок переломів. Також, задається довільне дійсне число $\varepsilon > 0$. Алгоритм рекурсивно проводить поділ ліній. Чим більше епсілон, тим більш апроксимованим буде контур. Вибирається дві точки (зазвичай початкова і кінцева точки ламаної) і між ними проводиться лінія. Відбувається пошук точки ламаної, яка є найбільш віддалена від даної лінії. Надалі поточні точки сполучаються із цією точкою і дана точка позначається як така, яка буде збережена . Проводиться перевірка відстані між іншими точками, які ще не були задіяними, і проведеними прямими. Якщо відстань між точною і прямою менше за ε, то така точка відкидається. Надалі проводимо нову ітерацію до тих пір, поки будуть оброблені всі точки. Таким чином, отримуємо апроксимовану ламану. На рисунку 2.2.2.6.3. показано приклад апроксимації ламаної.

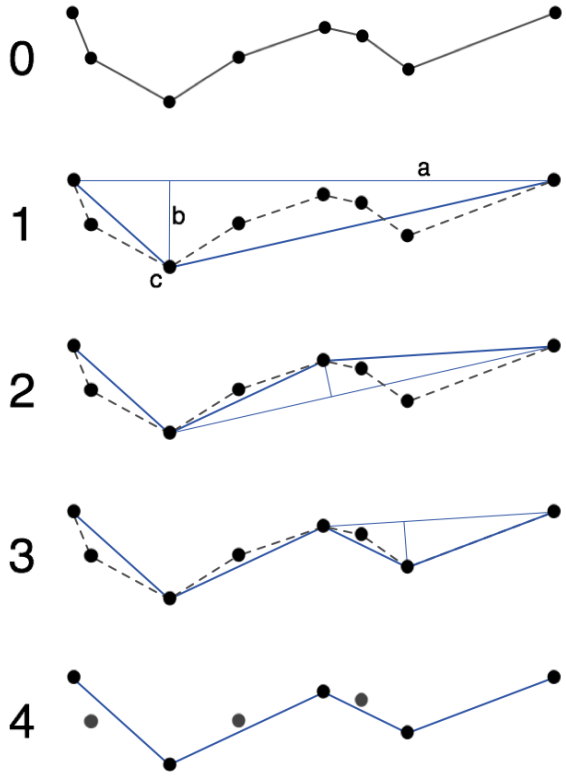


Рис. 2.2.2.6.3. – Приклад апроксимація ламаної

2.2.2.7. Поєднання роботи алгоритмів

Тепер, оглянувши алгоритми, можна навести приклад їхнього поєднання задля досягнення певних результатів. Наприклад – виявити об’єкт із заданим кольором. На рисунку 2.2.2.7.1. проілюстровано застосування алгоритмів і їх результат.

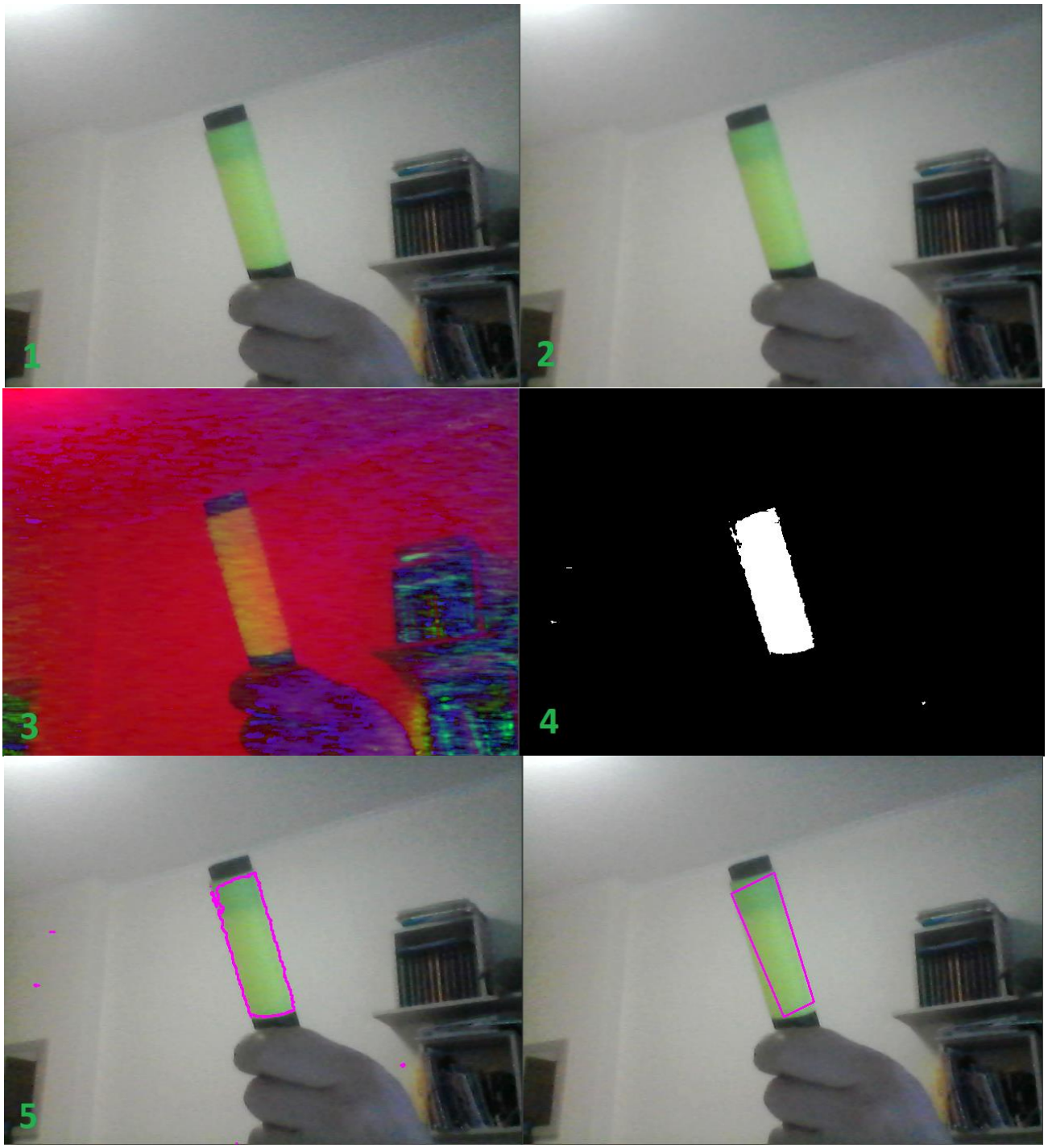


Рис. 2.2.2.7.1. – Процес застосування алгоритмів

Маємо наступну послідовність дій:

1) Отримуємо зображення з камери у кольоровій моделі RGB (у OpenCV використовується BGR)
2) Застосовуємо Гаусівське розмиття, задля зменшення кількості шумів
3) Конвертуємо RGB в HSV-модель
4) З HSV-моделі отримуємо необхідний колір, застосувавши порогові операції

5) З бінарного зображення отримуємо контури за допомогою аглоритма виявлення контурів (Сузукі 85)
6) Застосувавши алгоритм Рамера-Дугласа-Пекера, апроксимуємо контур

Отже, застосувавши алгоритми, отримали замкнутий колір, який позначає на зображенню місцерозташування шуканого об’єкта.

2.2.3. Система збірки Bazel

Система збірки Bazel – це програмне забезпечення з відкритим кодом, що забезпечує можливість збірки та тестування програм. Логотип продукту зображений на рисунку 2.2.3.1..

Рис. 2.2.3.1. – Логотип Bazel

Розроблений компанією Google, Bazel є подібний до Make, Maven, або Gradle за своїм функціоналом. Має можливість створювати ПЗ для різних платформ. Підтримує наступні мови програмування: C, C++, Java , Go, Python , Objective-C и Bourne shell-скрипти. Під час розроблення Bazel, увага була сфокусована на досягнення прискорення збірки, зокрема застосовуючи розпаралелювання. Також є можливість через даний інструмент підключати код, бібліотеки і інші файли, які знаходяться віддалено, потрібні для компіляції. Написання коду для налаштувань компіляції і збірки здійснюється на мові Starlark. Основою Bazel є робоча область (WORKSPACE), конфігураційні файли збірки (BUILD) та цілі (targets).

Структура BAZEL передбачає, що програмний код розбитий на «модулі» - окремі папки, які містять код, який виконує конкретну задачу. Дані папки об’єднуються через конфігураційні файли BUILD у яких прописані залежності через цілі. Таким чином, щоб скомпілювати програму, необхідно лише побудувати головний BUILD-файл, а через залежності, будуть побудовані і всі інші частини

проекту.

WORKSPACE – файл, який має бути в кореневій папці проекту. В даному файлі можна задати глобальні налаштування, наприклад вказати шлях до кореневих папок із бібліотеками.

BUILD – кожна окрема папка повинна мати свій власний BUILD-файл. Є головною одиницею у BAZEL. Тут прописуються всі налаштування конфігурацій, підключення файлів з програмним кодом, бібліотек і т.п. і вказуються інші конфігураційні файли BUILD.

Target – цілі, або правила, які підлягають до виконання. Через них вказуються дії, які необхідно виконати у BUILD-файлах.

Завдяки своїй структурі, Bazel забезпечує зручну компіляцію і аналіз конфігураційних файлів для компіляції.

2.2.4. Бібліотеки WiringPi та softPwm

Дані бібліотеки застосовуються для забезпечення зручної роботи з GPIO на Raspberry Pi. GPIO – це універсальний інтерфейс вводу-виводу, що надає можливість обмінюватися інформацією із різноманітною апаратурою, завдяки змінні напрузі на пінах. (рис. 2.2.4.1)

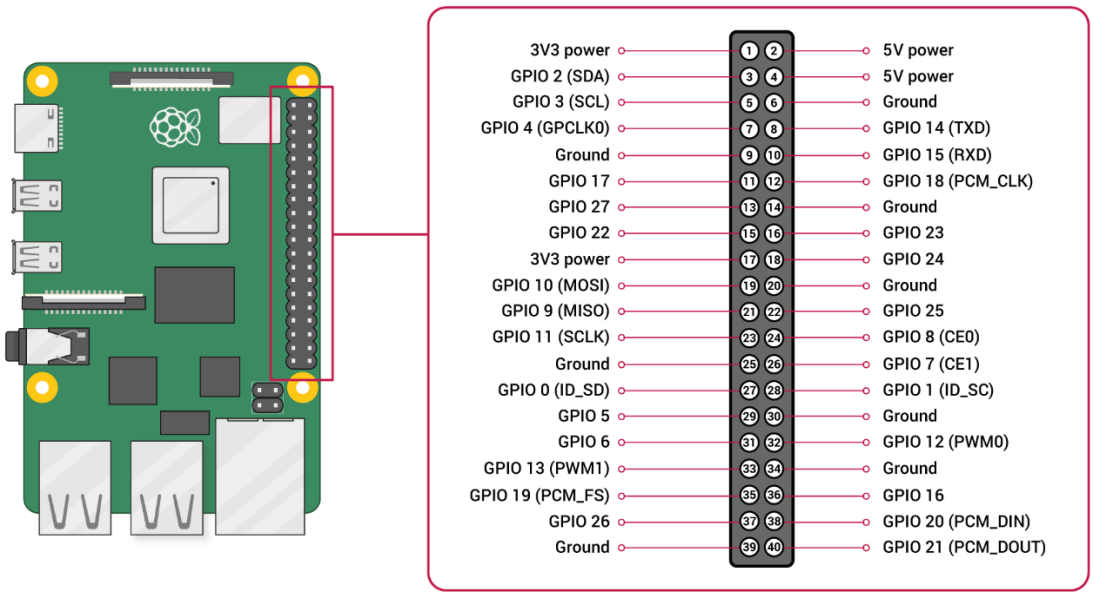


Рис. 2.2.4.1 – GPIO Raspberry Pi

Через те, що безпосередня робота з пінами Raspberry Pi є доволі важкою, через необхідність вдаватися до низькорівневого програмування, функції з бібліотек WiringPi та softPwm надають легкий спосіб використання цих пінів.

WiringPi - це бібліотека керування за допомогою GPIO на основі PIN-коду, написана на мові C для пристроїв на чіпах BCM2835, BCM2836 і BCM2837, які застосовані в усіх Raspberry Pi. Дана бібліотека розроблялася виключно для керування загальним інтерфейсом вводу-виводу на Raspberry Pi. Функції у бібліотеці наближені до функцій, які використовуються для написання програмного коду для апаратно-програмного пристрою Arduino на мові C/C++.

softPwm – бібліотека, яка використовує бібліотеку WiringPi для додавання певних можливостей, зокрема для реалізації цифрової PWM-модуляції (ШІМ-модуляція). Широтно-імпульсна модуляція - процес керування шириною (тривалістю) високочастотних імпульсів за законом, який задає низькочастотний сигнал. Використовується для керування сервоприводів.

Принцип роботи сервопривода на основі ШІМ – подання імпульсів певною тривалістю. В залежності від тривалості імпульсу, буде задаватися поворот сервопривода. На рисунку 2.2.4.2 наглядно показано, як від тривалості залежить кут повороту сервопривода.

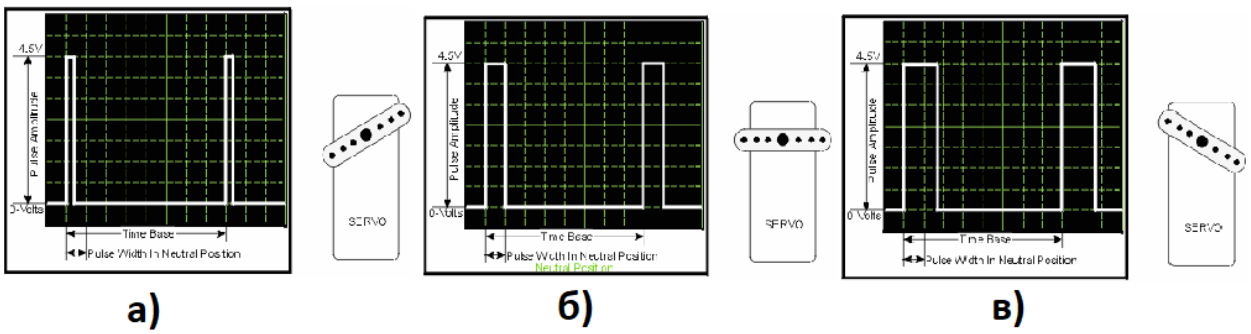


Рис. 2.2.4.2 – ШІМ сервоприводів

Як бачимо, залежно від тривалості сигналу, поворот буде здійснено уліво (варіант а) – тривалість менше тривалості «нейтральної позиції»), управо (варіант в) – тривалість більше тривалості «нейтральної позиції»), або прийняти положення без поворотів (варіант б) – тривалість рівна тривалості «нейтральної позиції»).

2.3. Опис нейронної мережі – алгоритму зворотного розповсюдження помилки.

Метод зворотного поширення помилки – це ітеративний градієнтний алгоритм, який забезпечує мінімізацію помилки багатошарової нейронної мережі. Як було сказано раніше, алгоритм використовує градієнтний спуск, що надає можливість знаходження локального мінімуму функції. Сама нейромережа – це функція багатьох змінних, а локальний мінімум – є наближенням до правильного значення функції при тих чи інших вхідних даних.

В чому полягає ідея методу? Під час роботи нейромережі, сигнали (дані) від вхідного шару нейромережі поширюються по прихованим шарам до вихідного шару, де формується відповідь нейромережі. Відповідь порівнюється із «правильною» відповіддю розраховується помилка нейромережі. Після цього отримана помилка розпочинає поширюватися у зворотному напрямку – від вихідного шару по прихованим до вхідного шару. У процесі поширення, коригуються значення ваг та персептронів.

Розглянемо загальну структуру нейромережі (рис. 2.3.1)

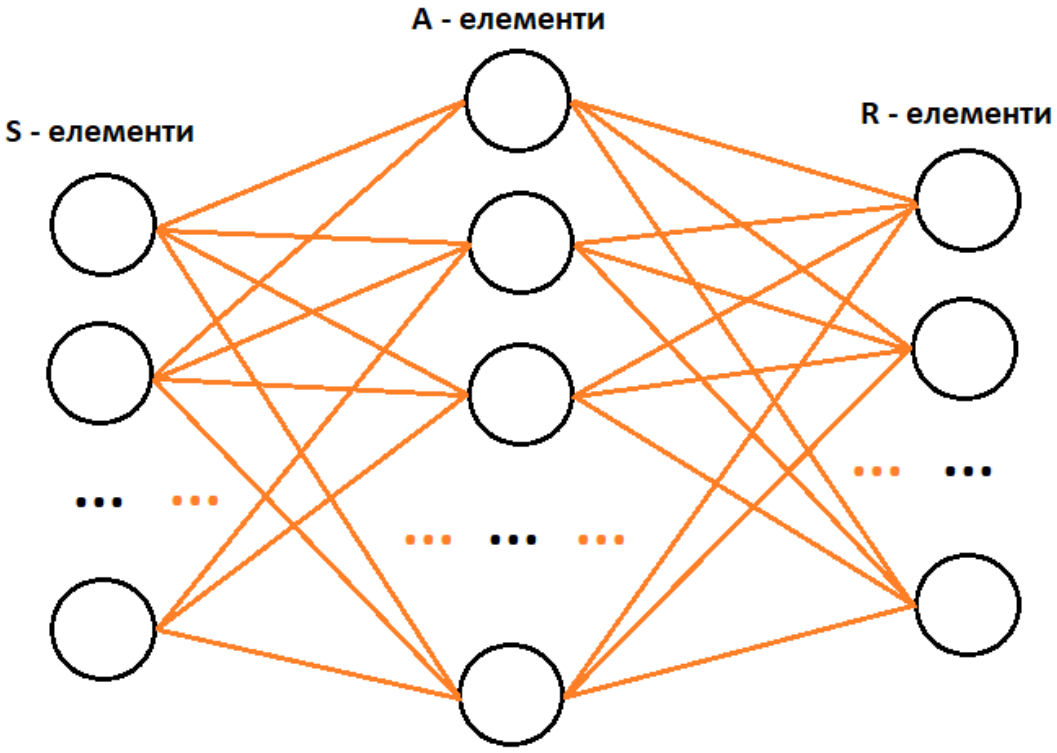


Рис. 2.3.1 – Загальна структура нейромережі

ПРОПУСК

Нейромережа може мати довільну кількість нейронів і прихованих шарів і завжди по одному вхідному і вихідному шарах.

Отже, введемо позначення:

$X = (x_1,\ldots,x_k)$ - вхідні вектори, $X_i \in \mathbb{R}^n$ (2.3.1)

$B = (b_1,\ldots b_k)$ - здвиги (2.3.2)

$A = (a_1,\ldots,a_k)$ - правильні вхідні вектора, $A_i \in \mathbb{R}^m$ (2.3.3)

(X, A) - навчальна вибірка (2.3.4)

W - вектор ваг нейронної мережі (2.3.5)

N(W, X) - функція, яка описує нейронну мережу (2.3.6)

Y = N(W, X) - відповідь нейронної мережі, $Y \in \mathbb{R}^m$ (2.3.7)

$D(Y, A) = \sum_{j=1}^{m}(y[j] - a[j])^2$ - функція помилки (2.3.8)

$D_i(Y) = D(Y, A_i)$ - функція помилки на і-му прикладі (2.3.9)

$E_i(W) = D_i(N(W, X_i))$ - помилка мережі на і-му прикладі (2.3.10)

$E(W) = \sum_{i=1}^{k} Ei(W)$ - помилка мережі на всій навчальній вибірці (2.3.11)

Завдання алгоритму можна звести до наступнийх цілей, а саме знайти:

Вектор W такий, що $E(W) \rightarrow \min$(навчання на всій вибірці)

Вектор W такий, що $E_i(W) \rightarrow \min$(навчання на всій вибірці)

ПРОПУСК

Для початку, необхідно зрозуміти, як саме працює персептрон – робоча одиниця в нейронній мережі. На рисунку 2.3.2 представлена загальна схема персептрона.

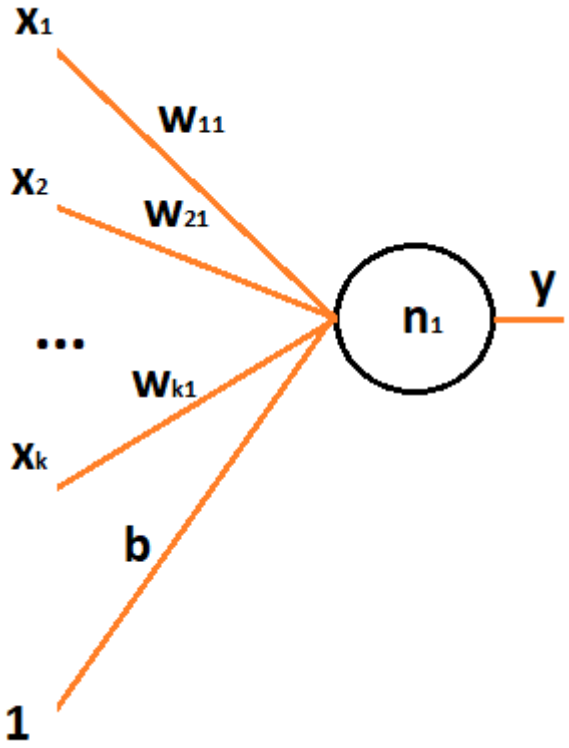


Рис. 2.3.2 – Схема перцептрону

Перцептрон – це реалізація певної математичної функції (2.3.13):

$$y = f\left(\sum_{i=1}^{k} w_{ij}x_i + b\right), \tag{2.3.13}$$

де f – функція активація;

w_{ij} – ваги;

x_i – вхідні значення;

b – здвиг;

y – результат обчислень пецсептрону.

До перцептрону підключені ваги w_{ij} (у даному випадку розглядається один перцептрон, і тому кількість ваг – $w_{11\ldots}w_{k1}$, де перша цифра позначає номер вхідних даних (x_i, $i \in [1\ldots k]$) а друга цифра позначає номер перцептрону, а через те що перцептрон один, то $j = 1$), які представляють собою дійсне число у межах (0…1).

$x_i \ldots x_k$ – набір вхідних значень, які передається перцептрону. Це може бути як звичайні дійсні числа, так і результат роботи інших перцептронів.

f – функція активації (або порогова). Дана функція надає результат у діапазоні [0…1] або [-1…1]. Вона симулює роботу біологічного нейрону – нейрон можу бути незбудженим (значення перцептрону близьке до 0 або -1), або бути збудженим (значення перцептрону близьке до 1). Розенблат використовував у своїй реалізації персептрону функцію сегмоїди (2.3.14):

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.3.14}$$

Графік функції сигмоїди зображений на рисунку 2.3.3.

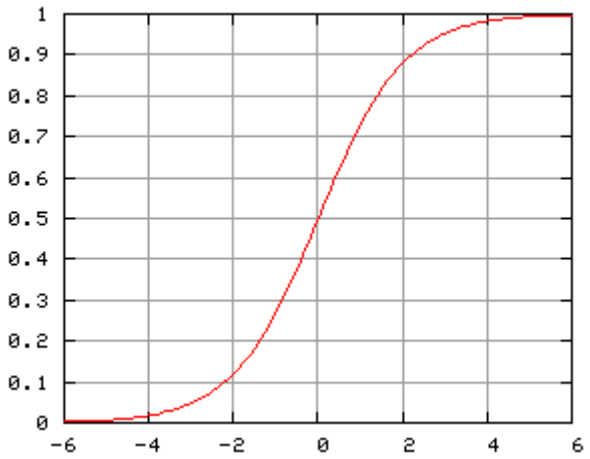


Рис. 2.3.3 – Графік функції сигмоїди

Функція сигмоїди надає результат у діапазоні [1…0].

Також, для реалізації роботи персептрону, можна використовувати інші функції активації, зокрема деякі з них наведен на рисунку 2.3.4.

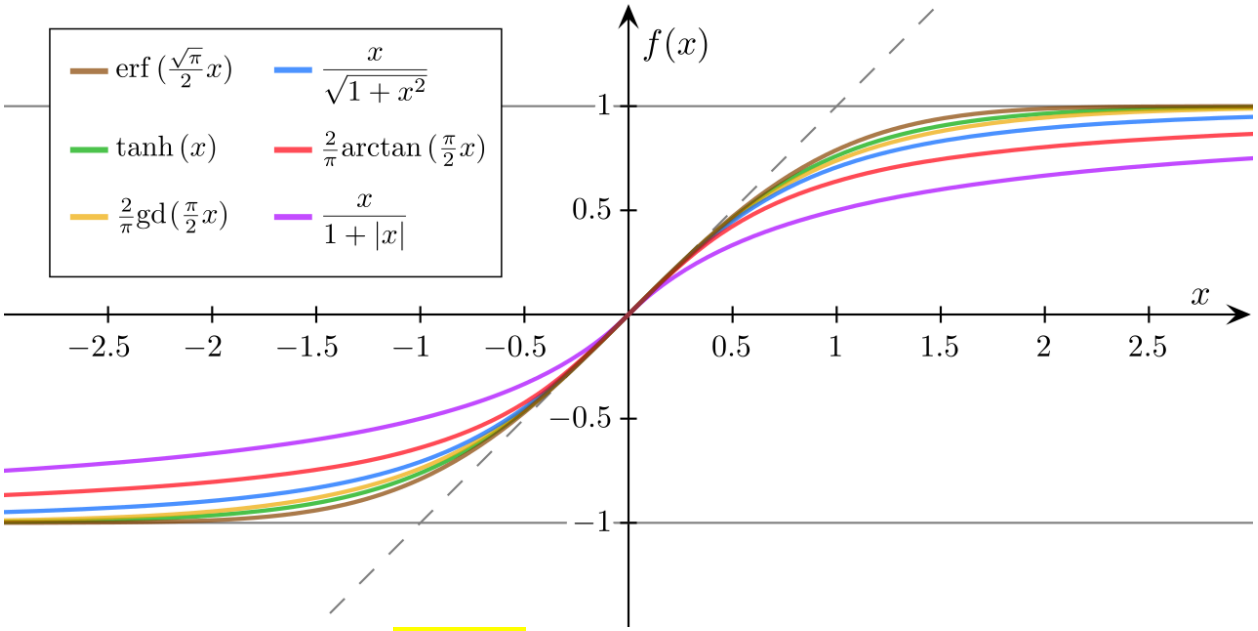
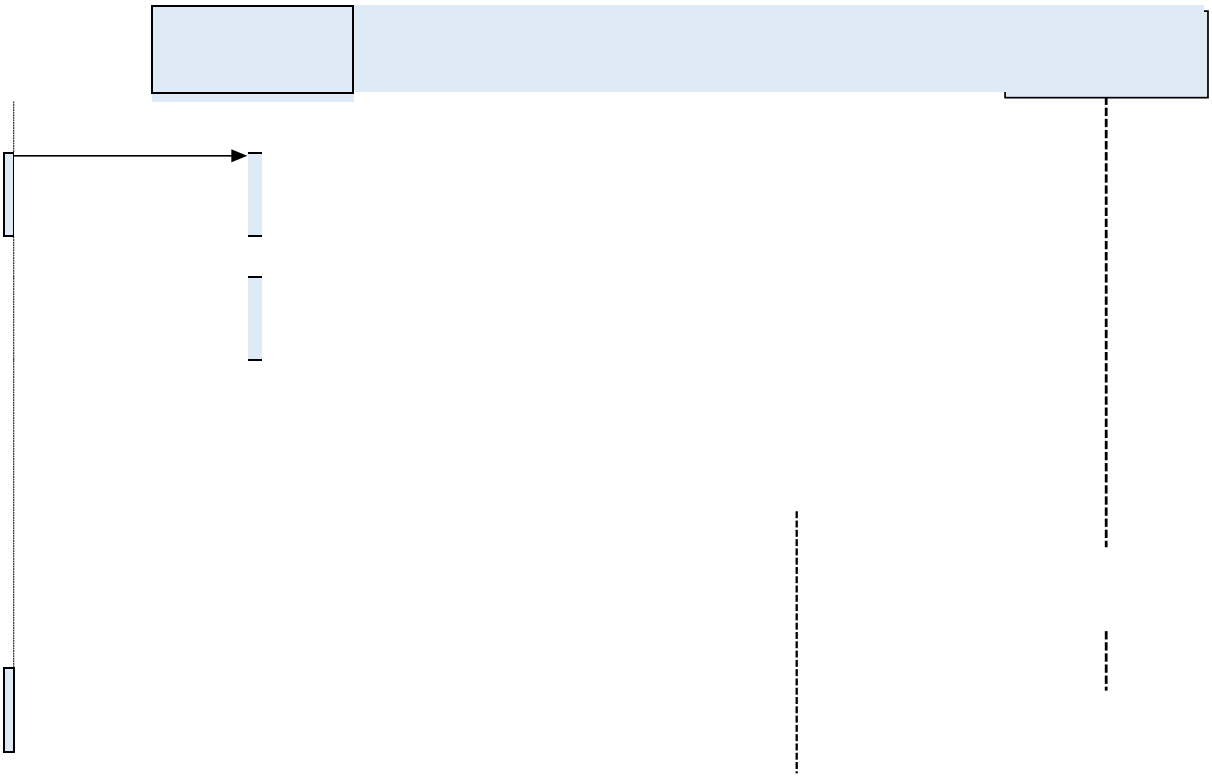


Рис. 2.3.4 – Функції активації

Здвиг b – це спеціальна вага, значення якої множиться на 1 та додається до суми добутків значень вхідних перцептронів на відповідні ваги. Вона грає роль певного «стимулятора», щоб у разі обчислень перцептрону не отримувати значення, рівне 0.

Ознайомившись з роботою перцептрону, перейдемо до процесу навчання.

Як було раніше зазначено, робота нейромережі базується на надані їй вхідних даних, які проходять через приховані шари нейромережі і прямують до вихідного шару, де сформовується остаточна відповідь. Дана відповідь залежить від значень ваг перцептронів. У разі, якщо нейромережа надає невірний результат, необхідно відкоригувати ці значення. Безпосередньо вручну видозмінити їх майже неможливо – по-перше, невідомо, наскільки сильно потрібно змінити ту чи іншу вагу, а по-друге, кількість ваг навіть у невеличкій нейромережі може бути тисячі. На допомогу приходить алгоритм зворотного поширення помилки. Алгоритм надає можливість встановити числову різницю між відповіддю, що надала нейромережа і еталонної відповідді. Після цього, дану помилку нейромережі алгоритм «протягує» через усю мережу – від вихідного шару, через приховані шари і до вхідного шару, змінюючи ваги і значення персептронів.



Розглянемо рисунок 2.3.5.

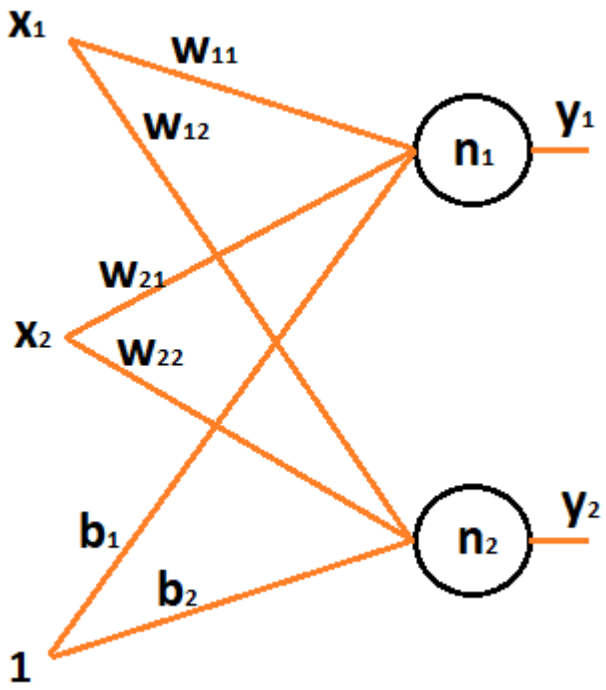


Рис. 2.3.5. – Приклад нейромережі

Дана нейромережа представлена двома перцептронами n_1 та n_2, двома входами x_1, x_2, вагами w_{11}, w_{12}, w_{21}, w_{22} та здвигами b_1, b_2

Проведемо умовні обчислення:

Значення першого перцептрону: $y_1 = sigm(w_{11}x_1 + w_{21}x_2 + b_1)$ (2.3.15)

Значення другого перцептрону: $y_2 = sigm(w_{12}x_1 + w_{22}x_2 + b_2)$ (2.3.16)

Також, нехай $S_1 = w_{11}x_1 + w_{21}x_2 + b_1$, (2.3.17)

$$S_2 = w_{12}x_1 + w_{22}x_2 + b_2 \quad (2.3.18)$$

Ми отримали відповідь нейромережі і тепер необхідно звіритися з еталонною відповіддю з навчальної вибірки (2.3.4). Для цього обчислимо функцію помилки двох змінних (2.3.19):

$$D_k(y_1,y_2) = (y_1 - a_1)^2 + (y_2 - a_2)^2 \quad (2.3.19)$$

Таким чинном, помилка нейромережі на певній навчальній вибірці (2.3.10) буде виглядати наступним чином:

$$E_k(W) = D_k(y_1(w_{11}, w_{21}, b_1), y_2(w_{12}, w_{22}, b_2)) \quad (2.3.20)$$

Алгоритм ЗПП надає можливість корегувати ваги перцептронів через похідку E_k від необхідної ваги перцептрона:

$$\frac{\partial E_k}{\partial w_{ji}} \tag{2.3.21}$$

Для цього необідно знайти часткові похідні. Розглянемо цей процес на прикладі нейромережі з попереднього рисунку 2.3.5.

Розглянемо знаходження похідної E_k від w_{21}. Ми маємо помилку нейромережі (2.3.20), яка основується на функції помилки двох змінних (2.3.19). Для того, щоб знайти похідну E_k *від* $w_{21,}$ необхідно знайти часткові похідні E_k від y_1 та E_k від y_2, а у свою чергу, треба буде знайти часткові похідні y_1 від w_{21} та w_{21} та y_1 від w_{21}. Таким чином, розпишемо формулу (2.3.19) для цього окремого випадку:

$$\frac{\partial E_k}{\partial w_{21}} = \frac{\partial E_k}{\partial y_1}\frac{\partial y_1}{\partial w_{21}} + \frac{\partial E_k}{\partial y_2}\frac{\partial y_2}{\partial w_{21}} \tag{2.3.20}$$

Розпишемо часткові похідні:

$$\frac{\partial E_k}{\partial y_1} = 2(y1 - a1) \tag{2.3.21}$$

$$\frac{\partial E_k}{\partial y_2} = 2(y2 - a2) \tag{2.3.21}$$

$$\frac{\partial y_1}{\partial w_{21}} = sigm'(S_1)x_2 \tag{2.3.22}$$

$$\frac{\partial y_2}{\partial w_{21}} = 0 \tag{2.3.23}$$

Таким чином, сформуємо остаточний варіант:

$$\frac{\partial E_k}{\partial w_{21}} = 2(y1 - a1)sigm'(S_1)x_2 + 0 \tag{2.3.24}$$

Отже, формула (2.3.24) надає помилку вказаної ваги. Тепер необхідно модифікувати вагу:

$$w_{21} := w_{21} - \frac{\partial E_k}{\partial w_{21}} \tag{2.3.25}$$

Сформуємо загальні формули для знаходження похідної E_k *від* вказаної ваги *w:*

$$D_k(y_1, \ldots y_n) = (y_i - a_i)^2 + \ldots + (y_2 - an)^2 \tag{2.3.26}$$

$$\frac{\partial E_k}{\partial y_1} = 2(y_1 - a_1) \tag{2.3.27}$$

$$S_i = \sum_{j=0}^{m} x_j w_{ji} \tag{2.3.28}$$

$$y_i = f(S_i) \tag{2.3.29}$$

$$\frac{\partial y_i}{\partial w_{ji}} = f'(S_i)x_j \tag{2.3.30}$$

$$E_k(W) = D_k(y_1(w_{10}, \ldots, w_{mn}, b_1), \ldots, y_n(w_{1n}, \ldots, w_{mn}, b_n)) \tag{2.3.31}$$

$$\frac{\partial E_k}{\partial w_{ji}} = \sum_{l=1}^{n} \frac{\partial E_k}{\partial y_l} \frac{\partial y_l}{\partial w_{ji}} = 2(y_i - a_i) f'(S_i) x_j \tag{2.3.32}$$

Таким чином, формули (2.3.26 - 2.3.32) задають правила для видозміннення ваг персептронів. Змінна здвигів відбувається за аналогічною послідовністю.

Проте, нейромережа може складатися з багатьох прихованих шарів. Необхідно надати можливість передавати помилку в глибину нейромережі. При фомуванні попередніх правил корегування ваг, ми брали значення помилки, сформоване в результаті роботи перцептрона y_1, y_2.

Проте, якщо розглядати вхідні значення (2.3.1) то можна побачити, що у свою чергу це також є результатом роботи перцептрона і до цих перцептронів можна застосувати ті самі правила по корегуванню ваг (рис. 2.3.6).

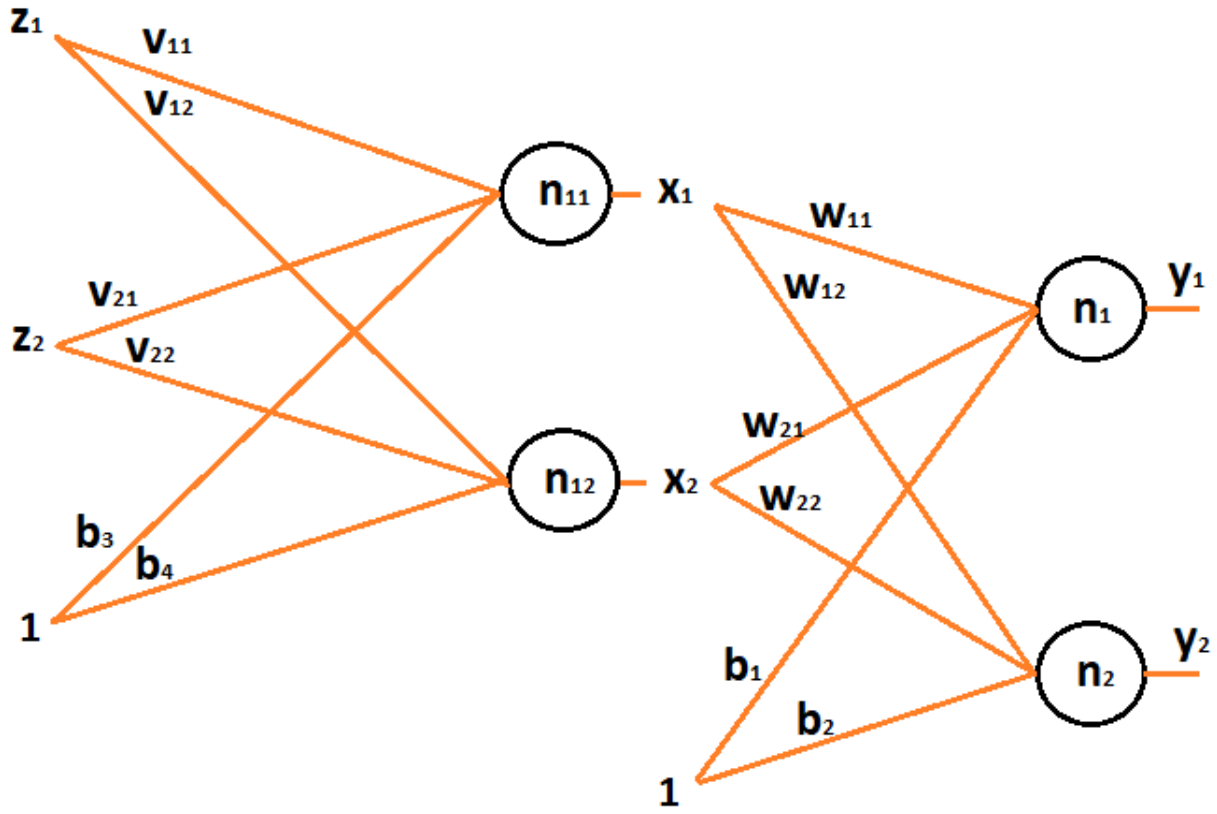


Рис. 2.3.6 – Нейромережа з прихованим шаром

Тепер, щоб відкорегувати ваги *v*, необхідно здійснити корегування значень перцептронів прихованого шару (n_{11}, n_{12}). Здійснити це можна, використавши формули (2.3.26 - 2.3.32), проте тепер необхідно знайти похідну E_k від вказаного значенян перцептрона *x*. Отримуємо формули для знаходження помилки значень перцептронів (2.3.33 - 2.3.39):

$$D_k(y_1,\dots y_n) = (y_i - a_i)^2 + \dots + (y_2 - a_n)^2 \quad (2.3.33)$$

$$\frac{\partial E_k}{\partial y_1} = 2(y_1 - a_1) \quad (2.3.34)$$

$$S_i = \sum_{j=0}^{m} x_j w_{ji} \quad (2.3.35)$$

$$y_i = f(S_i) \quad (2.3.36)$$

$$\frac{\partial y_i}{\partial x_j} = f'(S_i) w_{ji} \quad (2.3.37)$$

$$E_k(W) = D_k(y_1(w_{10},\dots,w_{mn}, b_1),\dots, y_n(w_{1n},\dots, w_{mn}, b_n)) \quad (2.3.38)$$

$$\frac{\partial E_k}{\partial x_j} = \sum_{l=1}^{n} \frac{\partial E_k}{\partial y_l} \frac{\partial y_l}{\partial w_{ji}} = 2(y_i - a_i) f'(S_i) w_{ji} \quad (2.3.39)$$

Таким чином, ми передаємо помилку на значення перцептронів, що знаходяться у прихованих шарах, і після цього ми мажмо змогу застосувати формули (2.3.26 - 2.3.32), які забезпечать корегування ваг перцептронів далі у нейромережі.

2.4. Бібліотека JSON

Бібліотека JSON (вимовляється як «джейсон») – це бібліотека, яка забезпечує текстовий формат обміну даними між різними ЕОМ. Окрім обміну, бібліотека надає можливості формувати файли для зберігання упорядкованої інформації і проводити читання таких файлів. Таким чином, зазвичай її використання вимагається саме при записі та читанню файлів із структоризованим вмістом.

Під час реалізації предмету дипломного дослідження, передбачається

використання бібліотеки для збереження даних, а саме: налаштувань нейромережі і навчальних даних.

2.5. Висновки до розділу

У другому розділі висвітлений перелік використаних технологій, які застосовувалися під час реалізації предмета дослідження дипломної роботи, наданий короткий опис, для кожної технології наведене коротке пояснення принципу її роботи. Окрім застосованих технологій і алгоритмів, розділ містить дані про обрані апаратні елементи, які будуть використані на етапі розробки модуля по керуванню жестами рук.

На завершення, підведемо загальний підсумок по кожному з пунктів Розділу 2, у якому буде коротко надано підсумування результатів, отриманих при написані:

- Опис апаратних елементів, які застосовувалися під час проектування модуля

  У даному підрозділі був наданий перелік апаратних елементів, які будуть використовуватися у процесі реалізації модуля по керуванню жнстами. Одноплатний комп'ютер Raspberry Pi, сервоприводи, модуль камери, альт-азимутальна установка та живлення є ключовими апаратними вузлами модуля, що надають можливість фізичної реалізації модуля. Для кожного елементу був наведений детальний опис і висвітлено його характеристики, також надано інформацію про призначення конкретного елементу.

- Технології, застосовані для обробки зображень, реалізації процеса виявлення жестів та керування альт-азимутальною установкою.

  Підрозділ містить інформацію, яка надає розуміння про весь процес обробки зображень, що в майбутньому використовується для виявлення об'єктік (жестів). Наданий детальний опис теоретичних відомостей по бібліотекам MediaPipe та OpenCV. Надано перелік алгоритмів і методів по обробці зображень, а саме: розмивання Гауса, кольорова модель HSV, поргові операції, алгоритм виявлення контурів, алгоритм Рамера-Дугласа-Пекера. Для кожного з алгоритмів наведений їх короткий опис і укінці наданий приклад їх застосування для виявлення об'єктів.

- Система збірки Bazel.

  Наведений опис системи збірки Bazel, яка використовуватиметься під час компіляції і збірки програмного забезпечення для модуля керування жестами. Наведений короткий опис переваг даної системи і принцип роботи.

- Бібліотеки WiringPi та softPwm.

Наданий опис бібліотек WiringPi та softPwm, які забезпечують можливість простої взаємодії із загальним інтерфейсом вводу/виводу GPIO. Коротко описано призначення та можливості даних бібліотек.

- Бібліотека JSON

  Описано призначення бібліотеки та надано відомості з експлуатації бібліотеки під час виконанян дипломної роботи.

- Опис нейронної мережі – алгоритму зворотного розповсюдження помилки.

  В останньому підрозділі висвітлені відомості про алгоритм щтучного інтелекту, який забезпечить розпізнавання логіки виявлених жестів, а саме алгоритм зворотного поширення помилки (ЗПП). Коротко описана суть методу, і наведене детальне пояснення роботи даного алгоритму, з відображенням усіх формул та обчислень, разом з ілюстративним супроводом. Також, розглянуто невеликі приклади у застосуванні даних формул.

РОЗДІЛ 3

ПРОГРАМНА ТА АПАРАТНА РЕАЛІЗАЦІЯ МОДУЛЯ РОЗПІЗНАВАНЯН ЖЕСТІВ РУК ДЛЯ КЕРУВАННЯ КВАДРОКОПТЕРОМ

3.1. Реалізація концепції модуля розпізнавання жестів

На початку робіт, то фізичному втіленню предмета дипломної роботи, а саме модуля розпізнавання жестів рук для керування квадрокоптером, необхідно було продумати загальну концепцію всієї системи. Модуль має відповідати наступним критеріям:

- Компактний розмір
- Автономність роботи

- Рухомий елемент, який забезпечуватиме достатній огляд для можливості розпізнання жесту
- Можливість кріплення модуля на коптер
- Інтерфейс, який забезпечить передачу даних від модуля до коптера

Враховуючи дані вимоги, і загальну схему систем розпізнавання жестів (Розділ 1, рис. 1.2.1) було обрано елементи для побудови (Розділі 2) і пропрацьована схема пристрою (рис. 3.1.1).

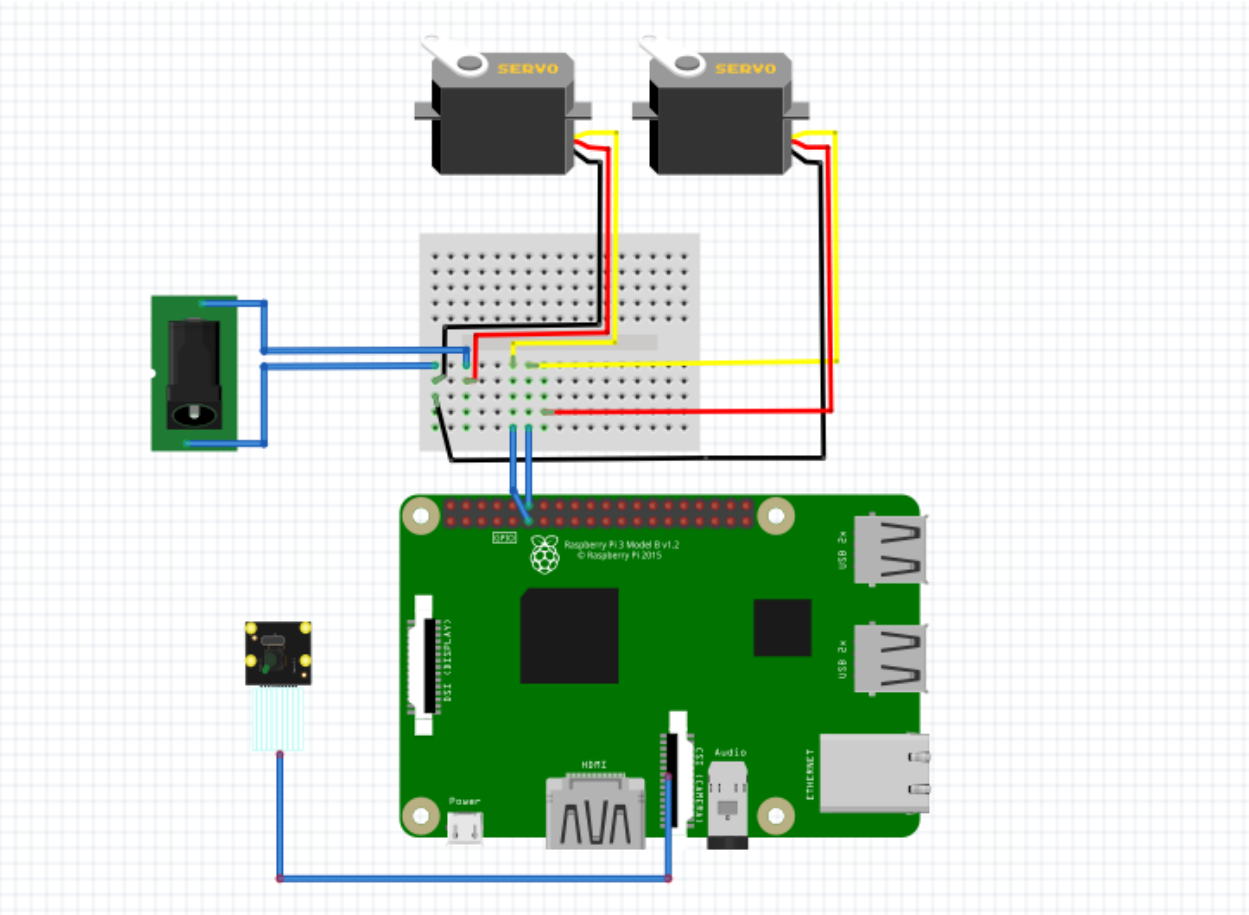


Рис. 3.1.1 – Схема підключення компонентів модуля розпізнавання жестів

Отримавщи усі необхідні компоненти, здійснено фізичну збірку модуля (рис. 3.1.2)

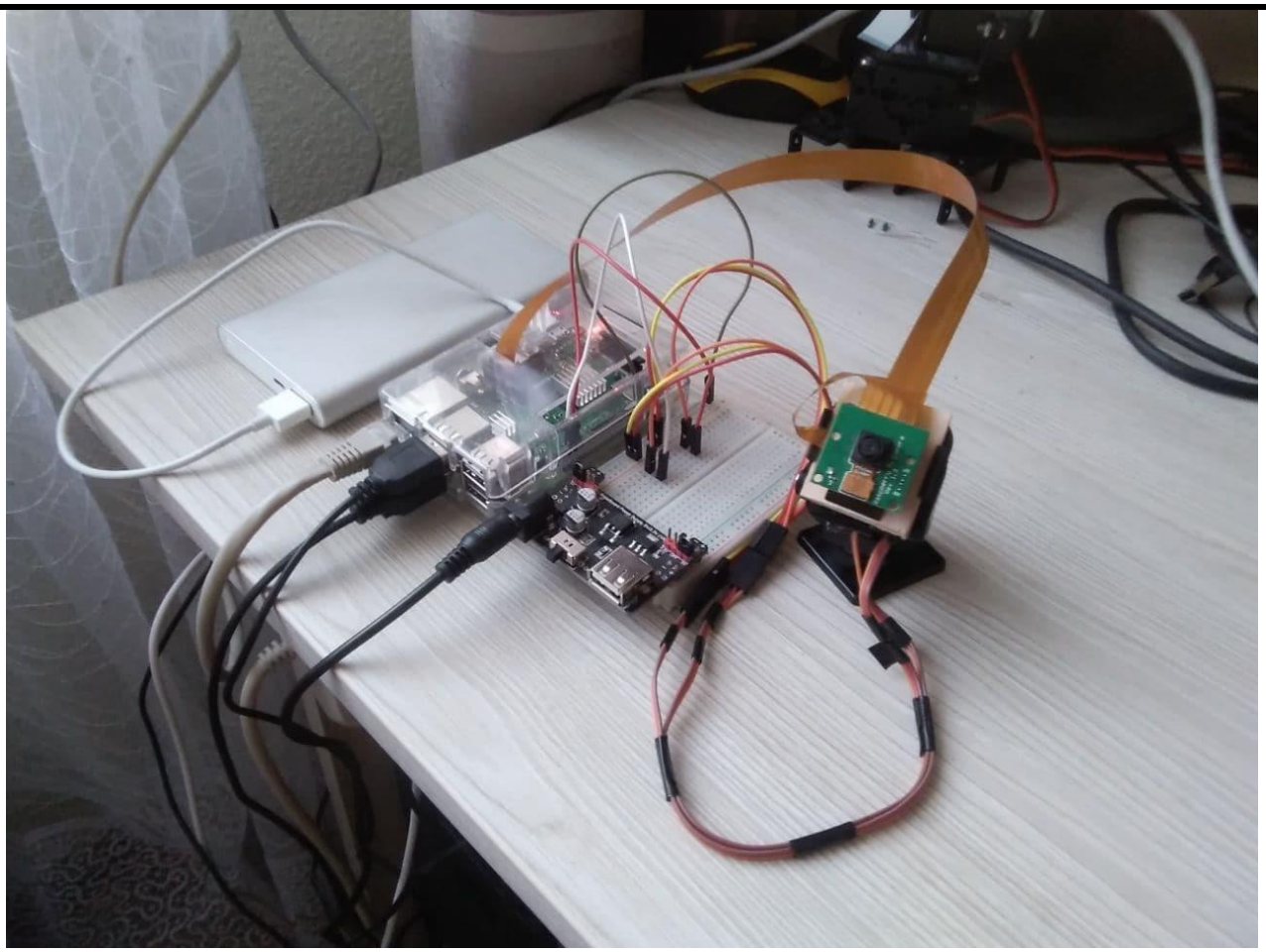


Рис. 3.1.2 – Фізична реалізація модуля

Після цього, було продумано інтегрування модуля у квадрокоптер і створення ескізу (рис. 3.1.3)

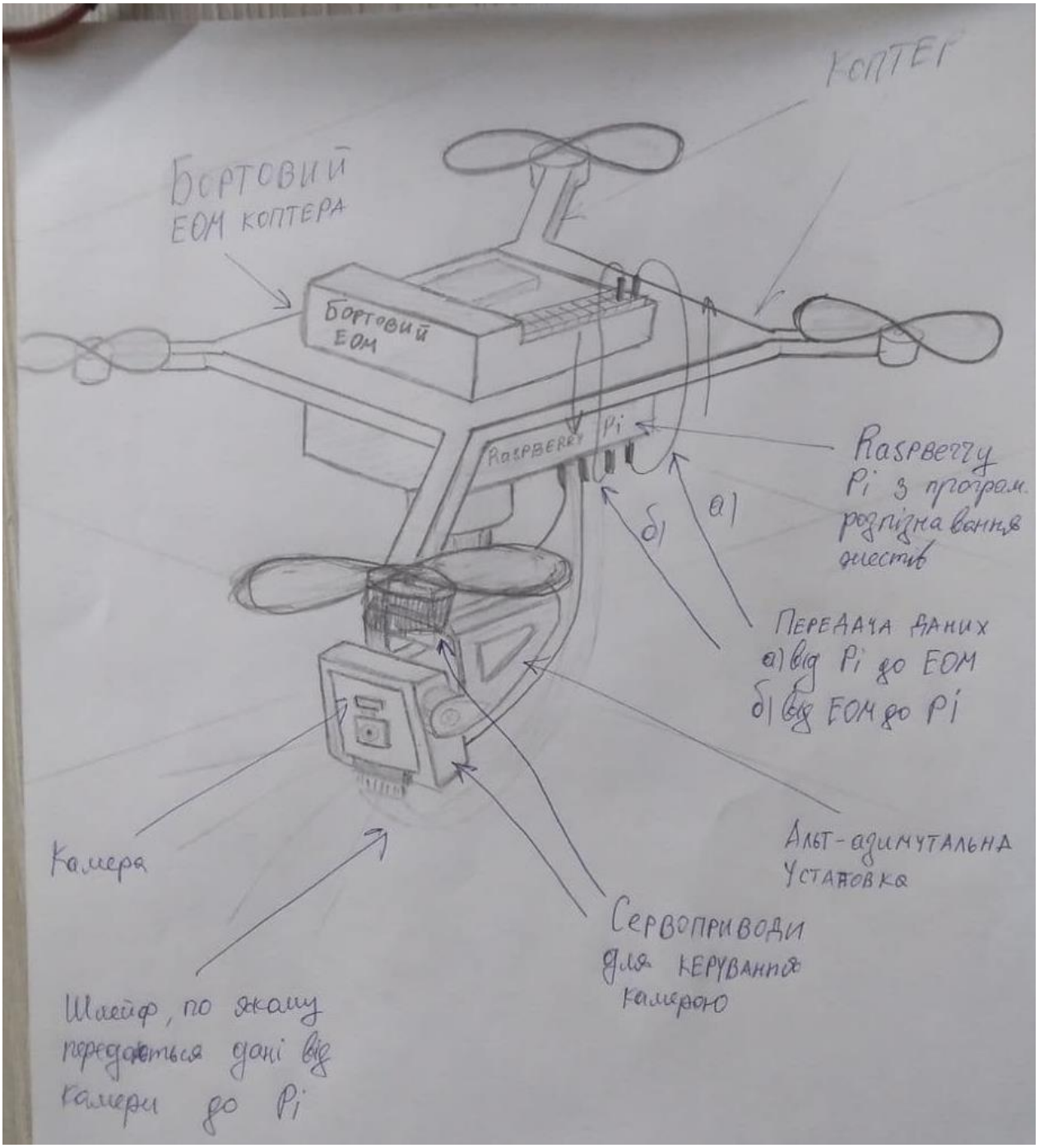


Рис. 3.1.3 – Ескіз поєднання модуля і квадрокоптера

Тепер, маючи загальне представлення про систему вцілому, за допомогою програмних засобів у AutoCAD, була створена 3D модель квадрокоптера у поєднані з модулем розпізнавання жестів (рис. 3.1.4 – 3.1.5).

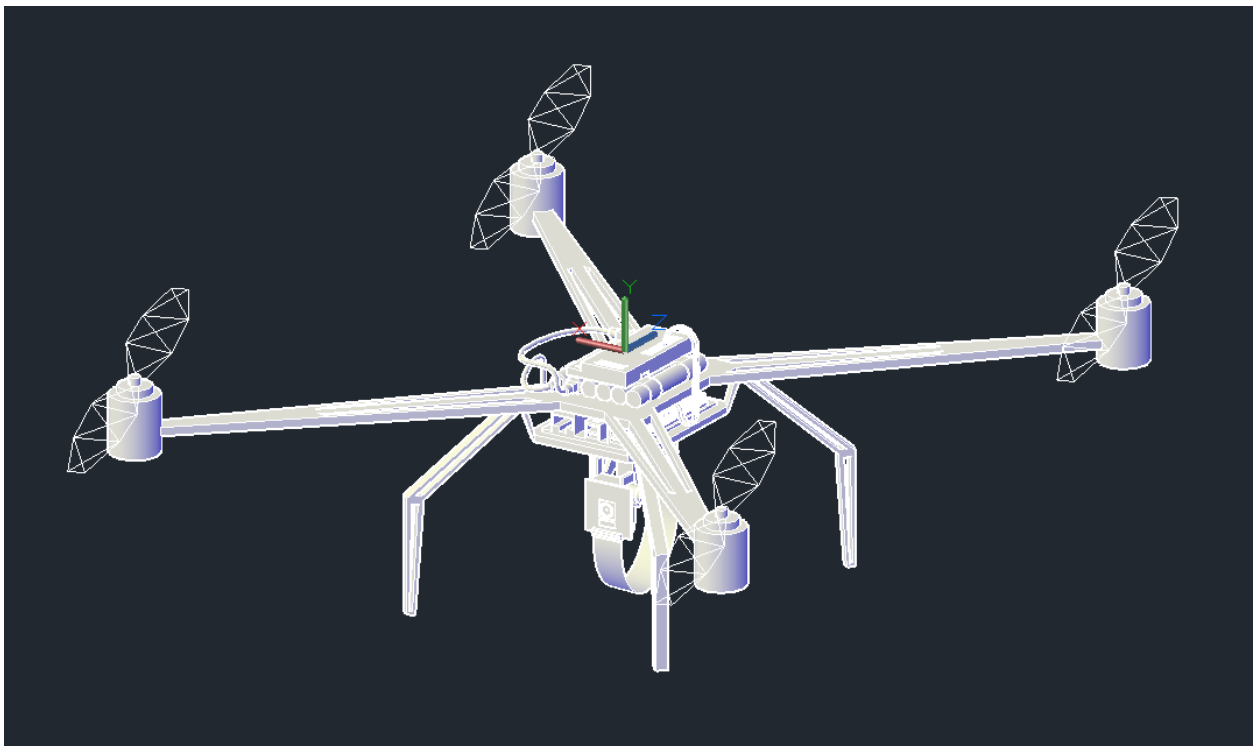


Рис. 3.1.4 – Модель у AutoCAD (Концептуальний)

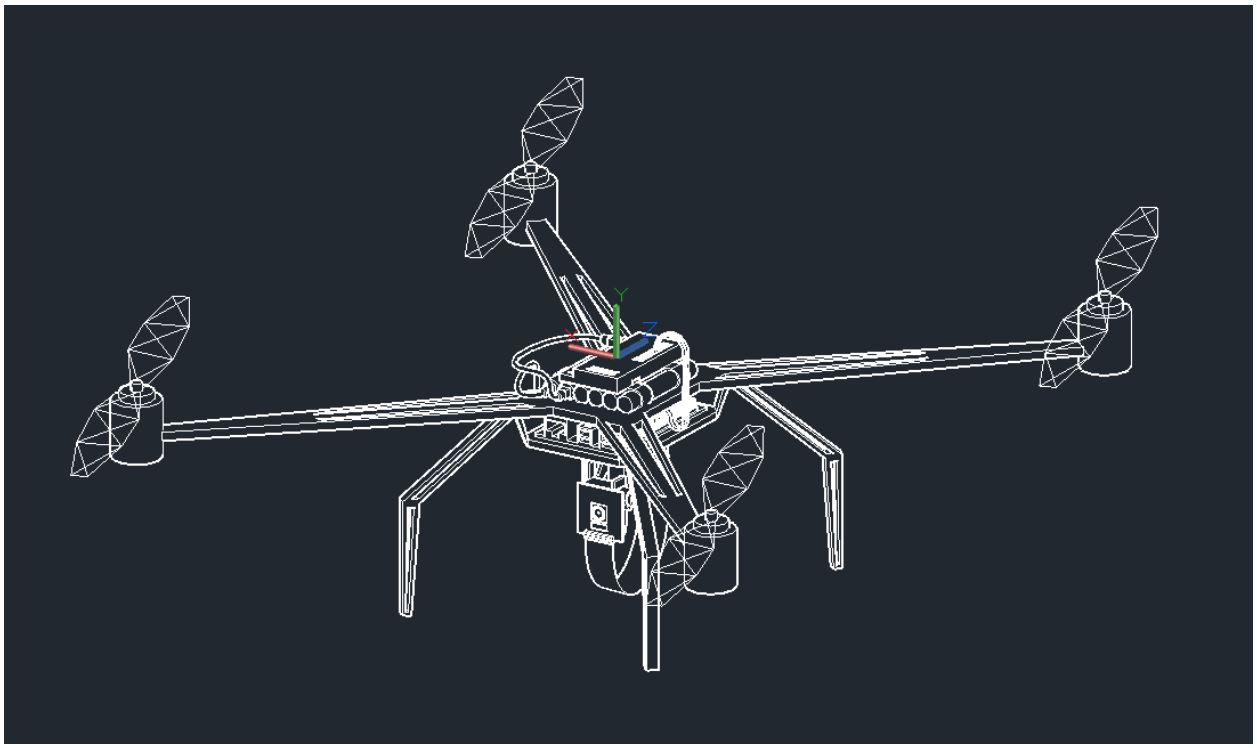


Рис. 3.1.5 – Модель у AutoCAD (Приховані лінії)

Наступним етапом є створення програмного забезпечення, яке здійснюватиме функції по розпізнаванню жестів і надання необхідних команд квадрокоптеру.

Дане ПЗ має виконувати наступні функції:

- Здійснювати керування сервоприводами для керування альт-азимутальної установкою
- Реалізовувати процес отримання кадру з відеопотоку і здійснбвати попередню обробку
- Реалізовувати процес виявлення жестів
- Реалізовувати процес розпізнавання жестів
- Надавати інформацію на бортовий комп’ютер квадрокоптера

Таким чином, для подальшої реалізації програмного коду, була створена діаграма прецедентів (рис. 3.1.6). (ПОСИЛАННЯ НА ГРАФ МАТЕРІАЛ)

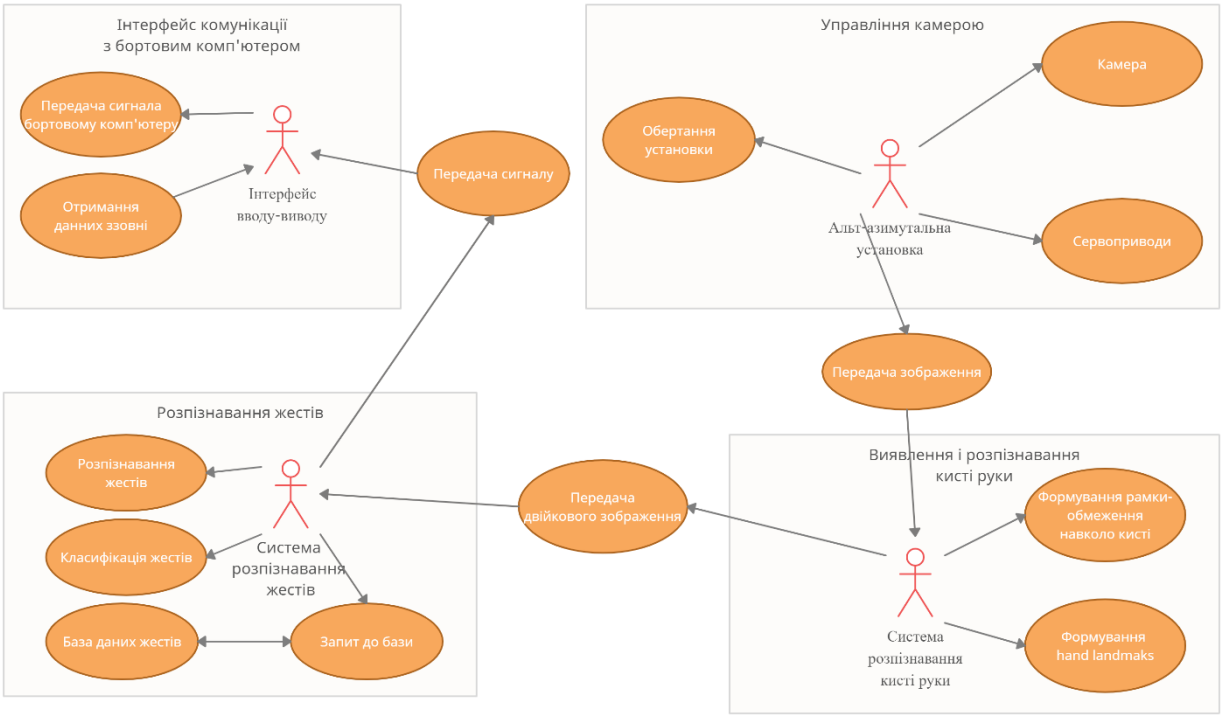


Рис. 3.1.6 – Діаграма прецедентів

Це граф, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи, асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання. Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання

(use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

На основі діаграми прецедентів, був створений програмний код (опис реалізації окремих класів у підрозділі 3.2.). Враховуючи написаний програмний код, була реалізована діаграма послідовностей (рис. 3.1.7) (ПОСИЛАННЯ НА ГРАФ МАТЕРІАЛ)

На основі програмного коду, для легшого розуміння структури програми, була реалізована діаграма класів. (ПОСИЛАННЯ НА ГРАФ МАТЕРІАЛ)

Загальний план роботи програми виглядає наступним чином. Після запуску програми через термінал з одним із аргументів: "default" (або не вказуючи ніяких параметрів), "learning", "make_data", виконується один із сценаріїв програми.

Сценарій при запуску з аргументом "default" (або без аргументів). Програма запускається у звичайному режимі. Створюється об’єкти класів HandTracking та AltAzimuth і запускається основний метод processTracking(), у якому відбувається основна логіка програми. Паралельно, в окремому процесі відбувається виконання методу mainLoop(), об’єкта класа AltAzimuth, який забезпечує обертання альт-азимутальної установки. Поки не буде виявлена кисть, альт-азимутальна установка буде проводити сканування, у разі виявлення кисті – буде фокусувати камеру на кисті руки. Під час виконання методу processTracking(), здійснюються наступні дії:

- Об’єкт класу NeuralNetwork (нейромережа) здійснює завантаження конфігураційних даних
- Створюється об’єкт класу MediaPipeHandsDetector, що забезпечуватиме роботу по виявленню жестів
- Розпочинається робота циклу, який працюватиме до моменту, поки працює програма (або до аварійного завершення)
- Отримуємо зображення з камери
- Передаємо зображення об’єкту класу MediaPipeHandsDetector для аналізу

на наявність кисті руки. Якщо кисть була виявленна, вектор ключових точок кисті руки передається на обробку у нейромережу, яка аналізує дані і виявляє логіку жеста. У разі успіху, передає сигнал на пін.

- Якщо жест був виявлений, здійснюється обробка кадру для подальшого знаходження центра мас кисті руки.
- Відбувається обчислення центра мас кисті руки і відносно положенню центра мас на зображенні, здійснюється корегування положення камери.
- Здійснюється відображення обробленого кадру зображення (якщо підключений дисплей)
- Відбувається наступна ітерація циклу.

Сценарій при запуску з аргументом "learning". Створюється об’єкт класу *HandTracking* і запускається метод *processLearning*. Під час роботи методу *processLearning*() здійснюється навчання нейромережі.

Сценарій при запуску з аргументом "make_data" (або без аргументів). Створюється об’єкти класів *HandTracking* та *AltAzimuth* і запускається метод, який забезпечує створення навчальних даних. Паралельно, в окремому процесі відбувається виконання методу *mainLoop*(), об’єкта класа *AltAzimuth*, який забезпечує обертання альт-азимутальної установки. Поки не буде виявлена кисть, альт-азимутальна установка буде проводити сканування, у разі виявлення кисті – буде фокусувати камеру на кисті руки. У процесі здійснюється захоплення кисті руки, і, натискаючи клавіші 0-8, здійснюється формування навчальних даних, одного із 9 жестів.

Приклад роботи програми зображено на рисунку

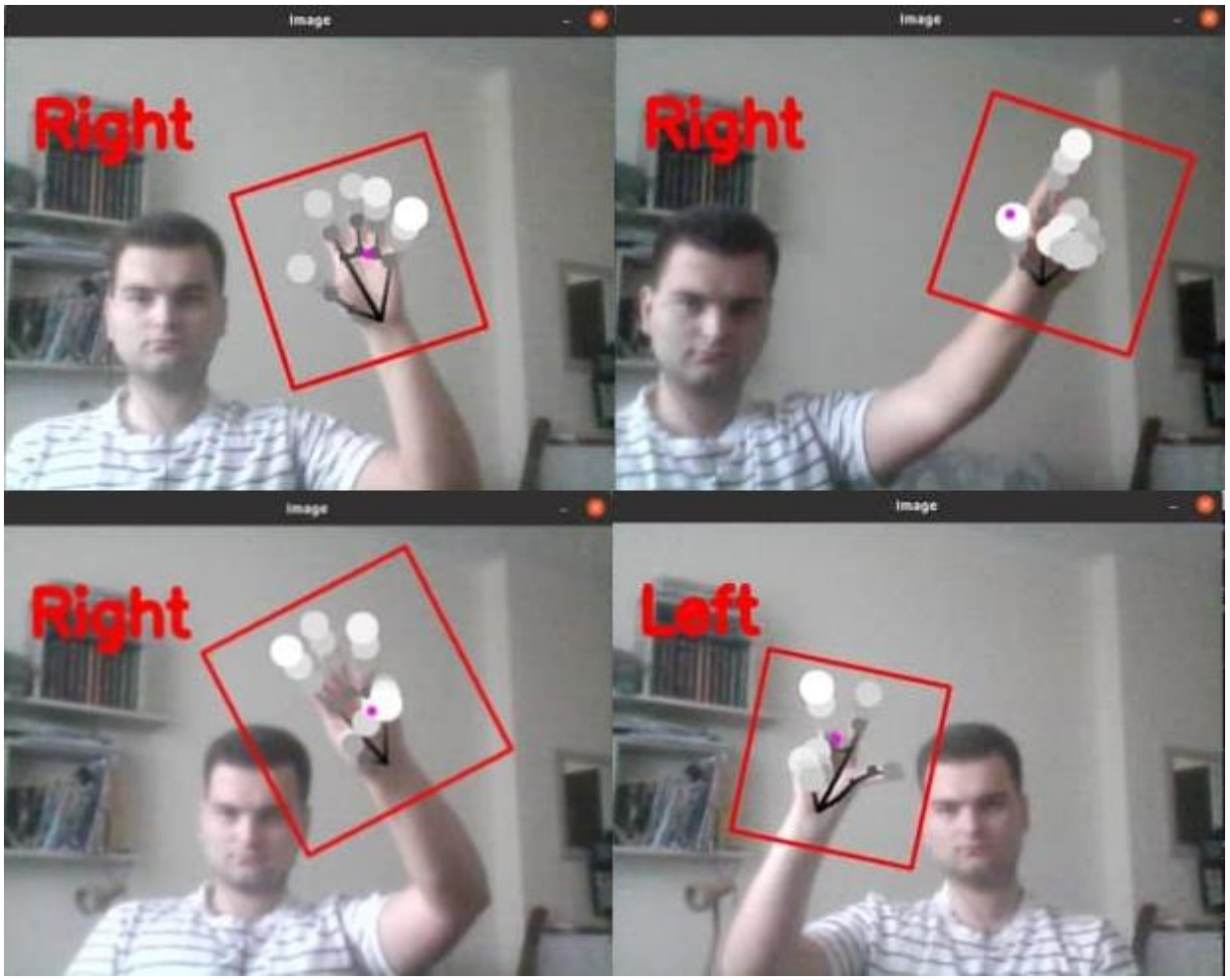


Рис. – Результат роботи програми.

3.2. Програмна реалізація модуля

3.2.1. Клас *AltAzimuth*

Загальний опис класу *AltAzimuth.*

Клас *AltAzimuth* реалізовує логіку обертання альт-азимутальної установки, на яку вмонтована камера, яка забезпечує надання системі зображень. В загальному, в класі реалізовано два основних метода, які спрацьовують залежно від обставин. На початку роботи модуля, спрацьовує метод пошуку кисті руки (*void searching()*), який здійснює сканування області. Сканування області відбувається шляхом обертання камери горизонтально у межах 0˚ - 180˚. Вертикально, наведення альт-азимутальної установки здійснюється у межах 0˚ - 165˚. Якщо кисть була виявлена, розпочнеться виконання методу відстеження (*void tracking()*). Даний метод працює на основі місцерозташування центру мас

точок кисті руки, відносно центру зображення. Надаючи загальний опис даному методу, обертання альт-азимутальної установки здійснюється наступним чином: якщо центр мас знаходиться правіше від центру зображення, камеру необхідно повернути ліворуч, і навпаки, якщо центр мас знаходиться лівіше від центру зображення – необхідно здійснити обертання праворуч. Подібним чином, логіка задана і для вертикального обертання – якщо центр мас вище – опускаємо камеру, якщо нижче – камера піднімається. Якщо центр мас знаходиться доволі близько від центру зображення (не далі відстані, що відповідає 15% половини висоти або ширини зображення. Тобто, якщо зображення має розміри 640 на 480 пікселів, то половини від цих розмірів будуть 320 та 240 відповідно, а 15% від даних розмірів складатимуть 48 та 36 пікселів відповідно. Якщо по ширині зображення, центр мас буде не далі ніж на 48 пікселів, а по висоті – не більше 36, то обертання камери не буде здійснюватися) то обертання припиняються.

Поля класу *AltAzimuth*.

bool isWork – змінна, яка відстежує роботи системи. Якщо значення змінної істино, то робота альт-азимутальної установки продовжується, інакше – припинити роботу.

cv::Point point – змінна, яка зберігає координати центра мас кисті руки. Це клас бібліотеки OpenCV, містить дві змінні *int x* та *int y*. Якщо дані змінні рівні -1, отже кисть не виявленна.

cv::Point centre – координати центра зображення.

int width, height – змінні зберігають ширину і висоту зображення відповідно.

int calibrationW, calibrationH - відстанні, які будуть викоритовуватися при заданні зони у якій, при знаходжені point, сервоприводи не будуть рухатися (15% від половини ширини і довжини зображення).

int d_x, d_y – буферні змінні. Містять різницю між координатами центра зображення та центра мас.

int pin0, pin1 – змінні зберігають номера пінів на GPIO Raspberry Pi, через які відбувається модуляція ШІМ (у роботі використані піни 24 та 23 відповідно).

int range – задає діапазон для ШІМ (у роботі використано значення 100).

int value – задає частоту для ШІМ (у роботі використано значення 50).

Методи класу *AltAzimuth*.

void tracking(int delayQuantity = 50) – даний метод реалізовує логіку

обертання камери, для здійснення пошуку кисті руки. Здійснюється др того моменту, поки *isWork* є істиною і значення змінних *x* та *y* змінної *point* рівні -1. Аргумент *delayQuantity* задає швидкість обертання камери.

void searching(int delayQuantity = 150) – метод, що реалізовує обертання альт-азимутальної установки, що імітує процес відстеження кисті руки. Як було раніше сказано, логіка методу побудована на основі обчислення положення цетру мас кисті руки відносно центра кадра зображення. Аргумент *delayQuantity* задає швидкість обертання камери.

void setPositionX(int delayQuantity) – здійснює обертання альт-азимутальної установки у вертикальній площині. Аргумент *delayQuantity* задає швидкість обертання камери.

void setPositionY(int delayQuantity) – здійснює обертання альт-азимутальної установки у горизонтальній площині. Аргумент *delayQuantity* задає швидкість обертання камери.

AltAzimuth(bool &isWork, cv::Point &point, int width = 640, int height = 480, pin0 = 23, pin1 = 24, range = 100, value = 50) – конструктор класу. Встановлює значення, для змінних *isWork, point, width, height, pin0, pin1, range, range, value.* Також проводить налаштування GPIO за допомогою бібліотек wiringPi та softPwm.

void mainLoop() – головний метод класу. Залежно від умов, активує метод пошуку кисті руки, або метод відстеження.

virtual ~AltAzimuth() – деструктор класу.

3.2.2. Клас *HandTracking*

Загальний опис класу *HandTracking.*

Клас *HandTracking* є головним класом у даному програмному забезпеченні. Він об’єднує у собі логіку всіх інших класів (окрім класу *AltAzimuth*, з яким зв’язаний опосередковано, надаючи дані про наявність кисті руки і центр мас). Даний клас може здійснювати три сценарії, залежно від аргументу, який задається у терміналі при запуску програми.

Якщо не було задано ніякого аргументу або задано аргумент «default», то розпочинається звичайне виконання роботи програми. Розпочинає виконання

метод *processTracking()*. Спочатку здійснюється завантаження конфігураційних даних нейромережі і відбувається її ініціалізація. Створюється об'єкт класу *MediaPipeHandsDetector*, який відповідає за процес виявлення кисті руки і передачі даних у нейромережу на аналіз. Надалі відбувається входження програми у цикл, який завершується при умові, якщо змінна *isWork* набуде хибного значення. Ітерація циклу виглядає наступним чином:

- Отримуємо зображення із камери;
- Дане зображення перевертається по діагоналі (через те, що розташування альт-азимутальної установки передбачається перевернутим).
- У разі, якщо ніяке зображення не було отримано, це сигналізує закінчення роботи. Змінні *isWork* присвоюється значення *false* і ітерація циклу преривається оператором *break*.
- Здійснюємо виклик методу *RunMPPGraph(cv::Mat &camera_frame_raw)* об'єкту класа *MediaPipeHandsDetector*. Даний метод здійснює обробку зображення і аналіз його на наявність кисті руки. У разі виявлення кисті руки, дані передаються у нейромережу на розпізнання жесту.
- Якщо жест виявлено, здійснюємо обробку кадра зображення, для отримання центру мас кисті руки.
- Проводимо обчислення центру мас кисті руки і передачу координат у об'єкт класа *AltAzimuth*.
- Якщо жест не виявлено, у об'єкт класа *AltAzimuth* передаємо значенян координат центру мас рівні –1 (що означає, що кисть не виявлена).
- На кінець ітерації, відображаємо оброблений кадр зображення.

Якщо був переданий аргумент «learning», то розпочинається процес навчання нейромережі. Розпочинає виконання метод *processLearning()*. Об'єкт класу NeuralNetwork здійснює виклик функції *study(std::string type)*, який розпочинає процес навчання нейромережі. Після закінчення навчання, нейромережа зберігає сформовує конфігураційні файли за допомогою виклику метода saveWeights(std::string path).

Якщо був переданий аргумент «make_data», то розпочинається процес створення навчальних даних для майбутнього навчання нейромережі. Робота схожа на роботу програми при запуску з аргументом «default», проте нейромережа неактивна і при натиснені клавіш цифр 0-8, здійснюється зберігання даних кисті

руки з міткою відповідного жеста.

Поля класу *HandTracking*.

bool isWork – змінна, яка відстежує роботи системи. Якщо значення змінної істино, то робота класу продовжується, інакше – припинити роботу.

bool isDetected – змінна, яка приймає значення істини, у разі, якщо кисть руки була виявлена, інакше змінна приймає значення хибності.

cv::VideoCapture capture – змінна типу даних з бібліотеки OpenCV для захоплення відеопотоку.

bool grab_frames – змінна позначає стан відеопотоку (*capture*). Якщо істина – процес захоплення відеопотоку працює, інакше – завершити процес захоплення відеопотоку.

cv::Mat camera_frame_raw – змінна зберігає кадр відеопотоку. *cv::Mat* – тип даних бібліотеки OpenCV і собою представляє матрицю, комірки якого описують зображення у певній кольоровій моделі (RGB, HSV).

std::string windowName – зберігає назву вікна, у якому відображаються кадри відео.

cv::Scalar lower{0,250,250} та *cv::Scalar upper{5,255,255}* – змінні задають кольоровий спектр для порогових операцій.

cv::Mat frameBlur – змінна містить зображення кадру, до якого було застосовано розмиття Гауса.

cv::Mat frameHSV – змінна зберігає зображення, конвертоване у HSV.

cv::Mat frameMask – змінна зберігає бінарне зображення, після застосування порогових операцій.

std::vector <std::vector<cv::Point>> contours - змінна зберігає контури, які отримано після застосування алгоритма Сузукі85.

std::vector <cv::Vec4i> hierarchy – змінна зберігає ієрархію контурів. У роботі не використовується і необхідна лише для реалізації роботи алгоритма Сузукі85.

std::vector <std::vector<cv::Point>> contoursPoly – змінна зберігає апроксимовані контури.

int area – змінна зберігає площу фігури, яку утворює контур. Застосовується, щоб відсіяти невеликі контури, які є «шумами».

double perimeter – зберігає периметр контура. Також застосовується для

відсіювання «шумів».

cv::Point point – змінна зберігає координати центра мас кисті руки.

int X, Y, numbPoints – буферні змінні, які використовуються при обчисленні центра мас кисті руки.

std::vector<int> hiddenLayers – вектор, що містить кількість персоптронів у прихованих шарах. Кількість елементів *hiddenLayers* позначає кількість прихованих шарів.

int inputPerceptrons – зберігає кількість перцептронів у вхідному шарі.

int outputPerceptrons – зберігає кількість перцептронів у вихідному шарі.

*NeuralNetwork *neuralNetwork* – вказівник на клас NeuralNetwork. На даний вказівник здійснюється динамічне виділення пам'яті, що забезпечує створення нейромережі.

Методи класу *HandTracking*.

void calculateCenterOfMass() – даний метод здійснює обчислення центру мас кисті руки, у разі її виявлення. Обчислення центру мас – це обчислення середнього арифметичного для координат *x* та *y* усіх ключових точок кисті руки. Після підрахунку координат центру мас, по координатам рисується на зображені невелике рожеве коло, що візуально позначає центр мас на зображенні.

void showVideo() – метод відображає кадри зображення.

void frameProcessing() – метод здійснює обробку зображення для виявлення центру мас кисті руки. У процесі роботи метода застосовуються такі алгоритми, як: розмиття Гауса, конвертація у іншу кольорову модель (HSV), порогові операції, алгоритм знаходження контурів (Сузукі85), апроксимація контурів за алгоритмом Рамера-Дугласа-Пекера.

HandTracking(int inputPerceptrons, int outputPerceptrons, std::vector<int> hiddenLayers) – конструктор класу *HandTracking*. У даному конструкторі відбувається створення об'єкту класа NeuralNetwork і його ініціалізація. Зокрема вхідні аргумени задаються кількість перцептронів у вхідному та вихідному шару нейромережі, також задається кількість прихованих шарів і їхня кількість перцептронів.

~HandTracking() – деструктор класа *HandTracking*.

void processTracking() – основний метод класа. Містить у собі виклики ключових методів об'єктів класів і задає логіку роботи даного програмного

забезпечення.

void processLearning() – метод, що реалізує процес навчання нейромережі і зберігання результату навчання.

bool &getIsWork() – метод повертає посилання на змінну siWork.

cv::Point &getPoint() – повертає посилання на змінну *point*.

3.2.3. Клас *MediaPipeHandsDetector*

Загальний опис класу *MediaPipeHandsDetector*.

Клас *MediaPipeHandsDetector* реалізовує логіку виявлення жестів за допомогою засобів бібліотеки *MediaPipe*. Принцип роботи бібліотеки, зокрема логіка виявлення кисті руки, висвітлено у Розділі 2. Загалом, принцип роботи даного класу полягає в наступному:

- Ініціалізація графу;
- Створення захоплювачів, через які буде здійснено отримання *Hand Landmarks*, кількість виявлених долонь (у програмному коді бібліотеки був змінений параметр, який відповідав за кількість одночасно виявлених кисть рук на зображені до 1), обробленого зображення із графічними мітками ключових точок кисті руки;
- Передача кадру зображення у граф і здійснення аналізу даного зображення графом на наявність кисті руки;
- Отримання за допомогою захоплювачів необхідних даних;
- Передача вектора з ключовими точками у нейромережу.

Поля класу *MediaPipeHandsDetector*.

bool isDetected – змінна, яка приймає значення істини, у разі, якщо кисть руки була виявлена, інакше змінна приймає значення хибності.

mediapipe::CalculatorGraph graph – граф MediaPipe.

std::unique_ptr<mediapipe::OutputStreamPoller> outputVideoPoller – захоплювач, через який здійснюється отримання обробленого зображення із графічними мітками ключових точок кисті руки на ньому.

std::unique_ptr<mediapipe::OutputStreamPoller> outputLandmarksPoller –

захоплювач, через який здійснюється отримання ключових точок кисті руки.

std::unique_ptr<mediapipe::OutputStreamPoller> outputHandCountPoller – захоплювач, через який здійснюється отримання кількості виявлених кисть рук.

std::string calculator_graph_config_contents - Змінна зберігає шлях до конфігураційного файлу (**.pbtxt*).

mediapipe::CalculatorGraphConfig config - Змінна зберігає конфігураційні дані.

sizeframe_timestamp_us_t – змінна, що зберігає поточну часову мітку.

Методи класу *MediaPipeHandsDetector*.

absl::Status Setup(std::string calculator_graph_config_file) – метод здійснює встановленя деяких конфігурацій для роботи графа.

absl::Status Shutdown() – метод для коректного завершення роботи об'єкта класу *MediaPipeHandsDetector*.

MediaPipeHandsDetector(bool &isDetected, std::string calculator_graph_config_file = "mediapipe/graphs/hand_tracking/custom_hand_tracking_desktop_live.pbtxt") – конструктор класа *MediaPipeHandsDetector*. Здійснює встановлення змінної *isDetected* і виклик метода *Setup()*.

virtual ~MediaPipeHandsDetector() – деструктор класа MediaPipeHandsDetector. Викликає метод *Shutdown()*.

absl::Status RunMPPGraph(cv::Mat &camera_frame_raw) – основний метод, що здійснює опрацювання кадру відеопотоку і надає можливіть отримати необхідну інформацію у вигляді вектора ключових точок кисті руки. Повертає *absl::StatusOk*, що є свідченням успішного виконання метода.

3.2.4. Класи *NeuralNetwork* та *Perceptrons*.

Загальний опис класу *Perceptrons*.

Даний клас є допоміжним для *NeuralNetwork* і реалізовує шар перцептронів. У тілі класу надано визначення структури *perceptron*, яка реалізовує окремий перцептрон. Клас *Perceptrons* містить у собі вектор, тип даних якого є *perceptron*.

Даний вектор є втіленням шару перцептронів.

Поля класу *Perceptrons*.

struct perceptron – структура, що реалізовує окремий перцептрон. Містить наступні змінні:

- *double value* – значення перцептрону;
- *double d_value* – значення помилки перцептрону;
- *std::vector<double> weights* – вектор ваг, які ідуть від перцептрона;
- *std::vector<double> d_weights* – вектор помилок ваг, які ідуть від перцептрона;
- *double derivative* – буферна змінна, яка використовується під час обчислень.

std::vector<perceptron> perceptronsLayer – вектор, який реалізовує шар перцептронів.

perceptron bias – здвиг.

int quantityOfPerceptrons – змінна зберігає кількість перцептронів у даному шару.

int quantityOfWeights – змінна зберігає кількість ваг, які ідуть від будь-якого перцептрона даного шару.

Методи класу *Perceptrons*.

Perceptrons(int numbOfPerceptrons, int numbOfWeights) – конструктор даного класу. Здійснює ініціалізацію шару пецептронів, в залежності вхідних аргументів.

~Perceptrons() – деструктор класу *Perceptrons.*

int getQuantityOfPerceptrons() – метод повертає кількість перцептронів у шару.

int getQuantityOfWeights() - метод повертає кількість ваг, що ідуть від будь-якого перцептрону у даному шару.

Загальний опис класу *NeuralNetwork.*

Клас *NeuralNetwork* здійснює реалізацію нейронної мережі на основі алгоритму зворотного поширення помилки. Детальний опис алгоритму наведено у другому розділі. Метод *void study(std::string type)* здійснює процес навчання.

Поля класу *NeuralNetwork.*

int countLayers - зберігає кількість слоїв у нейромережі.

std::vector<Perceptrons> layers* - масив вказівників на слої нейромережі.

bool isLearning – якщо істина – отже навчання нейромережі має продовжуватися, інакше – закінчити навчання.

std::string typeOfData – задає тип жесту.

int numbOfMotion – задає номер жесту.

std::vector<double> data - вектор, що містить дані для вхідного шару нейромережі.

std::vector<double> arr - Буферна змінна, для зберігання значень вихідного шару.

Методи класу *NeuralNetwork*.

NeuralNetwork(int inputPerceptrons, int outputPerceptrons, std::vector<int> hiddenLayers) – конструктор класу *NeuralNetwork*. За допомогою вхідних аргументів, задається кількість перцептронів на вхідному та вихідному шарах, кількість прихованих перцептронів і кількість перцептронів у кожному із прихованих шарів.

~NeuralNetwork() – деструктор класу *NeuralNetwork*.

void setInputPerceptrons(std::string type, std::vector<double> &data) - функція встановлює значення на вхідному шарові персептрона. *vector<double> data* - вектор, що містить значення вхідного слою персептронів.

void setPrimaryPerseptrons() - метод виставляє початкові значення для значень персептронів.

void setPrimaryWeights() - метод виставляє початкові значення для ваг персептронів.

double sigm(double x) – метод, що реалізовує сигмоїду.

double d_sigm(double x) - метод, що реалізовує похідну сигмоїди.

void calculatePerceptrons(int begin, int end, int layer) - метод обраховує персептрони наступного слою.

void calculateNeuralNetwork() - метод обраховує значення нейромережі.

void setTrueAnswer(int numbPerceptron) - метод виставляє в останьому слої нейромережі в змінних *d_x* "правильні значення" персептронів. Потім відбувається обрахування *d_x = x - d_x*.

void backPropagationBias(int layer) - обчислення помилки для здвигу за

методом ЗПП.

void backPropagationWeights(int layer) - обчислення помилки для ваг за методом ЗПП.

void weightsDifference(double coefficient = 0.02) – додає до ваг знайдені «посилки» (відбувається процес коренування ваг). *double coefficient* – задає швидкість навчання нейромережі.

void backPropagationPerceptrons(int layer) – метод, що реалізовує розповсбдження помилки на перцептрони.

void backPropagation() - реалізація методу ЗПП.

double levelOfEducation() - функція визначає рівень навчиності нейромережі. Чим значення *double*, що повертає функція, ближче до нуля, тим більше навчина нейромережа.

bool saveWeights(std::string path) – збереження даних нейромережі, а саме ваг.

bool loadWeights(std::string path, int numbOfLayer) - завантаження даних нейромережі, а саме ваг.

void loadData(std::string type) – метод реалізує завантаження параметрів нейромережі, а саме кількість шарів нейромережі і кількість перцептронів у цих шарах.

void study(std::string type) – один із основних методів нейромережі. Здійснює виклик вибірки методів, що забезпечує повний цикл навчання за алгоритмом ЗПП.

void calculateDirsAndFiles(std::string type, std::vector<int> &dirsAndFiles_out, int &quantityOfFiles_out) – технічний методі, для підрахування кількості папок і файлів у конкретній

int getQuantityOfItemInFolder(std::string pathToFolder) - технічний методі, для підрахування кількості навчальних даних у папці.

*void saveTrainData(std::string type , int numbOfFolder /*Номер жеста*/, const std::vector<double> &data)* – метод реалізовує логіку збереження навчальних даних по конкретному жесту.

void loadTrainData(std::string type, int numbOfFolder, int numbOfFile, std::string &type_out, int &numb_out, std::vector<double> &data_out) - метод реалізовує логіку завантаження навчальних даних по конкретному жесту.

3.3. Інструкція по експлуатації

Початок експлуатації модуля не є важким. Перш за все, необхідно переконатися у тому, що Raspberry Pi увімкнута. Після цього, необхідно відкрити термінал і перейти у теку, в якій знаходиться папка з проектом і зайти у дану папку. Після цього запустити команду:

bazel-bin/mediapipe/examples/desktop/controle_module/controle_module <аргумент>

де замість *<аргумент>* необхідно підставити один із наступних аргументів командного рядка:

- *default* – запускає модуль у звичайному режимі роботи;
- learning – запускає процес навчання нейромережі;
- *make_data* – запускає процес формування навчальних даних. Для того, щоб створити навчальний жест, необхідно надати кисті руки необхідного вигляд необхідного жесту, і натиснути клавішк 0-8 на клавіатури, що позначає необхідний жест.

У разі, якщо не було надано ніяких аргументів, або аргумент був прописаний невірно, програма запускається у режимі *default.*

3.4. Висновки до розділу

У даному розділу було надано детальний опис процесу реалізації предмету дипломного дослідження, а саме створення модуля розпізнавання жестів рук для керування квадрокоптером.

Було проведено роботу по висвітленню всього процесу побудови модуля. Детально наведена інформація по реалізації концепції модуля розпізнавання жестів, а саме наведенені критерії, за якими визначалося подальша розробка. Зокрема, модуль мав відповідати наступним вимогам:

- Компактність;
- Автономність;
- Рухома камера;
- Можливість розмістити модуль на квадрокоптері.

Враховуючи перелічені вимоги, була вибрана апаратура, яка необхідна для успішної реалізації модуля. Таким чином, основою, на якій базуватиметься робота усього пристрою, став одноплатний комп’ютер Raspberry Pi 3B+, який має

можливість керувати підключеними до нього модулями. На основі даного комп'ютера була проведена збірка модуля розпізнавання жестів, у яку увійшли камера, альт-азимутальна установка, сервоприводи та елементи живлення.

Після того, як була створена апаратна частина модуля, розпочалася реалізація програмного забезпечення. Для забезпечення якісної роботи програмного забезпечення, були обдумані функції, які має виконувати дане ПЗ. На основі даних функції, була створена діаграма прецеднентів, опираючись на яку, розпочалася робота над програмним кодом. Після написання коду, була проведена робота по забезпеченню кращого розуміння коду, тому були створенні діаграми послідовності та класів.

Таким чином, був реалізований предмет дипломного проектування. Окрім цього, даний модуль був створений таким чином, щоб в майбутньому була можливість проводити його модернізацію і розширювати можливості системи.

ВИСНОВКИ

Під час написання даної роботи було розроблено алгоритм для генерації тестового набору для ручного виконання і побудовано додаток, що реалізує наведений підхід. Алгоритм демонструє застосування теоретичних відомостей про скінчені автомати і графи для побудови технології, що може асистувати відділу тестування у найкритичнішій його активності – виборі скінченого числа перевірок серед їх безкінечної множини, які з найбільшою ймовірністю викриютьпомилки у програмному забезпечені.

Робота проводилась поетапно, у її процесі було детально розглянуто наступні проблеми і вирішено наступні задачі:

1. Розглянуто процес тестування і його складові. Виявлені проблема наявності помилок у існуючих системах, що є складними і часто асистують людству у критичних завданнях; необхідність тестування з метою віднаходження та попередження помилок під час використання; та конфлікт безкінечних можливостей для перевірок і скінчених ресурсів – людських та часу; і як результат, необхідність залучення інструментальної підтримки процесу тестування.

2. Залучення автоматизації до виконання і генерації тестів. Недоліки і переваги обох підходів та обґрунтування вибору автоматизації створення тестів як задачі дослідження.

3. Існуючи способи формального представлення вимог до програмного продукту – логіко-алгебраїчні та операційні моделі. Обґрунтування вибору на користь скінчено – автоматної моделі для описання еталону поведінки програмного забезпечення.

4. Розглянуто варіанти представлення скінченого автомату і обґрунтовано вибір на користь його візуального представлення діаграмою станів і переходів,що дає можливість залучення до розв’язання глобальної задачі дослідження такого потужного інструменту як теорія графів.

5. Розробка і обґрунтування алгоритмів побудови тестів і тестового

набору з використанням рекурсивного і жадібного алгоритмів.

6. Побудовано додаток, що реалізує підхід до генерації тестів і проведені експерименти з його використання.

Заявлений підхід налічує наступні переваги:

1. Гнучкість. Підхід може бути використаний для побудови тестового набору для продуктів з різних предметних областей, побудованих з використанням різних архітектур і технологій.

2. Фіксація максимальної кількості кроків тесту. У тестовий набір буде залучено тести перевищуючі максимальну встановлену довжину, тільки якщо існують переходи, недосяжні з початкової вершини за максимальну кількість кроків. Таким чином забезпечується зниження ймовірності допущення помилки під час виконання тестів, що є актуальною проблемою під час виконання довгих сценаріїв.

3. Недетермінованість. Евристичний алгоритм відбору тестів до тестового набору може обирати різні тести із встановленою метрикою тестового покриття, таким чином мінімізуючи ризик настання «парадоксу пестициду».

4. Мінімізація розміру тестового набору. Тестовий набір включає кращі тести для досягнення встановленого покриття, замість усіх можливих тестів.

5. Гарантованість досягнення встановленого рівні покриття. Для складних систем, що представлені графом із сотнею вершин і більше, гарантувати досягнення покриття усіх переходів створюючи тестовий набір у ручний спосіб складно і існує ймовірність помилки, на відміну від автоматичного способу.

6. Статичне тестування вимог. Під час перетворення представлення вимог на природній мові у скінчено-автоматну модель відбувається їх статичне тестування: виявляються їх недоліки, неточності і протиріччя.

Серед недоліків можна виділити наступні:

1. Обмеження на модель. Тестовий набір може бути побудований з використанням даного підходу лише для програмного забезпечення, еталонна модель якого може бути представлена у вигляді графу станів і переходів скінченого автомату.

2. Неповна автоматизація. Перетворення моделі графу – автомату у список

ребер і співставлення унікальних номерів ребер їх семантичному значенню не автоматизовано на даному етапі.

Додаток може бути успішно застосованим вже на поточному етапі у тестуванні програмних продуктів. Перспективним етапом дослідження є пошук такої модифікації алгоритму, за яким тести обираються до набору, який гарантував би покриття лише заданих переходів. Такий напрямок дослідження дасть змогу автоматично будувати набори для регресійного тестування, коли перевірці підлягатимуть лише переходи, які могли зазнати негативного впливу в результаті внесення змін до програмного продукту. Описаний підхід може бути вбудований у середовище підтримки неперервної інтеграцій, тоді відділ тестування буде кожного разу після внесення змін отримувати не лише нову версію продукту, а й набір перевірок, що треба провести.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *IP VPNs for Service Providers*: *The Foundation for Profitable http*://*www.cisco.com*/.
2. *Beazley D. Python essential reference* // *BHV*. – 2015. – 734 *с*.
3. *Jones C.A., Drake F.L. Python* & *XML* // *BHV*. – 2014. – 807 *с*.
4. ДСТУ 2851-94 Програмні засоби ЕОМ. Документування результатів випробувань.
5. НД ТЗІ 1.1-003-99. Термінологія у області захисту інформації в комп'ютерних системах від несанкціонованого доступу. // Департамент спеціальних телекомунікаційних систем і захисту інформації Служби безпеки України. – Київ, 1999.
6. Гамма Э., Холм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2006. – 366 с.
7. Доусон М. Программируем на *Python*. – СПб.: Питер, 2014. – 416 с.
8. Лутц М. Изучаем *Python*, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с.
9. Лутц М. Программирование на *Python*, том *I*, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.
10. Лутц М. Программирование на *Python*, том *II*, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.
11. Прохоренок Н.А. *Python* 3 и *PyQt*. Разработка приложений. – СПб.: БХВ-Петербург, 2012. – 704 с.
12. Прохоренок Н.А. Самое необходимое. — СПб.: БХВ-Петербург, 2011. — 416 с.
13. Фаулер М., Скотт К. *UML*. Основы. – Пер. с англ. – СПб.: Символ-Плюс, 2002.

14. Фаулер, Мартин – Архитектура корпоративных программных приложений.: Пер. с англ. – М.: Издательский дом "Вильяме", 2006. – 544 с.
15. Хахаев И.А. Практикум по алгоритмизации и программированию на *Python*. – М.: Альт Линукс, 2010. — 126 с.
16. Шаньгин В. Ф. Защита компьютерной информации, М: ДМК. 2008 – 544с.
17. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

Додаток А

Програмний код розробленого пакета

```
Collections_Test. js
#import "Join_It_Functions. js"
var target = UIATarget. localTarget ();
UIALogger. logStart ("Join It Temp Script");
target. delay (1);
target. frontMostApp (). mainWindow (). elements () [0]. tap (); // Cancel video
target. delay (2);
target. frontMostApp (). mainWindow (). buttons () ["Collections"]. tap ();
target. delay (1);
target. frontMostApp (). mainWindow (). buttons () ["Add Pictures"]. tap ();
target. delay (1);
target. frontMostApp (). mainWindow (). popover (). tableViews () [0]. cells ()
[Random (8)]. tap ();
target. delay (5);
for (i=0; i<3; i++)
{
target. frontMostApp (). mainWindow (). popover (). scrollViews () [0]. elements
() [i]. tap ();
target. delay (2);}
target. delay (5);
target. frontMostApp (). mainWindow (). popover (). navigationBar (). buttons ()
["Back"]. tap ();
target. delay (1);
target. frontMostApp (). mainWindow (). popover (). tableViews () [0]. cells ()
["My Photos"]. tap ();
target. delay (1);
```

```
target.frontMostApp().mainWindow().popover().tableViews()["Empty list"].cells()[0].tap();
target.delay(1);
target.frontMostApp().mainWindow().popover().tableViews()["Empty list"].cells()[20].images()[0].tap();
for(j=0; j<10; j++)
{
target.frontMostApp().mainWindow().tableViews()["Empty list"].cells()[j].tap();
target.delay(1);
}
target.frontMostApp().mainWindow().tableViews()["Empty list"].cells()["Join It - Original Collection"].tap();
target.delay(1);
target.frontMostApp().mainWindow().scrollViews()[0].elements()[0].tap();
target.delay(1);
target.frontMostApp().mainWindow().tableViews()["Empty list"].cells()["All Pictures"].tap();
target.delay(1);
target.frontMostApp().mainWindow().buttons()["Edit"].tap();
target.delay(1);
for(k=0; k<4; k++)
{
target.frontMostApp().mainWindow().scrollViews()[0].elements()[Random(16)].tap();
target.delay(1);
}
target.frontMostApp().mainWindow().buttons()["Delete"].tap();
target.delay(1);
```

```
target.frontMostApp().mainWindow().popover().buttons()["Delete Selected Pictures"].tap();
target.delay(1);
target.frontMostApp().mainWindow().buttons()["Back"].tap();
target.delay(1);
UIALogger.logPass("Done");
Smoke_Test.js
#import "Join_It_Functions.js"
var target = UIATarget.localTarget();
UIALogger.logStart("Join It Temp Script");
target.delay(1);
target.frontMostApp().mainWindow().elements()[0].tap(); // Cancel video
target.delay(1);
First_Launch_Check();
Choose_Random_Picture();
Change_Difficulty("Easy");
target.delay(1);
Change_Difficulty("Medium");
target.delay(1);
Change_Difficulty("Hard");
target.delay(1);
Change_Difficulty("Insane");
target.delay(1);
Change_Difficulty_Drag(Random(1));
target.delay(1);
Change_Difficulty("Easy");
target.delay(1);
Open_Options();
Tap_Sticking();
Tap_Rotation  ();
Tap_Timer();
```

```
Tap_Sound ();
Tap_Iphone ();
Tap_Ipad ();
Tap_Piece_Style (1);
Tap_Piece_Style (2);
Tap_Piece_Style (3);
Tap_Piece_Style (4);
Tap_Piece_Style ();
Choose_Background (Random (8));
Close_Options ();
Start_Game (5);
Game ();
UIALogger. logPass ("Done");
target. frontMostApp (). mainWindow (). images () [3]. doubleTap ();
Suspend_Test. js
#import "Join_It_Functions. js"
var target = UIATarget. localTarget ();
UIALogger. logStart ("Join It Temp Script");
target. delay (1);
target. frontMostApp (). mainWindow (). elements () [0]. tap (); // Cancel video
target. delay (2);
// First_Launch_Check ();
Deactivate (3);
target. delay (1);
target. frontMostApp (). mainWindow (). buttons () ["Collections"]. tap ();
target. delay (1);
Deactivate (3);
target. delay (1);
target. frontMostApp (). mainWindow (). scrollViews () [0]. elements () [0]. tap
();
target. delay (1);
```

```
Deactivate (3);
target. delay (1);
Open_Options ();
target. delay (1);
Deactivate (3);
target. delay (1);
Close_Options ();
target. delay (1);
Deactivate (3);
target. delay (1);
Start_Game (5);
target. delay (1);
Deactivate (3);
target. delay (1);
target. frontMostApp (). mainWindow (). staticTexts () ["Resume"]. tap ();
target. delay (1);
Exit_Game ();
target. delay (1);
Deactivate (3);
target. delay (1);
UIALogger. logPass ("Done");
Main_Screen_Test. js
#import "Join_It_Functions. js"
var target = UIATarget. localTarget ();
UIALogger. logStart ("Join It Temp Script");
target. delay (1);
target. frontMostApp (). mainWindow (). elements () [0]. tap (); // Cancel video
target. delay (2);
target. frontMostApp (). mainWindow (). buttons () ["HelpToolbar"]. tap ();
target. delay (1);
target. frontMostApp (). mainWindow (). buttons () ["BackToolbar"]. tap ();
```

```
target.delay (1);
Change_Difficulty ("Easy");
target.delay (2);
Change_Difficulty ("Medium");
target.delay (2);
Change_Difficulty ("Hard");
target.delay (2);
Change_Difficulty ("Insane");
target.delay (2);
target.frontMostApp ().mainWindow ().elements () [7].tap (); // Enter Hi Score
target.delay (1);
target.frontMostApp ().mainWindow ().elements () [8].tap (); // Exit Hi Score
target.delay (1);
Change_Difficulty_Drag (0);
target.delay (1);
Change_Difficulty_Drag (0.5);
target.delay (1);
if (target.frontMostApp ().mainWindow ().buttons () [2].isVisible () ==true)
{
target.frontMostApp ().mainWindow ().buttons () [2].tap ();
target.delay (1);
}
Open_Options ();
Close_Options ();
target.frontMostApp ().mainWindow ().images () [2].buttons () ["Info"].tap ();
target.delay (4);
target.frontMostApp ().mainWindow ().images () [2].buttons () ["saveButton"].
tap ();
target.frontMostApp ().mainWindow ().images () [2].buttons ()
["zoomButton"].tap ();
target.delay (1);
```

```
target.frontMostApp().mainWindow().images()[2].tap();
target.delay(1);
target.frontMostApp().mainWindow().buttons()["Next"].tap();
target.delay(2);
target.frontMostApp().mainWindow().buttons()["Previous"].tap();
target.delay(2);
target.flickFromTo({x:360, y:200}, {x:160, y:200}); // Flick Picture Change (NEXT)
target.delay(2);
target.flickFromTo({x:160, y:200}, {x:260, y:200}); // Flick Picture Change (PREVIOUS)
target.delay(2);
target.frontMostApp().mainWindow().buttons()["Collections"].tap();
target.delay(1);
target.frontMostApp().mainWindow().buttons()["Back"].tap();
target.delay(1);
target.frontMostApp().mainWindow().images()[2].buttons()["Go"].tap();
UIALogger.logPass("Done");
Difficulty_Test.js
#import "Join_It_Functions.js"
var target = UIATarget.localTarget();
UIALogger.logStart("Join It Temp Script");
target.delay(1);
target.frontMostApp().mainWindow().elements()[0].tap(); // Cancel video
target.delay(1);
Change_Difficulty("Easy");
Start_Game(5);
target.captureScreenWithName("Easy Game");
Exit_Game();
Change_Difficulty("Medium");
Start_Game(10);
```

```
target. captureScreenWithName ("Medium Game");
Exit_Game ();
Change_Difficulty ("Hard");
Start_Game (15);
target. captureScreenWithName ("Hard Game");
Exit_Game ();
Change_Difficulty ("Insane");
Start_Game (20);
target. captureScreenWithName ("Insane Game");
Exit_Game ();
UIALogger. logPass ("Done");
Help_Test. js
#import "Join_It_Functions. js"
var target = UIATarget. localTarget ();
UIALogger. logStart ("Join It Temp Script");
target. delay (1);
target. frontMostApp (). mainWindow (). elements () [0]. tap (); // Cancel video
target. delay (2);
target. frontMostApp (). mainWindow (). buttons () ["HelpToolbar"]. tap ();
target. delay (1);
target. captureScreenWithName ("Help_Screenshot_iPad 1");
target. delay (1);
for (i=2; i<5; i++)
{
target. frontMostApp (). mainWindow (). scrollViews () [0]. scrollViews () [0].
webViews () [0]. scrollDown ();
target. delay (1);
target. captureScreenWithName ("Help_Screenshot_iPad " +i);
target. delay (1);
}
target. delay (2);
```

```
target.frontMostApp().mainWindow().scrollViews()[0].scrollViews()[0].webViews()[0].staticTexts()["iPhone Controls"].tap();
target.delay(1);
target.captureScreenWithName("Help_Screenshot_iPhone 1");
target.delay(1);
for(i=2; i<5; i++)
{
target.frontMostApp().mainWindow().scrollViews()[0].scrollViews()[0].webViews()[0].scrollDown();
target.delay(1);
target.captureScreenWithName("Help_Screenshot_iPhone " +i);
target.delay(1);
}
target.delay(2);
target.frontMostApp().mainWindow().scrollViews()[0].scrollViews()[0].webViews()[0].links()["\nJoin It website"].staticTexts()["Join It website"].tap();
target.delay(1);
target.frontMostApp().mainWindow().buttons()["BackToolbar"].tap();
UIALogger.logPass("Done");
Picture_Change_Test.js
#import "Join_It_Functions.js"
var target = UIATarget.localTarget();
UIALogger.logStart("Join It Temp Script");
target.delay(1);
target.frontMostApp().mainWindow().elements()[0].tap(); // Cancel video
target.delay(1);
target.frontMostApp().mainWindow().buttons()["Collections"].tap();
target.delay(1);
target.frontMostApp().mainWindow().scrollViews()[0].elements()[0].tap();
target.delay(2);
```

```
for (i=1; i<16; i++)
{
target.flickFromTo ({x: 360, y: 200}, {x: 160, y: 200}); // Flick Picture Change (NEXT)
target.delay (2);
}
for (i=1; i<16; i++)
{
target.flickFromTo ({x: 160, y: 200}, {x: 260, y: 200}); // Flick Picture Change (PREVIOUS)
target.delay (2);
}
for (j=1; j<16; j++)
{
target.frontMostApp ().mainWindow ().buttons () ["Next"].tap (); // Button Picture Change (NEXT)
target.delay (2);
}
for (j=1; j<16; j++)
{
target.frontMostApp ().mainWindow ().buttons () ["Previous"].tap (); // Button Picture Change (PREVIOUS)
target.delay (2);
}
UIALogger.logPass ("Done");
Piece_Style_Test.js
#import "Join_It_Functions.js"
var target = UIATarget.localTarget ();
UIALogger.logStart ("Join It Temp Script");
target.delay (1);
target.frontMostApp ().mainWindow ().elements () [0].tap (); // Cancel video
```

```
target.delay (1);
// First_Launch_Check ();
for (j=1; j<5; j++)
{
Open_Options ();
Tap_Piece_Style (j);
target.delay (1);
Close_Options ();
Start_Game (5);
target.captureScreenWithName ("Piece_Syle " + j);
Exit_Game ();
}
UIALogger.logPass ("Done");
High_Score_Test.js
#import "Join_It_Functions.js"
var target = UIATarget.localTarget ();
UIALogger.logStart ("Join It Temp Script");
target.delay (1);
target.frontMostApp ().mainWindow ().elements () [0].tap (); // Cancel video
target.delay (2);
target.frontMostApp ().mainWindow ().elements () [7].tap (); // High SCore
Change_Difficulty ("Easy");
target.delay (2);
target.captureScreenWithName ("High Score for Easy");
Change_Difficulty ("Medium");
target.delay (2);
target.captureScreenWithName ("High Score for Medium");
Change_Difficulty ("Hard");
target.delay (2);
target.captureScreenWithName ("High Score for Hard");
Change_Difficulty ("Insane");
```

```
target. delay (2);
target. captureScreenWithName ("High Score for Insane");
target. frontMostApp (). mainWindow (). buttons () ["HelpToolbar"]. tap ();
target. frontMostApp (). mainWindow (). buttons () ["BackToolbar"]. tap ();
Open_Options ();
Close_Options ();
target. frontMostApp (). mainWindow (). elements () [8]. tap (); // Main Menu
UIALogger. logPass ("Done");
Join_It_Functions. js
var target = UIATarget. localTarget ();
function Deactivate (t)
{
UIATarget. localTarget (). deactivateAppForDuration (t);
}
function Start_Game (t)
{
target. frontMostApp (). mainWindow (). images () [3]. buttons () ["Go"]. tap ();
target. delay (t);
target. tap ({x: 350.00, y: 350.00});
}
function Game ()
{
Open_Options ();
Set_Reference_Picture_Visibility (Random (1));
Tap_Sticking ();
Tap_Picture_Frame ();
Set_Table_Scale (Random (1));
Tap_Zoom_Lock ();
Tap_Rotation ();
Tap_Timer ();
Tap_Sound ();
```

```
Tap_Sharing ();
Choose_Background (Random (8));
Close_Options ();
Exit_Game ();
}
function Exit_Game ()
{
target. frontMostApp (). mainWindow (). buttons () ["Pause"]. tap ();
target. delay (1);
target. frontMostApp (). mainWindow (). staticTexts () ["Main Menu"]. tap ();
}
function First_Launch_Check ()
{
if (target. frontMostApp (). mainWindow (). staticTexts () ["What's New"].
isVisible () ==true)
{
target. delay (20);
target. frontMostApp (). mainWindow (). buttons () ["Continue"]. tap ();
target. delay (1);
target. frontMostApp (). mainWindow (). buttons ()
["iPadButtonSwitchPressed"]. tap ();
target. delay (1);
target. frontMostApp (). mainWindow (). buttons () ["OK"]. tap ();
target. delay (1);
}
}
function Open_Options ()
{
if (target. frontMostApp (). mainWindow (). buttons () ["HelpToolbar"]. isVisible
() ==true)
{
```

```
target. frontMostApp (). mainWindow (). buttons () ["SettingsSmall"]. tap (); //
The Options
}
else
{
target. frontMostApp (). mainWindow (). buttons () ["SettingsControl"]. tap (); //
The InGame Options
}
target. delay (1);
}
function Close_Options ()
{
target. tap ({x: 651.00, y: 63.00});
}
function Random (p)
{
if (p<=1)
{
r = Math. random ();
}
else
{
r = parseInt (Math. random () * p) +1;
}
UIALogger. logMessage ("Random Number is " +r)
return r;
}
function Set_Reference_Picture_Visibility (r)
{
target. frontMostApp (). mainWindow (). scrollViews () [0]. sliders () [0].
dragToValue (r);
```

```
	}
	function Tap_Sticking ()
	{
	target. frontMostApp (). mainWindow (). scrollViews () [0]. elements () [6]. tap
(); // STICKING y={95: 123}
	target. delay (1);
	}
	function Tap_Picture_Frame ()
	{
	target. frontMostApp (). mainWindow (). scrollViews () [0]. elements () [10]. tap
();
	}
	function Set_Table_Scale (r)
	{
	target. frontMostApp (). mainWindow (). scrollViews () [0]. sliders () [1].
dragToValue (r);
	}
	function Tap_Zoom_Lock ()
	{
	target. frontMostApp (). mainWindow (). scrollViews () [0]. elements () [10]. tap
();
	}
	function Tap_Rotation ()
	{
	target. frontMostApp (). mainWindow (). scrollViews () [0]. elements () [22]. tap
(); // ROTATION y={430: 460}
	target. delay (1);
	}
	function Tap_Timer ()
	{
```

```
target.frontMostApp().mainWindow().scrollViews()[0].elements()[25].tap(); // TIMER y={479: 509}
target.delay(1);
}
function Tap_Sound()
{
target.frontMostApp().mainWindow().scrollViews()[0].elements()[28].tap(); // SOUND y={526: 556}
target.delay(1);
}
function Tap_Sharing()
{
target.frontMostApp().mainWindow().scrollViews()[0].elements()[30].tap(); // SHARING y={573: 603}
target.delay(1);
}
function Tap_Iphone()
{
target.frontMostApp().mainWindow().scrollViews()[0].elements()[34].tap(); // IPHONE
target.delay(1);
}
function Tap_Ipad()
{
target.frontMostApp().mainWindow().scrollViews()[0].elements()[35].tap(); // IPAD
target.delay(1);
}
function Tap_Piece_Style(Piece_Num)
{
switch(Piece_Num)
```

```
{
case 1:
target.frontMostApp().mainWindow().scrollViews()[0].scrollViews()[0].elements()[0].tap(); // PIECE STYLE 1
target.delay(1);
break;
case 2:
target.frontMostApp().mainWindow().scrollViews()[0].scrollViews()[0].elements()[1].tap(); // PIECE STYLE 2
target.delay(1);
break;
case 3:
target.frontMostApp().mainWindow().scrollViews()[0].scrollViews()[0].elements()[2].tap(); // PIECE STYLE 3
target.delay(1);
break;
case 4: target.frontMostApp().mainWindow().scrollViews()[0].scrollViews()[0].elements()[3].tap(); // PIECE STYLE 4
target.delay(1);
break;
default:
target.frontMostApp().mainWindow().scrollViews()[0].scrollViews()[0].elements()[2].tap(); // PIECE STYLE 3
target.delay(1);
}
}
function Choose_Background(r)
{
target.frontMostApp().mainWindow().scrollViews()[0].scrollViews()[0].elements()[3].scrollToVisible();
target.delay(1);
```

```
        target.frontMostApp().mainWindow().scrollViews()[0].buttons()[8].tap();
        target.delay(1);
        switch(r)
        {
        case 1:
        target.frontMostApp().mainWindow().scrollViews()[0].buttons()[1].tap();
// PIECE STYLE 1
        target.delay(1);
        break;
        case 2:
        target.frontMostApp().mainWindow().scrollViews()[0].buttons()[2].tap();
// PIECE STYLE 2
        target.delay(1);
        break;
        case 3:
        target.frontMostApp().mainWindow().scrollViews()[0].buttons()[3].tap();
// PIECE STYLE 3
        target.delay(1);
        break;
        case 4:
        target.frontMostApp().mainWindow().scrollViews()[0].buttons()[4].tap();
// PIECE STYLE 4
        target.delay(1);
        break;
        case 5:
        target.frontMostApp().mainWindow().scrollViews()[0].buttons()[5].tap();
// PIECE STYLE 1
        target.delay(1);
        break;
        case 6:
```

```
        target.frontMostApp().mainWindow().scrollViews()[0].buttons()[6].tap();
// PIECE STYLE 2
        target.delay(1);
        break;
        case 7:
        target.frontMostApp().mainWindow().scrollViews()[0].buttons()[7].tap();
// PIECE STYLE 3
        target.delay(1);
        break;
        case 8:
        target.frontMostApp().mainWindow().scrollViews()[0].buttons()[8].tap();
// PIECE STYLE 4
        target.delay(1);
        break;
        default:
        target.frontMostApp().mainWindow().scrollViews()[0].buttons()[1].tap();
// PIECE STYLE 3
        target.delay(1);
        }
        target.delay(1);
        }
        function Change_Difficulty(Difficulty)
        {
        switch(Difficulty)
        {
        case "Easy":
        target.frontMostApp().mainWindow().elements()[3].tap(); // EASY
        target.delay(1);
        break;
        case "Medium":
        target.frontMostApp().mainWindow().elements()[4].tap(); // MEDIUM
```

```
target.delay (1);
break;
case "Hard":
target.frontMostApp (). mainWindow (). elements () [5]. tap (); // HARD
target.delay (1);
break;
case "Insane":
target.frontMostApp (). mainWindow (). elements () [6]. tap (); // INSANE
target.delay (1);
break;
default:
target.frontMostApp (). mainWindow (). elements () [4]. tap (); // MEDIUM
target.delay (1);
}
}
function Change_Difficulty_Drag (Difficulty_Drag)
{
target.frontMostApp (). mainWindow (). sliders () [0]. dragToValue
(Difficulty_Drag);
target.delay (2);
}
function Choose_Random_Picture (Random_Image)
{
target.frontMostApp (). mainWindow (). buttons () ["Collections"]. tap ();
r=Random (6);
if (r==1)
{
r++;
}
target.frontMostApp (). mainWindow (). tableViews () ["Empty list"]. cells () [r].
tap ();
```

```
target.delay (2);
r=Random (15);
target.frontMostApp ().mainWindow ().scrollViews () [0].elements () [r].tap
();
target.delay (1);
r++;
if (target.frontMostApp ().mainWindow ().buttons () ["Edit"].isVisible ()
==true)
{
target.delay (30);
target.frontMostApp ().mainWindow ().scrollViews () [0].elements () [r].tap
();
target.delay (1);
}
}
```