**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. а

**Керівник:**  к.т.н. доц. Глазок О.М.

**Консультанти з окремих розділів пояснювальної записки:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Нормоконтролер:**  Тупота Є.В. а

**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.

ЗАВДАННЯ

НАВИКОНАННЯДИПЛОМНОЇРОБОТИСТУДЕНТА

Кашкевич І-Ф. Ф.

(прізвище*,* ім'я*,* по батькові)

**1.**Т**ема проєкту (роботи):**  “Методика використання платформи .*NET* при

створенні тригерів бази даних ”

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

**2.**Т**ермін виконання проєкту (роботи):** з 10.03.2014 до 22.06.2014

**3.** В**ихідні дані до проекту (роботи):** постановка задачі до виконання роботи*,*

мова програмування *C*#*,* середовище програмування *MS Visual Studio 2010,*

СУБД *MS* *SQL Server*

**4.** З**міст 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.** П**ерелік 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 а

Керівник дипломної роботи Глазок О.М.

(підпис)

ЗавданняприйнявдовиконанняКашкевич 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 6](#_bookmark0)  [ВСТУП 7](#_bookmark1)  [РОЗДІЛ 1 АНАЛІЗ ПІДХОДІВ ДО ТЕСТУВАННЯ ПРОГРАМНОГО](#_bookmark2) [ЗАБЕЗПЕЧЕННЯ 11](#_bookmark2)   * 1. [Зв'язок між тестування і забезпеченням якості 15](#_bookmark3)   2. [Автоматизація процесу тестування програмного](#_bookmark4)   [забезпечення 16](#_bookmark4)   * 1. [Обґрунтування генерації тестів 23](#_bookmark5)   2. [Висновки до розділу 25](#_bookmark6)   [РОЗДІЛ 2 ОБГРУНТУВАННЯ ПІДХОДУ ДО РОЗРОБКИ СИСТЕМИ 26](#_bookmark7)   * 1. [Формальна модель представлення вимог 26](#_bookmark8)   2. [Аналіз сучасних методик та засобів тестування програмних](#_bookmark9) [застосувань для мобільних пристроїв 31](#_bookmark9)   3. [Побудова тестового набору 39](#_bookmark10)   4. [Висновки до розділу 44](#_bookmark11)   [РОЗДІЛ 3 ОПИСАННЯ РОБОТИ СИСТЕМИ ВІДСЛІДКОВУВАННЯ](#_bookmark12) [ПОМИЛОК В ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ](#_bookmark12)  [*WEB*-ДОДАТКІВ 45](#_bookmark12)   * 1. [Логіка роботи додатку 45](#_bookmark13)   2. [Структура програмного додатку 48](#_bookmark14)   3. [Інтерфейс додатку та параметри запуску 50](#_bookmark15)   4. [Висновки до розділу 55](#_bookmark16)   [ВИСНОВКИ 57](#_bookmark17)  [СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ 60](#_bookmark18)  [ДОДАТОК А 62](#_bookmark19) | |
|  | *5* |

|  |  |
| --- | --- |
| ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ*,* СКОРОЧЕНЬ*,* ТЕРМІНІВ  *ЕОМ – Електронна обчислювальна машина*  *ШІ – Штучний інтелект IoT - Internet of Things, Інтернет речі*  *ОПЗ – Оперативний запам’ятовуючий пристрій*  *ПЗЗ – Пристрій із зарядним зв’язком*  *CCD – Charge-coupled device, (те саме, що і ПЗЗ)*  *КМОН – Комплементарна структура метал-оксид-напівпровідник*  *CMOS – Complementary-symmetry/metal-oxide semiconductor (те саме, що і а КМОН)*  *Мп – 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* | |
|  | *6* |

|  |  |
| --- | --- |
| 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. | |
|  | *7* |

|  |  |
| --- | --- |
| 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. | |
|  | *8* |

|  |  |
| --- | --- |
| 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 | |
|  | *9* |

|  |  |
| --- | --- |
| 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. | |
|  | *10* |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| РОЗДІЛ 1  АНАЛІЗ ЗАСОБІВ, МЕТОДІВ ТА АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ, ЯКІ ЗАСТОСОВУЮТЬ ПРИ СТВОРЕННІ ІНТЕРАКТИВНИХ СИСТЕМ КЕРУВАННЯ ЖЕСТАМИ   * 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. | | | | | | | | | |
| Кафедра КСУ | | | | НАУ вввввв 000 ПЗ | | | | | |
| **Виконав** |  |  |  | Аналіз підходів до тестування програмного забезпечення | **Літера** | | | **Аркуш** | **Аркушів** |
| **Керівник** |  |  |  | Д |  |  | *11* | *61* |
| **Консульт*.*** |  |  |  | СП 4….. 123 | | | | |
| **Норм*.* контр*.*** |  |  |  |
| **Зав*.* Каф*.*** |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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. 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 (градуси, ˚) | 86 | 72 | 48 | 30 | 25 | 17 | | 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.  image  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 (28). 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 | |
|  | *12* |

|  |  |
| --- | --- |
| 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 | |
|  | *13* |

|  |  |
| --- | --- |
| 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 | |
|  | *14* |

|  |  |
| --- | --- |
| 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.).  Сервоприводы купити  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. 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 | |
|  | *15* |

|  |  |
| --- | --- |
| 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). | |
|  | *16* |

|  |  |
| --- | --- |
| 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 | |
|  | *17* |

|  |  |
| --- | --- |
| 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:  *x2 + y2 = R2*,  де, *x* та *y* – координати центра кола,  *R* – діаметр кола. | |
|  | *18* |
| 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 кольоворої 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). | |
|  | *19* |

|  |  |
| --- | --- |
| 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 (A-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 (n0 – n2) та один вихідний (y0), 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. | |
|  | *20* |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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. | |
|  | *21* |

|  |  |
| --- | --- |
| РОЗДІЛ 2  ПРОЕКТУВАННЯ МОДУЛЯ РОЗПІЗНАВАНЯН ЖЕСТІВ РУК ДЛЯ КЕРУВАННЯ КВАДРОКОПТЕРОМ   * 1. Опис апаратних елементів, які застосовувалися під час проектування модуля.      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 | |
|  | *22* |

|  |  |
| --- | --- |
| 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 A.  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.) | |
|  | *23* |

|  |  |
| --- | --- |
| 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. | |
|  | *24* |

|  |  |
| --- | --- |
| 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 1А.   * 1. Технології, застосовані для обробки зображень, реалізації процеса виявлення жестів та керування альт-азимутальною установкою.      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. | |
|  | *25* |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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(). | | | | | | | | | | | |
|  | | | | | | | | | | | *25* |
|  | | | | | | | | | | | |
| афедра КСУ | | | | НАУ вввввв 000 ПЗ | | | | | | | |
| **Виконав** |  |  |  | Обгрунтування підходу до розробки системи | **Літера** | | | **Аркуш** | **Аркушів** | | |
| **Керівник** |  |  |  | Д |  |  | *26* | *61* | | |
| **Консульт*.*** |  |  |  | СП 4……. 123 | | | | | | |
| **Норм*.* контр*.*** |  |  |  |
| **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 С/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 GraphConfigнаявна 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 і двох основних підграфів, можна ознайомитися на рисунку 2.2.1.2..  Вставити 3 рисунка графів | |
|  | *27* |

|  |  |
| --- | --- |
| 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. | |
|  | *28* |

|  |  |
| --- | --- |
| плином часу. Значення таймерів можна використовувати в умовах переходів*,* зміну змінних або значеннях параметрів реакцій. Крім цього*,* таймери можна запускати в діях*,* пов'язаних з переходами. Запущений таймер починає відлік часу з 0. Час може бути дискретним чи безперервним*,* що дозволяє моделювати поведінку різноманітних систем реального часу.  Гібридні автомати застосовувані для моделювання систем*,* що взаємодіють з безперервними фізичними процесами. У цих автоматах частина змінних зазвичай має значення*,* зміна яких описується системою диференціальних рівнянь  Широке використання операційних моделей пов’язано з їх наступними перевагами:  У моделі звичайно враховуються не всі властивості модельованого програмного забезпечення*,* а тільки важливі для розглянутої в даний момент завдання. Тому моделі виявляються значно простіше модельованих систем*,* їх набагато зручніше аналізувати.  Віртуальні машини використовуються на практиці мов програмування дуже складні і визначені нечітко*,* а віртуальні машини моделей значно більш прості і доступні для огляду. Це дозволяє провести вичерпний аналіз можливої поведінки моделі*,* виявити всі класи можливих при її роботі ситуацій.  Моделі проміжного типу мають риси як логіко-алгебраїчних*,* так і операційних. Основні їх види такі.  Логіки Хоара є специфічним видом логік*,* затвердження яких складаються з формул логіки деякого виду та програмних команд. У найпростішому вигляді це трійки {*Q*} *S* {*R*}*,* де *S* – частина програми на певній мові*,* а *Q* і *R* – формули обчислення висловлювань*,* що залежать від змінних*,* що входять до *S*. *Q* інтерпретується як умова*,* виконання перед початком виконання *S* (передумова)*,* а *R* – як умову*,* що має виконуватись після виконання *R* (післяумови). Якщо *R* дійсно завжди істинне після виконання *S* у стані*,* де істинно *Q,* така трійка теж вважається дійсною і має назву «трійка Хоара». У логіці Хоара для деякої мови програмування*,* семантика цієї мови задається у вигляді правил виведення*,* які дозволяють з тавтологій виводити крок за кроком істинні трійки Хоара. | |
|  | *29* |

|  |  |
| --- | --- |
| Узагальненням логік Хоара є динамічні чи програмні логіки. Вони є спеціальним типом модальних логік*,* в яких оператори модальності пов'язані з інструкціями програм. Зазвичай використовуються оператори [*S*] і <*S*>*,* де *S* – деяка програма. Твердження [*S*]*Q* означає*,* що завжди після виконання програми *S* формула *Q* істинна*,* а <*S*> *Q* – що після виконання *S, Q* може виявитися істинною. Трійка Хоара {*Q*} *S* {*R*} може бути представлена у динамічній логіці як *Q*=>[*S*]*R*.  Програмні контракти*,* є окремим випадком логіки Хоара*,* що звужує можливості використання логічних формул. Програмний контракт представляє собою опис поведінки набору програмних компонентів представлених у вигляді опису сигнатур операцій кожного з цих компонентів*,* структур їх станів*,* а також передумов і післяумов для кожної операції та наборів інваріантів для кожного компонента окремо. Інваріант компонента є предикатом*,* що залежить від елементів стану цього компонента*,* який повинен бути виконаний в станах*,* коли жодна з операцій компоненту не виконується. Інваріанти описують обмеження цілісності внутрішніх даних компонента. Передумову операції компонента являє собою предикат*,* що залежить від елементів стану цього компонента і параметрів цієї операції. При виклику операції з порушенням передумови її поведінка не визначена. Передумова є частиною контракту*,* яку зобов'язане дотримуватися оточення компонента*,* щоб забезпечити його коректну роботу. Післяумова операції являє собою предикат*,* що залежить від параметрів операції*,* її результату*,* елементів стану компонента до виклику операції і тих же елементів після закінчення її роботи. Контракти часто неможливо виконати безпосередньо*,* оскільки післяумова не визначає прямо коректні результати операцій і наступні стани*,* а лише оцінює передані ним дані.  Серед усіх формальних підходів*,* що можуть бути застосовані до представлення специфікацій широке практичне застосування у проектуванні і тестуванні засобів автоматики*,* зв’язку*,* обчислювальної техніки і програмного забезпечення знайшли скінчені автомати.  Вибір на користь скінченого автомату зумовлений тим*,* що побудова логіко  – алгебраїчних та проміжних моделей практично не вигідна в силу величезних | |
|  | *30* |

|  |  |
| --- | --- |
| витрат на розробку додаткових аксіом*,* тверджень та умов(передумов) та післяумов*,* що визначають заключні правила отримання правильного результату.  2.2. Аналіз сучасних методик та засобів тестування програмних застосувань для мобільних пристроїв  Існуючі на сьогоднішній день методи тестування ПЗ не дозволяють однозначно і повністю виявити всі дефекти і встановити коректність функціонування аналізованої програми*,* тому всі існуючі методи тестування діють у рамках формального процесу перевірки досліджуваного або розроблюваного ПЗ.  Такий процес формальної перевірки*,* чи верифікації*,* Може довести*,* що дефекти відсутні з точки зору використовуваного методу. (Тобто немає ніякої можливості точно встановити або гарантувати відсутність дефектів у програмному продукті з урахуванням людського фактора*,* присутнього на всіх етапах життєвого циклу ПЗ).  Існує безліч підходів до вирішення завдання тестування і верифікації ПЗ*,* але ефективне тестування складних програмних продуктів - це процес у вищій мірі творчий*,* не зводиться до слідування строгим і чітким процедурам або створенню таких [8].  З точки зору *ISO* 9126*,* якість програмного забезпечення можна визначити як сукупну характеристику досліджуваного ПЗ з урахуванням наступних складових:   * надійність; * супроводжуваність; * практичність; * ефективність; * мобільність; * функціональність.   Існує кілька ознак*,* за якими прийнято проводити класифікацію видів тестування. Зазвичай виділяють наступні: | |
|  | *31* |

|  |  |
| --- | --- |
| 1) за об'єктом тестування:   * функціональне тестування (*functional testing*); * тестування продуктивності (*performance testing*); * навантажувальне тестування (*load testing*); * стрес-тестування (*stress testing*); * тестування стабільності (*stability* / *endurance* / *soak testing*); * юзабиліті-тестування (*usability testing*); * тестування інтерфейсу користувача (*UI testing*); * тестування безпеки (*security testing*); * тестування локалізації (*localization testing*); * тестування сумісності (*compatibility testing*);   2) за наявністю доступу до системи:   * тестування Чорної Скриньки (*black box*); * тестування білої скриньки (*white box*) * тестування сірої скриньки (*grey box*);   3) за ступенем автоматизації:   * ручне тестування (*manual testing*); * автоматизоване тестування (*automated testing*); * напівавтоматизоване тестування (*semiautomated testing*);   4) за ступенем ізольованості компонентів:   * компонентне (модульне) тестування (*component*/*unit testing*); * інтеграціонне тестування (*integration testing*); * системне тестування (*system*/*end*-*to*-*end testing*);   5) за часом проведення тестування:   * альфа-тестування (*alpha testing*); * тестування при прийомі (*smoke testing*); * тестування нової функціональності (*new feature testing*); * регресійне тестування (*regression testing*); * тестування при здачі (*acceptance testing*); * бета-тестування (*beta testing*);   6) за ознакою позитивності сценаріїв: | |
|  | *32* |

|  |  |
| --- | --- |
| * позитивне тестування (*positive testing*); * негативне тестування (*negative testing*);   7) за ступенем підготовленості до тестування:   * тестування по документації (*formal testing*); * тестування *ad hoc* або інтуїтивне тестування (*ad hoc testing*).   Необхідність автоматизації тестування класифікується за наступними ознаками:  1) за об’єктом тестування:   * функціональному тестуванні (*functional testing*); * тестуванні продуктивності (*performance testing*); * навантажувальному тестуванні (*load testing*); * стрес-тестуванні (*stress testing*); * тестуванні стабільності (*stability* / *endurance* / *soak testing*);   2) за наявністю доступу до системи:  – тестування чорної скриньки (*black box*);  3) за ступенем ізольованості компонентів:  – системне тестування (*system*/*end*-*to*-*end testing*);  4) за часом проведення тестування:   * альфа-тестування (*alpha testing*) * тестування при прийомі (*smoke testing*) * регресійне тестування (*regression testing*) * тестування при здачі (*acceptance testing*) * Бета-тестування (*beta testing*)   Розглянемо більш детально види тестування в яких може застосовуватись автоматизація:  1) функціональне тестування – це тестування ПЗ в цілях перевірки можливості реалізації функціональних вимог*,* тобто здатності ПЗ в певних умовах вирішувати задачі*,* необхідні користувачам.  Функціональні вимоги визначають*,* що саме робить ПЗ*,* які задачі воно вирішує.  Функціональні вимоги включають в себе: | |
|  | *33* |

|  |  |
| --- | --- |
| * функціональну пригодність (англ. *suitability*); * точність (англ. *accuracy*); * здатність до взаємодії (англ. *interoperability*). * Відповідність стандартам і правилам (англ.*compliance*). * Захищеність (англ. *security*).   2) тестування продуктивності в інженерії програмного забезпечення - тестування яке проводиться з метою визначення*,* як швидко працює система або її частину під певним навантаженням. Також може служити для перевірки і підтвердження інших атрибутів якості системи*,* таких як масштабованість*,* надійність і споживання ресурсів.  Тестування продуктивності – це одна зі сфер діяльності розвивається в галузі інформатики інженерії продуктивності*,* яка прагне враховувати продуктивність на стадії моделювання та проектування системи*,* перед качаном Основному стадії кодування. У тестуванні продуктивності розрізняють наступні напрямки:   * стрес (*stress*); * навантажувальне (*load*); * тестування стабільності (*endurance or soak or stability*); * конфігураційне (*configuration*).   3) навантажувальне тестування (англ. *Load Testing*) – визначення або збір показників продуктивності і годині відгуку програмно-технічної системи або пристрою у відповідь на зовнішній запит з метою встановлення відповідності вимогам*,* що пред'являються до даної системи (пристрою). Для дослідження годині відгуку системи на високих або пікових навантаженнях проводиться стрес-тестування*,* при якому створювана на систему навантаження перевищує нормальні сценарії його використання. Не існує чіткої межі між навантажувальним та стрес-тестуванням*,* однак ці поняття не варто змішувати*,* так як ці види тестування відповідають на різні бізнес-питання і використовують різну методологію.  Термін тестування навантаження може бути використаний у різних значеннях в професійному середовищі тестування ПЗ. У загальному випадку він | |
|  | *34* |

|  |  |
| --- | --- |
| означає практику моделювання очікуваного використання додатка за допомогою емуляції роботи декількох користувачів одночасно. Таким чином*,* подібне тестування найбільше підходить для екстермінатуса мультикористувацьких систем*,* частіше - використовують клієнт-серверну архітектуру (наприклад*,* веб- серверів). Однак і інші типи систем ПЗ можуть бути протестовані подібним способом. Наприклад*,* текстовий або графічний редактор можна змусити прочитати дуже великий документ; а фінансовий пакет - згенерувати звіт на основі даних за декілька років. Найбільш адекватно спроектований навантажувальний тест дає більш точні результати.  Основна мета навантажувального тестування полягає в тому*,* щоб*,* створивши певну очікувану в системі навантаження (наприклад*,* за допомогою віртуальних користувачів) і*,* звичайно*,* використавши ідентичне програмне і апаратне забезпечення*,* спостерігати за показниками продуктивності системи. В ідеальному випадку в якості критеріїв успішності навантажувального тестування виступають вимоги до продуктивності системи*,* які формулюються і документуються на стадії розробки функціональних вимог до системи до початку програмування основних архітектурних рішень. Однак часто буває так*,* що такі вимоги не були чітко сформульовані або не були сформульовані зовсім. У цьому випадку перше навантажувальний тестування буде являтися пробним (*exploratory load testing*) і ґрунтуватися на розумних припущеннях про очікувану навантаженні і споживанні апаратної частини ресурсів.  Розробка методів побудови математичних моделей "чорної" скриньки є однією з важливих кібернетичних проблем. За умови наявності математичної моделі "чорної" скриньки з'являється можливість віднести його до якогось одного класу*,* всі системи якого ізоморфні по поведінці.  Для науки метод “чорної скриньки” має дуже велике значення. З його допомога в науці було зроблено дуже багато видатних відкриттів. Наприклад*,* вчений Гарвей Галі в *XVII* столітті передбачив будову серця. Він моделював роботу серця насосом*,* запозичивши ідеї із зовсім іншої області сучасних йому знань - гідравліки. Практична цінність методу “чорної скриньки” полягає по- перше*,* в можливості дослідження дуже складних динамічних систем*,* і*,* по-друге*,* | |
|  | *35* |

|  |  |
| --- | --- |
| в можливості заміни одного "ящика" іншим. Навколишня дійсність і біологія дають масу прикладів виявлення будови систем методом "чорної" скриньки.  Принципи тестування чорного ящика – у цьому методі програма розглядається як чорний ящик. Метою тестування ставиться з'ясування обставин*,* в яких поведінка програми не відповідає специфікації. Для виявлення всіх помилок у програмі необхідно виконати вичерпне тестування*,* тобто тестування на всіх можливих наборах даних. Для більшості програм таке неможливо*,* тому застосовують розумне тестування*,* при якому тестування програми обмежується невеликою підмножиною можливих наборів даних. При цьому необхідно вибирати найбільш відповідні*,* підмножини з найвищою імовірністю виявлення помилок.  Прийоми тестування чорного ящика:   * еквівалентне розбиття; * аналіз граничних значень; * аналіз причинно-наслідкових зв’язків; * допущення про помилку; * системне тестування.   2.2.1. Тестування через еквівалентне розбиття Основу методу складають два положення:   1. вихідні данні необхідно розбити на кінцеве число класів еквівалентності; 2. в одному класі еквівалентності містяться такі тести*,* що*,* якщо один тест з класу еквівалентності виявляє деяку помилку*,* то й будь який інший тест з цього класу еквівалентності має виявляти цю ж помилку.   Кожен тест має включати*,* по можливості*,* максимальну кількість класів еквівалентності*,* щоб мінімізувати загальне число тестів.  Розробка тестів цим методом здійснюється в два типа: виділення класів еквівалентності і побудова тесту.  Класи еквівалентності виділяються шляхом вибору кожної вхідної умови*,* які беруться з допомогою технічного завдання або специфікації і розбиваються на дві та більше групи. | |
|  | *36* |

|  |  |
| --- | --- |
| Виділення класів еквівалентності є евристичним способом*,* однак існує ряд правил:  Якщо вхідна умова описує область значень*,* наприклад "Ціле число приймає значення від 0 до 999"*,* то існує один правильний клас еквівалентності і два неправильних.  Якщо вхідна умова описує число значень*,* наприклад "Число рядків у вхідному файлі лежить в інтервалі (1.6)"*,* то також існує один правильний клас і два неправильних.  Якщо вхідна умова описує безліч вхідних значень*,* то визначається кількість правильних класів*,* рівна кількості елементів у множині вхідних значень. Якщо вхідна умова описує ситуацію "повинно бути"*,* наприклад "Перший символ має буті заголовним"*,* тоді один клас правильний і один неправильний.  Якщо є підстава вважати*,* що елементи всередині одного класу еквівалентності можуть програмою трактуватися по-різному*,* необхідно розбити даний клас на підкласи. На цьому кроці тестувальник на основі таблиці має скласти тести*,* що покривають собою всі правильні і неправильні класи еквівалентності. При цьому тестувальник має мінімізувати загальне число тестів.  Визначення тестів:  Кожному класу еквівалентності присвоюється унікальний номер.  Якщо ще залишилися не включені в тести правильні класи*,* то пишуться тести*,* які покривають максимально можливу кількість класів.  Якщо залишилися не включені в тести неправильні класи*,* то пишуть тести*,* які покривають тільки один клас [13].  2.2.2. Тестування через аналіз граничних значень  Граничні умови - це ситуації*,* що виникають на вищих і нижніх межах вхідних класів еквівалентності.  Аналіз граничних значень відрізняється від еквівалентного роздроблення наступним:  Вибір будь-якого елемента в класі еквівалентності в якості | |
|  | *37* |

|  |  |
| --- | --- |
| представницького здійснюється таким чином*,* щоб перевірити тестом кожний кордон цього класу.  При розробці тестів розглядаються не тільки вхідні значення (простір входів)*,* але і вихідні (простір виходів).  Метод вимагає певної міри творчості та спеціалізації в розглянутій задачі. Існує декілька правил:  Побудувати тести з неправильними вхідними даними для ситуації незначного виходу за межі області значень. Якщо вхідні значення повинні буті в інтервалі [-1.0. +1.0]*,* Перевіряємо - 1.0*,* 1.0*,* - 1.000001*,* 1.000001.  Обов'язково писати тести для мінімальної і максимальної межі діапазону.  Використовувати перші два правила для кожного з вхідних значень (використовувати пункт 2 для Всіх вихідних значень).  Якщо вхід і вихід програми представляє впорядкована множина*,* зосередити увагу на першому і останньому елементі списку.  Аналіз граничних значень*,* якщо він застосований правильно*,* дозволяє виявити велику кількість помилок. Однак визначення цих кордонів для кожної задачі може бути окремим важким завданням. Також метод не перевіряє комбінації вхідних значень.  2.2.3. Тестування через аналіз причинно-наслідкових зв'язків  У специфікації визначаються множини причин і наслідків. Під причиною розуміється окрема вхідна умова або клас еквівалентності. Слідство являє собою вихідну умову або перетворення системи. Тут кожній причині і слідству присвоюється номер.  На основі аналізу семантичного (смислового) змісту специфікації будується таблиця істинності*,* в якій послідовно перебираються всі можливі комбінації причин і визначаються слідства для кожної комбінації причин.  Таблиця забезпечується примітками*,* які задають обмеження і описують комбінації*,* які неможливі. Недоліком цього підходу є погане дослідження граничних умов. | |
|  | *38* |

|  |  |
| --- | --- |
| * + 1. Тестування через припущення про помилку   Тестувальник з великим досвідом вишукує помилки без всяких методів*,* але при цьому він підсвідомо використовує метод припущення про помилку. Даний метод у значній мірі ґрунтується на інтуїції. Основна ідея методу полягає в тому*,* щоб скласти список*,* який перераховує можливі помилки і ситуації*,* в яких ці помилки могли проявитися. Потім на основі списку складаються тести.   * + 1. Системне тестування   Системне тестування програмного забезпечення - це тестування програмного забезпечення (ПЗ)*,* що виконується на повній*,* інтегрованій системі*,* з метою перевірки відповідності системи вихідним вимогам. Системне тестування відноситься до методів тестування “чорної скриньки”*,* і*,* тим самим*,* не вимагає знань про внутрішню побудову системи.  Основним завданням системного тестування є перевірка як функціональних*,* так і не функціональних вимог до системи в цілому. При цьому виявляються дефекти*,* такі як невірне використання ресурсів системи*,* непередбачені комбінації даних користувацького рівня*,* несумісність з оточенням*,* непередбачені сценарії використання*,* відсутня або невірна функціональність*,* незручність використання і т.д. Для мінімізації ризиків*,* пов'язаних з особливостями поведінки системи в тій чи іншій середовищі*,* під час тестування рекомендується використовувати оточення максимально наближене до того*,* на яке буде встановлений продукт після видачі.  2.3. Побудова тестового набору  Побудова тестового набору передбачає віднаходження такої множини *R,* яка входить у *T* і забезпечує покриття всіх переходів модельного графу – формула  *co*(*R*) = *co*(*T*)*,*  де *T –* множина усіх тестів (шляхів графу); | |
|  | *39* |

|  |  |
| --- | --- |
| *R* – результуючий тестовий набір;  *co*(*R*) – покриття графу *G,* що досягається набором шляхів *R*;  *co*(*T*) *–* покриття графу *G,* що досягається набором шляхів *T,* такого що є розв’язком задачі оптимізації min*T*  min *ti*  *R* .  Тестовий набір будується з використанням жадібного алгоритму (рис. 2.1).  Жадібний алгоритм дозволяє отримати оптимальне рішення задачі шляхом здійснення ряду виборів.  У кожній точці прийняття рішення в алгоритмі робиться вибір*,* який в даний момент виглядає найкращим.  У записі алгоритму використовуються наступні умовні позначення:  | *S* | – потужність множини *S*;  *len*(*t*) – довжина тесту *t,* яка вимірюється у кількості тестових впливів;  {} – Порожня множина;  <> – Порожній упорядкований список.  Фіксуємо обмеження довжини тестів *M,* яке є параметром даного алгоритму. *C* – повний набір доступних транзакцій*, T* – повний набір раніше побудованих тестів.  Для кожного тесту *t* з *T* обчислюємо *co* (*t*)*,* як безліч тестових ситуацій з *C,*  що покриваються даним тестом. Якщо *co* (*t* )={}*,* то видаляємо *t* з *T*. *procedure* ПОБУДОВА\_ТЕСТОВОГО НАБОРУ  (*M*: максимальна довжина кроків тесту*,*  *P*: вага допустимого перевищення максимальної довжини на користь покриття );  *begin*  *t*1 *T* оголошуємо кращим  *for* кожного тесту *ti* з множини тестів *T*  *if len*(*t*1) ≤ *M* && *len*(*ti*) > *M,* то тест *t*1 кращий  *if len*(*t*1) ≤ *M* && *len*(*ti*) ≤ *M* && *co*(*t*1) > *co*(*ti*)*,* то тест *t*1 кращий  *if co*(*t*1) ≥ *co*(*ti*) && *len*(*t*1) < *len*(*ti*)*,* то тест *t*1 кращий  *if co*(*t*1) > *co*(*ti*) && *len*(*t*1) ≤ *len*(*ti*)*,* то тест *t*1 кращий | |
|  | *40* |

Початок

Отримання переліку вершин

Визначення

поточного індексу

Цикл опитування вершин

Присутні вихідні

ребра

ні

так

Вершина не

відвідувалась

ні

так

Відмітка про

відвідування

Відмітка про

вихідні ребра

ні

Остання

вершина

так

Кінець

Рис. 2.1. Схема алгоритму побудови всіх маршрутів на графі

*41*

|  |  |
| --- | --- |
| *if len*(*t*1)  *M* && *len*(*ti*)*,* а (*len*(*t*1) – *len*(*ti*)) *P* < (*co*(*t*1) – *co*(*ti*))*,* то тест *t*1 кращий. Додаємо знайдений кращий тест у набір *R*. Видаляємо із безлічі непокритих ситуації *C* всі елементи множини *co*(*t*кращий).  *end; end*;  Перше правило –гарантує*,* що в першу чергу всі можливі тестові ситуації будуть покриватися тестами з довжиною*,* що не перевищує максимальну.  *len*(*t*1) ≤ *M* && *len*(*ti*) > *M*.  Друге правило – формула вибирає тести з максимально можливим (в межах обмеження довжини) тестовим покриттям – у більшості випадків ця евристика дозволяє зменшити сумарну довжину тестів в наборі за рахунок зменшення їх кількості.  *len*(*t*1) ≤ *M* && *len*(*ti*) ≤ *M* && *co*(*t*1) > *co*(*ti*).  Трете і четверте правила – віддають перевагу тестам*,* що поліпшують покриття або довжину і не погіршує притому іншу з цих двох характеристик.  *if co*(*t*1) ≥ *co*(*ti*) && *len*(*t*1) < *len*(*ti*) *if co*(*t*1) > *co*(*ti*) && *len*(*t*1) ≤ *len*(*ti*)  Згідно з п*’*ятим правилом з двох тестів*,* що перевищують максимальну довжину*,* один краще іншого*,* якщо збільшення довжини компенсується (з урахуванням вагового параметра *P*) збільшенням покриття.  *len*(*t*1) > *M* && *len*(*ti*) > *M* && (*len*(*t*1) – *len*(*ti*)) *P* < (*co*(*t*1) – *co*(*ti*)).  Оскільки алгоритм евристичний (рис. 2.2)*,* і в ньому використовується перебір по неупорядкованій множині*,* в загальному випадку результати його роботи недетерміновані | |
|  | *42* |

Початок

Отримання всіх *M* і *Р*

Розрахунок

*C*, *T*, *R*

Для кожного тесту з

набору *Т*

L (кращ)<*M && L(*поточ*)<M*

ні

так

L (кращ)<*M &&*

*c(кращ)<c*(поточ)

ні

так

L (поточ)<*M &&*

*c(*кращ*)<c*(поточ)

ні

так

параметри кращого = параметри поточного

ні

Останній *М*

так

Кінець

Рис. 2.2. Схема алгоритму побудови тестового набору

*43*

|  |  |
| --- | --- |
| На різних запусках на одних і тих же даних можлива генерація різних тестових наборів*,* а проте в більшості випадків побудовані таким чином набори мають однакові метрики.  Допустимі значення: ціле позитивне число або + ∞. *P* задає вагу перевищення максимальної довжини тестів по відношенню до підвищення покриття. При великих значеннях параметра алгоритм в першу чергу вибирає з тестів*,* що перевищують максимальну довжину і додають хоча б якесь тестове покриття*,* самі короткі*,* що дозволяє зменшити максимальну довжину тестів*,* але може приводити до істотного збільшення сумарної складності тестового набору; при малих значеннях алгоритм в першу чергу вибирає тести*,* що покривають більшу кількість тестових ситуацій*,* що дозволяє мінімізувати сумарну складність тестового набору за рахунок додавання в нього дуже довгих тестів. Допустимі значення: невід’ємні числа (рис. 2.2).  У ході апробації алгоритму встановлено*,* що в більшості випадків оптимальне значення для параметра *M* трохи перевершує довжину найдовшого можливого в графі маршруту без циклів*,* що веде з початкового стану в кінцеве*,* а для параметра *P* лежить в діапазоні.  2.4. Висновки до розділу  Скінчений автомат є математичною абстракцією в теорії алгоритмів*,* що дозволяє описувати шляхи зміни стану об’єкта в залежності від його поточного стану і вхідних даних*,* за умови*,* що загальна можливу кількість станів скінчене. У процесі роботи скінченого автомату відбувається послідовна зміна скінченого числа його внутрішніх станів*,* причому стан автомата в певний момент часу однозначно визначається вхідним і вихідним сигналами. Такі автомати являють собою основу всієї сучасної обчислювальної техніки і всіляких дискретних систем автоматичного контролю і управління.  На даний час скінчені автомати – це один з основних інструментів побудови систем тестування. | |
|  | *44* |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| РОЗДІЛ 3  ОПИСАННЯ РОБОТИ СИСТЕМИ ВІДСЛІДКОВУВАННЯ ПОМИЛОК В ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ *WEB*-ДОДАТКІВ  3.1. Логіка роботи додатку  Додаток повинен реалізувати наступну функціональність для користувача:   * для кожного з зареєстрованих користувачів вести облік проектів для тестування; * користувач повинен мати змогу будувати діаграму станів і переходів для продукту*,* що необхідно перевірити; * користувач запускає додаток*,* надаючи йому на вхід множину ребер графу переходів; * система повинна будувати усі шляхи графа (множину тестових випадків) та тестові набори*,* користуючись заданим критерієм покриття; * система повинна передбачати генерування випадкової послідовності взаємодій з додатком*,* що тестується; * користувач співставляє *id* станів їх описанню на природній мові*,* отримуючи таким чином тестові випадки для перевірки цільової системи.   Розроблюваний пакет має складатися з декількох модулів*,* що містять по кілька функціональних тестів. У підсумку пакет буде покривати весь базовий функціонал програмного забезпечення сайту.  На етапі моделювання системи проведемо аналіз основних видів її використання. На основі видів діяльності (рис. 3.1)*,* побудуємо послідовність подій (вхід до системи (рис. 3.2) та тестування *web*-додатків (рис. 3.3)). | | | | | | | | | |
| Кафедра КСУ | | | | НАУ ввввввв 000 ПЗ | | | | | |
| **Виконав** |  |  |  | Описання роботи системи відслідковування помилок в програмному забезпеченні *web*-додатків | **Літера** | | | **Аркуш** | **Аркушів** |
| **Керівник** |  |  |  | Д |  |  | *45* | *61* |
| **Консульт*.*** |  |  |  | СП 4….. 123 | | | | |
| **Норм*.* контр*.*** |  |  |  |
| **Зав*.* Каф*.*** |  |  |  |

|  |  |
| --- | --- |
| Введення імені і паролю  Перевірка наявності імені  Інакше  Реєстрація  Виведення списку поточних проектів  Генерація тестового набору  Запуск тесту на основі отриманого тестового  Підтвердження результатів тестування  Інакше  Формування звіту про тестування    Рис. 3.1. Діагарма основних видів діяльності при тестуванні програмного забезпечення в системі  Форма входу Модуль Інтерфейс Користувач в систему ідентифікації користувача   1. Введеня імені і паролю 2. Передача даних на 3. Повідомлення про помилку 4. Відкриття проектів та   історії дій користувача  Рис. 3.2. Діагарма послідовності дій для входу користувача в систему | |
|  | *46* |



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
| Користувач | | | | Меню користувача | | | Форма вибору Функція Модуль  проектів генерації тесту проведення |  |
|  | | | |  |  | |  | |
|  | | | 1. Вибір елементу ме | | ню "Проекти тестування" | | | |
|  | |  |  |  |  | 1. Виклик форми 2. Формування тестової послідовності 3. Виконання тестування 4. Вимога до   повторної генерації тестового набору  проходження тестування | | |
|  | |  |  |  |  |
|  | | |  | |  |
| 6. Звіт | | з |
|  | |  |  |  |  | | | |
|  | |  |  |
| Рис. 3.3. Діаграма послідовності дій для тестування програмного забезпечення  Усі модулі пакета є незалежними. Результат тестування*,* як і самі модулі знаходяться в єдиному файлі з розширенням *Trace*. Тести*,* що відповідають модулям пакета представлені у табл. 3.1.  Таблиця 3.1  Відповідність тестів модулям пакета | | | | | | | | |
|  | *Test*\_*Func* | | | | | Набор функцій*,* що використовуються в модулях пакета | |  |
| *Test\_App* | | | | | Запуск додатку*,* Меню Опцій*,* Меню Колекції*,* Головний  Екран | |
| *Test\_Collect* | | | | | Повний тест функціоналу меню Колекції | |
| *Test\_Diff* | | | | | Тест розбиття зображення по Важкості (Використання  кнопок) | |
| *Test\_Help* | | | | | Повний тест функціоналу Екрану Допомоги | |
| *Test\_Main* | | | | | Повний тест функціоналу Головного Екрану | |
| *Test\_Picture*\_*Contr* | | | | | Тестування переключення зображень кнопками  Наступна*,* Попередня | |
| *Test\_Susp* | | | | | Додатковий тест для визначення поведінки сайту в  момент згортання | |
| *Test\_Orient* | | | | | Додатковий тест для визначення поведінки сайту в момент зміни положення пристрою (для мобільних  браузрів) | |
|  |  | | | | | | |  |
|  | | | | | | | | *47* |



|  |  |
| --- | --- |
| * 1. Структура програмного додатку      1. Класи програмного додатку   Клас *graph\_test\_coverage* являє собою абстракцію покриття модельного графу для формування дерева варіантів . Він містить контейнер з ребрами та відповідним кожному ребру булевою змінною*,* що містить інформацію про те*,* чи покрите ребро чи ні. Об’єкт класу *graph\_coverage* використовується під час побудови тестового набору.  *#include < graph\_test\_coverage.h>*  Реалізовані наступні методи класу:  *graph\_test\_coverage <Graph>(const Graph & graph)*  Конструктор класу*,* створює об’єкт – абстракцію покриття графу і ініціалізує значення покриття кожного ребра графу *graph* як *FALSE* (непокрите).  *aditional\_coverage\_for\_path(std::list<size\_t> path) const*  Функція повертає додатне число типу *size*\_*t,* якщо шлях *path*покриває хоча б одне досі непокрите ребро*,* і 0 – у протилежному випадку.  *fully\_covered() const*  Функція повертає булеве значення. *TRUE,* якщо кожне ребро модельного графу покрито шляхами(тестами)*,* що увійшли до результуючого набору*,* і *FALSE*  – у протилежному випадку.  *add\_coverage\_for\_path(const std::list<size\_t>& path)*  Функція помічає всі вершини*,* що зустрічаються у шляху *path* як покриті.  Об’єкт класу *test\_edge*\_*id* містить унікальне описання ребра графу*,* яким слугують його початого і цільова вершини.  *#include " test\_edge.h"*  *test\_edge\_id(size\_t vertex\_source, size\_t vertex\_target)*  Конструктор класу*,* ініціалізує *m\_vertex\_source* та *m\_vertex\_target*  значеннями переданих параметрів.  Клас *Graph* є синонімом класу *boost::adjacency\_list* і використовується для представлення модельного графу. | |
|  | *48* |

|  |  |
| --- | --- |
| *#include <boost/graph/adjacency\_list.hpp>*  Наступні методи із тих*,* що пропонує клас *boost::adjacency\_list* було використано під час реалізації додатку:  *out\_edges(vertex\_descriptor u, const adjacency\_list& g)*  Повертає пару літераторів*,* що ітерує діапазон ребер*,* що виходять вершини  *U* графу *G*.  *target(edge\_descriptor e, const adjacency\_list& g)*  Функція повертає значення типу *vertex\_descriptor,* що є значенням кінцевої вершини ребра *e* графу *g.*  *num\_vertices(const adjacency\_list& g)*  Функція повертає кількість вершин графу *g. vertex(vertices\_size\_type n, const adjacency\_list& g)* Функція повертає *n-*у вершину графу *g*.  3.2.2. Методи  Основні функціъ*,* що використовують при розв’язанні задачі:  *read\_values\_in\_set<T>(std::istream & input, size\_t values\_count, std::set<T>\* result)*  Функція*,* що читає із вхідного потоку *input, values\_count* кількість ребер*,* що представлені парами вершин і зберігає їх у контейнер *result.*  *output\_test(const std::list<size\_t>& path\_accumulator, std::ostream & output)*  Функція*,* що виводить у вихідний потік *output* шлях на графі(тест)*,* що міститься у *path\_accumulator.*  *save\_to\_collection(const std::list<size\_t>& path, std::list<std::list<size\_t>>& pathes\_collection)*  Функція поміщає шлях *path* у *pathes\_collection* – фінальний набір тестів.  Функція*,* що реалізує алгоритм побудови всіх маршрутів на графі із заданим обмеженням (відвідування однієї вершини не більше двох разів). Результат поміщається у *pathes\_collection.* Функція рекурсивна.  Параметрами виступають модельний граф *graph,* поточна вершина  *current\_vertex,* множина кінцевих вершин графу *final\_vertices,* контейнер для | |
|  | *49* |

|  |  |
| --- | --- |
| шляху*,* що будується *path\_accumulator, visits\_count\_vector –* вектор*,* в якому міститься інформація про кількість відвідування поточної вершини*, pathes\_collection –* тестовий набір*,* контейнер*,* в який буде поміщено *path\_accumulator,* коли буде досягнуто одну з вершин множини *final\_vertices.*  *read\_case(Graph & states\_graph, std::set<size\_t>& final\_states)*  Функція*,* що відповідає за отримання усіх вхідних даних для генерації тестового набору*,* а саме – значення кількості вершин модельного графу*,* значення кількості кінцевих вершин*,* множина кінцевих вершин і множина ребер графу.  *generate\_all\_tests(const Graph & states\_graph, const std::set<size\_t>& final\_states, std::list<std::list<size\_t>>& tests\_collection)*  Функція*,* яка через виклик функції *trace\_graph(),* будує усі шляхи на графі.  *choose\_best\_tests(const size\_t M, const double P, const Graph& states\_graph, const size\_t states\_count, std::list<std::list<size\_t> >& tests\_collection, std::list<std::list<size\_t> >& best\_tests)*  Функція*,* що реалізує алгоритм побудови тестового набору. Вхідні параметри: *M –* максимальна довжина переходів у тесті*, P* – вага допустимого перевищення довжини тесту на користь покриття*, states\_graph –* модельний граф*, states\_count –* кількість станів модельного графу *states\_graph, tests\_collection –* повний набір тестів*,* що побудовано у результаті роботи функції *generate\_all\_tests(),best\_tests –* контейнер*,* що містить результуючу тестову послідовність.   * 1. Інтерфейс додатку та параметри запуску      1. Модуль тестування сценаріїв   Розробка тестів*,* заснованих на використанні сценаріїв*,* здійснюється за такою методикою:  – визначається модель використання*,* що включає операційне оточення продукту і "акторів". Актором може бути користувач*,* інший продукт*,* апаратна | |
|  | *50* |

|  |  |
| --- | --- |
| частина тощо*,* тобто все*,* з чим продукт обмінюється інформацією. Поділ на оточення і акторів умовно і служить для опису оптимальних способів використання продукту;   * розробляються сценарії використання продукту. Опис сценарію в залежності від продукту і обраного підходу може бути строго певним*,* параметризованим або дозволяти певний рівень невизначеності; * розробляється набір тестів*,* що покривають задані сценарії. З урахуванням ступеня невизначеності*,* закладеної в сценарії*,* кожен тест може покривати один сценарій*,* кілька сценаріїв*,* або*,* навпаки*,* частина сценарію.   Використання сценаріїв не вимагає наявності повної формальної специфікації вимог*,* але зате може зажадати більше часу на розробку і аналіз.  Для кожного проекту можна обрати окремий сценарій тестування (рис.  3.4)*,* що дозволяє уніфікувати систему.  Рис. 3.4. Вікно з переліком активних проектів  Ще одна особливість тестування сценаріїв полягає в тому*,* що цей метод направляє тестування на перевірку конкретних режимів використання продукту*,* що дозволяє знаходити дефекти*,* які метод тестування за вимогами може пропустити.  3.3.2. Модуль генераціъ тестів  Тест формалізований у наступному вигляді (рис. 3.5). Тут кожна стрілка з позначкою *DoTransaction, DoAddressTenure* або *DoDataTenure* представлет собою | |
|  | *51* |

|  |  |
| --- | --- |
| виклик відповідної функції продукту з передачею параметрів. Стрілка *checkTr* відповідає перевірці проходження по шині транзакції з відповідними параметрами. Кожна зі стрілок діаграми генератором тестів перетворюється в здійсненний код*,* при цьому стрільцям*,* які представляють собою виклики функцій може відповідати досить простий і невелику ділянку коду*,* що викликає відповідну функцію і перевіряючий її вихідне значення на наявність помилок.  Рис. 3.5. Формальний запис сценарного тесту  Слід зазначити*,* що стрілки*,* відповідні перевірки транзакцій*,* можуть після генерації перетворитися в досить складний код*,* який буде виконувати очікування появи транзакції на шині протягом заданого при генерації часу - тайм-ауту*,* перевіряти фази транзакції і звіряти обчислені значення параметрів із заданими еталонними значеннями.  В результаті в розглянутому прикладі виграш від застосування генераційні підходу досягається в основному за рахунок використання наочного візуального представлення тестів*,* що може бути нівельовано витратами на створення генераційні сценарію на *MSC*.  Можлива куди більш ефективна формалізація сценарію для генерації тестів (рис. 3.6)*,* яка представляє інший спосіб формалізації тесту*,* що виконує ті ж самі перевірки.  На рис. 3.6 перевірки транзакцій згруповані з породжують їх викликами в | |
|  | *52* |

|  |  |
| --- | --- |
| окремі фрагменти*,* а паралелізм*,* який використовується при виконанні фрагментів*,* заданий через *Par* - формальну конструкцію*,* яка застосовується для зображення паралелізму.  Рис. 3.6. Формальний запис сценарного тесту з використанням паралелізму При генерації тестів по діаграмі рис. 3.6 тестовий генератор перебирає всі  можливі і не повторюються варіанти виклику тестованих функцій*,* зберігаючи при цьому коректність порядку перевірок*,* що в даному прикладі дає три згенерованих тесту. Нескладно бачити*,* що витрати на створення діаграми.  Для успішного розвитку технологій тестування на основі моделей потрібно одночасно підтримувати їх еволюцію в трьох напрямках: нарощувати функціональність технологій*,* нарощувати спряженість цих технологій з сучасними процесами розробки і нарощувати зручність їх використання. Особливо важливо збільшувати зручність використання технологій для вирішення найбільш часто зустрічаються завдань.  3.3.3. Описання інтерфейсу системи  Наша Система дозволяє:  – зберігати інформацію про помилку в наступному виді: хто повідомив про проблему*,* дата і час*,* коли була виявлена проблема; серйозність проблеми; опис | |
|  | *53* |

|  |  |
| --- | --- |
| неправильної поведінки програми; стан помилки (рис. 3.7).  Рис. 3.7. Вікно з описанням помилки в роботі програмного забезпечення   * дозволяє менеджеру проектів прийняти помилку для подальшого усунення та відіслати программісту для її усунення; * надає можливість відсортувати баг за серйозністью*, id,* тегами (рис. 3.8)*,* а також провести пошук конкретної помилки (рис. 3.9); * за всіма тестами можна сформувати звітність та передивитись статистику (рис. 3.10).   Рис. 3.8. Вікно з списком знайдених помилок в роботі програмного забезпечення | |
|  | *54* |

|  |  |
| --- | --- |
| Рис. 3.6. Вікно знайдених помилок з інструментарієм пошуку  Рис. 3.10. Вікно історії проведення тестів  3.4. Висновки до розділу  Побудовано *web*-додаток*,* що реалізує підхід до генерації тестів і проведені експерименти з його використанням.  Заявлений підхід налічує наступні переваги:   * гнучкість; * фіксація максимальної кількості кроків тесту; | |
|  | *55* |

* недетермінованість;
* мінімізація розміру тестового набору;
* гарантованість досягнення встановленого рівні покриття;
* статичне тестування вимог.

*56*

|  |  |
| --- | --- |
| 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 викриють помилки 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 | |
|  | *57* |

|  |  |
| --- | --- |
| 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 | |
|  | *58* |

|  |  |
| --- | --- |
| 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. | |
|  | *59* |

|  |  |
| --- | --- |
| СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ   1. *IP VPNs for Service Providers*: *The Foundation for Profitable* [*http*://*www*.*cisco*.*com*/.](http://www.cisco.com/) 2. *Beazley D*. *Python essential reference* // *BHV*. – 2015. – 734 *c*. 3. *Jones C*.*A*.*, Drake F*.*L*. *Python* & *XML* // *BHV*. – 2014. – 807 *c*. 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. | |
|  | *60* |

|  |  |
| --- | --- |
| 1. Фаулер*,* Мартин – Архитектура корпоративных программных приложений.: Пер. с англ. – М.: Издательский дом "Вильяме"*,* 2006. – 544 с. 2. Хахаев И.А. Практикум по алгоритмизации и программированию на   *Python*. – М.: Альт Линукс*,* 2010. — 126 с.   1. Шаньгин В. Ф. Защита компьютерной информации*,* М: ДМК. 2008 – 544с. 2. Бойченко С.В.*,* Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ*,* 2017. – 63 с. | |
|  | *61* |

|  |  |
| --- | --- |
| 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); | |
|  | *62* |

|  |  |
| --- | --- |
| *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); | |
|  | *63* |

|  |  |
| --- | --- |
| *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* (); | |
|  | *64* |

|  |  |
| --- | --- |
| *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); | |
|  | *65* |

|  |  |
| --- | --- |
| *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* (); | |
|  | *66* |

|  |  |
| --- | --- |
| *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); | |
|  | *67* |

|  |  |
| --- | --- |
| *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); | |
|  | *68* |

|  |  |
| --- | --- |
| *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); | |
|  | *69* |

|  |  |
| --- | --- |
| *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); | |
|  | *70* |

|  |  |
| --- | --- |
| *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* | |
|  | *71* |

|  |  |
| --- | --- |
| *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*"); | |
|  | *72* |

|  |  |
| --- | --- |
| *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* (); | |
|  | *73* |

|  |  |
| --- | --- |
| *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*)  { | |
|  | *74* |

|  |  |
| --- | --- |
| *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*); | |
|  | *75* |

|  |  |
| --- | --- |
| }  *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* ()  { | |
|  | *76* |

|  |  |
| --- | --- |
| *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*) | |
|  | *77* |

|  |  |
| --- | --- |
| {  *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); | |
|  | *78* |

|  |  |
| --- | --- |
| *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: | |
|  | *79* |

|  |  |
| --- | --- |
| *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* | |
|  | *80* |

|  |  |
| --- | --- |
| *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* (); | |
|  | *81* |

|  |  |
| --- | --- |
| *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);  }  } | |
|  | *82* |