Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

УДК 004.93 *На правах рукопису*

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: “ Розробка програмної технології візуального супроводження об’єктів для систем реального часу ”

Спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ – 65.00.00.000

Студент
ІПЗ-44 МС _____ /Микола МОРОЗ/

Науковий керівник
к.т.н., ас. _____ /Денис БЕРЕСТОВ/

Консультант
з питань нормоконтролю
фахівець _____ /Тамара ЧАПОВСЬКА/

Допускається до захисту
Завідувач кафедри
д.т.н., проф. _____ /Олексій БИЧКОВ/

Київ – 2021

Рішенням Екзаменаційної комісії
випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії

професор, доктор техн. наук Андрій БОНДАРЧУК

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО:
Завідувач кафедри програмних систем і технологій
_____ (Олексій БИЧКОВ)
„___”_____ 2021 р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ
СТУДЕНТУ

Морозу Миколі Валерійовичу
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної бакалаврської роботи “Розробка програмної технології візуального супроводження об’єктів для систем реального часу”, керівник роботи Берестов Денис Сергійович, к.т.н, асистент затверджені наказом вищого навчального закладу від „11” листопада 2020 р. № 6.

2. Строк здачі студентом закінченої роботи _____

3. Вихідні дані до роботи теоретична складова процесу візуального супроводження об’єктів, алгоритми візуального відстежування об’єктів

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)

1. Аналіз і огляд предметної галузі

2. Проектування системи

3. Особливості реалізації системи

4. Опис здобутих результатів

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) _____

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
РОЗДІЛ 1	Денис БЕРЕСТОВ		
РОЗДІЛ 2	Денис БЕРЕСТОВ		
РОЗДІЛ 3	Денис БЕРЕСТОВ		
РОЗДІЛ 4	Денис БЕРЕСТОВ		

7. Дата видачі завдання _____

Керівник _____ (Денис БЕРЕСТОВ)

Завдання прийняв до виконання _____ (Микола МОРОЗ)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1	Огляд процесу візуального супроводу	10.11.2020	виконано
2	Вибір алгоритму супроводження	24.12.2020	виконано
3	Реалізація алгоритму відстеження	16.01.2021	виконано
4	Інтеграція алгоритму відстеження до вбудованої системи	05.02.2021	виконано
5	Оптимізація рішення	20.04.2021	виконано
6	Оцінка швидкодії системи	03.05.2021	виконано
7	Оформлення пояснювальної записки	06.06.2021	виконано

Студент – бакалавр _____ (Микола МОРОЗ)

Керівник роботи _____ (Денис БЕРЕСТОВ)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 72 с., 11 рис., 3 табл., 1 додат., 24 джерела.

Тема: Розробка програмної технології візуального супроводження об’єктів для систем реального часу

Об’єкт дослідження: візуальне супроводження об’єктів на відеопослідовностях.

Мета роботи: огляд та аналіз наявних алгоритмів візуального супроводження об’єктів на відеопослідовностях; проектування та розробка комплексу програмного та апаратного забезпечення для автоматизованого візуального супроводу об’єкта у відеопотоці в режимі реального часу для вбудованих систем.

Предмет дослідження: розробка системи для візуального супроводження об’єктів в режимі реального часу для вбудованих систем.

Результати дослідження:

Проведено огляд наявних алгоритмів супроводження та обґрунтовано алгоритм для розробки рішення. Оптимізовано алгоритм для забезпечення роботи вбудованої системи на платформах з мікроархітектурою ARM, розроблено графічний інтерфейс користувача. Проведено оцінювання швидкодії роботи системи.

Висновок

В результаті дослідження було розроблено програмну технологію для забезпечення автоматизованого супроводу візуальних об’єктів, яка здатна супроводжувати ціль зі швидкістю 27 кадрів на секунду.

АВТОМАТИЗОВАНИЙ СУПРОВІД ВІЗУАЛЬНОГО ОБ’ЄКТА, ВБУДОВАНІ СИСТЕМИ, КОМП’ЮТЕРНИЙ ЗІР, РЕАЛЬНИЙ ЧАС, ФІЛЬТР КОРЕЛЯЦІЇ

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 72 с., 11 рис., 3 табл., 1 прил., 24 источников.

Тема: Разработка программной технологии визуального сопровождения объектов для систем реального времени

Объект исследования: визуальное сопровождение объектов на видеопоследовательностях.

Цель работы: обзор и анализ существующих алгоритмов визуального сопровождения объектов на видеопоследовательностях; проектирование и разработка комплекса программного и аппаратного обеспечения для автоматизированного визуального сопровождения объекта в видеопотоке в режиме реального времени для встроенных систем.

Предмет исследования: разработка системы для визуального сопровождения объектов в режиме реального времени для встроенных систем.

Результаты исследования:

Проведён обзор существующих алгоритмов сопровождения и обосновано алгоритм для разработки решения. Алгоритм был оптимизирован для обеспечения работы встроенной системы на платформах с микроархитектурой ARM, был разработан графический интерфейс пользователя. Была проведена оценка быстродействия работы системы.

Вывод

В результате исследования было разработано программную технологию для обеспечения автоматизированного сопровождения визуальных объектов, которая способна сопровождать цель со скоростью 27 кадров в секунду.

АВТОМАТИЗИРОВАННОЕ СОПРОВОЖДЕНИЕ ВИЗУАЛЬНОГО ОБЪЕКТА, ВСТРОЕННЫЕ СИСТЕМЫ, КОМПЬТЕРНОЕ ЗРЕНИЕ, РЕАЛЬНОЕ ВРЕМЯ, ФИЛЬТР КОРЕЛЯЦИИ

ANNOTATION

Graduation qualifying bachelor's thesis: 72 pages, 11 images, 3 tables, 1 application, 24 sources.

Topic: Development of visual object tracking software technology for real-time systems

Object of study: visual object tracking in video sequences

The goal of the work: review and analyze existing algorithms for visual object tracking in video sequences; design and development of a software and hardware complex for automated real-time visual object tracking in a video stream for embedded systems.

Subject of study: development of a system for a real-time visual object tracking for embedded systems.

Results of the research:

A review of existing tracking algorithms was made and the algorithm for the development of the solution was justified. The algorithm has been optimized to support execution on an embedded system on platforms with the ARM microarchitecture, a graphical user interface has been developed. The performance of the system has been assessed.

Conclusion

As a result of the research, the program technology for automated visual object tracking was developed. The system is capable of tracking a target with the speed of 27 frames per second.

AUTOMATED VISUAL OBJECT TRACKING, EMBEDDED SYSTEMS, COMPUTER VISION, REAL-TIME, CORRELATION FILTER

ЗМІСТ

РОЗДІЛ 4

ОПИС ЗДОБУТИХ РЕЗУЛЬТАТІВ

ВСТУП

Актуальність роботи

Супроводження (відстеження, трекінг) об’єктів на сьогоднішній день є досить важливим і актуальним завданням у галузі комп’ютерного зору. Відстеження об’єктів має багато практичних застосувань [1], серед яких відеоспостереження, взаємодія між людиною та комп’ютером, навігація роботів, різноманітні системи спостереження та безпеки, моніторинг трафіку дорожнього руху, розпізнавання діяльності та інші.

Відстеження об’єктів націлене на те, щоб надати комп’ютеру можливість отримувати кращу модель реального світу. Оскільки комп’ютери за допомогою додаткових пристроїв не можуть бачити навколишнє середовище так сам як це робить людина, то відстеження об’єктів сприяє наближенню комп’ютерного сприйняття дійсності до людського, тим самим покращуючи взаємодію між людиною і комп’ютером.

Мета і задачі дослідження

Метою роботи є огляд та аналіз наявних алгоритмів візуального супроводження об’єктів на відеопослідовностях, а також проектування та розробка комплексу програмного та апаратного забезпечення для автоматизованого візуального супроводу об’єкта у відеопотоці в режимі реального часу для вбудованих систем.

Досягнення мети включало розв’язання таких задач:

1) огляд проблеми візуального супроводу об’єктів;

2) огляд наявних алгоритмів візуального відстежування об’єктів;

3) вибір алгоритму, який задовольнятиме вимогам швидкодії та обмеженості апаратних ресурсів;

4) реалізація алгоритму;

5) інтеграція алгоритму до вбудованої системи;

6) програмна оптимізація системи;

7) тестування швидкодії системи;

Об’єктом дослідження є візуальне супроводження об’єктів на відеопослідовностях.

Предметом дослідження є розробка системи для візуального супроводження об’єктів в режимі реального часу для вбудованих систем.

Практичне значення одержаних результатів

Розроблена система може бути використана для автоматизованого супроводу за об’єктом. Подібні системи використовуються у військовій галузі для стабілізації прицілу зброї та ведення більш ефективного вогневого враження противника. За умови доповнення розробленого рішення алгоритмом розпізнавання бажаних цільових об’єктів, рішення може перетворитись у автоматичний відстежувач цілей

Особистий внесок студента

Основним результатом є:

1. оптимізація обраного алгоритму супроводження для роботи в умовах обмежених обчислювальних можливостей на вбудованих системах на платформі з мікроархітектурою ARM;
2. розробка комплексу програмного та апаратного забезпечення для автоматизованого відстежування візуальних об’єктів за допомогою Raspberry Pi 3B, камери та механізму повороту-нахилу.

Апробація результатів випускної кваліфікаційної бакалаврської роботи

Результати кваліфікаційної роботи були представлені на 8-й Східно-Європейській конференції «Математичні та програмні технології Internet of Everything», що відбулася в м. Києві 14 травня 2021 р.

Публікації

За результатами досліджень та розробок, проведених у кваліфікаційній роботі, підготовлено доповідь для участі в 8-й Східно-Європейській конференції «Математичні та програмні технології Internet of Everything», що відбулася в м. Києві 14 травня 2021 р.

РОЗДІЛ 1
АНАЛІЗ ТА ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ

1.1. Огляд вимог до відстежувача

Завданням комплексу апаратних та програмних компонентів відстежувача є автоматизований супровід візуального об’єкта у відео потоці в режимі реального часу. Відстежувач також повинен мати програмний графічний інтерфейс користувача для цілевказівки та проведення супроводу.

Спосіб взаємодії користувача з відстежувачем такий. Користувачу транслюється зображення з камери в режимі реального часу. Користувач повинен мати можливість у відео потоці в будь-який момент часу виділити об’єкт, за яким він бажає проводити візуальний супровід, обравши прямокутну область, в якій розташовується об’єкт. Спостереження проводиться лише за одним об’єктом у певний момент часу. Після цього в користувача також має бути можливість припинити стеження за обраним об’єктом.

Після отримання вказівки на супроводження відбувається пошук вказаного об’єкта на наступних кадрах і виділення цього об’єкта кольоровою прямокутною рамкою. Супровід за об’єктом продовжується до моменту примусової зупинки стеження. Такий випадок може виникнути через переривання, отримане від користувача, або ж в разі неможливості продовжувати стеження.

Ключовими вимогами до відстежувача є:

- цільовий об’єкт може мати будь-які розміри, форми та колір, оскільки завчасно невідомо, який об’єкт необхідно буде супроводжувати;
- цільовий об’єкт може не мати певної визначеної траєкторії руху, про яку можна знати завчасно;
- цільовий об’єкт може перебувати на будь-якому фоні, у будь-якому візуальному середовищі;

- алгоритм повинен бути швидким для можливості відстеження в режимі реального часу, тобто програма має відображати опрацьовані кадри зі швидкістю 20 кадрів у секунду та більше.

1.2. Аналіз предметної галузі

З описаних вимог можна побудувати такі похідні твердження, завдяки яким можна буде обрати найбільш відповідний для заданого завдання алгоритм супроводу:

- стеження відбувається лише за одним об’єктом у певний момент часу;
- об’єкт для спостереження задається під час роботи, тобто вибір та ініціалізація цілі відбувається на певному кадрі і всі дані про об’єкт можна буде дістати лише з цього кадру;
- під час спостереження можна буде використовувати дані про об’єкт лише з попередніх кадрів, оскільки не буде можливості зазирнути в майбутні кадри;
- оскільки завчасно не відомо про особливості об’єктів, за якими буде відбуватися спостереження, їхній фон та навколишнє оточення, то не можна ґрунтувати розроблення алгоритму на певних характеристиках, особливостях та обмеженнях, які можуть бути притаманні об’єктам (наприклад, для відстеження м’яча у футболі головним припущенням було б те, що м’яч круглий і має певний колір, що значно спростило б завдання);
- алгоритм повинен бути достатньо швидким, оскільки важлива робота в реальному часі, а дані будуть надходити безпосередньо з камери.

Визначивши ключові вимоги можна приступати до огляду наявних джерел. У джерелі [2] у процесі візуального супроводу цільового об’єкта виділяють 4 важливих компоненти:

а) ініціалізацію цілі;

б) модель подання цілі на онові її особливостей;

в) прогноз руху;

г) пошук та отримання положення цілі.

Далі кожний із компонентів буде описано більш детально.

Ініціалізація цілі полягає у виборі позиції об’єкта або певної зони навколо нього за допомогою таких допоміжних візуальних елементів як: обмежувальна прямокутна зона, еліпс, центроїд, скелет фігури, контур, силует та інших. Згідно з заданими вимогами для такого відстежувача ініціалізація відбуватиметься в ручному режимі за допомогою виділення обмежувальної прямокутної зони.

Модель подання цілі зазвичай складається з певних візуальних особливостей, які притаманні об’єкту. Чим більш унікальними будуть особливості цілі, тим більш ефективним та точним буде виявлення. Вибір особливостей візуального представлення цільового об’єкта є важливим завданням, оскільки від них залежатиме ефективність і якість супроводу. Оскільки у вимогах немає конкретних вказівок щодо зовнішнього вигляду та візуальних особливостей цілі, то й до подання моделі цілі також немає конкретних вимог. У такому випадку необхідно, щоби модель подання була універсальною в тому сенсі, що вона має бути отримана з вказаної зони кадру й не має передбачити інших знань про об’єкт.

Під час прогнозування руху в наступних кадрах оцінюється найбільш імовірне нове розташування цілі. На цьому етапі, зазвичай, використовують модель руху об’єкта. Це корисно у випадках, коли завчасно відомо про можливий рух цілі. Модель руху об’єкта може бути строго детермінованою, тобто, знаючи початкову точку руху, можна точно спрогнозувати розташування шуканого об’єкта в наступних кадрах. Також модель руху може мати локальну пам’ять, наприклад, якщо в декількох попередніх кадрах об’єкт рухався в одному напрямку, то зі значною вірогідністю, можна буде припустити, що в поточному кадрі тенденція руху в такому ж напрямку збережеться. Проте у випадках, коли рух цілі не детермінований або може стрімко змінюватись, використання моделі руху може бути надлишковим або повністю зайвим. Згідно з заданими вимогами, цільовий об’єкт не має завчасно відомого напрямку і характеру руху, тому для даного відстежувача модель руху може бути відсутньою або ж мати локальну пам’ять.

Під час пошуку цілі відбувається локалізація шуканого об’єкта на кадрі. За наявності прогнозування руху етап пошуку може залежати від його результату. У деяких випадках така взаємодія сприяє значному підвищенню продуктивності. Під час пошуку цілі, зазвичай, використовуються різноманітні спеціалізовані детектори (розпізнавачі) цілей.

У джерелі [3] також наголошується на важливості гарної моделі подання цільового об’єкта і виділяються такі основні візуальні особливості, на яких може ґрунтуватись відстежування об’єкту:

а) колір;

б) контури об’єктів;

в) оптичний потік;

г) текстура.

Далі кожна із особливостей буде описано більш детально.

Зазвичай колір представлено трьома каналами кольорів RGB – червоного, зеленого та синього. У загальному, об’єкти не часто змінюють власний колір, тому можна вважати колір досить властивою характеристикою. Проте варто також брати до уваги, що колір об’єкта може й не змінюватись, але сприйняття об’єкта залежить від освітленості. Також на колір об’єкта може впливати його власна здатність до відбиття або поглинання видимого світла.

Об’єкти мають контури й внаслідок різниці кольорів між об’єктом та його фоном можна визначити межу переходу кольорів, тобто знайти зміну в інтенсивності кольору. На відміну від особливостей, які ґрунтуються на кольорі об’єкта, контури є менш чутливими до змін освітлення, адже вони не прив’язуються до конкретних кольорів, а залежать від різких переходів значень кольору з одного в інший. Проте головним недоліком контурів є ситуації, коли кольори є неоднорідними та коли колір об’єкта зливається з фоном.

Оптичний потік – це шаблон руху об’єктів на зображенні між двома послідовними кадрами, викликаний рухом об’єкта чи камери. Він представлений двовимірним векторним полем, де кожен вектор є вектором переміщення, який показує рух точок від першого кадру до другого. Завдяки оптичному потоку можна

здобути дані щодо руху та переміщення кожного з пікселів між послідовними кадрами. Обчислення оптичного потоку відбувається через введення додаткових обмежень на яскравість кадрів, тобто яскравість відповідних пікселів у послідовних кадрах вважається сталою, завдяки чому і відбувається пошук відповідності. Даний вид особливостей зазвичай використовується у відстеженні та сегментації на основі руху.

Текстуру можна описати як міру зміни інтенсивності поверхні, завдяки якій можна кількісно визначити такі показники, як плавність та регулярність або повторюваність. Отримання текстури вимагає додаткового обчислювального кроку, призначення якого полягає в обробці зображення для створення спеціальних дескрипторів. Текстура, як і краї, також є менш чутливою до змін значень освітленості в порівнянні з особливостями на основі кольорів.

Ці особливості є основними, проте є і більш комплексні та специфічні особливості. Особливості об’єктів можуть бути поєднані між собою для досягнення більш чітких та надійних результатів, а можуть, навпаки, використовуватись окремо, наприклад, для забезпечення більшої швидкості відстежування або для супроводу на основі специфічних особливостей об’єктів чи їхнього навколишнього оточення. Вибір візуальних особливостей для відстеження напрямку впливає на етап розпізнавання або пошуку об’єкту. У джерелі [1] представлено варіанти деяких підходів до розпізнавання, серед яких:

а) точкові розпізнавачі;

б) видалення фону;

в) сегментація;

г) контрольоване навчання або навчання під наглядом;

ґ) метод тимчасової різниці.

Виявлення на основі точкових розпізнавачів полягає в знаходженні особливих точок на послідовностях кадрів. Особлива точка – це невеликий регіон, який належить об’єкту і який є досить виразним, завдяки чому об’єкт може бути співставлений на суміжних кадрах на основі цих точок. Особлива точка має бути нечутливою до змін освітленості на зображенні та доступною з різних сторін під час

переміщення камери та зміни ракурсу об’єкта, що не завжди може бути можливим. Зазвичай, в якості особливих точок використовують кути. Також особливими точками можуть слугувати кінці ліній і точки екстремумів інтенсивності.

Розпізнавання на основі видалення фону ґрунтується на отриманні об’єкта завдяки відділенню об'єкта переднього плану від його фону. Для цього, зазвичай, будують модель фону. Завдяки цьому у кожному кадрі проводиться порівняння з моделлю й через різниці значень у певних зонах можна визначати рух. Для цього фонова модель має бути еталонною, постійно оновлюватись і не містити рухомих об’єктів, адже це вплине на якість розпізнавання. Цей метод не є складним у реалізації, проте метод є дуже чутливим до змін зовнішнього середовища. Саме тому він не буде дієздатним, якщо камера буде постійно змінювати своє положення.

Сегментація зображення – це його поділ на декілька регіонів на основі однаковості певних значень, наприклад, поділ за кольорами або текстурою. Цей підхід може бути корисним для визначення меж об’єктів. Є багато реалізацій такого підходу. Такими представниками є зсуви середніх значень кольорів для пошуку кластерів або активні контури, які спрямовані на пошук меж та ліній за допомогою додаткових обмежень.

Основна ідея контрольованого навчання полягає в тому, що розпізнавання об’єкта може відбуватись завдяки навчанню на різних прикладах об’єкта з різних ракурсів та в різних формах і масштабах. Методи навчання передбачають наявність певного набору даних про вигляд об’єкта, які будуть використані як приклади для навчання, а в результаті навчання буде створено функцію, яка буде відображати вхідні значення на певні вихідні значення. Використовуючи вихідні дані, отримані від такої функції, можна буде визначати на скільки ймовірно те, що це є дійсно цільовий об’єкт. Цей процес називають класифікацією. Важливим елементом у навчанні є правильність вибору специфічних особливостей об’єктів у навчальних прикладах. До методів контрольованого навчання належать, наприклад, нейронні мережі та опорні вектори (SVM).

Метод тимчасової різниці здатний визначити попіксельні зміни між двома або трьома послідовними кадрами. Завдяки даному порівнянню можна визначати рух

об’єкта. Проте, якщо об’єкт рухається повільно, метод працює погано, оскільки він не одержує всі відповідні пікселі. Іншим недоліком методу є випадки, коли рух об'єкта припиняється. Оскільки змін між кадрами немає, то виконати пошук об’єкта також неможливо. У таких випадках можна використовувати принцип локальної пам’яті та повертати останній регіон, у якому було знайдено ціль.

Розглянуті особливості супроводжувачів підтверджують комплексність та складність цієї царини. Можуть бути різні підходи та поєднання, які будуть надавати різні переваги і вирішувати різні проблеми. Оскільки відстежування є дослідницькою діяльністю, кращим рішенням буде обрати певний алгоритм із кола доступних, адже розроблені алгоритми є надійними та чітко обґрунтованими.

1.3. Класифікація алгоритмів відстеження

Згідно з джерелом [2], алгоритми відстеження можна розділили на дві загальні категорії: відстежувачі з використанням кореляційних фільтрів (ВКФ) та відстежувачі без використання кореляційних фільтрів (ВБКФ).

Дискримінаційні фільтри кореляції активно використовуються в різних галузях комп'ютерного зору, серед яких розпізнавання об'єктів, реєстрація зображень, перевірка обличчя та розпізнавання дій. У відстеженні об'єктів кореляційні фільтри (КФ) використовуються для поліпшення надійності та ефективності.

Схеми відстеження на основі КФ виконують обчислення послуговуючись частотними характеристиками для мінімізації обчислювальної вартості. Загальна архітектура цих алгоритмів використовує підхід «відстеження за виявленням» і представлена на рис. 1.1. Фільтри кореляції ініціалізуються спеціально вказаною й обмеженою зоною, у якій розташовується ціль, на початковому кадрі. Під час подальшого відстеження місце розташування цілі в новому кадрі оцінюється завдяки використанню попереднього положення цілі. Щоб ефективно представити зовнішній вигляд цілі, застосовуються відповідні методи вилучення характеристик, які

використовуються для побудови мапи характеристик (мапи особливостей). Крайні межі зони згладжуються завдяки застосуванню косинусного фільтра. Згладжування відбувається через те, що основні характеристики цілі зосереджені ближче до центу, а межі можуть вносити додаткий шум або ж можуть містити об’єкти, які не належать цілі.

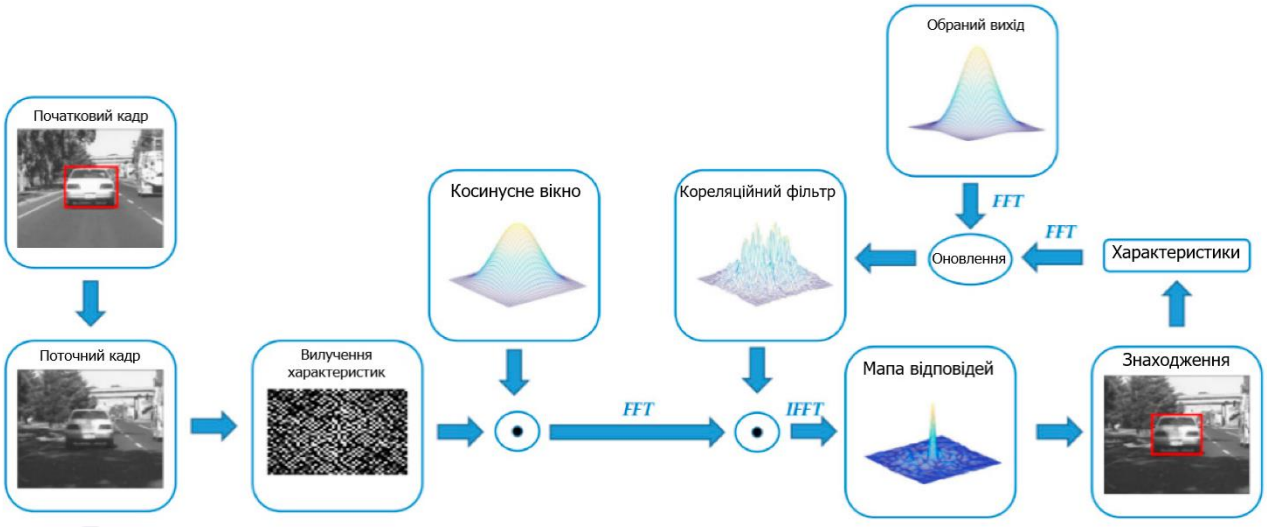


Рис. 1.1. Загальна схема алгоритмів відстеження на основі КФ

Операція кореляції виконується замість операції згортки, оскільки операція кореляції є швидшою. Мапа відповідей (результат пошуку в частотній області) обчислюється через поелементне множення між адаптивним фільтром навчання та характеристиками, вилученими за допомогою дискретної трансформації Фур'є (DFT). DFT працює з частотними даними за допомогою швидкого перетворення Фур'є (FFT). Мапа довіри (результат пошуку у просторовій області) отримується в просторовій області, застосовуючи обернену FFT (IFFT) до мапи відповідей. Максимальне значення на мапі довіри вказує на нову позицію цілі. У результаті модель цілі оновлюється завдяки вилученню характеристик із нової позиції, після чого ці характеристики використовуються для оновлення кореляційного фільтра.

Схеми відстеження на основі КФ стикаються з різними труднощами, як-от тренування зовнішнього вигляду цілі (орієнтації та форми), оскільки воно може змінюватися з часом. Іншим завданням є вибір ефективного представлення

характеристик, оскільки деякі характеристики можуть покращувати ефективність ВКФ. Іншим важливим завданням для ВКФ є адаптація масштабу, оскільки розмір кореляційних фільтрів фіксований під час відстеження. Проте ціль із часом може змінювати свій розмір залежно від дальності до неї. Крім того, якщо ціль втрачена, її неможливо відновити знову.

Згідно з цією класифікацією всі відстежувачі, які не використовують кореляційні фільтри називаються відстежувачами без використання кореляційного фільтра (ВБКФ). ВБКФ можна класифікувати на такі підкатегорії:

- відстежувачі з навчанням патчу;
- відстежувачі на основі розрідженості;
- відстежувачі на основі суперпікселів;
- відстежувачі на основі графів;
- відстежувачі на основі навчання з декількома екземплярами;
- відстежувачі на основі частин;
- відстежувачі на основі сіамських мереж.

Ці відстежувачі є дискримінаційними, за винятком відстежувачів на основі розрідженості, які є генеративними відстежувачами.

Відстежувачі з навчанням патчів використовують патчі цілі й патчі фону. Відстежувач навчається на позитивних та негативних зразках. Підготовлений відстежувач тестується на низці зразків, а максимальна відповідь відстежувача на одному з вхідних зразків вказує на положення цілі.

Під час відстеження на основі навчання декількох екземплярів замість розгляду певних патчів, як у попередньому методі, навчальні зразки розміщують у комірки й кожній комірці надається мітка. Позитивна мітка присвоюється комірці, якщо комірка містить принаймні один позитивний зразок, а негативна комірка містить лише негативні зразки. Позитивна комірка може містити позитивні та негативні екземпляри. Під час навчання мітки для екземплярів невідомі, але мітки на комірках відомі. За такого відстеження екземпляри використовуються для побудови слабких класифікаторів, а кілька екземплярів вибираються та об'єднуються для формування сильного класифікатора.

Відстежувач, що базується на сіамській мережі, здійснює відстеження на основі механізму відповідності. Процес навчання використовує загальні варіації зовнішнього вигляду цілі. Сіамські мережеві відстежувачі узгоджують шаблони цілі з кандидатськими зразками, щоб здобути схожість між патчами.

Відстежувачі на основі суперпікселей. Суперпікселі являють собою групу пікселів, що мають однакові значення. Цільова зона сегментується на декілька суперпікселів, над якими проводиться класифікація для дискримінації об’єкта.

Відстежувачі на основі графів. Граф має вершини (це можуть бути пікселі, суперпікселі або частини об'єкта) та ребра (відповідність серед вершин). Графи використовуються для прогнозування міток незазначених вершин. Як правило, відстежувачі на основі графів використовують суперпікселі як вузли для представлення зовнішнього вигляду об'єкта, а ребра представляють внутрішню геометричну структуру. Інша стратегія – побудувати графи між частинами об’єктів у різних кадрах.

Відстежувачі на основі частин. Частинне моделювання активно використовується для обробки частин, які можуть змінюватись. Локальні частини об'єкта використовуються для моделювання трекера.

Відстежувачі на основі розрідженості. Усі вивчені до цього часу алгоритми є дискримінаційними методами відстеження. З іншого боку, генеративні методи вивчають представлення цілі та пошук цілі в кожному кадрі з мінімальною помилкою відновлення. Розріджене представлення – хороший приклад для генеративних моделей. Мета – виявити оптимальне подання цілі, яке є досить розрідженим і мінімізує помилку відновлення.

1.4. Вибір алгоритму відстеження

Для вибору алгоритму відстеження необхідно сфокусуватися на особливостях цільових об’єктів, особливостях середовища, обчислювальних можливостях

пристрою та на перевагах наявних алгоритмів відстеження. Для більш зваженого рішення необхідно виконати порівняння наявних алгоритмів.

Ресурс [4] представляє спільноту «VOT Challenge», яка спрямована на створення вільнодоступного сховища наявних алгоритмів візуального відстеження. Щорічно ця спільнота проводить конкурс, у результаті якого порівнює й оцінює подані роботи. Це оцінювання спрямоване на визначення якості відстежувача на основі підготовлених та анотованих відеопослідовностей. Тестові випадки включають основні проблеми відстеження, такі як: зміни освітлення, зміни розмірів об’єкта, перекриття об’єкта та інші. Для автоматичного оцінювання якості відстежувачів використовується спеціально розроблений фреймворк. Також важливим є те, що у відкритому доступі перебувають як тестові дані із очікуваними результатами, так і результати порівняння трекерів за різними показниками.

Серед змагань «VOT Challenge» є також і змагання, які орієнтуються на відстежувачі, що здібні працювати в режимі реального часу. Здатність працювати в режимі реального часу означає, що відстежувач має повідомляти про розташування цілі на кожному кадрі з частотою, що перевищує або дорівнює частоті кадрів у відео послідовності.

Спосіб тестування відстежувачів реального часу є таким. Спочатку відбувається ініціалізація відстежувача вхідними даними, а саме – цільовою зоною, у першому кадрі послідовності за допомогою відповідного інструментарію. Після цього інструмент оцінювання очікує вихідні дані відстежувача про положення цілі. Якщо відстежувач не встигне відповісти про нове положення цілі за наданий йому інтервал часу, то як відповідь відстежувача на такому кадрі приймається остання відповідь відстежувача.

Змагання «VOT» використовують методологію, що базується на перезапусках. У випадках, коли відстежувач надає результат у вигляді зони, в якій визначено нове розташування об’єкта, і ця зона має нульове перекриття з еталонними даними, то відбувається збій, а відстежувач повторно ініціалізується через п’ять кадрів після збою. Також у «VOT» є власні основні міри для оцінювання відстежувачів.

Точність та робастність – це основні міри, що застосовуються для дослідження ефективності відстежувача в експериментах на основі перезапусків. Точність – це середнє перекриття між результатами відстежувача та еталонними даними упродовж успішних періодів супроводження. Робастність вимірює, як часто під час супроводу відстежувач втрачає ціль, унаслідок чого виконує повторну ініціалізацію. Потенційні відхилення в оцінюванні точності, які пов’язані з перезапусками, зменшуються завдяки ігноруванню десяти кадрів після повторної ініціалізації, що є досить консервативним запасом.

Третьою основною мірою оцінювання є очікуване середнє перекриття (EAO), яке поєднує значення точності в кожному кадрі та кількість перезавантажень відстежувача. Ця міра відповідає за очікуване середнє перекриття відстежувача на подібних за візуальними властивостями до такого набору даних короткотермінових послідовностях. EAO вимірює очікуване перекриття зон відстежувача без перезавантажень на короткостроковій послідовності.

Для вибору та порівняння алгоритмів можна скористатися даними щодо переможців у категорії реального часу за 2017 – 2020 роки, дані про яких було взято з відповідних джерел [5][6][7][8]. Порівняння алгоритмів наведено у таблиці 1.1.

Таблиця 1.1

Порівняння найкращих алгоритмів на VOT RT2017 – VOT RT2020

Алгоритм	EAO	Точність	Робастність	Рік	GPU
AlphaRef	0.486	0.754	0.788	2020	Так
OceanPlus	0.471	0.679	0.824	2020	Так
AFOD	0.458	0.708	0.780	2020	Так
SiamMargin	0.366	0.577	0.321	2019	Так
SiamFCOT	0.350	0.601	0.386	2019	Так
DiMP	0.321	0.582	0.371	2019	Так
SiamRPN	0.383	0.586	0.276	2018	Так
SA_Siam_R	0.337	0.566	0.258	2018	Так
SA_Siam_P	0.286	0.533	0.337	2018	Так

Продовження таблиці 1.1.

CSRDCF++	0.212	0.459	0.398	2017	Так
SiamFC	0.182	0.502	0.604	2017	Так
ECOhc	0.177	0.494	0.571	2017	Ні
Staple	0.170	0.530	0.688	2017	Ні

У таблиці наведено дані щодо оцінювання кожного алгоритму за значеннями середнього очікуваного перекриття, точності, робастності та використання алгоритмом обчислювальних ресурсів відеокарти (GPU) у процесі роботи. Також варто зазначити про трактування значень кожної з мір оцінювання. Більші значення очікуваного середнього перекриття є кращими. Більші значення точності є кращими. Менші значення робастності є кращими.

З наведеної таблиці очевидно, що збільшується тенденція до використання GPU та нейронних мереж у алгоритмах відстеження. Проте, не всі вбудовані системи мають у своїй апаратній архітектурі GPU, а якщо й мають, то наявний GPU не завжди доцільний для виконання важких завдань. Наприклад, Raspberry Pi хоч і має GPU, проте він не є достатньо потужним для виконання завдання відстежування і використовується, зазвичай, для завдань мультимедіа. Якщо ж реалізувати ці алгоритми без використання GPU, то їхня швидкість буде значно менша. Тому одним із критеріїв, який вплине на вибір алгоритму, є направленість алгоритму на виконання на CPU.

Для подальшого розгляду можна одразу відкинути алгоритми, які використовують GPU. Відповідно, залишилися лише алгоритми Staple та ECOhc. Алгоритм ECOhc є кращим за показником робастності, а також має трохи кращий показник EAO. З іншого боку, Staple має кращий показник точності. Якщо звернутися до джерел, у яких описано ці відстежувачі [9][10], то в них можна знайти інформацію щодо їхньої швидкодії. Кожен із них запускався на процесорі Intel i7. ECOhc зумів досягти 60 кадрів на секунду, у той час як Staple – 80 кадрів на секунду, при чому автори відстежувача Staple додатково вказали на модифікації, які збільшують швидкість відстежування до 100 кадрів на секунду. Саме тому перевага у виборі

алгоритму відстежування буде надана алгоритму Staple, оскільки швидкодія є вирішальним показником у цьому випадку.

1.5. Опис алгоритму роботи відстежувача

Алгоритм відстеження Staple в процесі роботи спирається на комбінацію фільтра кореляції (з використанням гістограми спрямованих градієнтів як характеристики) та глобальної кольорової гістограми. Він працює згідно з парадигмою відстежування на основі розпізнавання. Далі на основі джерела [6] буде описано особливості роботи цього алгоритму.

Проблему пошуку об’єкта на зображенні x_t у порядковому кадрі t можна звести до задачі вибору такої прямокутної зони p_t з набору S_t, яка збільшуватиме оцінку в формулі

$$p_t = \arg\max_{p \in S_t} f(T(x_t, p), \theta) \tag{1.1}$$

де функція T – таке перетворенням зображення, що $f(T(x, p), \theta)$ присвоює оцінку прямокутній зоні p на зображенні x відповідно до параметрів моделі θ;

θ – параметри моделі, які мають обиратися так, щоби мінімізувати значення функції витрат, яка залежить від попередніх зображень та положень об’єкта на цих зображеннях.

Для пошуку об’єкта на зображенні використовується оцінювальна функція. Оцінювальна функція є лінійною комбінацією шаблонної оцінки та оцінки гістограми і визначена як

$$f(x) = \gamma_{tmpl} f_{tmpl}(x) + \gamma_{hist} f_{hist}(x) \tag{1.2}$$

де γ_{tmpl}, γ_{hist} – коефіцієнти, які визначають вплив шаблонної оцінки та оцінки гістограми на результуюче значення оцінки, при чому $\gamma_{tmpl} + \gamma_{hist} = 1$.

Шаблонна оцінка є лінійною функцією характеристичного *K*-канального зображення φ_x, отриманого з зображення *x* та визначеного на скінченній сітці $\tau \subset \mathbb{Z}^2$, обчислюється як

$$f_{tmpl}(x;h) = \sum_{u\in\tau} h[u]^T \phi_x[u], \tag{1.3}$$

де *h* – шаблон, інше характеристичне *K*-канальне зображення.

Оцінка гістограми є лінійною функцією середнього пікселя й обчислюється з характеристичного *M*-канального зображення ψ_x, отриманого з зображення *x* і визначеного на скінченній сітці $H \subset \mathbb{Z}^2$:

$$f_{hist}(x;\beta) = \beta^T \left(\frac{1}{|\mathrm{H}|}\sum_{u\in\mathrm{H}} \psi_x[u]\right), \tag{1.4}$$

де β – ваговий вектор гістограми, шаблон.

Оскільки зовнішній вигляд об'єкта може суттєво змінюватися впродовж всієї відео послідовності, то не ефективно будувати його модель лише на основі першого кадру та використовувати цю єдину модель, щоби знайти об'єкт у всіх інших кадрах. Тому цей алгоритм застосовує адаптацію моделі для того, щоби скористатися інформацією, наявною в пізніших кадрах. Для цього він використовує прогнози (знайдене положення цілі) у нових кадрах як навчальні дані, завдяки яким можна оновити модель. Проте в цьому підході існує небезпека того, що невеликі помилки з часом накопичуються й в результаті можуть викликати дрейф моделі.

Отримання кореляційного фільтра (шаблону) для шаблонної оцінки відбувається за формулою

$$\hat{h}[u] = 1/(\hat{d}[u] + \lambda)\cdot \hat{r}[u], \tag{1.5}$$

де ^ позначає операцію дискретного перетворення Фур'є;

$\hat{d}$ – знаменник шаблону, який визначено у формулі (1.6);

$\hat{r}$ – чисельник шаблону, який визначено у формулі (1.7);

λ – коефіцієнт, який використовується для обмеження складності моделі.

$\hat{d}[u]$ є сумою елементів головної діагоналі матриці $\hat{s}[u]$ з розмірністю *K* на *K* з елементами $\hat{s}^{ij}[u] = (\hat{\phi}^j)^* \otimes \hat{\phi}^i$, де $\hat{\phi}$ є характеристичним *K*-канальним зображенням $\varphi_{T(x,p)}$, яке переведено до простору Фур'є. $\hat{d}[u]$ обчислюється за формулою

$$\hat{d}[u] = \sum_{i=1}^{K} (\hat{\phi}^i)^* \otimes \hat{\phi}^i, \tag{1.6}$$

де * означає комплексне спряжене число;

⊗ означає операцію поелементного множення.

$\hat{r}[u]$ є К-вимірним вектором з елементами $\hat{r}^i[u]$ й обчислюється за формулою

$$\hat{r}^i[u] = (\hat{y})^* \otimes \hat{\phi}^i, \tag{1.7}$$

де *y* – бажаний вихід, зазвичай у формі функції Гаусса з піком в одиниці.

Формули (1.6) та (1.7) завдяки простору Фур'є представляють собою операції крос-кореляції. Операція кореляції застосовується замість виснажливої щодо обчислювальних ресурсів операції згортки. Мапа відповідей обчислюється за допомогою поелементного множення між адаптивним кореляційним фільтром навчання та отриманими з зображення характеристиками за допомогою дискретної трансформації Фур'є (DFT). DFT працює в частотній області за допомогою швидкого перетворення Фур'є (FFT). Мапа довіри в просторовій області отримується за допомогою оберненого FFT (IFFT) над мапою відповідей. Максимальні значення на мапі довіри вказують на нову цільову позицію об'єкта.

Навчання для шаблонної оцінки відбувається через адаптацію чисельника $\hat{r}_t$ та знаменника $\hat{d}_t$ шаблону $\hat{h}$. Адаптація чисельника $\hat{r}_t$ представлена формулою

$$\hat{r}_t = (1-\eta_{tmpl})\hat{r}_{t-1} + \eta_{tmpl}\hat{r}_t', \tag{1.8}$$

де η_{tmpl} – коефіцієнт адаптивності шаблонної моделі;

$\hat{r}_{t-1}$ – чисельник шаблону, отриманий з попередніх кадрів;

$\hat{r}_t'$ – чисельник шаблону, який обчислюється за формулою (1.7) для поточного кадру.

Адаптація знаменника $\hat{d}_t$ представлена формулою

$$\hat{d}_t = (1-\eta_{tmpl})\hat{d}_{t-1} + \eta_{tmpl}\hat{d}_t', \tag{1.9}$$

де $\hat{d}_{t-1}$ – знаменник шаблону, отриманий з попередніх кадрів;

$\hat{d}_t'$ – знаменник шаблону, який обчислюється за формулою (1.6) для поточного кадру.

Отримання шаблону кольорової гістограми для характеристичного каналу зображення $j = 1..M$ відбувається за формулою

$$\beta_t^j = \frac{p^j(O)}{p^j(O) + p^j(B) + \lambda}, \tag{1.10}$$

де $p^j(O)$ – частка пікселів у регіоні об’єкта, для якої характеристика j не дорівнює нулю;

$p^j(B)$ – частка пікселів у регіоні фону об’єкта, для якої характеристика j не дорівнює нулю.

Навчання моделі на основі гістограми відбувається за допомогою адаптації моделей зони фону та зони об’єкта. Адаптація зони об’єкта представлена формулою

$$p_t(O) = (1-\eta_{hist})\, p_{t-1}(O) + \eta_{hist}\, p_t'(O), \tag{1.11}$$

де η_{tmpl} – коефіцієнт адаптивності гістограмної моделі;

$p_t(O)$ – вектор з елементів $p_t^j(O)$ для $j = 1..M$;

$p_t'(O)$ – кольорова гістограма зони об’єкту, обчислена у кадрі t.

Адаптація зони фону представлена формулою

$$p_t(B) = (1-\eta_{hist})\, p_{t-1}(B) + \eta_{hist}\, p_t'(B), \qquad (1.12)$$

де $p_t(B)$ – вектор з елементів $p_t^j(B)$ для $j = 1..M$;

$p_t'(B)$ – кольорова гістограма зони фону біля об’єкту, обчислена у кадрі t.

Також, на відміну від опису реалізації в джерелі [6] в цій реалізації алгоритму не буде виконуватися пошук за масштабом об’єкта, тобто якщо під час відстежування об’єкт змінюватиме свої розміри, то зона виділення об’єкту збільшуватися не буде. Це рішення прийняте для того, щоби збільшити ефективність алгоритму, адже контроль за розмірами та масштабом об’єкту не є критичною умовою.

Висновки до розділу

Візуальне відстежування об’єктів є популярною проблемою в межах комп’ютерного зору. Завдяки популярності на цей час існує багато підходів для вирішення цієї проблеми. Проте для відстежування у режимі реального часу за обмежених апаратних ресурсів питання вибору відповідного алгоритму є вирішальним. На вибір відстежувача також впливають вимоги і обмеження щодо супроводжуваних об’єктів та попередні знання чи припущення щодо них. Для вибору відстежувача було проведено огляд існуючих алгоритмів, які доступні на ресурсі «VOT Challenge». В результаті огляду було обрано алгоритм Staple та обґрунтовано його придатність. Цей алгоритм має гарну швидкодію, а також помірні вимоги щодо обчислювальних ресурсів, що є вирішальним для роботи на вбудованих системах.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИСТЕМИ

2.1. Загальний опис системи

Розроблювана система для супроводження візуальних об’єктів складається з апаратної та програмної реалізації. Апаратна реалізація включає у себе вбудовану систему і камеру для зйомки в режимі реального часу. Оскільки кут огляду камери достатньо малий, то для збільшення оглядовості до апаратної реалізації можна додати платформу повороту та нахилу, побудовану на основі серводвигунів. Програмна реалізація складається з графічного інтерфейсу користувача та програмних засобів для візуального супроводу, отримання зображення з камери, контролю серводвигунів механізму повороту-нахилу.

Таким чином, програмну частину можна розбити на 4 модулі:

- модуль графічного інтерфейсу користувача;
- модуль взаємодії з камерою;
- модуль обробки зображень;
- модуль керування серводвигунами в механізмі нахилу та повороту.

Завдання модуля взаємодії з камерою – отримання зображення з камери за запитом. Отримане зображення повинно бути представлено за допомогою кольорової моделі BGR. Також модуль має мати змогу встановлювати та отримувати параметри зйомки, як-от значення висоти та ширини (роздільна здатність зображення) чи кількість кадрів, яку здатна надавати камера, за секунду (FPS).

Модуль обробки зображень отримує на вхід зображення з модуля взаємодії з камерою, а на виході надає оброблене зображення. Обробку зображення можна розділити на три режими:

а) обробка відсутня;

б) виділення цільового об’єкта;

в) відстежування.

У першому режимі модуль працює в режимі трансляції зображення, отриманого з камери. Якщо немає завдання на супровід цілі, то виконуватиметься саме цей вид обробки.

До етапу відстежування користувач має спершу виділити цільовий об’єкт на зображенні ручним способом. Для цього йому треба вказати область, в якій знаходиться об’єкт. За це відповідає режим виділення цільового об’єкта. Виділення цілі відбувається за допомогою малювання над об’єктом прямокутної рамки. Цей вид обробки можна назвати підготовчим або кроком ініціалізації цілі. Коли область виділено, можна проводити відстежування.

Після вказання цільової області для відстежування обробка зображення полягає у пошуку заданої цілі у кожному кадрі, а для цього застосовується алгоритм відстежування, описаний у підрозділі 1.5. Результатом роботи відстежувача є виділення знайденого об’єкта за допомогою прямокутної рамки. При отриманні команди зупинки відстежування обробка зображення знову повертається до п.1.

Ці типи обробок чергуються між собою і відповідають за різні стани процесу відстежування, тому їх зручно сприймати у вигляді станів і переходів між ними. Діаграму станів і переходів для послідовностей обробок наведено на рис. 2.1.

Рис. 2.1. Діаграма станів для модуля обробки зображень

Модуль керування серводвигунами повинен забезпечувати плавний рух серводвигунів у напрямку руху спостережуваного об’єкта. Принцип контролю руху серводвигунів – зберігати супроводжуваний об’єкт у центрі зображення. Для цього модуль повинен отримувати інформацію про положення цілі на зображенні та

обраховувати різницю центру зображення та центру супроводжуваного об’єкта. Після чого керувати серводвигунами для зменшення цієї різниці.

Модуль графічного інтерфейсу користувача повинен показувати користувачу зображення з камери, надавати можливість змінювати параметри зйомки (роздільна здатність, кількість кадрів за секунду), виділяти цільовий об’єкт та зупиняти супровід. Цей модуль безпосередньо взаємодіє з модулем обробки зображення.

2.2. Опис особливостей середовища розробки

За цільову вбудовану систему було обрано Raspberry Pi 3 Model B. Дана платформа є досить універсальним та популярним середовищем для розробки та розв’язання широкого кола завдань.

Raspberry Pi – це одноплатний комп’ютер. Головною особливістю Raspberry Pi є універсальність. Raspberry Pi має операційну систему на базі ядра Linux та піни загального призначення для введення / виведення (GPIO), які полегшують взаємодію з зовнішнім середовищем завдяки підключенню сенсорів та перемикачів. Також система має низьку вартість (~40$), достатню потужність, багато апаратних інтерфейсів (HDMI, USB, Ethernet, GPIO та ін.), наявність широкої спільноти та вичерпної документації.

Raspberry Pi 3B має 64-розрядний чотирьохядерний процесор ARM Cortex-A53 з тактовою частотою 1,2 ГГц. Також він володіє 1 Гб оперативної пам’яті і графічним процесором VideoCore IV. Операційна система встановлюється на microSD карту. Такий підхід дозволяє легко розширювати сховище та перемикатися між різними операційними системами шляхом заміни карт microSD.

Також важливою характеристикою є наявність у Raspberry Pi спеціального слоту для камери. Найпопулярнішими модулями камери є Camera Module V2 та V1. V2 має 8-мегапіксельний сенсор Sony IMX219, а V1 має 5-мегапіксельний сенсор OmniVision OV5647. Модуль камери можна використовувати для зйомки відео з

високою чіткістю, а також для фотографій. Обидва модулі підтримують такі режими зйомки: 1080p30, 720p60 та VGA90. Камера під’єднується за допомогою стрічкового кабеля довжиною 15 см до порту CSI на Raspberry Pi.

Стандартною операційною системою для Pi є Raspbian. Raspbian – це безкоштовна 32-бітна операційна система, заснована на Debian та оптимізована для обладнання Raspberry Pi. Raspbian постачається з великою кількістю програмних пакетів – спеціально зібраного програмного забезпечення, яке постачається у зручному форматі з метою зручної установки на Raspberry Pi. Завдяки цьому розробка програм не відрізняється від десктопної розробки на Linux-подібній ОС і не має зайвих труднощів.

2.3. Опис допоміжних бібліотек для розробки системи

Для програмування системи буде використано мову C++. Ця мова програмування підтримує імперативну, об’єктно-орієнтовану та узагальнену парадигми програмування. C++ надає значно більший контроль над тим, як програмне забезпечення використовує ресурси. Саме тому за допомогою C++ можна досягти ефективного використання наявних ресурсів. Найчастіше C++ застосовують там, де важливі час та ефективність. Проте, за такі переваги мова розплачується довшим часом розробки програмного забезпечення і вимагає наявності більш ґрунтовних знань у розробника.

Для розробки графічного інтерфейсу буде використано фреймворк Qt [11]. Qt є кросплатформним і має переносимість на рівні сирцевого коду. Фреймворк має всі основні класи, які можуть бути потрібні для розробки прикладного ПЗ, починаючи з елементів графічного інтерфейсу й закінчуючи класами для роботи з мережею. Бібліотека також дозволяє керувати потоками, працювати з мережею та забезпечує кросплатформовий доступ до файлів. Іншою особливістю Qt є наявність механізму сигналів та слотів. Це елемент слабкого зв’язування сутностей.

Сигнал формується коли відбувається певна подія. Слот – це функція, яка викликається у відповідь на певний сигнал. Механізм сигналів і слотів є безпечним у відношенні до типів. Сигнатура сигналу повинна збігатися з сигнатурою слота-одержувача. Клас, який виробляє сигнал, не відає про те, які слоти його отримають. Так само, і слот – нічого не відає про те, на які сигнали він реагуватиме. Один сигнал можна підключати до багатьох слотів, а один слот може реагувати на декілька сигналів. Всі класи, успадковані від QObject прямо чи опосередковано можуть мати і використовувати сигнали і слоти. Головною перевагою даного механізму, як вже було сказано, є слабка зв’язаність між слотами та сигналами. Проектування взаємодій між об’єктами із застосуванням такого механізму є більш простішим і дозволяє будувати більш загальні і незалежні класи та сутності.

Оскільки розроблювана система часто матиме справу із зображеннями, то для роботи з ними буде використано бібліотеку OpenCV [12]. OpenCV — бібліотека функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення з відкритим вихідним кодом. Бібліотека розроблена і підтримується Intel. Вихідний код бібліотеки написаний мовою C++ і поширюється під ліцензією BSD. Бібліотека може вільно використовуватися в академічних та комерційних цілях.

Головною особливістю бібліотеки OpenCV, яка буде використовуватись під час розробки системи, є наявність ієрархії класів для подання зображення у формі багатовимірних матриць. У цих класах визначено зручний інтерфейс для доступу до будь-яких елементів зображення, а також наявні операції із зображеннями, такі як додавання матриць, масштабування значень та розмірів матриці, виділення каналу, накладання маски та інші.

Для взаємодії з серводвигунами механізму повороту та нахилу буде використано піни GPIO на Raspberry Pi. Керування серводвигунів зазвичай відбувається за допомогою широтно-імпульсної модуляції. Для зручного доступу до пінів та для можливості передачі модульованого сигналу цими пінами буде використано бібліотеку pigpio [13].

Ще одним необхідним завданням є отримання кадру із камери. У підрозділі 3.1 буде описано підходи та бібліотеки для цього, а також порівняння швидкодії цих підходів.

2.4. Проектування графічного модуля користувача

Модуль графічного інтерфейсу користувача використовує всі переваги віджетів, які надаються фреймворком Qt. Віджети можуть відображати дані, отримувати введення від користувача, бути контейнером для інших віджетів, які слід об’єднати. Діаграма класів для модуля графічного інтерфейсу користувача наведена на рис. 2.2.

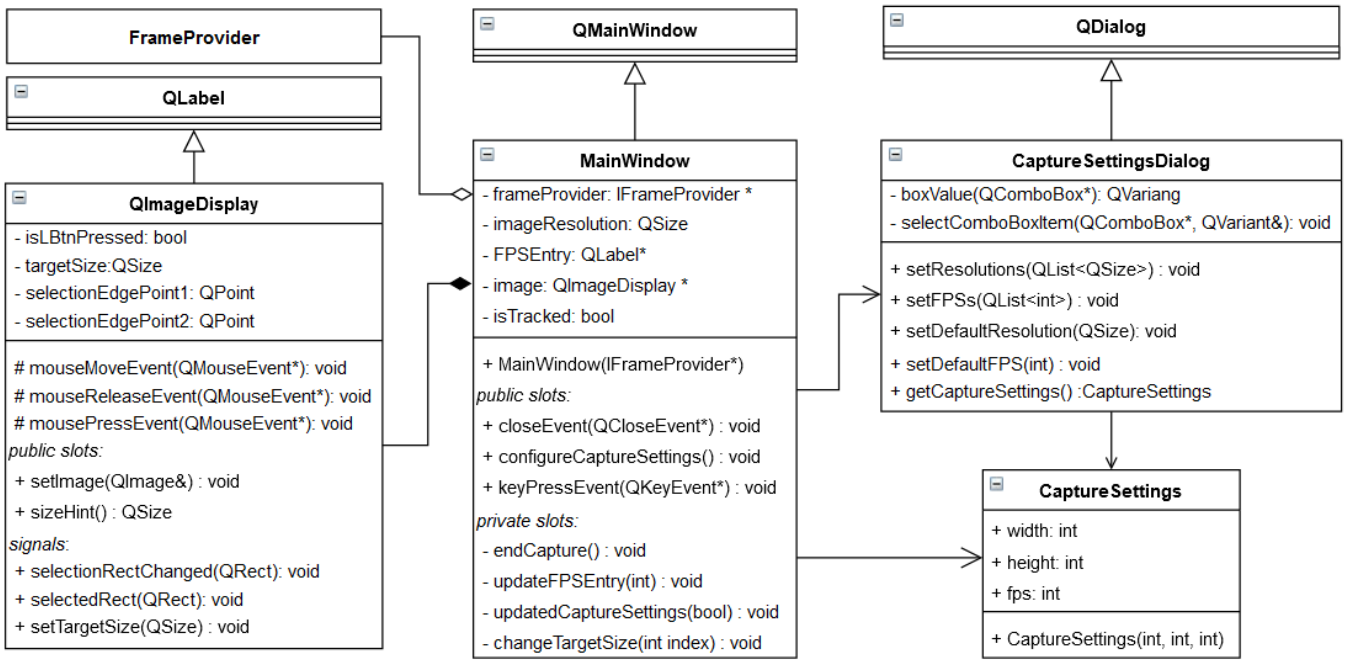


Рис. 2.2. Діаграма класів для модуля графічного інтерфейсу користувача

Більша частина графічного інтерфейсу користувача розташована у головній формі, яка представлена класом MainWindow. Цей клас є нащадком класу QMainWindow, який використовується для розробки головного вікна додатку. В класі MainWindow зосереджені необхідні графічні елементи для взаємодії користувача з

програмою. Через графічний інтерфейс відбувається диспетчеризація команд до класу FrameProvider. Взаємодія з модулем обробки зображень, представленим класом FrameProvider, відбувається з використанням механізму сигналів та слотів. Таким чином, при наявності нового обробленого зображення графічному інтерфейсу подається сигнал, який обробляється через відображення зображення, а коли користувач виділяє цільовий об’єкт, обробнику зображень надходять сигнали з координатами миші, які він масштабує та зображує у вигляді прямокутника.

Відображення зображення покладено на віджет QImageDisplay. Він також призначений для надання користувачу можливості вибору цільової області за рахунок миші. Даний віджет є нащадком QLabel. За рахунок перевизначення таких обробників подій як mouseMoveEvent, mousePressEvent, mouseReleaseEvent даний віджет може реагувати на події від миші заданим чином.

Клас CaptureSettingsDialog є нащадком класу QDialog і є модульним діалоговим вікном. Він слугує для встановлення роздільної здатності зображення та бажаної швидкості отримуваних з камери кадрів (FPS). Бажані налаштування, які задаються класом CaptureSettings, передаються до модуля отримання зображення через інтерфейс класу FrameProvider.

2.5. Проектування модуля обробки зображень

Даний модуль представлений класами FrameProvider та інтерфейсами ITracker, ICameraCapture, ITargetHightlighter. Клас FrameProvider виконує обробку зображення за допомогою описаних інтерфейсів і надає готовий кадр через механізм сигналів та слотів – у вигляді сигналу. Клас StapleTracker реалізує інтерфейс ITracker і працює згідно з математичною моделлю описаною у підрозділі 1.5. Інтерфейс ICameraCapture представляє собою модуль отримання зображень із камери. Інтерфейс ITargetHighlighter використовується для виділення цільового об’єкта рамкою під час

супроводження і під час ініціалізації цілі користувачем. Діаграма класів для модуля обробки зображень наведена на рис. 2.3.

Клас FrameProvider є нащадком QThread і має перевизначений метод run, який при інстанціюванні цього класу буде виконуватись в окремому потоці. Проте, оскільки всі методи, в тому числі і конструктор з деструктором, які викликаються у нащадка QThread будуть виконуватись у викликаючому потоці, то треба синхронізувати доступ до змінних, які використовуються у методах класу FrameProvider. Синхронізація доступу виконується за допомогою QReadWriteLock.

Завдяки використання інтерфейсів розроблювана система буде менш зв’язаною, а також передбачатиме можливе розширення. Таким чином, у систему може бути додано новий алгоритм відстеження, або ж додано інший спосіб отримання кадру із камери.

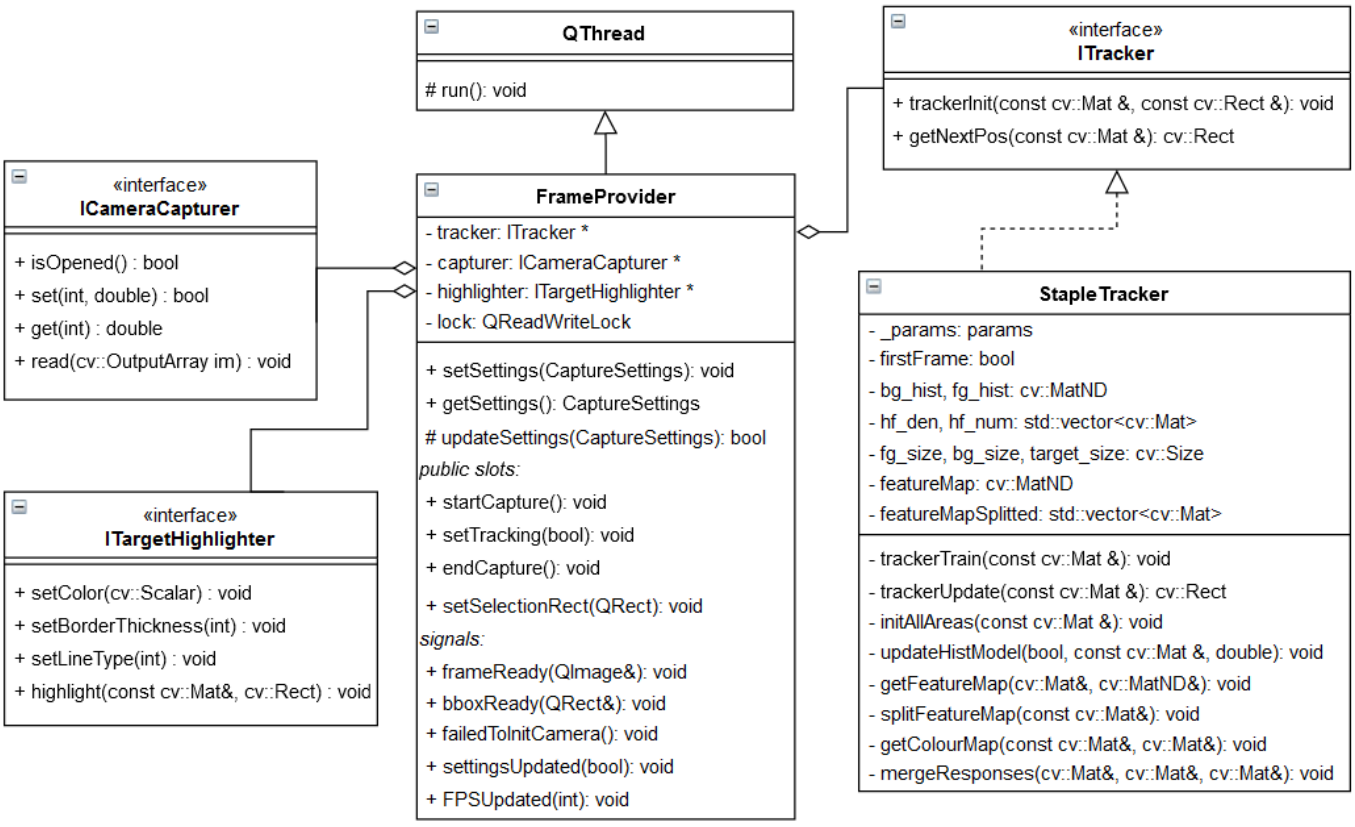


Рис. 2.3. Діаграма класів для модуля обробки зображень

2.6. Проектування модуля керування механізмом повороту та нахилу

Діаграма класів для модуля керування механізмом повороту та нахилу наведена на рис. 2.4. Механізм повороту і нахилу побудований на основі двох серводвигунів, один з яких відповідальний за поворот у горизонтальному напрямку, а інший – нахил у вертикальному напрямку. Кожний із серводвигунів представлений інтерфейсом IServoController, а механізм повороту та нахилу – абстрактним класом PanTiltController. Клас PanTiltController керує двома серводвигунами через слот handleShift, який реагує на зміну положення об’єкта у кадрі і змінює положення серводвигунів таким чином, щоб мінімізувати різницю між центром зображення та центром об’єкта, а також забезпечити плавний рух серводвигунів, аби уникнути зайвих тремтінь та розмивань у наступних кадрах.

Проектування програмного модуля здійснено з використанням інтерфейсів через те, що різні серводвигуни можуть мати різні специфікації щодо їхнього керування, а описані інтерфейси задають узагальнений доступ до функціональності серводвигунів. Тому для розробки модуля також треба буде реалізувати такий інтерфейс для окремого серводвигуна.

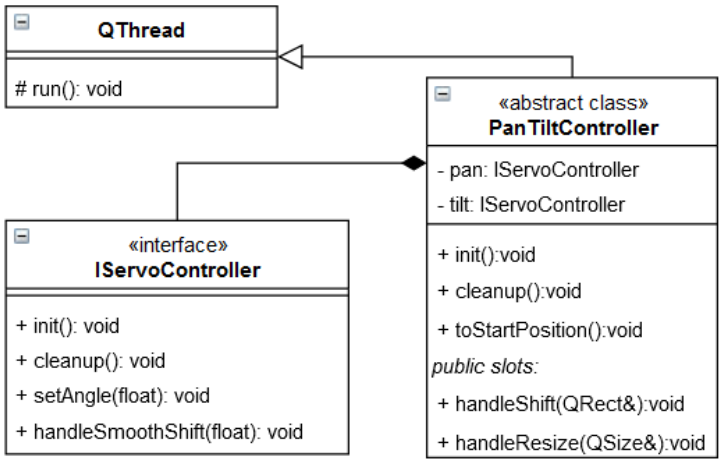


Рис. 2.4. Діаграма класів для модуля керування механізмом повороту та нахилу

Завдяки наслідуванню від класу QThread цей модуль виконується в окремому потоці і взаємодія з ним відбуватиметься в асинхронному режимі. Таке рішення було

прийняте через те, що контроль серводвигунів зазвичай вимагає витримування певного часу очікування. Тому, щоб не затримувати загальний процес, краще виконувати керування серводвигунами асинхронно. Якщо вказівок щодо зміни положення цілі на модуль надходити не буде, то модуль не здійснюватиме жодних дій, а лише очікуватиме. Натомість при надходженні даних щодо руху цілі модуль буде асинхронно керувати серводвигунами.

Висновки до розділу

Для розробки рішення було обрано мову C++ за її можливості щодо ефективного керування пам’яттю. Цільовою вбудованою платформою було обрано Raspberry Pi 3B за її популярність, доступність та гарну підтримку. Додатково було обрано фреймворк Qt оскільки він дозволяє зробити архітектуру додатку більш гнучкою, а також має необхідні засоби для розробки графічного інтерфейсу.

У розділі було наведено загальний опис системи, який необхідний для її розуміння, а також подальших прийнятих рішень. З опису системи було виділено модулі та наведено порядок їх взаємодії. Кожний модуль було описано, а також спроектовано зовнішні та внутрішні інтерфейси за допомогою діаграм класів. Описані модулі є слабко зв’язаними між собою, що забезпечує гнучкість системи. Кожний з модулів може бути доповнений або змінений завдяки використанню внутрішніх інтерфейсів.

РОЗДІЛ 3
ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ СИСТЕМИ

3.1. Порівняння підходів отримання кадру з камери

Для Raspberry Pi наявно декілька можливих програмних засобів для отримання зображення з камери Camera Module V1/V2.

Бібліотека OpenCV пропонує клас VideoCapture [14] для отримання кадрів із відеофайлу, камери або мережевого потоку. Клас VideoCapture є зручною обгорткою для багатьох API бекендів вводу-виводу відео. Деякі бекенди, як-от Direct Show (DSHOW), Video For Windows (VFW), Microsoft Media Foundation (MSMF), Video 4 Linux (V4L) тощо є інтерфейсами до бібліотеки вводу-виводу відео, що надається операційною системою. Деякі інші бекенди, як-от OpenNI2 для Kinect, Intel Perceptual Computing SDK, GStreamer, XIMEA Camera API тощо, є інтерфейсами до пропрієтарних драйверів або до зовнішніх бібліотек. Проте варто зазначити, що API бекенду у VideoCapture буде доступно лише у випадку, коли необхідні бекенди використовувалися для створення бінарних файлів бібліотеки OpenCV.

Для порівняння буде використано бекенди V4L2 та GStreamer, оскільки вони доступні на Raspberry Pi в операційній системі Raspbian. V4L2 постачається з Raspbian і для ввімкнення цього модуля необхідно виконати в терміналі команду sudo modprobe bcm2835-v4l2. GStreamer не постачається за замовчуванням, тому його необхідно встановити за допомогою команди sudo apt-get install gstreamer1.0-tools.

Оскільки клас VideoCapture може вносити додаткові витрати до процесу отримання зображення, то можна спробувати використовувати драйвер V4L2 безпосередньо, як описано в джерелі [15].

У дистрибутиві Raspbian також наявні утиліти raspistill та raspivid для отримання фото та запису відео з камери [16]. Ці утиліти використовують MMAL (Multimedia Abstraction Layer) API, який працює над іншою бібліотекою OpenMAX. API MMAL надає простіший інтерфейс, ніж OpenMAX. Варто також зазначити, що

MMAL — це специфічний для Broadcom API, який використовується лише в системах VideoCore 4 (GPU), тобто у Raspberry Pi 4 описаний API не використовується.

Відомою бібліотекою, яка використовує MMAL API є Picamera, проте ця бібліотека написана для мови Python. Іншою бібліотекою, яка також використовує MMAL API, але написана для мови C++ є Raspicam [17]. У цій бібліотеці також міститься клас RaspiCam_CV, який спрощує процес отримання кадру з камери та подальшого використання кадру за допомогою функцій бібліотеки OpenCV.

Для зручного порівняння швидкодії та можливостей бібліотек було створено загальний інтерфейс CameraCapture, який надає можливості для підключенню до камери, отримання кадру, встановлення й читання властивостей камери та отриманих кадрів. З використанням вищеописаних бібліотек було реалізовано цей інтерфейс і було отримано 4 класи для порівняння.

Для оцінювання було обрано множину розмірів популярних роздільних здатностей зображень, з урахуванням апаратних обмежень модуля камери. Кожен із порівнюваних об’єктів по черзі ініціалізувався і встановлював бажаний розмір кадру відповідно до заданої роздільної здатності. Також по черзі встановлювалася бажана швидкість отримання кадрів від камери, яка дорівнювала 30 та 40 кадрам у секунду. Тестова множина кадрів для кожного оцінювання складала 1000 кадрів. Після чого фіксувався загальний час отримання зображень і з нього обраховувалося середнє значення кількості одержаних кадрів за секунду через ділення кількості кадрів на час роботи.

Оцінювання часу виконувалося програмним методом за допомогою бібліотеки OpenCV. OpenCV має функції getTickCount() та getTickFrequency() [18]. Перша функція повертає кількість тактових циклів від певного моменту початку відліку (наприклад, від моменту старту системи) до виклику цієї функції. Друга функція повертає кількість тактових циклів у секунді. Отже, якщо на початку та наприкінці певної події викликати getTickCount(), а потім знайти різницю між значеннями, то буде отримано витрачений час у тактових циклах процесора. Якщо ж після цього отримане значення поділити на значення функції getTickFrequency(), то результуючим значенням буде час у секундах. Також варто зазначити, що

вимірювання часу в тактових циклах процесора є більш точним, тому було обрано саме такий спосіб підрахунку часу.

Згідно з описаним вище принципом оцінювання та довідкою щодо функцій OpenCV для вимірювання швидкодії, програмне оцінювання для кожного дослідження включало 5 кроків.

1. Викликати і зберегти значення функції getTickCount().
2. Розпочати отримання кадрів із камери за допомогою заданого підходу, для заданої роздільної здатності та заданої бажаної швидкості одержання кадрів.
3. Викликати і зберегти значення функції getTickCount() після того, як отримано необхідну кількість кадрів.
4. Знайти загальний час обробки через ділення різниці значень викликів getTickCount() на значення функції getTickFrequency().
5. Знайти середнє значення кількості кадрів за секунду через ділення кількості оброблених кадрів на загальний час.

Результати оцінювання, проведеного на Raspberry Pi 3B з ОС Raspbian, при бажаній швидкості отримання кадрів 30 FPS наведено в табл. 3.1.

Таблиця 3.1

Швидкодія досліджуваних підходів при 30 FPS

	OpenCV V4L2	OpenCV GStreamer	RaspiCam	V4L2
320 x 240	26,76	30,26	28,13	30,22
640 x 480	26,7	30,28	28,15	30,25
1024x768	5,52	5,83	14,22	5,44
1280x720	26,45	30,22	28,02	30,08
1280x768	5,37	5,63	27,94	5,54
1280x960	5,39	5,67	27,53	5,59
1280x1024	5,47	5,74	9,62	5,64
1600x1200	5,24	5,57	12,5	5,39
1920x1080	5,41	5,58	13,01	5,51
1920x1200	5,35	5,53	10,35	5,54

У табл. 3.2 наведено результати оцінювання при бажаній швидкості 40 FPS.

Таблиця 3.2

Швидкодія досліджуваних підходів при 40 FPS

	OpenCV V4L2	OpenCV GStreamer	RaspiCam	V4L2
320 x 240	32,93	40,49	36,78	40,73
640 x 480	32,84	40,43	36,7	40,63
1024x768	5,51	5,78	12,76	5,73
1280x720	32,41	40,38	33,07	40,68
1280x768	5,37	5,65	31,37	5,57
1280x960	5,39	5,68	22,83	5,61
1280x1024	5,49	5,78	9,35	5,7
1600x1200	5,36	5,59	11,62	5,44
1920x1080	5,52	5,65	12,54	5,61
1920x1200	5,3	5,54	9,74	5,44

На рис. 3.1 та рис. 3.2 наведено дані з табл. 3.1 та табл. 3.2 відповідно у вигляді діаграми, яка є більш наочною для сприйняття.

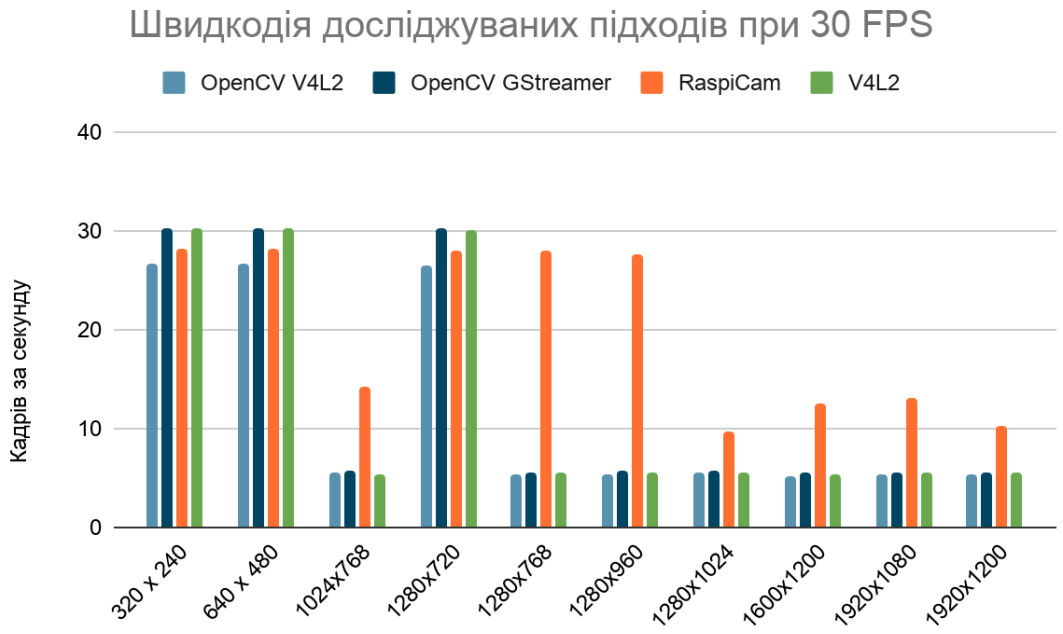


Рис. 3.1. Швидкодія досліджуваних підходів при 30 FPS

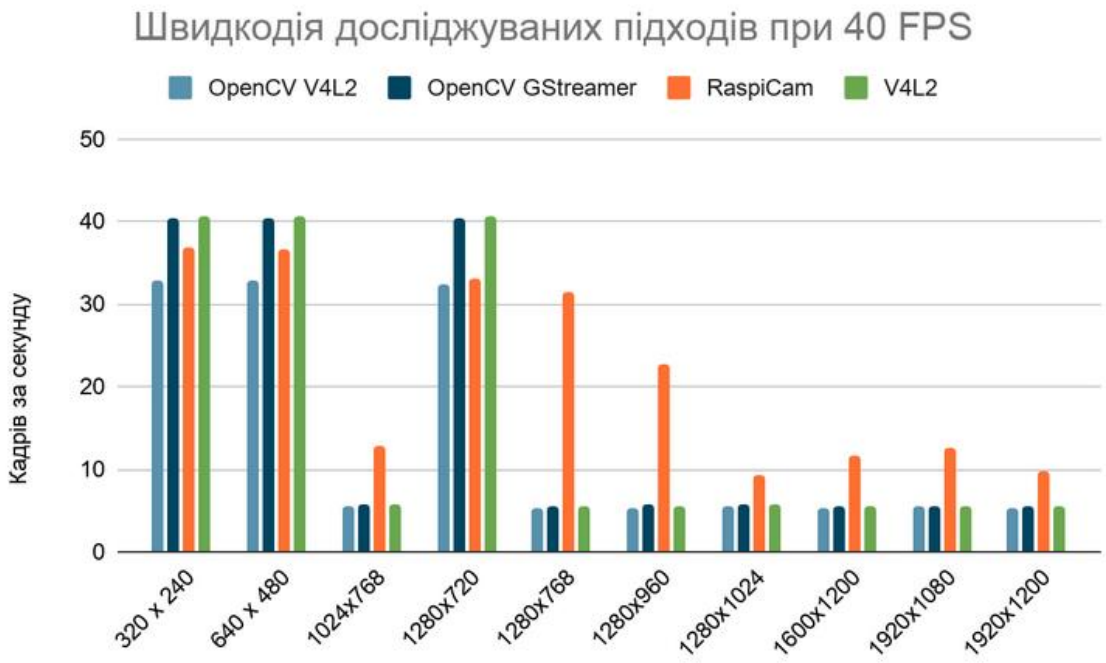


Рис. 3.2. Швидкодія досліджуваних підходів при 40 FPS

З отриманих результатів можна зробити такі висновки. По-перше, для деяких роздільних здатностей швидкодія відповідає встановленому значенню швидкості одержання кадрів. По-друге, для інших роздільних здатностей найкращі результати має підхід на основі RaspiCam. По-третє, для цих роздільних здатностей задання більших швидкостей отримання кадрів призводить до зменшення швидкодії, тому для них краще встановлювати таку швидкість, на рівні якої вони виконуються. Отже, для роздільних здатностей 320x240, 640x480 та 1280x720 краще використовувати підхід на основі GStreamer за швидкості 40 FPS, а для інших роздільних здатностей — RaspiCam.

3.2. Огляд програмних оптимізацій відстежувача

Оскільки реалізація алгоритму відстеження та її запуск на платформі Raspberry Pi 3B під час тестування швидкодії (див. пункт 3.2.5.) не виконала вимоги щодо обробки у режимі реального часу, то було прийнято рішення оптимізувати алгоритм

відстеження. Опис найбільш впливових та значних оптимізацій наведено у наступних розділах.

3.2.1. Видалення зайвих копіювань

Алгоритм STAPLE під час роботи обробляє багато масивів зображень, які подані як масиви пікселів. Наприклад, зображення з роздільною здатністю 640x480 складається з 307200 пікселів, які зі свого боку мають по 8 біт на кожний із трьох кольорів (синій, зелений, червоний), тобто по 3 байти на піксель і 921600 байт на зображення відповідно. Операції копіювання або ж ініціалізації таких масивів даних можуть значною мірою впливати на час виконання алгоритму. Саме тому зайві операції копіювання та ініціалізації великих даних є потенційними претендентами для проведення оптимізацій, призначених для збільшення швидкодії алгоритму.

На стадіях пошуку цілі та тренування алгоритм потребує використання мапи особливостей. Використовуючи припущення про те, що ці стадії будуть завжди виконуватися послідовно одна за одною, можна знаходити мапу особливостей лише один раз замість двох у кожній стадії. Також оскільки в алгоритмі використовується підхід пошуку фіксованого розміру цілі, заданого в першому кадрі, то мапа особливостей також не змінюватиметься в розмірі під час виконання алгоритму. У такий спосіб, можна ініціалізувати мапу особливостей лише один раз — під час ініціалізації відстежувача, тим самим, уникнувши зайвих ініціалізацій під час стадій пошуку цілі та тренування.

Окрім мапи особливостей є й інші дані зі значним розміром, для яких можна зменшити зайві копіювання та ініціалізації. Серед них і два подання шаблонної моделі. Оскільки розміри шаблонної моделі не змінюються під час виконання алгоритму, то нові тимчасові подання шаблоної моделі, які використовуються для оновлення глобальної шаблонної моделі, можна так само ініціалізувати лише один раз, уникаючи зайвих подальших ініціалізацій під час обрахунків.

Описані дії в програмному коді можна застосувати за допомогою одного з двох підходів. Оскільки відстежувач представлений класом, то можна додати нові змінні в приватну частину класу, ініціалізувати дані один раз у конструкторі або в методі для ініціалізації нової цілі та використовувати додані змінні в необхідних методах. Інший підхід полягає у використанні ключового слова static [19] поряд із визначенням змінної в межах методу (для мови C++). Змінна, визначена як static, у межах методу має статичну тривалість зберігання й ініціалізується лише один раз під час першого виклику методу, у якому вона описана. Проте при використанні такого підходу для змінних, розмір яких залежить від розміру цілі, тобто визначається під час виконання, необхідно додати спеціальні програмні конструкції для коректної ініціалізації цих змінних. Наприклад, через використання булевої змінної, яка позначає нову ціль і, відповідно, необхідність ініціалізації чи переініціалізації.

Загальним правилом для визначення того, який із двох вищеописаних підходів варто застосувати, є врахування необхідної видимості змінних. Тобто, якщо змінна використовується лише в одному методі і є локальною, до доцільніше скористатися підходом, який використовує ключове слово static. В іншому випадку необхідно додати змінну до приватних полів класу.

Під час проведення такого типу оптимізації, окрім мапи особливостей та тимчасових змінних для отримання глобальної шаблонної моделі, описаним чином було також змінено тимчасові змінні для оновлення кольорової моделі та одержання мапи відповідей.

3.2.2. Оптимізація циклів

В алгоритмі STAPLE обробка масивів даних часто заснована на циклах. Проте, залежно від порядку доступу до елементів масиву швидкість обробки значно відрізняється. Наразі в комп’ютерній архітектурі кожне з ядер процесора має власний кеш і дані з оперативної пам’яті проходять через цей кеш і зберігаються у ньому для

збільшення швидкодії, оскільки кожне звертання до пам’яті є дорогою операцією для процесора. Отже, якщо під час обробки масиву дані будуть отримуватися послідовним способом, подібним до фізичного розташування даних у пам’яті, то швидкодія буде максимальною. Це відбувається через те, що при необхідному зверненні до пам’яті відбувається читання не заданої адреси, а цілого блоку, який потім зберігається у кеші і при наступному зверненні до наступної адреси дані вже будуть в кеші.

Натомість, якщо масив буде великим і в кожній ітерації будуть необхідними такі значення, які розташовані на значній відстані один від одного, то швидкодія буде значно нижчою через те, що кеш буде швидко застарівати й оновлюватись.

Оскільки зображення у OpenCV мають формат BGR, тобто по 3 кольори на піксель, то найбільш швидкою буде така послідовність циклів, яка спочатку (зовнішній цикл) ітерується по рядках, потім – по стовпчиках, а потім – по кольорах.

Спершу для обробки масивів використовувалася функція forEach класу Mat з бібліотеки OpenCV. У документації [20] зазначено, що функтор, який надано в якості аргументу функції виконується для всіх елементів матриці паралельно. Проте в результаті порівняння цієї функції зі звичайним вкладеним циклом було виявлено, що цикл виконується швидше.

Для прикладу, операція виділення з багатовимірної матриці по одному каналу та перетворення його в двовимірну матрицю для операцій з комплексними числами з використанням підходу на основі forEach займала 1158851 нс, а підхід на основі звичайного циклу по кожному каналу займає 512128 нс, тобто у 2 рази менше.

Іншою, але більш мінорною оптимізацією є використання вказівників у якості індексів циклу. Операція інкременту вказівника є швидшою за операцію індексації масиву, яка виконує додавання зсуву до адреси масиву.

3.2.3. Оптимізації, доступні для швидкого перетворення Фур'є

Шаблонне оцінювання працює з використанням швидкого перетворення Фур'є (ШПФ) для виявлення кореляції між шаблоном та поточним фоном навколо спостережуваного об'єкта. Операція ШПФ виконується для 28 каналів характеристичного зображення, внаслідок чого вона виконується відносно повільно. Але відомо [21], що ШПФ виконується швидше, якщо розміри масиву (кількості рядків та стовпців) є степенями двійки, або ж кратні 2, 3, 5.

Тому для оптимізації ШПФ було додано функцію пошуку оптимальних розмірів для фону об'єкта на основі перебору. Якщо ж знайти оптимальні розміри не вдається, то натомість використовуються розміри, які визначені у оригінальному алгоритмі. Також варто зазначити, що пошук розмірів відбувається лише один раз – під час ініціалізації відстежувача.

Одним із прикладів цієї оптимізації є тестова відеопослідовність, використана у розділі 3.2.5, із ціллю розміром 41 x 46 і розміром фону 96 x 95. За такої конфігурації відстежувач у середньому здатний оброблювати кадри зі швидкістю 21,58 кадрів на секунду. Натомість, за використання пропонованої оптимізації розмір фону змінюється до 76 x 113, а швидкість обробки кадрів збільшується у середньому до 28,07, тобто збільшується у 1,3 рази.

3.2.4. Використання потоків для паралельного виконання операцій

Оскільки Raspberry Pi 3B має 4 ядра, то теоретично можна скористатись наявними ядрами для розподілення роботи і збільшення швидкодії за рахунок паралельного виконання.

У роботі алгоритму можна виділити декілька операцій, які можна розподілити для виконання різними потоками. Серед цих операцій є обчислення фільтра кореляції, навчання шаблонної моделі, навчання моделі на основі гістограми. Описані операції

виконуються у циклах і їх можна легко розподілити для паралельного виконання за допомогою використання директив зі стандарту OpenMP [22].

В результаті додавання паралельного виконання операцій приріст швидкодії був незначним. Це пов’язано з тим, що операції створення, взаємодії, організації, планування та диспетчеризації потоків також забирають певну кількість процесорного часу. А пропоновані паралельні операції відносно описаних операцій за значеннями часу виконання не є значно вищими. Саме тому паралельні операції трохи швидші за послідовні версії операцій.

3.2.5. Оцінка впливу оптимізацій на роботу відстежувача

Вплив оптимізацій на кількість оброблених кадрів за секунду наведено на акумулятивній діаграмі (тобто кожна з оптимізацій використовує всі попередні оптимізації зліва), яка зображена на рис. 3.3.

Оцінка проводилась на Raspberry Pi 3B (крім пункту «RPi 2B»). Для оцінювання використовувалась відеопослідовність з роздільною здатністю 1280 x 720. Розмір обраної цілі – 41 x 46 пікселів. Фіксувався лише загальний час обробки кадрів відстежувачем, після чого цей час ділився на загальну кількість кадрів, в результаті чого було отримано кінцеве значення кількості оброблених кадрів за секунду. Вибірка кожної з оптимізацій включала 30 запусків, після чого отримані швидкості зводились до середнього арифметичного.

Також варто зазначити, що у наведеному вище оцінюванні не було залучено камеру, а використовувалась однакова відеопослідовність для уніфікації вхідних даних та зменшення впливу інших факторів на роботу відстежувача. Камера ж вносить додаткові залежності по часу за рахунок того, що отримання наступного кадру є відносно довготривалою операцією.

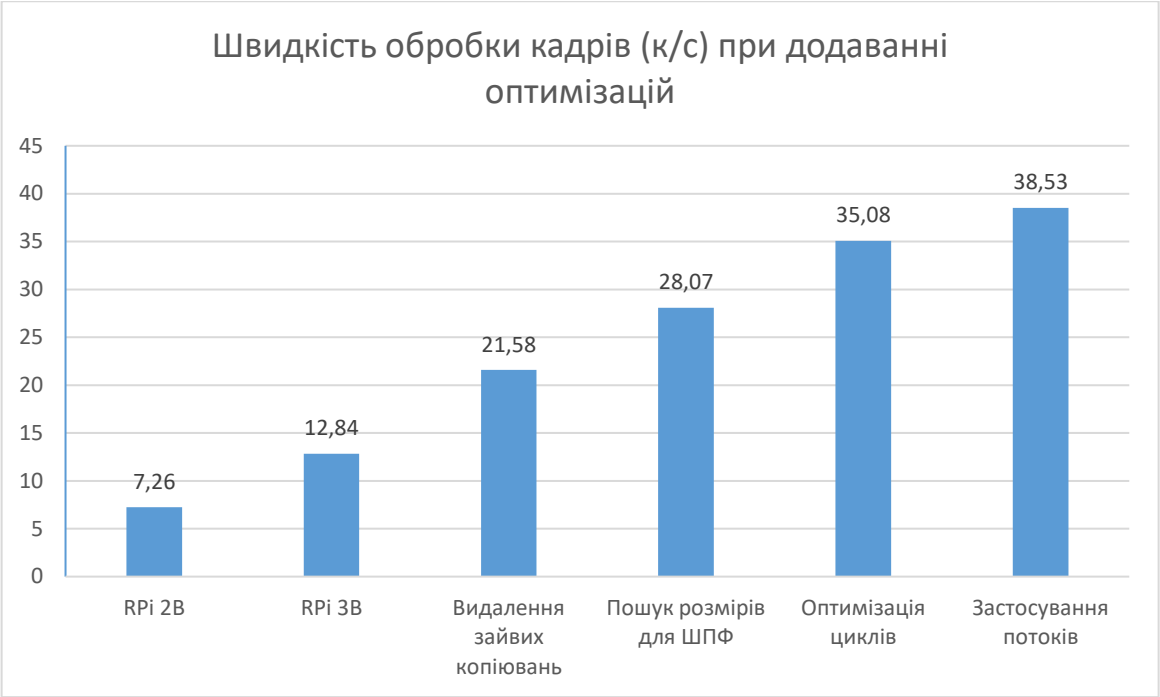


Рис. 3.3. Залежність швидкодії від послідовного застосування оптимізацій

3.3. Опис використаних апаратних засобів та їх з'єднань

Окрім Raspberry Pi у роботі також присутній механізм нахилу та повороту і камера. Механізм нахилу та повороту складається з двох серводвигунів типу SG-90. Ці серводвигуни змонтовані у спеціальний корпус, до якого кріпиться камера. Серводвигуни мають кут повороту 180° і швидкість повороту 60° за 0.1 секунду [23]. Кожний з серводвигунів має по 3 контакти: живлення (VCC), заземлення (GROUND) та контакт широтно-імпульсної модуляції (PWM).

Бажаний кут, у який треба встановити серводвигун, встановлюється за допомогою контакту PWM. В залежності від довжини сигналу і його тривалості розраховується необхідний кут. Серводвигун працює з частотою 50 Гц, тобто на один цикл припадає 20 мс. Довжина імпульсу для даного типу серводвигунів у циклі може бути від 1 мс до 2 мс. Тобто, при довжині імпульсу в 1 мс положення серводвигуна буде на позначці 0°, а при довжині в 2 мс – на позначці 180°. З даного співвідношення можна розраховувати значення необхідного кута.

Серводвигуни по контакту PWM було під’єдано відповідних пінів Raspberry Pi з номерами 12 та 13, оскільки ці піни мають апаратну підтримку широтно-імпульсної модуляції. З іншої сторони, на інших пінах Raspberry Pi, за потреби, можна створити програмну широтно-імпульсну модуляцію, проте апаратна підтримка в даному випадку краща за програмну.

Для з’єднання всієї схеми використовується додаткова перехідна плата разом з шиною і Т-подібним конектором, який дублює піни Raspberry Pi. З’єднання на додатковій перехідній платі детально показано на схематичному зображенні на рис. 3.4. Один кінець шини під’єднується до Raspberry Pi 3B, а інший – до конектора. Для більшої безпеки до кожного з контактів широтно-імпульсної модуляції також під’єднується резистор з розмірністю в 1 кОм.

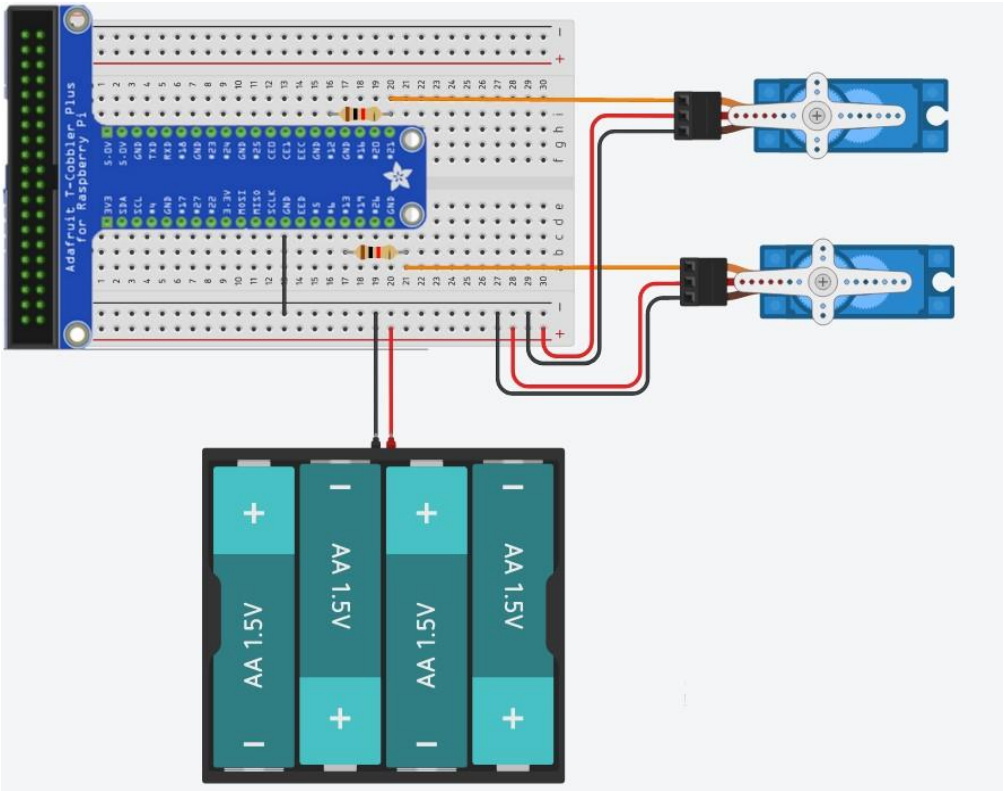


Рис. 3.4. Схема з’єднань механізму повороту та нахилу

Живлення серводвигунів відбувається через зовнішнє джерело струму з напругою у 5V. Також можна було виконати живлення серводвигунів через спеціальні піни живлення з напругою у 5V на Raspberry Pi, проте якщо виникне проблема з одним із серводвигунів, то Raspberry Pi може через це постраждати. Саме

тому живлення серводвигунів відбувається через зовнішнє джерело струму. Проте заземлення на схемі відбувається на спеціальний пін GND на Raspberry Pi, який відповідає за заземлення.

Реальний вигляд рішення подано на рис. 3.5.

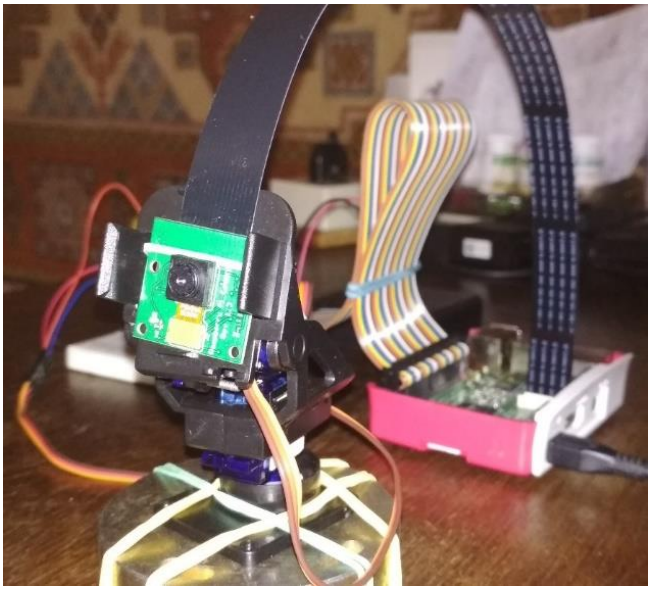


Рис. 3.5. Реальний вигляд рішення

Висновки до розділу

Загальна швидкодія системи залежить від етапів отримання зображення та відстежування об’єкту, саме тому у даному розділі було описано особливості реалізації відповідних модулів та їх оптимізацій. Для етапу отримання зображення з камери було проведено тестування, на основі якого було обрано найшвидше рішення. Для етапу відстежування було описано оптимізації, які дозволили збільшити швидкість відстежувача у 3 рази. Вплив кожної з оптимізацій також було оцінено та наведено для огляду. Також у розділі було описано механічні з’єднання та принцип роботи механізму нахилу-повороту камери, який побудований на основі двох серводвигунів.

РОЗДІЛ 4
ОПИС ЗДОБУТИХ РЕЗУЛЬТАТІВ

4.1. Порядок оцінювання системи

Головним критерієм системи відстежування є швидкодія. Для оцінки швидкодії система в режимі реального часу тестувалась на трьох основних роздільних здатностях вхідних зображень: 320 x 240, 640 x 480, 1280 x 720. Роздільна здатність 1920 x 1080 не використовувалась для тестування, оскільки операція отримання такого зображення є довгою і загальна швидкодія системи в такому разі теж є далекою від вимог реального часу. Для кожного розміру зображення було проведено тестування на різних розмірах цілей: 64 x 64, 100 x 100, 200 x 200. Таким чином, було протестовано 9 випадків. Мета тестування – визначити середню швидкість обробки кадрів за хвилину для заданих роздільних здатностей та розмірів цілей. Загальний час для тестування одного окремого випадку – 5 хвилин. За такого тестування відстежувач при однакових розмірах цілі на різних роздільних здатностях виконує однакову кількість роботи, оскільки пошук цілі проводиться в околі її попереднього положення, а не на всьому зображенні.

Оскільки перше тестування проводилось на різних цілях (зі збільшенням роздільної здатності розмір цілі відносно нього зменшувався), то додатково було проведено інше тестування. Для вказаних роздільних здатностей відбувалось відстежування однієї цілі, розміри якої відповідно до роздільних здатностей є такими: 32 x 32, 64 x 64, 128 x 128. Мета даного тестування – оцінити вплив роздільної здатності вхідного зображення разом з розміром відповідної цілі на швидкодію відстежувача та системи в цілому.

Оцінка середньої швидкості обробки кадрів виконувалась за таким принципом. Загальний час, витрачений на відстежування об’єкта, та кількість оброблених кадрів фіксувалися програмним способом. Для отримання середнього значення кількості кадрів за секунду необхідно кількість оброблених кадрів поділити на загальний час

обробки. Програмна оцінка часу виконувалась програмним методом за допомогою можливостей бібліотеки OpenCV і функцій getTickCount() та getTickFrequency(). Ці функції та спосіб їх застосування вже було описано у розділі 3.2.5. Тестування відбувалось у ручному режимі, тобто цілі виділялись користувачем. Відлік часу та кадрів відбувався лише після ініціалізації цілей.

4.2. Результати оцінювання

Результати оцінювання швидкодії відстежувача за кількістю оброблених кадрів на секунду відносно розміру кадру та розміру цілі, зазначених у попередньому розділі, наведено на рис. 4.1. З отриманих результатів можна зробити висновок, що чим більшим є роздільна здатність вхідного зображення і розмір цілі, тим довше триває обробка кадру.

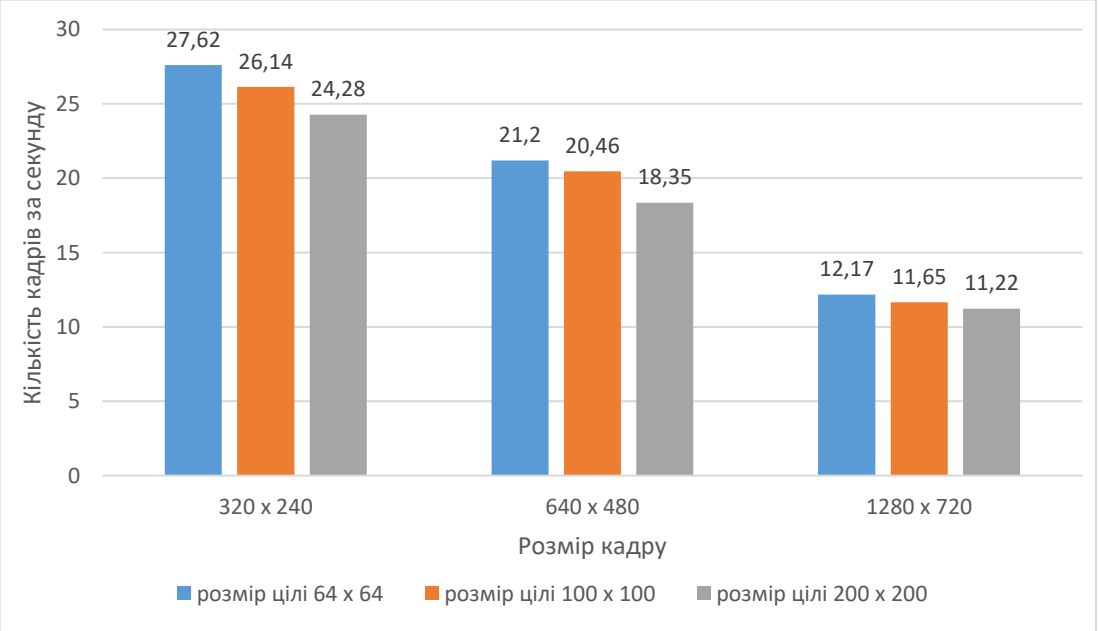


Рис. 4.1. Оцінка швидкодії розробленого рішення

Проте варто зазначити, що в загальний час також було включено і час отримання зображення з камери. Якщо додатково взяти до уваги результати, отримані у розділі 3.2.5, то можна помітити, що залежність загальної швидкодії системи від

камери є значною. Також варто враховувати, що операції копіювання, масштабування та відображення операцій з більшою роздільною здатністю займають більше часу. Швидкодія відстежувача залежить лише від розміру цілі.

Результати іншого тестування показують залежність між розміром цілі та швидкодією відстежувача. Для розміру вхідного зображення 320 x 240 та розміру цілі 32 x 32 швидкодія склала 27.81 кадрів на секунду, для 640 x 480 і розміру цілі 64 x 64 – 21.26, а для розміру 1280 x 720 і розміру цілі 128 x 128 – 12.36. Таким чином, закономірність є наступною: максимальна швидкодія буде за найменших розмірів вхідного зображення та цілі.

4.3. Огляд можливих застосувань системи

Розроблене рішення забезпечує візуальне відстежування об’єкту у відеопотоці. Таким чином, воно може бути використано для автоматизованого супроводу за об’єктом, який представляє інтерес для користувача. Одним із яскравих прикладів застосування подібних систем є військова галузь, а саме – стабілізація прицілу зброї за рухом супроводжуваної цілі. Прикладом подібних розробок є оптико-електронна система керування стрільбою артилерійських установок «Sens-2» [24], розроблена державним підприємством «НДІ «КВАНТ». Дана система призначена для керування вогнем корабельної артилерії малого калібру по повітряним, надводним і береговим цілям у режимах ракурсної колонки або стабілізованого оптико-електронного прицілу (ОЕП). Основною особливістю даної системи є те, що оператор артилерійської установки може вказувати цільовий об’єкт для супроводу. Після отримання завдання на супровід система буде виконувати переміщення камери та артилерійських установок за рухом вказаного об’єкта. Таким чином буде збільшено точність наведення, автоматизовано супровід та прицілювання, а оператора буде звільнено від необхідності виконувати тривале націлювання на об’єкт.

За умови доповнення розробленого рішення алгоритмом розпізнавання окремих цільових об’єктів, рішення може перетворитись у автоматичний відстежувач цілей, який працюватиме без втручання людини. Оскільки операція розпізнавання у кожному кадрі є досить виснажливою щодо обчислювальних ресурсів та довготривалою, то її можна запускати з певною невеликою частотою для пошуку цілі. При вдалому знаходженні цілі буде проведено ініціалізацію цілі для відстежувача і запущено супровід на основі рішення, розробленого у цій роботі. Загальна архітектура матиме вигляд конвеєра або пайплайну (англ. pipeline), коли вихідні дані з одного модуля є вхідними даними для іншого.

За таким підходом можуть працювати детектори руху з фіксацією порушників і візуальним супроводом їх подальшого руху. Такі ж підходи застосовуються в охоронних системах відеоспостереження.

Висновки до розділу

У розділі було наведено порядок тестування швидкодії системи та його результати, які є цілком логічними – чим менша роздільна здатність зображення та менша ціль, тим більшою є швидкодія. Таким чином, за розмірів зображення 320 х 240 та розмірів цілі 64 х 64 швидкість відстежування складала 27 кадрів на секунду. Проте дана роздільна здатність є достатньо низькою на цей час. Якщо ж розглянути роздільну здатність 640 х 480, то вона має прийнятну швидкодію на рівні 20 кадрів на секунду.

У кінці розділу було наведено можливі застосування системи. Вона може бути використана для автоматизованого супроводу за об’єктом. Подібні системи використовуються у військовій галузі для стабілізації прицілу зброї та ведення більш ефективного вогневого враження противника. За умови доповнення розробленого рішення алгоритмом розпізнавання бажаних цільових об’єктів, рішення може перетворитись у автоматичний відстежувач цілей.

ВИСНОВКИ

У випускній кваліфікаційній бакалаврській роботі було проведено ознайомлення з проблемою візуального відстеження об’єктів, яка на сьогоднішній день є досить перспективною і має широкий спектр застосувань, починаючи процесами спостереження за виробничими процесами та закінчуючи використанням у військових цілях. В роботі було розглянуто особливості візуального відстежування об’єктів, загальну будову та стадії алгоритмів відстежування, основні підходи та принципи їх реалізації, наявні популярні алгоритми відстеження, а також основні перешкоди, які виникають під час візуального супроводження об’єктів.

Для вибору алгоритму відстеження було використано публічні дані щодо оцінювання та порівняння відстежувачів, які були проведені у рамках змагань «VOT Challenge», які проводились у попередні роки. Обравши переможців змагань та оцінивши їх відповідно до заданих вимог, було прийнято рішення про вибір алгоритму відстежування Staple, який базується на фільтрі кореляції та кольоровій гістограмі. Це рішення було прийнято на основі гарних показників швидкодії алгоритму, а також у зв’язку з тим, що він не використовує графічний процесор для виконання завдання, тобто є помірним щодо обчислювальних ресурсів.

Цільовою вбудованою системою було обрано Raspberry Pi 3 Model B, оскільки вона є популярною, бюджетною та достатньо потужною платформою для різноманітних проєктів. Raspberry Pi також має багато інтерфейсів, серед яких піни загального призначення, за допомогою яких можна взаємодіяти з зовнішнім середовищем, а також слот для камери. Для розширення поля зору камери було вирішено додати до рішення механізм повороту та нахилу, який побудований на основі двох серводвигунів SG-90.

За допомогою мови програмування C++, бібліотеки комп’ютерного зору OpenCV, фреймворку Qt для графічного інтерфейсу, бібліотеки pigpio для взаємодії з пінами Raspberry Pi та обраного алгоритму відстежування було розроблено програмне

забезпечення з графічним інтерфейсом користувача для візуального супроводу заданого користувачем об’єкту.

Також у роботі було проведено декілька тестувань. Оскільки відстежування відбувається у режимі реального часу, то було проведено тестування швидкодії отримання кадрів із камери при різних бажаних роздільних здатностях зображення з використанням різних доступних підходів. Результат дослідження був цілком передбачуваним і показав, що краща швидкодія забезпечується за менших роздільних здатностей, які кратні 320 x 240.

Інше тестування полягало в окремій оцінці швидкодії алгоритму відстежування. Початкова версія алгоритму мала невелику швидкодію на рівні 12 кадрів на секунду для цілі розміром 41 x 46. Тому було застосовано ряд програмних оптимізацій, після чого було оцінено їх вплив на швидкодію відстежування. Після застосування всіх оптимізацій швидкодія склала 38 кадрів на секунду.

Загальна швидкодія системи під час фінального тестування мала найкращий результат при роздільній здатності 320 x 240 при розмірі цілі 64 x 64 і складала 27 кадрів на секунду під час супроводження.

Розроблене рішення може бути використаним для автоматизованого візуального супроводу об’єкта у режимі реального часу. Також у роботі описано можливі доповнення отриманого рішення, а також потенційні сфери застосування, серед яких охоронні системи відеоспостереження та стабілізація прицілу зброї за рухом візуального об’єкта.

Результати досліджень та розробок, проведених у кваліфікаційній роботі, були представлені на 8-й Східно-Європейській конференції «Математичні та програмні технології Internet of Everything», що відбулася в м. Києві 14 травня 2021 р.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Object tracking methods and their areas of application: A meta-analysis. A thorough review and summary of commonly used object tracking methods / Sanna Ågren. – 45 pages. – [Електронний ресурс]. – Режим доступу: http://www8.cs.umu.se/education/examina/Rapporter/SannaAgrenFinal.pdf. (дата звернення: 21.05.2021).

2. Handcrafted and Deep Trackers: Recent Visual Object Tracking Approaches and Trends / Mustansar Fiaz, Arif Mahmood, Sajid Javed, and Soon Ki Jung. – 36 pages. – [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/1812.07368.pdf. (дата звернення: 21.05.2021).

3. Object tracking: A survey / Yilmaz A., Javed O., and Shah M. – 2006. – 45 pages. – [Електронний ресурс]. – Режим доступу: https://dl.acm.org/doi/pdf/10.1145/1177352.1177355. (дата звернення: 21.05.2021).

4. Змагання «VOT Challenge». – [Електронний ресурс]. – Режим доступу: https://www.votchallenge.net/. (дата звернення: 22.05.2021).

5. Результати змагання «VOT 2017». – [Електронний ресурс]. – Режим доступу: https://www.votchallenge.net/vot2017/results.html. (дата звернення: 22.05.2021).

6. Результати змагання «VOT 2018». – [Електронний ресурс]. – Режим доступу: https://www.votchallenge.net/vot2018/results.html. (дата звернення: 22.05.2021).

7. Результати змагання «VOT 2019». – [Електронний ресурс]. – Режим доступу: https://www.votchallenge.net/vot2019/results.html. (дата звернення: 22.05.2021).

8. Результати змагання «VOT 2020». – [Електронний ресурс]. – Режим доступу: https://www.votchallenge.net/vot2020/results.html. (дата звернення: 22.05.2021).

9. ECO: Efficient Convolution Operators for Tracking / M. Danelljan et. al., In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017. – Режим доступу: https://openaccess.thecvf.com/content_cvpr_2017/papers/Danelljan_ECO_Efficient_Convolution_CVPR_2017_paper.pdf. (дата звернення: 22.05.2021).

10. Staple: Complementary Learners for Real-Time Tracking / L. Bertinetto et. al. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016. – Режим доступу: https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Bertinetto_Staple_Complementary_Learners_CVPR_2016_paper.html. (дата звернення: 23.05.2021).

11. C++ Qt framework. – [Електронний ресурс]. – Режим доступу: https://www.qt.io/. (дата звернення: 23.05.2021).

12. Open Source Computer Vision Library OpenCV. – [Електронний ресурс]. – Режим доступу: https://opencv.org/. (дата звернення: 23.05.2021).

13. The pigpio library. – [Електронний ресурс]. – Режим доступу: http://abyz.me.uk/rpi/pigpio/. (дата звернення: 23.05.2021).

14. OpenCV VideoCapture Class Reference – OpenCV 4.5.2 documentation. – [Електронний ресурс]. – Режим доступу: https://docs.opencv.org/4.5.2/d8/dfe/classcv_1_1VideoCapture.html. (дата звернення: 24.05.2021).

15. Molloy D. Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux. Indiana : John Wiley & Sons, Inc. 2016. pp. 625 – 626.

16. Raspberry Pi Camera Module – Raspberry Pi documentation. – [Електронний ресурс]. – Режим доступу: https://www.raspberrypi.org/documentation/raspbian/applications/camera.md. (дата звернення: 24.05.2021).

17. RaspiCam: C++ API for using Raspberry camera with/without OpenCV – AVA research group. – [Електронний ресурс]. – Режим доступу: https://www.uco.es/investiga/grupos/ava/node/40. (дата звернення:).

18. Measuring Performance with OpenCV – OpenCV documentation. – [Електронний ресурс]. – Режим доступу: https://docs.opencv.org/master/dc/d71/tutorial_py_optimization.html. (дата звернення: 24.05.2021).

19. C++ static local variables reference – CppReference. – [Електронний ресурс]. – Режим доступу: https://en.cppreference.com/w/cpp/language/storage_duration#Static_local_variables. (дата звернення: 24.05.2021).

20. OpenCV forEach function – OpenCV documentation. – [Електронний ресурс]. – Режим доступу: https://docs.opencv.org/master/d3/d63/

classcv_1_1Mat.html#a952ef1a85d70a510240cb645a90efc0d. (дата звернення: 24.05.2021).

21. Performance Optimization of DFT. – [Електронний ресурс]. – Режим доступу: https://docs.opencv.org/master/de/dbc/tutorial_py_fourier_transform.html. (дата звернення: 24.05.2021).

22. OpenMP 4.0 API C/C++ Syntax Quick Reference Card. – [Електронний ресурс]. – Режим доступу: https://www.openmp.org/wp-content/uploads/ OpenMP-4.0-C.pdf. (дата звернення: 24.05.2021).

23. SG90 Servo Datasheet. – [Електронний ресурс]. – Режим доступу: https://datasheetspdf.com/pdf/791970/TowerPro/SG90/1. (дата звернення: 25.05.2021).

24. Оптико-електронна система керування стрільбою артустановок «Sens-2». – [Електронний ресурс]. – Режим доступу: http://kvant-ukr.org/sens.html. (дата звернення: 27.05.2021).

ДОДАТКИ

Додаток А

Software Architecture Document

Visual object tracker system
Software Architecture Document (SAD)

CONTENT OWNER: Mykola Moroz

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
• 1.0.0	• 1.0.0	• 30.05.2021

Table of Contents

Documentation Roadmap

1.1 Document Management and Configuration Control Information

- Revision Number: 1.0.0
- Revision Release Date: 30.05.2021
- Purpose of Revision: first release of the document

1.2 Purpose and Scope of the SAD

This SAD specifies the software architecture for **visual object tracker system.** All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents. The Software Architecture Document (SAD) provides a comprehensive architectural overview of the Visual object tracker system. It presents a number of different architectural views to depict the different aspects of the system. These structures will be represented in the views of the software architecture that are provided in Section 3.

1.3 Viewpoint Definitions

As required by ANSI/IEEE 1471-2000, this SAD employs a stakeholder-focused, multiple view approach to architecture documentation. A viewpoint identifies the set of concerns to be addressed and a view is a viewpoint applied to a system. In current section are presented and defined viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns. In the next subsections each viewpoint is shortly presented.

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	Viewpoint(s) that apply to that class of stakeholder's concerns
Developer	Module Uses, Data flow, Deployment
Maintainer	Module Uses, Data flow, Deployment
End user	Data flow, Deployment
Acquirer	Module Uses, Deployment

1.3.1 Uses viewpoint definition

1.3.1.1 Abstract. Views conforming to the uses viewpoint partition the system into a unique non-overlapping set of hierarchically decomposable implementation units – modules and show which modules use which other modules. A module uses another module if its correctness depends on the correctness of the other

1.3.1.2 Stakeholders and Their Concerns Addressed. Stakeholders and their concerns addressed by this viewpoint include

- developers, who use the elements to produce a running version of the system;
- maintainers, who are tasked with modifying the software elements;
- acquirers, who are concerned that projected changes to the system over its lifetime can be made economically by confining the effects of each change to a small number of elements.

1.3.1.3 Elements, Relations, Properties, and Constraints. Elements of the uses viewpoint are modules, which are units of implementation that provide defined functionality. Modules connected with "uses" relation, whereby one module requires the correct implementation of another module for its own correct functioning. This view makes explicit which modules use which other modules to achieve their responsibilities. Properties of elements include their names, the functionality assigned to them and their software-to-software interfaces.

1.3.1.4 Language(s) to Model/Represent Conforming Views. UML, using subsystems or packages to represent elements (modules) and the uses relation can be showed as a dependency with the stereotype <<uses>>.

1.3.2 Data flow viewpoint definition

1.3.2.1 Abstract. Views conforming to the data flow (pipeline) viewpoint represent the system as a computational model in which components act as data transformers and connectors transmit data from the outputs of one component to the inputs of another.

1.3.2.2 Stakeholders and Their Concerns Addressed. Stakeholders and their concerns addressed by this viewpoint include

- developers, who use the elements to produce a running version of the system;
- maintainers, who are tasked with modifying the software elements;
- users, who will use the final system and need to know basic architecture overview.

1.3.2.3 Elements, Relations, Properties, and Constraints. Elements of the pipeline viewpoint are pipe and filter. Filter is a component that transforms data read on its input ports to data written on its output ports. Pipe is a connector that conveys data from a filter's output ports to another filter's input ports. The "attachment" relation associates filter output ports with data-in roles of a pipe, and filter input ports with data-out roles of pipes. Properties of elements include their names, the functionality assigned to them and the data transferred between them. The main constraint is that pipes connect filter output ports to filter input ports.

1.3.2.4 Language(s) to Model/Represent Conforming Views. (a) UML, activity diagram or (b) dataflow diagram.

1.3.3 Deployment viewpoint definition

1.3.3.1 Abstract. Deployment viewpoints allocate software elements which are native to a component & connector style to the hardware of the computing platform on which the software executes.

1.3.3.2 Stakeholders and Their Concerns Addressed. Stakeholders and their concerns addressed by this viewpoint include

- developers, who use the elements to produce a running version of the system;
- maintainers, who are tasked with modifying the software elements;
- users, who will use the final system and need to know basic architecture overview;
- acquirers, who are concerned that projected changes to the system over its lifetime can be made economically by confining the effects of each change to a small number of elements.

1.3.3.3 Elements, Relations, Properties, and Constraints. Elements of the deployment viewpoint are software elements and environmental elements. Software element is an elements from a component & connector view. Environmental element is a hardware of the computing platform — processor, memory, disk, network and so on. Possible relations are “allocated-to”, “migrates-to”, “copy-migrates-to”, “execution-migrates-to”. Properties of elements include their names, the functionality assigned to them and the data transferred between them. The allocation topology is unrestricted. However, the required properties of the software must be satisfied by the provided properties of the hardware.

1.3.3.4 Language(s) to Model/Represent Conforming Views. (a) informal graphical notations that use boxes, circles, lines, arrows, and so on to represent the software and environmental elements; (b) UML, a deployment diagram is a graph of nodes connected by communication associations; (c) Architecture Analysis and Design Language (AADL) or (d) SysML which are architecture description languages that provide formal notations for describing deployment views.

2 Architecture Background

2.1 Problem Background

2.1.1 System Overview

The purpose of the system which consists of combination of hardware and software components is the automated tracking of a visual object that is selected by a user in the video stream from camera in real-time. The tracker system must also have a graphical user interface for target selection and tracking progress visualization.

The method of user interaction with the tracker system is as follows. Images from the camera in real time are shown to the user. After some time and when the user will be ready, the user selects the desired object in the video stream by choosing the rectangular area in which the object is located. As a constraint, only one object is tracked at given time. After that, the user can stop tracking the selected object.

When user selected a target for tracking, the specified object is searched for in the next frames and the object is highlighted with a colored rectangular frame. Object tracking continues until a forced stop of tracking. This may be due to an interrupt received from the user, or if it is impossible to continue monitoring.

Also, it should be noted that target object can have any size, any form and shape and user selects an object manually by placing rectangular box over target. If camera and its software supports several image resolutions, resolution settings should be available to user. System should have wide range of view, so pan and tilt mechanisms should be added. And pan-tilt mechanism should center the target on the image and move the camera according to the movement of tracked object.

System should be compact and run on an embedded system. Cost of the final system (including software and hardware) must be cheap enough (~50$). System should be modular and flexible to allow further expansion and modification, support hardware and software component changes.

The key requirements for the tracker are:

- the target object can have any size, shape and color, because it is not known beforehand which object will need to be tracked;
- the target object may not have a certain defined trajectory of movement, which can be known in advance;
- the target object can be on any background, in any visual environment;
- the algorithm must be fast to be able to track target in real time, ie the program must display the processed frames at a speed of 20 frames per second or more;
- as camera has small angles of view system also must be supplied with pan-tilt mechanism;
- tracker system should run on an embedded system.

2.1.2 Goals and Context

As described in previous section, solution should be modular to support extension of the system and components change. For solution to be modular software architecture should be modular too. From description several modules can be formed: camera capture module, GUI module, tracker module, pan-tilt control module. These modules can be changed in future (in software or hardware part), so every module should have one stable interface.

Solution should work with required speed (20+ FPS) so these constraints on development tools and requires for efficient tracking algorithm and camera capture method.

As user should have ability to select target with rectangular box some software methods in GUI should be provided. GUI should consist of camera output and simple interface for target selecting and for tracking cancel. User interface should support settings with camera resolution, and there should be ways to set it from GUI to camera.

Pan-tilt mechanism can be built with servomotors. This is the part that is easily can be changed in hardware. So there should be general interface and ways for pan-tilt module expansion. Pan-tilt should be connected to camera and move it according to target moves. Also module should center target, so special routines should be provided.

System can be presented as a pipeline. Images are taken from camera, then they are processed, location of target found and is shown to user, pan-tilt centers target.

2.1.3 Significant Driving Requirements

The main key requirements are speed of the solution (20+ FPS) and architecture modularity. The former can be achieved with efficient tracking algorithm (or with some optimizations) and fast image capturing and processing routines. The latter can be achieved with modular style of system designing, and with required level of control of cohesion and coupling between software classes.

2.2 Solution Background

2.2.1 Architectural Approaches

Main architectural style for the system is a pipeline pattern architecture as there are stages (capturing, tracking, showing, pan-tilt moving) that are based on previous stages and are input to next stages. Every stage can be represented as separate module that has some input and output. In main module these modules are connected to form pipe pattern. Views selected in Section 1.3 (uses and data flow viewpoints) help to understand given architectural decisions and approaches.

Pipeline architecture in some cases allows for parallel execution of stages. And in given solution threading is also used to speed up general execution time.

Modularity of architecture can be achieved with help of software modules. From system description can be formed next modules: tracker module, image processing module, camera capture module, GUI module and pan-tilt control module. Each module performs one task and can be extended or modified to meet new changes that can be presented with new requirements, hardware and/or software changes. Each module has open external interface seen to other modules to reduce coupling between modules and increase cohesion inside modules.

2.2.2 Analysis Results

To provide evidence that software architecture is fit for required purpose several analyses were made. To measure the processing speed of the solution it was tested with in-build software functions for time measurement. In real-time every 100 captured frames are benchmarked. Count of captured frames is divided on processing time and finally a FPS value found. In result solution showed 21 FPS with image resolution 640x480 pixels and target size 64x64 pixels and 27 FPS with image resolution 320x240 pixels and target size 64x64 pixels.

To measure modularity metrics to measure cohesion and coupling were used. For coupling metrics CE (efferent coupling) and CA (afferent coupling) were used. For cohesion metric LCOM was used. As solution is small enough and has small amount of classes metrics show small values, which are interpreted as good results.

2.2.3 Requirements Coverage

Requirements addressed by the software architecture are: processing speed of the system and system modularity. Processing speed is achieved with efficient tracking algorithm in tracking module and in using threads for parallel execution of stages. System modularity is addressed in flexibility, open for extension, possible component changes requirements and this aspect is achieved with module-based design of software.

The main requirements are covered and results of analysis provided in previous section.

3 Views

This section contains the views of the software architecture described in Section 1.3.

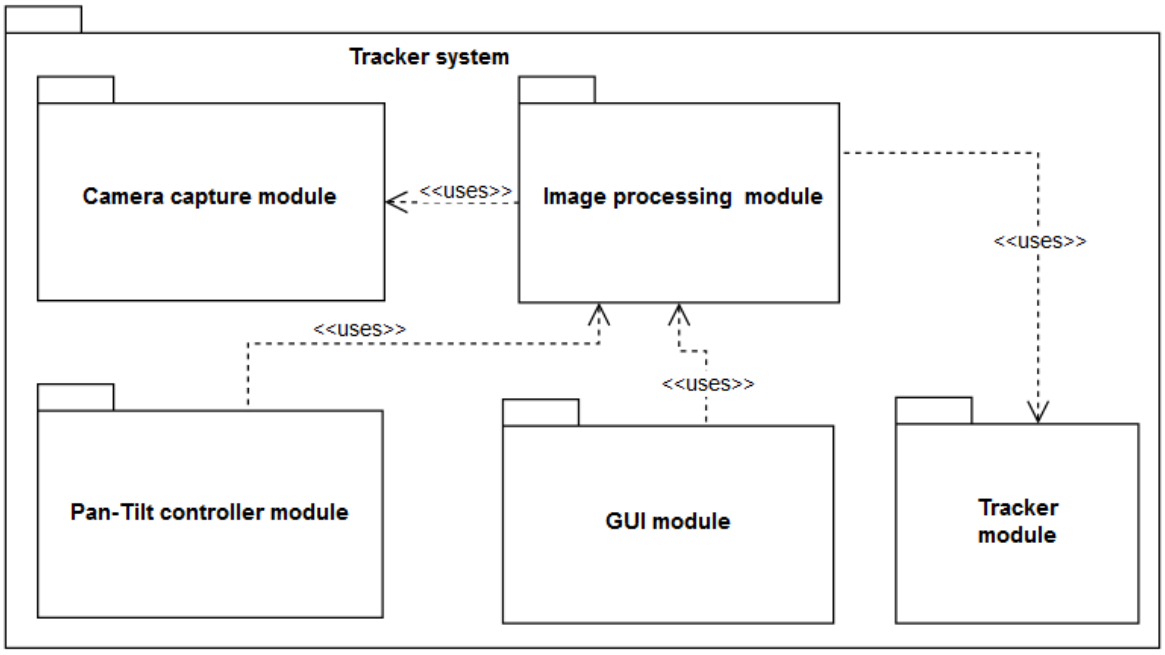
3.1 Uses View

3.1.1 View Description

This view partitions the system into a unique non-overlapping set of hierarchically decomposable modules and shows the relations between them. The modules that described earlier are given on a diagram.

3.1.2 View Overview

View represented as diagram built based on UML package diagram notation.



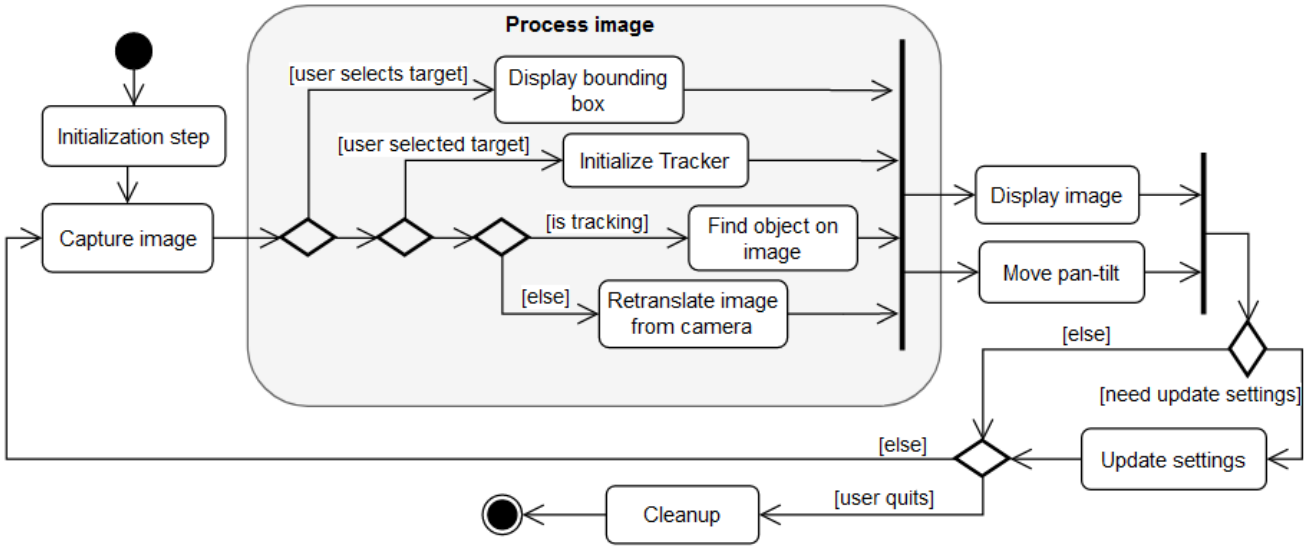
3.2 Data Flow View

3.2.1 View Description

This view represents the system as a computational model in which components act as data transformers and connectors transmit data from the outputs of one component to the inputs of another. System in given view presented as pipeline of modules described earlier.

3.2.2 View Overview

View represented as diagram built based on UML activity diagram notation.



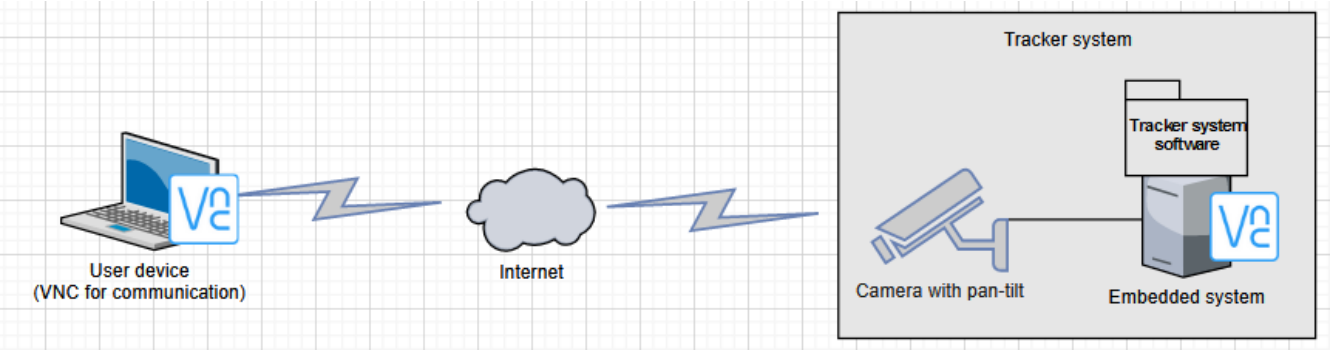
3.3 Deployment View

3.3.1 View Description

This view allocates software elements which are native to a component & connector style to the hardware of the computing platform on which the software executes.

3.3.2 View Overview

View represented as diagram built with informal graphical notations that use boxes, circles, lines, arrows, and so on to represent the software and environmental elements. VNC server used for remove control of embedded system from user's work stations.



4 Referenced Materials

Clements 2010	Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford, *Documenting Software Architectures: Views and Beyond*, Addison Wesley Longman, 2010.
IEEE 1471	ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000.

5 Directory

5.1 Glossary

Term	Definition
software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
view	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
viewpoint	A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.

5.2 Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
IEEE	Institute of Electrical and Electronics Engineers
OS	Operating System
GUI	Graphical user interface
SAD	Software Architecture Document
UML	Unified Modeling Language