МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Автоматизована система відстежування об'єктів у реальному часі з використанням технології Apriltag**

Виконав: студент 2 курсу, групи 8.1212-іпз-2
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
О.О. Поляков
(ініціали та прізвище)
Керівник доцент, к.т.н., Н.П. Полякова
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ Дісітел П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра __електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Тетяна Критська_____
"_01_" __вересня_____2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

___Полякову Олександру Олександровичу___
(прізвище, ім'я, по батькові)

1.Тема роботи __Автоматизована система відстежування об'єктів у реальному часі з використанням технології Apriltag_____

керівник роботи ____Полякова Наталія Петрівна, доцент, кандидат технічних наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 09.10.2023 №1577-с_____

2. Строк подання студентом кваліфікаційної роботи_____30.11.2023_____
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми відстеження об'єктів у реальному часі;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми і розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
16 слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.09.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	02.09-10.09.23	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.09-12.09.23	виконано
3	Аналіз існуючих методів рішення	13.09-14.09.23	виконано
4	Дослідження засобів виявлення об’єктів на зображеннях	15.09-20.09.23	виконано
5	Дослідження засобів виявлення тегів Apriltag	21.09-26.09.23	виконано
6	Узгодження подальших дій з науковим керівником	27.09-28.09.23	виконано
7	Збір зразків відеоматеріалу для виконання роботи та його попередня підготовка	29.09-13.10.23	виконано
8	Реалізація функціоналу для отримання сканованих тегів за допомогою підготовленого відеоматеріалу.	14.10-16.10.23	виконано
9	Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження	17.10-19.10.23	виконано
10	Реалізація функціоналу для отримання положення тегу в кадрі та перетворення координат тегу у систему моделі.	20.10-09.11.23	виконано
11	Порівняння вихідних характеристик роботи системи за допомогою вхідних тестових відео	10.11-17.11.23	виконано
12	Реалізація користувацького інтерфейсу для комп’ютерної системи	18.11-22.11.23	виконано
13	Оформлення звіту	23.11-27.11.23	виконано

Студент _____ Поляков О.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Полякова Н.П.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 83

Рисунків: 21

Таблиць: 5

Джерел: 17

Поляков О. О. Автоматизована система відстежування об’єктів у просторі і часі з використанням технології Apriltag : кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник Н. П. Полякова. Запоріжжя : ЗНУ, 2023. 83 с.

Мета і завдання дослідження полягають у вивченні нових способів відстежування об’єктів у просторі та часі, а також у розробці комп’ютерної системи, яка буде здатна вирішувати цю проблему, використовуючи відео потік з камери та технологію Apriltag.

У процесі дослідження була розглянута проблема відстежування об’єктів у просторі та часі та використання технології Apriltag для її вирішення. Результатом цього дослідження є розроблена система, яка здатна відстежувати об’єкти за допомогою технології Apriltag та визначати їх координати у просторі, використовуючи в якості вхідних даних відео потік з камери. Також був розроблений механізм відображення отриманих результатів на 3D моделі.

Ключові слова: комп’ютерна система, розпізнавання об’єктів, визначення положення об’єктів, Комп’ютерний зір, Apriltag, Open CV, ThreeJS.

SUMMARY

Pages: 83

Drawings: 21

Tables: 5

Source: 17

Poliakov O. O. Automated tracking system of objects in space and time using Apriltag technology: qualification work of the master of specialty 121 "Software engineering" / Science head advisor N. P. Poliakova. Zaporizhzhia: ZNU, 2023. 83 p.

The goal and task of the research is to study new ways of tracking objects in space and time, as well as to develop a computer system that will be able to solve this problem using the video stream from the camera and the Apriltag technology.

In the process of research, the problem of tracking objects in space and time was considered and the use of Apriltag technology for its solution. The result of this research is a developed system that is able to track objects using Apriltag technology and determine their coordinates in space, using the video stream from the camera as input data. A mechanism for displaying the obtained results on a 3D model was also developed.

Keywords: computer system, object recognition, object position determination, Computer vision, Apriltag, Open CV, ThreeJS.

ЗМІСТ

ВСТУП

Актуальність теми

З того часу як комп’ютерні технології стали більш розповсюджені та дешевші, а комплектуючі та комп’ютерна периферія стали більш доступні, перед людством постало питання автоматизації багатьох процесів. Однією з таких задач є відстежування положення об’єкту або продукції у просторі та часі.

Система відстежування об’єктів у реальному часі може застосовуватись у різних сферах, де потрібно відстежувати значні об’єми об’єктів у замкнутому просторі, або лінії товарів з замкнутим циклом виробництва. Систему можливо застосовувати на підприємствах, складах та фірмах, оскільки саме ці установи частіше за все страждають через неякісне відстежування товару та отримують великі збитки через це.

Застосування GPS датчиків або антен є дуже затратним та веде до додаткових фінансових витрат. Використання RTSP потоку з камери є розумною альтернативою, яка зменшує кінцеві витрати на усю систему в цілому, а використання тегів для машинного зору, створених спеціально для робототехніки допомагає зменшити вимоги до потужності апаратного комплексу на якому вона буде працювати.

Таким чином є актуальним створення комп’ютерної системи для вистежування товару у реальному часі застосовуючи RTSP потік з відеокамери та систему AprilTags.

Мета і завдання дослідження

Мета і завдання полягають у вивченні сучасних підходів до визначення положення товару у реальному часі в обмеженому просторі, а також у розробці системи, що буде ефективно вирішувати цю задачу, застосовуючи потоковий протокол реального часу (Real Time Streaming Protocol, RTSP) з відеокамери або звичайний відео потік та систему AprilTags.

Об’єкт дослідження

Об’єктом дослідження є потік відеокадрів з камери.

Предмет дослідження

Предметом дослідження є визначення положення об’єкту у обмеженому просторі та у реальному часі на основі потоку відеокадрів камери.

Методи дослідження

Для вирішення поставленої задачі використовуються наступні методи дослідження:

1. Аналіз особливостей та існуючих рішень для проблеми визначення положення об’єкту в обмеженому просторі.
2. Аналіз способів розпізнавання об’єктів у реальному часі.
3. Аналіз бібліотек комп’ютерного зору та розпізнавання Apriltags.
4. Аналіз RTSP потоку з наявними тегами.
5. Аналіз методів визначення положення об’єктів відносно обмеженого простору.
6. Синтез отриманих знань в процесі дослідження проблеми та методів її вирішення.

7. Експериментування з використанням різних підходів для прискорення роботи системи.
8. Експериментування зі зміною параметрів роботи розробленої системи.

Наукова новизна одержаних результатів

Наукова новизна одержаних результатів дослідження полягає у тому, що для вирішення задачі визначення положення об’єкту в обмеженому просторі у реальному часі був використаний новий та ефективний підхід, а саме: була розроблена система для розпізнавання об’єкту за допомогою AprilTags, що працює у парі з алгоритмом визначення положення об’єкту відносно обмеженого простору та відображення на 3D моделі.

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає у тому, що була розроблена комп’ютерна програма, яка у реальному часі визначає положення товару у просторі, використовуючи перетворення координат. Дана система є дуже дешевою з точки зору апаратного забезпечення та дозволяє розгорнути систему не використовуючи додаткових засобів та високопродуктивного обладнання.

Глосарій

Розпізнавання об’єктів (англ. Object detection) — це визначення положення об’єктів на зображенні за допомогою обмежувальних рамок, а також вірогідності їхньої приналежності до того чи іншого класу.

AprilTag — це візуальна довірча система, корисна для широкого спектру завдань, включаючи доповнену реальність, робототехніку та калібрування камери. Мішені можна створювати на звичайному принтері, а програмне забезпечення для виявлення AprilTag обчислює точне 3D-положення, орієнтацію та ідентичність тегів відносно камери. Бібліотека AprilTag реалізована на C без зовнішніх

залежностей. Бібліотека призначена для легкого включення в інші програми, а також для перенесення на вбудовані пристрої. Продуктивність у реальному часі може бути досягнута навіть на процесорах рівня мобільних телефонів.

CMake — це кросплатформна система генерації файлів, необхідних для компіляції програмного забезпечення із вихідного коду.

OpenCV — це відкрита бібліотека комп'ютерного зору, що включає у себе алгоритми для обробки та перетворення зображень, функціонал для роботи з відео, а також модуль для використання глибоких нейронних мереж класифікації, розпізнавання, виявлення ключових точок та сегментації об'єктів на зображенні.

Python — це інтерпретована об'єктно-орієнтована мова програмування високого рівня із динамічною типізацією.

Three.js — бібліотека кроссбраузера JavaScript, що використовується для створення і відображення анімованої комп'ютерної 3D графіки при розробці вебдодатків. Three.js скрипти можуть використовуватись спільно з елементами HTML5 CANVAS, SVG або WebGL.

YOLO (You Only Look Once) — популярна модель виявлення об'єктів і сегментації зображення, розроблена Джозефом Редмоном і Алі Фархаді з Вашингтонського університету. YOLOv8 — остання версія YOLO від Ultralytics. YOLOv8 підтримує повний спектр завдань штучного інтелекту зору, включаючи виявлення, сегментацію, оцінку пози, відстеження та класифікацію.

BLE – технологія бездротової персональної мережі, розроблена Bluetooth Special Interest Group (Bluetooth SIG), призначена для нових застосунків у галузі охорони здоров'я, фітнесу, радіомаяків, безпеки та індустрії домашніх розваг. Ця технологія не залежить від класичного Bluetooth, але Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) і LE можуть співіснувати. Оригінальна специфікація була розроблена Nokia у 2006 році під назвою Wibree, яка була інтегрована в Bluetooth 4.0 у грудні 2009 року як Bluetooth Low Energy.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ ВИЗНАЧЕННЯ ПОЛОЖЕННЯ ОБ'ЄКТУ В ОБМЕЖЕНОМУ ПРОСТОРІ У РЕАЛЬНОМУ ЧАСІ

1.1 Огляд проблеми визначення положення об'єкту в обмеженому просторі у реальному часі

Відстеження об'єктів у замкнутому просторі є на разі актуальною темою та метою багатьох наукових досліджень. Існують багато способів відстежувати об'єкт у замкнутому просторі - ми бачили системи локації в приміщеннях на основі інфрачервоного випромінювання, ультразвуку, вузькосмугового радіо, потужності сигналу WiFi, UWB, зору та багатьох інших [1].

Багато систем працюють використовуючи технологію YOLO. Проте, якщо об'єкти будуть виглядати майже однаково, є велика вірогідність, що нейронна мережа буде видавати результат, що містить дуже багато false-positive результатів. Інші системи, що використовують технології інфрачервоного випромінювання, ультразвуку, вузькосмугового радіо, потужності сигналу WiFi стикаються з низкою проблем, характерних для них, а саме:

1. Сильна багатопроменевість через відбиття сигналу від стін та меблів.
2. Необхідність прямої видимості.
3. Ослаблення або розсіювання сигналу через архітектуру будівлі.
4. Необхідність високої точності роботи системи.

Через це на даний момент ми не маємо дешевої та надійної системи для роботи у замкнутому просторі для відстеження та з можливістю обробки великого масиву об'єктів з високою точністю.

1.2 Технології для визначення положення об’єкту в обмеженому просторі у реальному часі

Для вирішення проблеми розпізнавання об’єктів більшість систем, розрахованих на відстежування об’єктів такого типу використовують технологію BLE. Технологія BLE знаходить широке застосування у багатьох господарських галузях. Одним із ключових напрямів її використання є відстеження активів усередині приміщень. Впроваджуючи пристрої трекінгу по Bluetooth в інфраструктуру підприємства, можна виконувати моніторинг об'єктів, що рухаються в режимі реального часу і отримувати детальну аналітику переміщень для оптимізації роботи компанії. До недоліків цієї системи можно віднести велику вартість багатьох маячків та невисоку точність (до 1 м). Джефрі Хайтауер у своєму дослідженні [2] наводить перелік систем, що можуть використовуватись для вирішення цієї проблеми:

1. Активні бейджі (точність у межах кімнати), проте ця система не підтримує велику масштабованість та є велика вірогідність похибок через сонячне світло, що буде заважати інфрачервоному випромінюванню.
2. Технологія Active Bats (точність від 9 см). Для цієї технології потрібна стельова сенсорна сітка та дуже велика кількість сенсорів.
3. Технологія Motion Star (точність від 1 мм). Технологія потребує дуже високопродуктивного апаратного забезпечення та дуже точного встановлення контрольного апарату.
4. Технологія VHF, що має майже ідеальну точність, але потребує дуже дорогої інфраструктури, а її поле роботи обмежується полем зору.
5. Технологія Cricket має покриття 4x4 фути, проте не має централізованого менеджменту.

Інші системи, що перелічені у цьому дослідженні мають незадовільну точність (>1м), тому не приводяться.

В таблиці 1 зображені усі системи, що використовуються у цьому дослідженні [2].

Таблиця 1 – Системи, що використовуються для відстежування об’єктів у замкнутому просторі

Технологія	Принцип дії	Точність
Активні беджі	Дифузна інфрачервона стільникова близькість	Розмір кімнати
Active Bats	Ульразвуковий час підльоту	9 см (95%)
MotionStar	Аналіз сцени, затримка	1 мм, 1мс, 0,1 градус
Радар MSR	802.11 RF аналіз сцени та триангуляція	3-4,3м (50%)
PinPoint 3D-iD	Затримка RF	1-3м
Easy Living	Машинний зір, триангуляція	Варіативно
Smart Floor	Фізичний контакт	Проміжки між сенсорами тиску

До недоліків перерахованих систем можна віднести велику вартість додаткового обладнання, специфічний масштаб та обмеження фізичного характеру. Згідно дослідження [2] Джефрі Хайтауера перелічені методи мають недоліки. Детально про ці обмеження вказано у таблиці 2.

Таблиця 2 – Таблиця обмежень систем, що використовуються для відстежування об’єктів у замкнутому просторі

Технологія	Масштаб	Вартість	Обмеження
Активні беджі	1 станція на кімнату, 1 бейдж на 1 станцію впродовж 10 сек.	Адміністративні витрати, бейджі та бази	Сонячне світло та флуоресцентне світло перешкоджає інфрачервоному випромінюванню
Active Bats	1 база на 10 квадратних метрів, 25 розрахунків на кімнату в секунду	Адміністративні витрати, бейджі, бази та сенсори	Потрібна сенсорна решітка на стелі
Motion-Star	Контролер на сцену, 108 сенсорів на сцену	Контрольовані сцени, велика вартість обладнання	Точна установка
Радар MSR	3 бази на поверх	Встановлення мережі 802.11, близько 100 бездротових NIC	Потрібні бездротові NIC
Pin-Point 3D-iD	Декілька баз на будівлю	Інфраструктурне встановлення, велика вартість обладнання	Перешкоди від 802.11
Easy Living	3 камери на маленьку кімнату	Потужність обробки, встановлення камер	Вразливість камер
Smart Floor	Сенсорні грати на усю кімнату	Інсталяція сенсорних грат, створення датасету тиску ніг	Точність розпізнавання не масштабується до великих натовпів.

Доречно буде привести дані з дослідження Райнера Маутца у роботі [3]. Наведено 13 найбільш популярних технологій для визначення положення об’єктів у обмеженому просторі. Усі дані приведені у таблиці 3.

Таблиця 3 – Технології для визначення положення об’єктів в обмеженому просторі за точністю роботи та областю покриття

Технологія	**Точність**	**Покриття (м)**
Камери	0.1 мм – 1 дм	1 – 10
Інфрачервоне випромінювання	1 см – 1 м	1 – 5
Тактильні та комбіновані полярні системи	1 мкм – 1 мм	3 – 2000
Звук	1 см	2 – 10
Wi-Fi	1 м	20 – 50
RFID	1 дм – 1 м	1 – 50
Ultra-Wideband	1 см – 1 м	1 – 50
Високочутливі супутникові системи навігації	10 м	Глобальні
Псевдоліти	1 см – 1 дм	10 – 1000
Інші радіочастоти (ZigBee, Bluetooth тощо)	1 м	10 – 1000
Інерціальна навігація	1 %	10 – 100
Магнітні системи	1 мм – 1 см	1 – 20
Інфраструктурні системи	1 см – 1 м	Будівля

Використання камер для вирішення цієї проблеми є найбільш оптимальним рішенням, бо використання камер дозволяє отримати задовільну точність (від 0.1 мм), не потребує розгортання додаткової інфраструктури, не потребує додаткового апаратного або програмного забезпечення для роботи з ними. Сучасні технології машинного зору дозволяють використовувати RTSP потік таких камер для знаходження шуканого об’єкту.

1.3 Сфери застосування визначення положення об’єкту в обмеженому просторі у реальному часі

Вирішення проблеми визначення положення об’єкту в обмеженому просторі у реальному часі є зараз дуже популярним питанням у різноманітних наукових колах та є дуже затребуваним великим та малим бізнесом. Багато передових компаній світу займається вирішенням цього питання для різноманітних потреб.

Система, що буде визначати положення об’єкту у реальному часі та транслювати на інтерактивну мапу знайде великий попит на багатьох підприємствах із замкнутим циклом виробництва, складах, торгівельних центрах та інших установах, де є переміщення великої кількості об’єктів у однаковій або дуже схожій тарі.

Комерційно значимими для масового ринку є інтерактивні сервіси моніторингу продукції. Велика кількість виробництв мають дуже застарілий спосіб відстежування переміщення продукції та чіткого слідування технологічним процесам. Наприклад, система може визначати положення товару у просторі, прорахувати його дійсний технологічний маршрут та прислати повідомлення на смартфон керівника, що продукція знаходиться у правильному або неправильному місці. За допомогою маркування з використанням бази даних об’єкт на підприємстві буде мати зрозумілий та прозорий технологічний процес, який виключає або значно зменшує вірогідність помилки при виконанні особливо трудомістких та коштовних операцій. Це дозволить значно пришвидшити багато технологічних процесів та упорядкувати контроль за продукцією.

Іншим прикладом може бути розміщення товару на складах, де можливо відстежувати велику кількість контейнерів та записувати їх місцезнаходження у базі даних, сповіщати повідомленнями через смартфон або e-mail. Керівник складу або уповноважена особа зможе у реальному часі керувати на інтерактивній мапі розташуванням усіх об’єктів та буде переконана, що усі товари знаходяться у потрібних локаціях.

У лікарнях дуже корисним буде відстежування контейнерів з цінними ліками або іншим медичним обладнанням. Інтерактивна мапа дозволить пришвидшити

знаходження необхідних ліків, обладнання та інструментів. З внесенням до бази даних працівника, що отримує вантаж ця система дозволить зробити облік у цій сфері більш прозорим та зрозумілим. Це дозволить покращити час на виконання екстрених операцій та облік препаратів, що використовуються.

1.4 Сучасні підходи комп'ютерного зору до визначення положення об'єкту в обмеженому просторі у реальному часі

Для визначення положення об'єкту у обмеженому просторі у реальному часі та демонстрації на 3D мапі потрібно вирішити декілька питань:

1. Як розпізнати певний з багатьох однакових контейнерів?
2. Як визначити положення об'єкту у системі камери?
3. Як трансформувати координати з системи камери до системи 3D мапи?

Для вирішення першого питання було обрано систему тегів Apriltags. Дані теги вперше були використані у робототехніці для калібрування роботів. Вони добре розпізнаються камерами, можливо отримати ідентифікаційний номер тегу, його якірні кути та знайти обмежувальну рамку навколо нього. Положенням контейнеру можливо враховувати положення центру тегу. Для вирішення другого питання потрібно визначити фокусну довжину камери та виконати перетворення координат тегу у систему камери, відносно її центру. Для вирішення третього питання обрано розрахунок, що базується на матриці обертання R та вектору трансформації t.

1.4.1 Ефективність використання тегів для розпізнавання об’єктів

Система маркерів AprilTag, зображених на рисунку 1, була розроблена April Robotics Laboratory Мічиганського університету (Олсон, 2011).

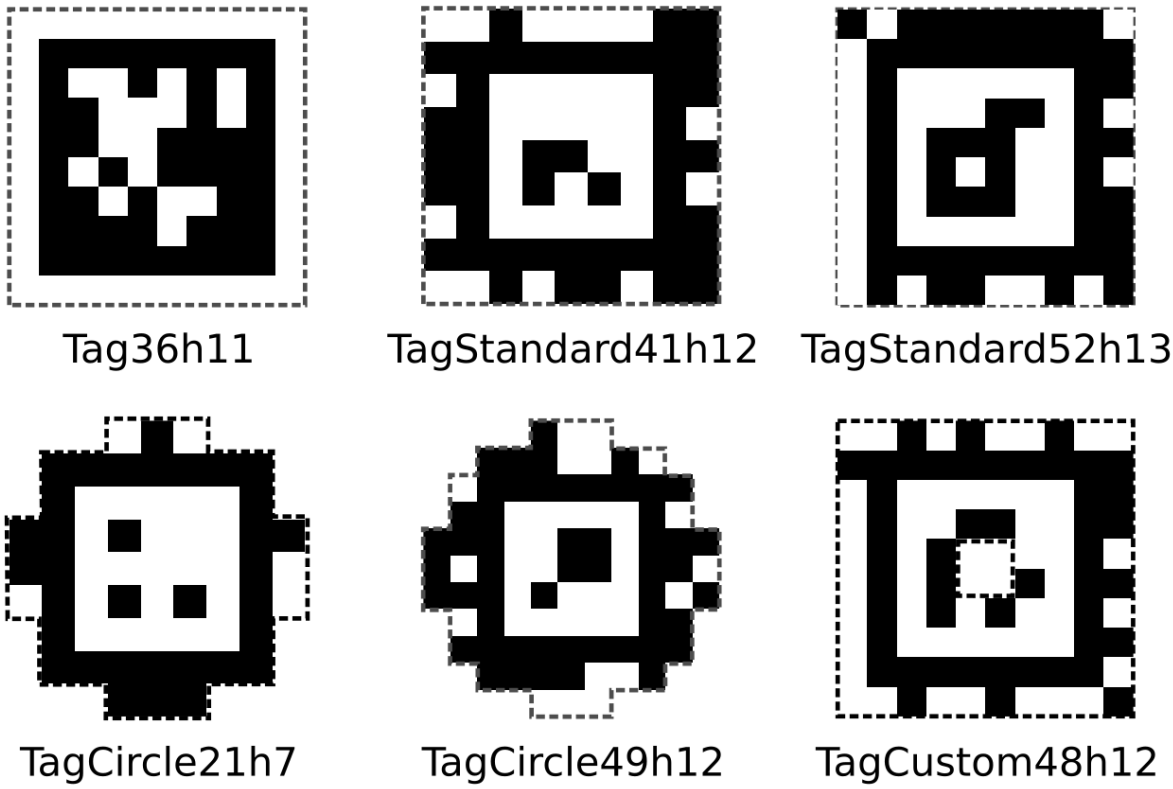

Рисунок 1 – Основні сімейства Apriltag

AprilTag застосовується для широкого кола завдань: калібрування камери, робототехніка, доповнена реальність тощо. Він дозволяє розрахувати точне положення, орієнтацію та ідентичність маркера відносно камери. Процес виявлення складається з кількох етапів: пошук лінійних сегментів, виявлення квадратів, обчислення положення та орієнтації мітки, декодування штрих-коду. Пошук спрямованих лінійних сегментів використовується, подібно до підходу ARTtag, а потім послідовності сегментів обробляються для формування квадрата. Виявлення квадратів застосовує рекурсивний 4 – рівневий пошук глибини, і на кожному рівні дерево додає одну сторону квадрата. На етапі ідентифікації перевіряється дійсність штрих-коду всередині виявленого маркера. Щоб закодувати внутрішнє зображення, AprilTag використовує систему лексикодів, що характеризується двома

параметрами: кількістю бітів кодового слова (внутрішнього шаблону) та мінімальною відстанню Хеммінга між будь-якими двома кодами. Lexicode генерує коди для тегів, що дозволяє виявляти та виправляти помилки бітів, наприклад: «Tag36h11» означає 36-бітний маркер (масив 6x6) з мінімальною відстанню Хеммінга в 11 біт між будь-якими двома кодами; «Tag16h5» відноситься до 16-бітного маркера (масив 4x4) з мінімальним значенням Хеммінга 5 біт між ними. будь-які два коди. Система AprilTag характеризується збільшеною кількістю різних кодів, збільшеною кількістю бітових помилок, які можна виявити та виправити, зменшенням помилкових спрацьовувань і плутанини між тегами, зменшеною загальною кількістю бітів у тегу та зменшеним розміром маркера.

1.4.2 Задача розпізнавання об’єкту у системі камери

Сайєд Мухаммад Аббас та Салман Аслан у дослідженні [4] розглянули принцип за яким працює механізм визначення положення тегу у системі камери. AprilTag використовує вбудований 2D-кодований маркер для виявлення тегів і для того, щоб відрізняти його від інших тегів. Візуальний маркер може бути будь-якого розміру з квадратним розміром. Бирка надрукована на білому фоні з чорним контурним квадратом. Усередині квадрата вбудований чорний штрих-код. AprilTag використовує унікальний алгоритм виявлення для швидкого, надійного розпізнавання та мінімізації впливу невеликих оклюзій. На рисунку 3 показано кроки алгоритму AprilTag. На першому кроці він обчислює величину та напрямок градієнта в кожному пікселі зображення, яке містить AprilTag. Для визначення лінійних сегментів використовуючи метод найменших квадратів на кластерах з схожими піксельними градієнтами. Після цього ці обчислені градієнти групуються в кластери, які називаються компонентами, на основі подібних атрибутів градієнта за допомогою методу на основі графіків. Беручи за основу напрямок градієнту на зображенні визначаються всі можливі квади – квадратні сегменти однакового розміру. Використовуючи метод зважених найменших квадратів, лінія підбирається для кожного компонента таким чином, що напрямок градієнтів визначає напрямок підігнаної лінії. Крім того, напрямок градієнта визначає напрямок відрізків лінії. Отже, кожна

лінія має темну сторону ліворуч і світлішу сторону праворуч. Крім того, після ідентифікації всіх ліній виявляються можливі чотирикутні фігури, як показано на кроці 3 на рисунку 2. Квад з правильною кольоровою схемою витягується для визначення позиції об’єкту. Витягується квадратна фігура з правильною кодовою схемою. Крім того, положення мітки з шістю ступенями свободи (6 degrees of freedom, 6DOF) у системі відліку камери повертається за допомогою гомографії та внутрішньої оцінки над витягнутим тегом а отже розташування тегу у системі камери повертається, використовуючи гомографічну та внутрішню оцінку.

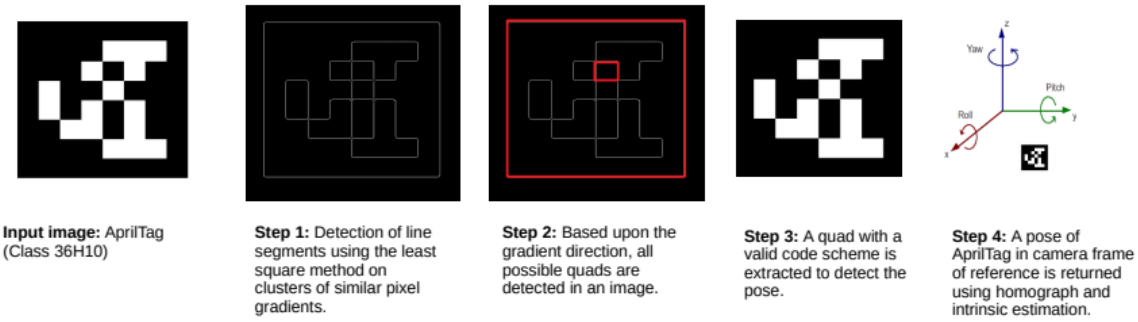

Рисунок 2 – Кроки алгоритму Apriltag.

1.4.3 Задача визначення положення розпізнаної об’єкту у системі 3D моделі

Після розпізнавання тегу у кадрі та визначення його положення у системі камери, необхідно визначити де саме знаходиться об’єкт, щоб коректно відобразити його на 3D моделі. Необхідно розуміти, що камери, які розташовані у робочому просторі мають різне положення, кут нахилу, поле зору та фокусну відстань. Відповідно до цього отримуємо питання: 1 м від центра камери – це скільки метрів або сантиметрів реального простору? Для вирішення цього питання ми звернемося до дослідження [5]. Пошук оптимального/найкращого повороту та переміщення між двома наборами відповідних даних 3D-точок, щоб вони були вирівняні/зареєстровані, є поширеною проблемою. На рисунку 3 наведено ілюстрацію задачі для найпростішого випадку з 3 відповідними точками (мінімальна кількість необхідних точок для вирішення).

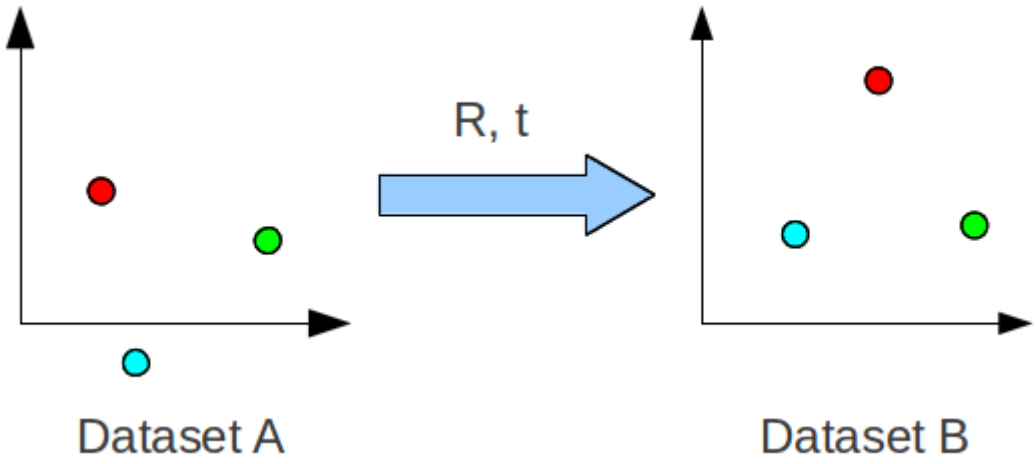

Рисунок 3 – Ілюстрація задачі трансформації точок

Відповідні точки мають однаковий колір, R – обертання, а t – переміщення. Необхідно знайти найкраще обертання та трансляцію, які вирівняють точки в наборі даних A з набором даних B. Тут «оптимальний» або «найкращий» є з точки зору помилок найменших квадратів. Це перетворення іноді називають евклідовим або жорстким перетворенням, оскільки воно зберігає форму та розмір. Це на відміну від афінного перетворення, яке включає масштабування та зсув.

Використання цього методу дозволяє виконувати таке перетворення багато разів знаючи тільки точки, які відповідають положенню у системі камери, матрицю обертання R, що буде унікальною для кожної з камер та вектор трансформації t, що також буде унікальним.

1.5 Аналіз програмного забезпечення для визначення положення людини в обмеженому просторі у реальному часі

1.5.1 Рішення від компанії Oriient

Oriient – це ізраїльська компанія, що спеціалізується на сфері визначення положення людей у замкнутому просторі. Також компанія пропонує рішення (рисунок 4) для торгових центрів, молів, різноманітних приміщень та офісних будівель. У якості технологій для позиціонування вони використовують магнітне поле Землі, яке існує навколо нас, а геомагнетизм — це наука про властивості цього магнітного поля. Усередині приміщення кожна будівля вносить унікальне спотворення магнітного поля Землі зі своїми власними магнітними особливостями. Orient використовує датчики, які вже є всередині смартфонів, щоб ідентифікувати та записувати ці конкретні внутрішні магнітні ландшафти, що дозволяє нам створити карту місця, просто прогулюючись зі смартфоном у руках, а потім точно визначити розташування інших смартфонів у тій самій будівлі. виключно на основі показань приладів. Компанія Orient розробила найсучасніші алгоритми, які дозволяють здійснювати позиціонування в приміщенні лише за допомогою програмного забезпечення. Запатентована технологія, яка зіставляє магнітні показання з базою даних будівель, дозволяє точно визначити місцезнаходження будь-якого смартфона, який рухається всередині будь-якої будівлі, у будь-якій точці світу.

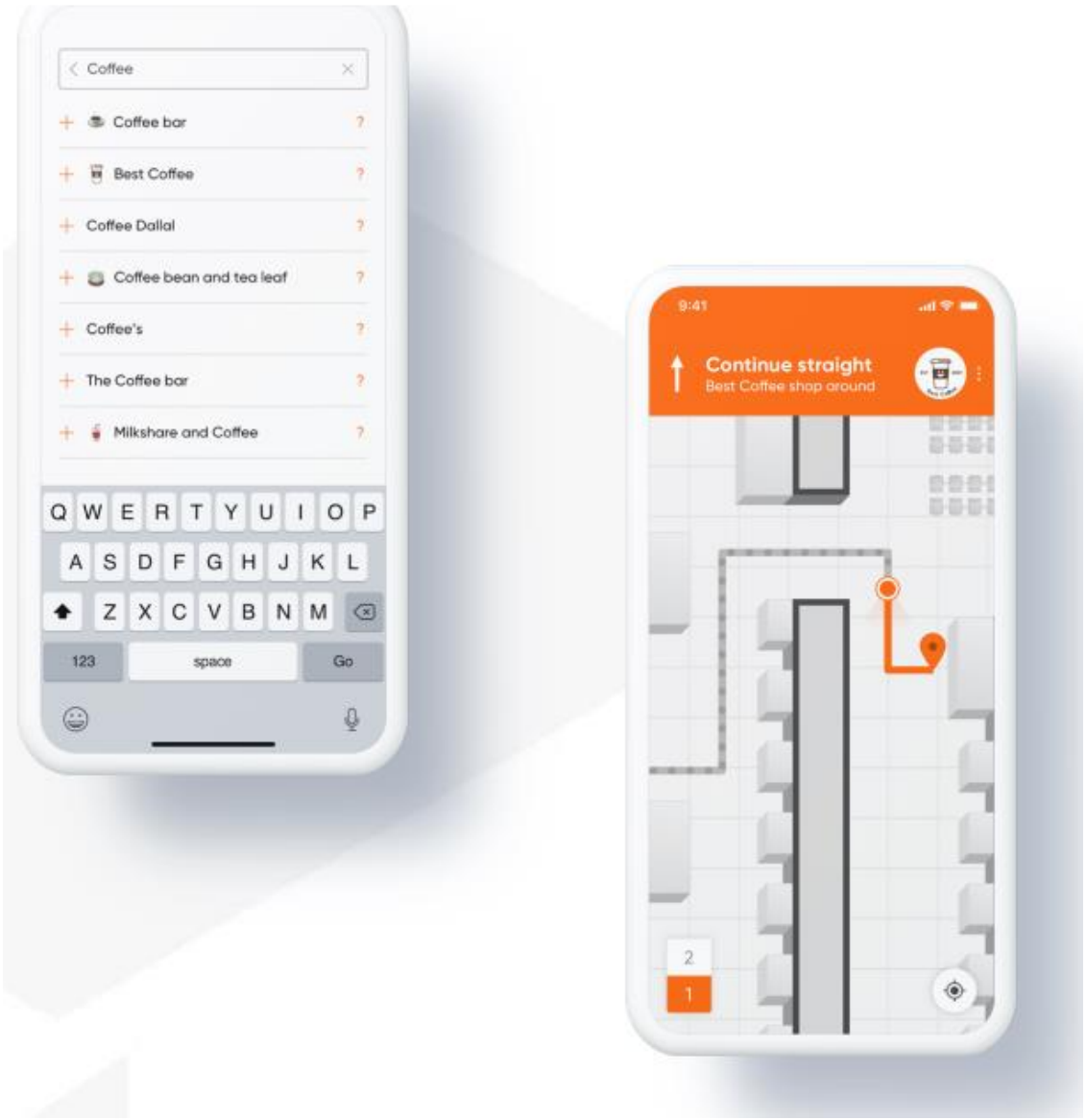

Рисунок 4 – Приклад програмного забезпечення від компанії Oriient

До переваг цього рішення можно віднести простоту виконання, а до недоліків – велику вартість та відсутність можливості відстежувати неживий об’єкт, тобто об’єкт без мобільного телефону.

1.5.2 Рішення від компанії InSoft

На платформі доступні широкі веб-додатки (рисунок 5) для керування даними та візуалізації. infsoft LocAware пропонує двонаправлене з’єднання зі

сторонніми системами через численні інтерфейси для об’єднання внутрішніх і зовнішніх потоків даних.

Рішеннями цієї компанії користуються F. Hoffmann-La Roche, Roche Diagnostics, Siemens, Siemens Healthineers, Франкфуртський аеропорт і Швейцарська федеральна залізниця (SBB).

Рисунок 5 – Приклад програмного забезпечення від компанії Infsoft

У великих будівлях або на широких територіях кампусу відвідувачі та співробітники хочуть дістатися до багатьох місць. Infsoft Wayfinding допомагає користувачам швидко та легко дістатися до потрібного місця.

Продукт можна використовувати в таких будівлях, як лікарні, аеропорти та офісні комплекси у внутрішніх і відкритих приміщеннях. Рішення містить карту розташування з оглядом приміщень. Крім того, об’єкти інтересу (POI) можна знайти безпосередньо на карті, а також за допомогою панелі пошуку та відобразити найкоротший шлях. На мобільних пристроях програма також може пропонувати

покрокову навігацію в реальному часі та додатково допомагати користувачеві. Залежно від вимог до рішення у відповідному місці, infsoft Wayfinding може бути реалізовано як нативний додаток, прогресивний веб-додаток або стаціонарний термінал. Також є можливість комбінувати додатки.

Використання інструментів налаштування завжди включено в рішення. За допомогою редактора карт infsoft, CMS infsoft, маршрутів infsoft і калібрування infsoft програму можна гнучко налаштувати відповідно до потреб замовника. Використовуючи SDK (набір для розробки програмного забезпечення), також можна інтегрувати технологію в існуючі програми. Для офісних середовищ infsoft Wayfinding можна також інтегрувати в додаток Workplace Experience.

До переваг рішень від компанії infsoft можна віднести надання повного апаратно-програмного комплексу для вирішення проблеми визначення положення в обмеженому просторі, можливість використання різного апаратного забезпечення для одного й того ж рішення та інтерактивність, наявність SDK для впровадження системи в існуючі мобільні застосунки.

До недоліків рішень від компанії Infsoft можна віднести те, що рішення компанії в основному зосереджені на використанні додаткової інфраструктури замість наявної, компанія активно використовує Bluetooth маячки та інші додаткові засоби для визначення місцезнаходження об’єкту чи людини. Це призводить до зайвих витрат для бізнесу.

1.5.3 Рішення від компанії Sewio

Sewio Networks є виробником системи визначення місцезнаходження в реальному часі (Real time location system, RTLS), яка забезпечує бізнес-результати для виробників, складів, центрів розподілу, виробників обладнання тощо. Система Sewio побудована на основі ультраширокосмугової технології (Ultra-Wide Band, UWB) і поставляється з RTLS Studio, програмним забезпеченням для дистанційного керування та візуалізації.

Він надає партнерам і клієнтам точне, просте в інтеграції, надійне і повністю масштабоване рішення IoT для відстеження розташування всередині приміщень,

яке дозволяє компаніям досягти більшої ефективності, прибутковості та безпеки. Заснована в 2014 році компанія Sewio має офіси в США, Великобританії, Німеччині та Чехії. Sewio має понад 80 партнерів із системної інтеграції та обслуговує клієнтів у 45 країнах. Серед клієнтів Sewio: Volkswagen, Toyota, Budweiser Budvar, TPCA, Škoda, ENEL.

Основним продуктом компанії є використання ультра широкосмугової системи визначення положення об'єктів та людей. Внутрішня система позиціонування складається з приймачів, які називаються «якорями», на стелі та передавачів, які називаються «мітками», на відстежуваних об'єктах. Теги надсилають сигнали на прив'язки, які під'єднані до мережі за допомогою кабелів або бездротового зв'язку, потім вони отримують ці сигнали та надсилають їх на сервер, який обчислює точне положення тегів у режимі реального часу.

Переваги використання системи від Sewio:

1. Перезаряджувані, з великим терміном служби батареї, мітки UWB RTLS для відстеження співробітників.

2. Повністю масштабоване рішення, яке дозволяє налаштовувати наявні теги, додавати тисячі відстежуваних активів і збільшувати охоплення відповідно до зростання ваших потреб.

3. Точність 30 см, що забезпечує повну гнучкість і варіативність віртуальних зон без будь-яких змін інфраструктури.

До недоліків можна віднести необхідність закупівлі великої кількості якорів, маячків та розгортання додаткової інфраструктури.

1.6 Висновок

Системи для визначення положення людини в обмеженому просторі у реальному часі поділяються на два типи: ті, що потребують спеціалізованої інфраструктури та ті, що використовують широкодоступну інфраструктуру: датчики смартфонів, Wi-Fi та камери. Головна проблема використання датчиків смартфонів та Wi-Fi полягає у потребі постійного носіння пристрою, що неможливо коли є проблема визначення положення неживого об’єкту, наприклад контейнеру. Передові дослідження у машинному зорі та використання реперних міток дають можливість розпізнавати контейнери у реальному часі з використанням камери спостереження, не використовуючи додаткове обладнання або іншу інфраструктуру. Після розпізнавання ми можемо визначити положення контейнеру в обмеженому просторі та зобразити його на 3D моделі. Проблемою визначення положення в обмеженому просторі займаються передові компанії, тому створення подібних комп’ютерних систем є актуальним.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ ВИЗНАЧЕННЯ ПОЛОЖЕННЯ ОБ'ЄКТУ В ОБМЕЖЕНОМУ ПРОСТОРІ У РЕАЛЬНОМУ ЧАСІ

2.1 Технологія Apriltags

Технологія Apriltag, що була розроблена у Мічиганському університеті пройшла довгий шлях до того вигляду, який має зараз. Вперше вона була застосована у 2011 році і з того часу безперервно вдосконалювалась для покращення надійності та швидкості розпізнавання. Зараз найбільш розповсюдженим є сімейство тегів 36h11. На рисунку 6 ми бачимо три теги сімейства 36h11, що розташовані на паперових кубах. Для використання тегів вони повинні бути надруковані та розташовані на рівній поверхні. Це зображення є відправною точкою для аналізу тегів.

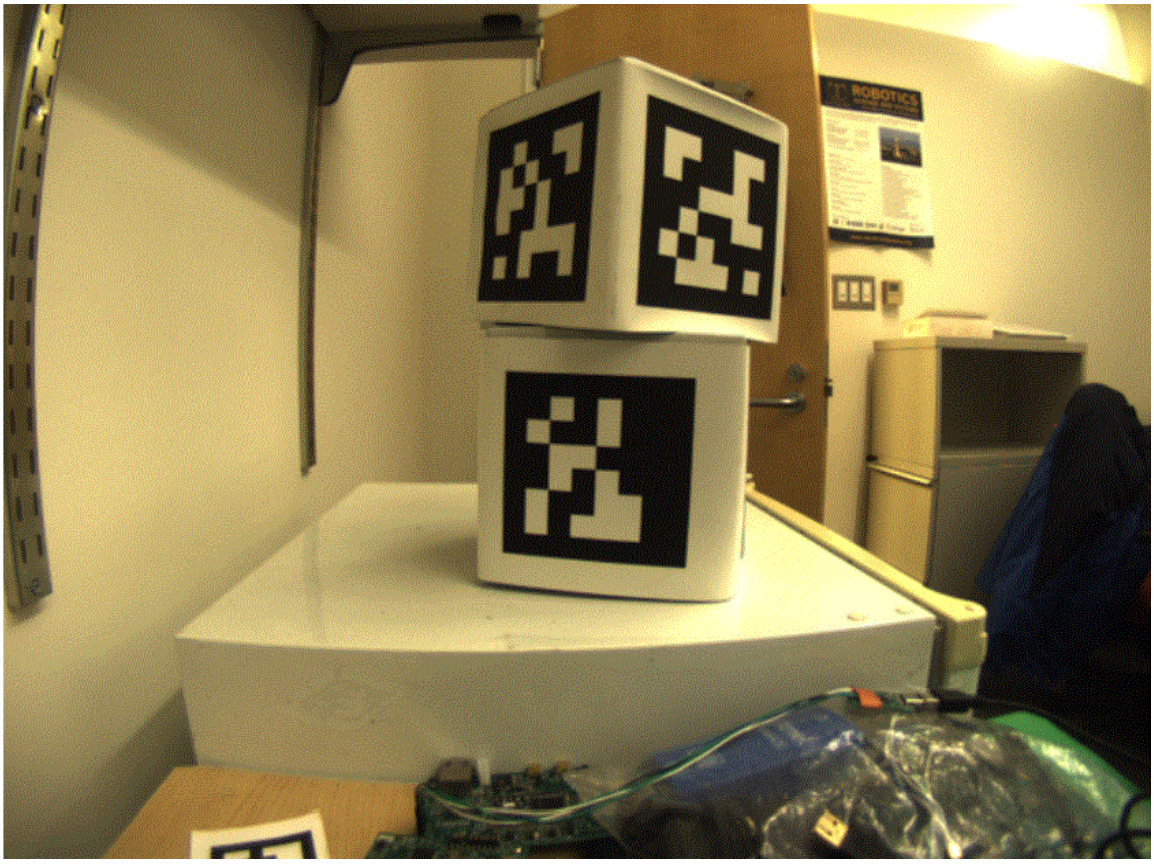

Рисунок 6 – Теги сімейства 36h11

Першим кроком є видалення кольорів на зображенні. На рисунку 7 ми бачимо зображення з видаленими кольорами.

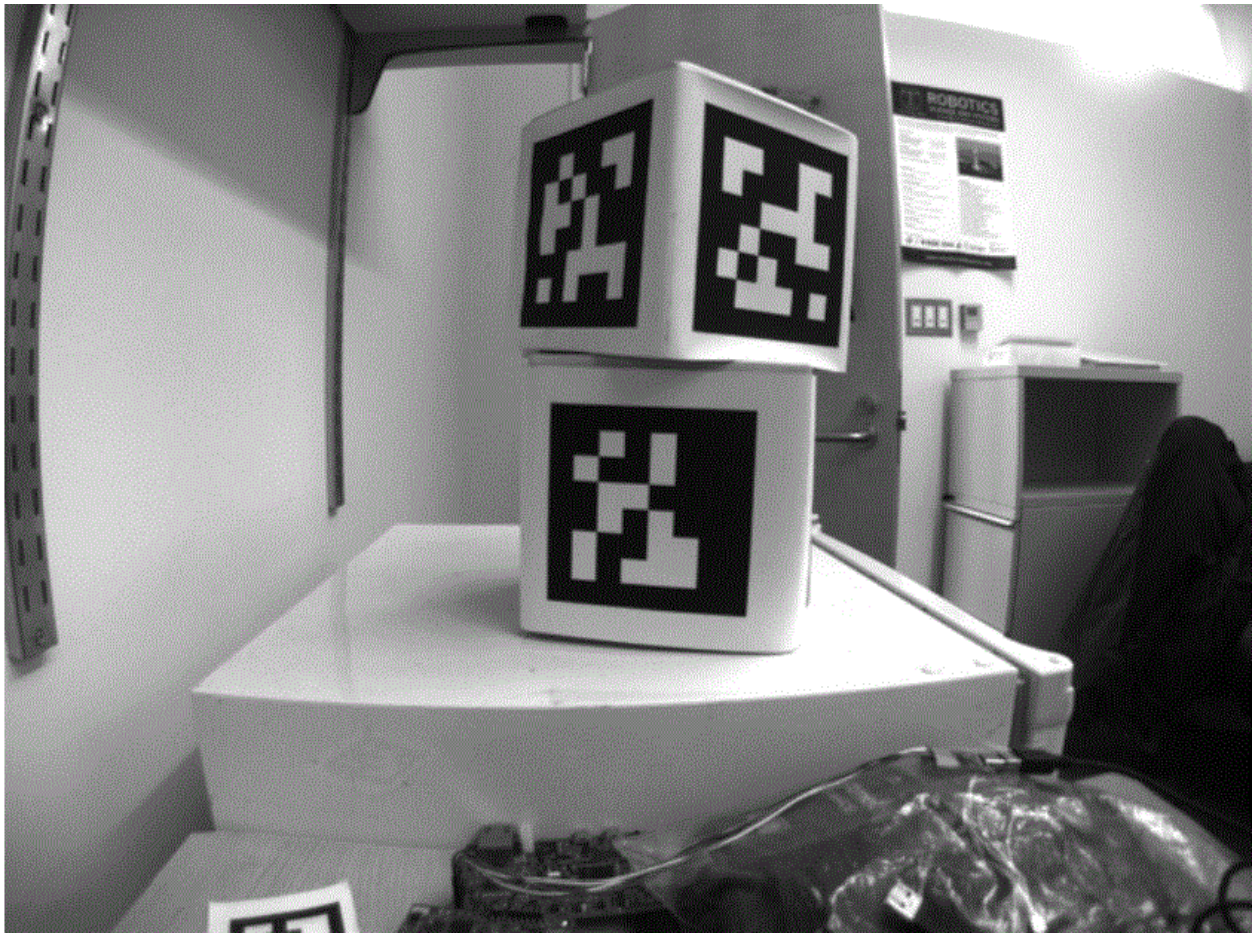

Рисунок 7 – Видалення кольорів на зображенні

Другим кроком є конверсія зображення до меншого розподілення (рисунок 8). Чим менше пікселів, тим швидше працює алгоритм. Це не впливає на кінцеву якість зображення та на розпізнавання тегу.

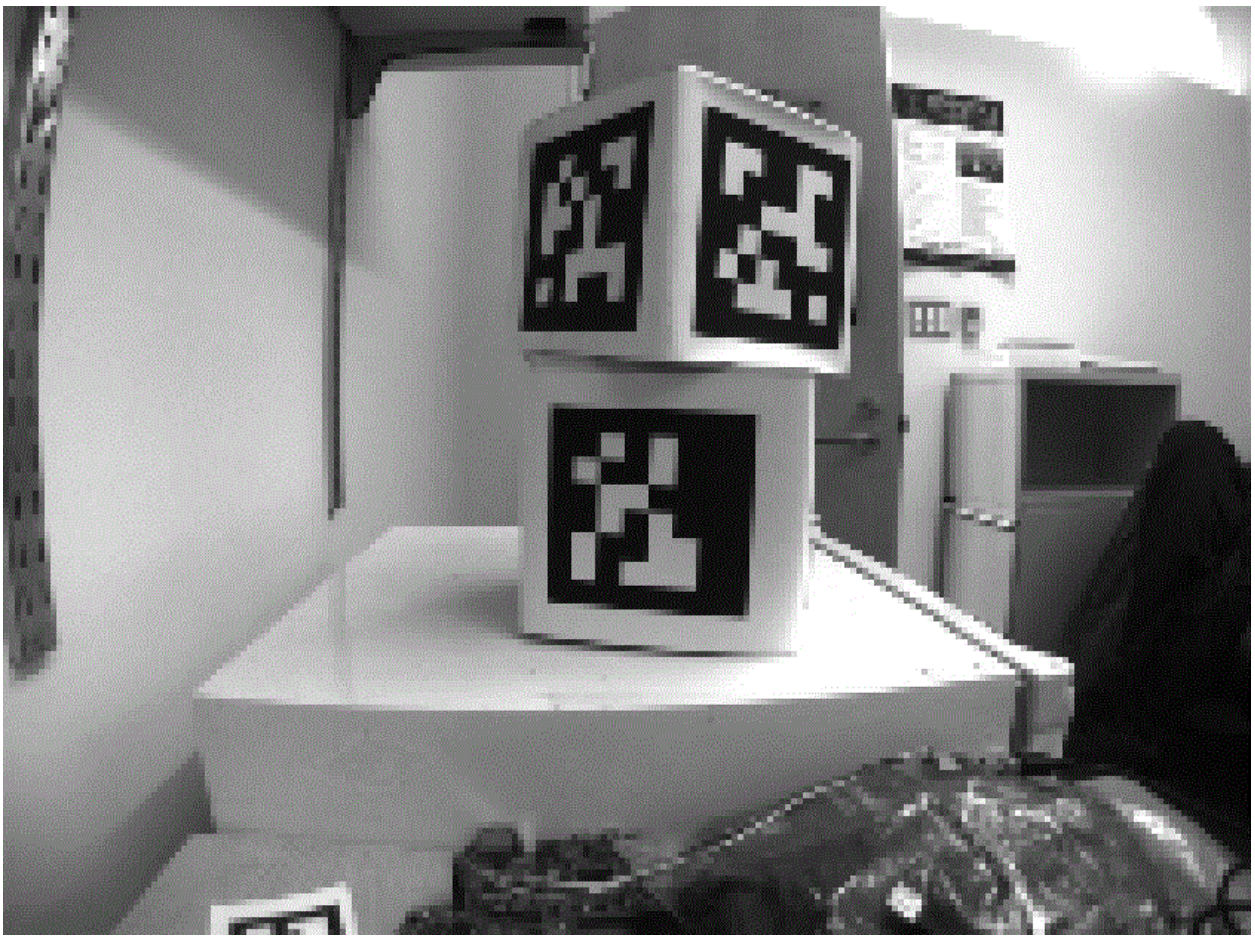

Рисунок 8 – Конверсія зображення до меншого розподілення

Наступним кроком застосовується алгоритм адаптивного порогу (рисунок 9). Він класифікує пікселі як «вірогідно світлі», «вірогідно темні» або «не впевнений».

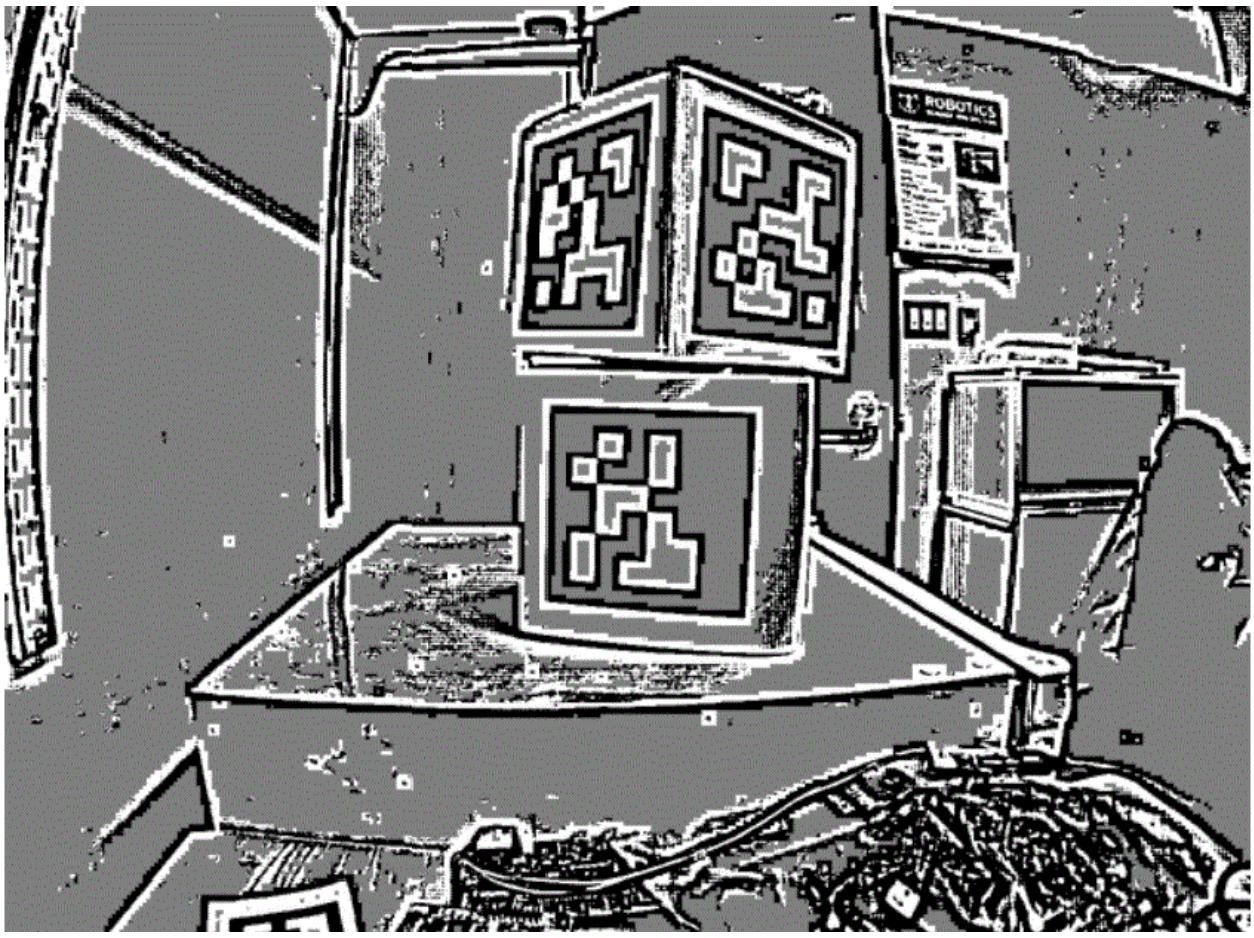

Рисунок 9 – Застосування алгоритму адаптивного порогу

Як видно на рисунку 9 пікселі визначаються шляхом порівняння яскравості з найбільшим оточенням пікселів. Після цього пікселі сегментуються, як зображено на рисунку 10, проте якщо сегмент дуже малий, щоб бути вагомою частиною тега він відкидається.

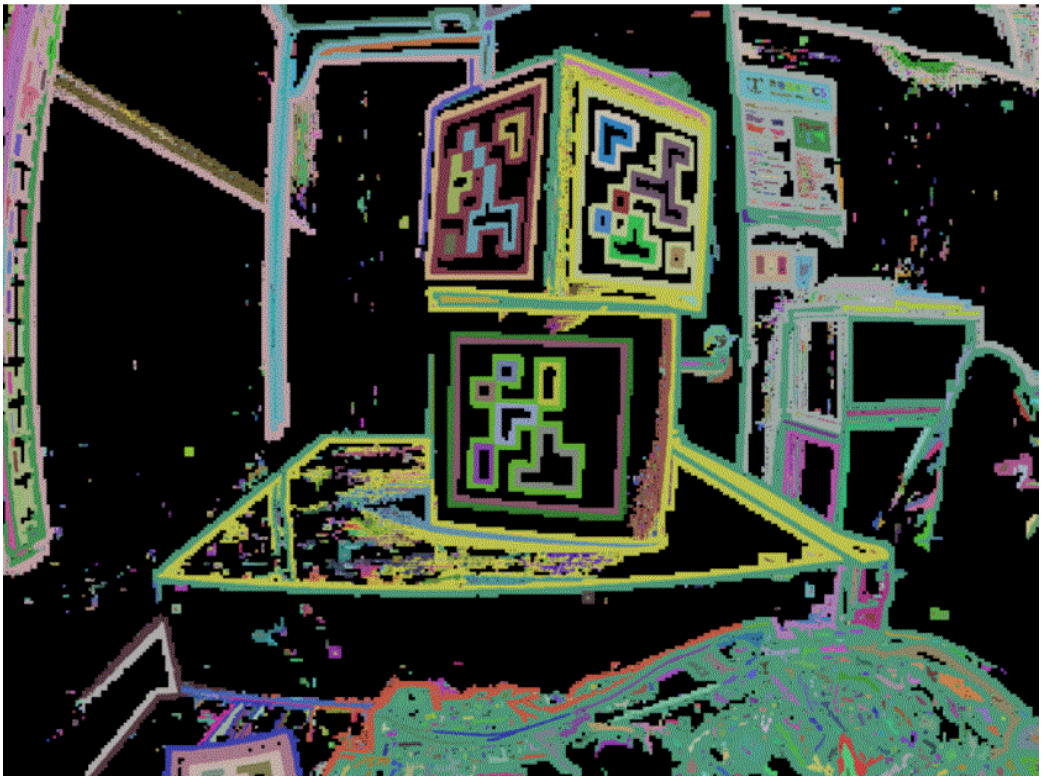

Рисунок 10 – Сегментація пікселів

1. Алгоритм для визначення квадів для кожного сегменту виглядає наступним чином:

2. Визначити найбільш вірогідного кандидата з пікселів, що лежить у обох вимірах.

3. Ітерувати через усі можливі комбінації кутів, покращуючи розмір кожного разу.

4. Обрати найкращий сегмент

Враховуючи набір усіх чотирикутників, визначається підмножина чотирикутників, яка найбільш вірогідно є тегом. Один великий зовнішній чотирикутник з множиною внутрішніх чотирикутників, вірогідно є добрим кандидатом. Якщо усі кроки пройшли успішно, то лишається чотиристоронння зона пікселів, зображена на рисунку 11, яка, вірогідно є тегом.

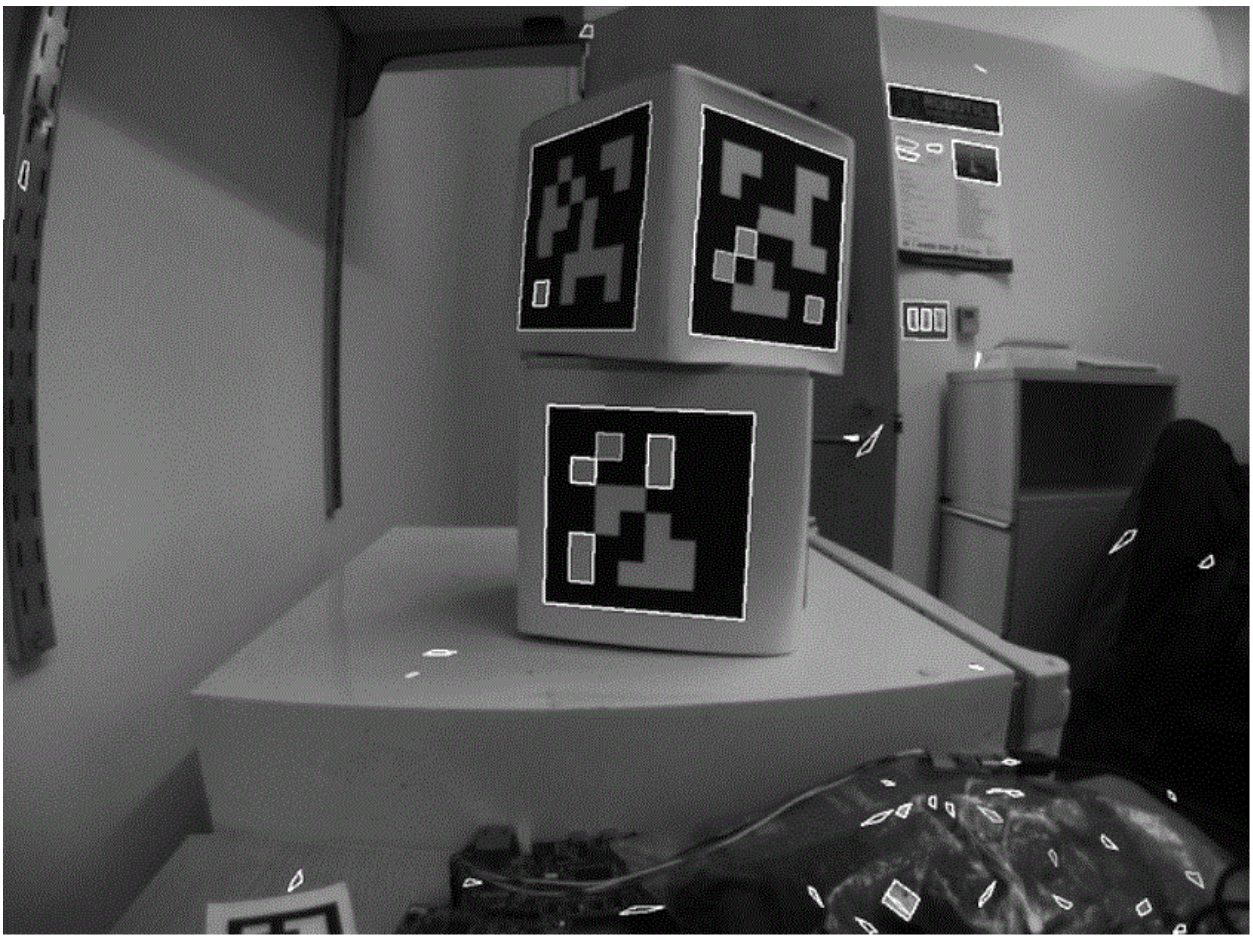

Рисунок 11 – Розпізнавання квадів

Після отримання областей, що можуть бути тегами, визначається id тега. Це робиться шляхом «розшифровки» візерунка світлих та темних квадратів на внутрішній стороні. Далі обчисляються очікувані координати внутрішніх пікселів у центрі кожного біта. На рисунку 12 зображено як кожне місце позначається як «0» або «1», порівнюючи інтенсивність пікселя з порогом.

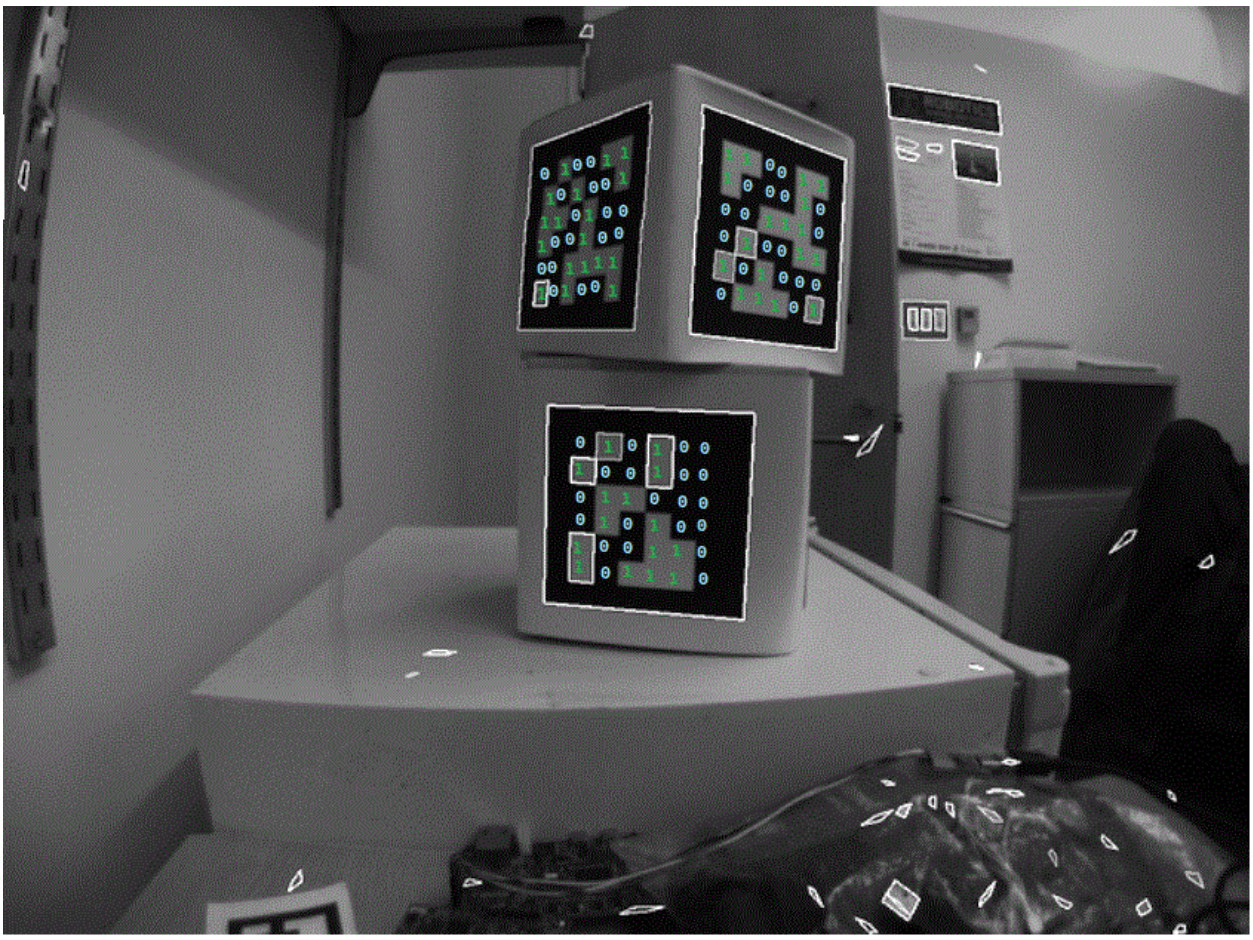

Рисунок 12 – Декодування id тега

Знаходиться ідентифікатор тегу, який найбільше відповідає тому, що видно на зображенні, враховуючи одну чи дві бітові помилки. Якщо немає дійсного ідентифікатора тегу, який відповідає цьому тегу, то процес декодування припиняється. Наступним кроком потрібно знати точне розташування кутів мітки або її центру. В обох випадках операція зі зниження роздільної здатності, яка була виконана на початку, спотворила зображення, і потрібно скасувати ці ефекти. Спочатку використовується виявлене розташування тегу, щоб визначити область інтересу на зображенні з вихідною роздільною здатністю, потім обчислюється градієнт у попередньо визначених точках у області інтересу, щоб визначити, де зображення найбільш різко переходить від чорного до білого. Потрібно використати ці вимірювання градієнта, щоб швидко змінити зовнішній чотирикутник із повною роздільною

здатністю, та використати геометрію, щоб обчислити точний центр перебудованого чотирикутника.

2.2 Використання Apriltags для розпізнавання об'єктів у реальному часі

На кожному зображенні виконується пошук Apriltag за допомогою алгоритму, описаного на у розділі 2.1. Використовуючи припущення про те, як об’єктив камери спотворює 3D-світ на 2D-масив пікселів у камері, обчислюється оцінка положення камери відносно тегу. Щоб припущення щодо поведінки об’єктива були точними, потрібне хороше калібрування камери.

Ідентифікатор тегу також розшифровується із зображення. Враховуючи ідентифікатор кожного тегу, можна переглянути положення тегу в полі.

Знаючи положення мітки на полі та положення камери відносно мітки, класи 3D-геометрії можна використовувати для оцінки положення камери на полі.

2.2.1 Визначення векторів положення та трансформації

Після знаходження тегу потрібно визначити вектори положення та трансформації для подальших розрахунків. Вектори положення та трансформації пов’язані з камерою, та вираховуються лише один раз.

Як вказано у [5] можливо використати рішення, яке буде працювати як з даними без шуму, так і з даними з шумом. Для вірного рішення потрібно мінімум 3 унікальних позиції.

Основна формула буде мати вигляд:

$$RA + t = B\,, \tag{2.1}$$

де A і B – набори трьохмірних точок з відомими співвідношеннями, R – матриця обертання 3x3, t – вектор трансформації

Проте перед тим, як вирахувати вектори потрібно вирахувати фокусні параметри камери (рисунок 13).

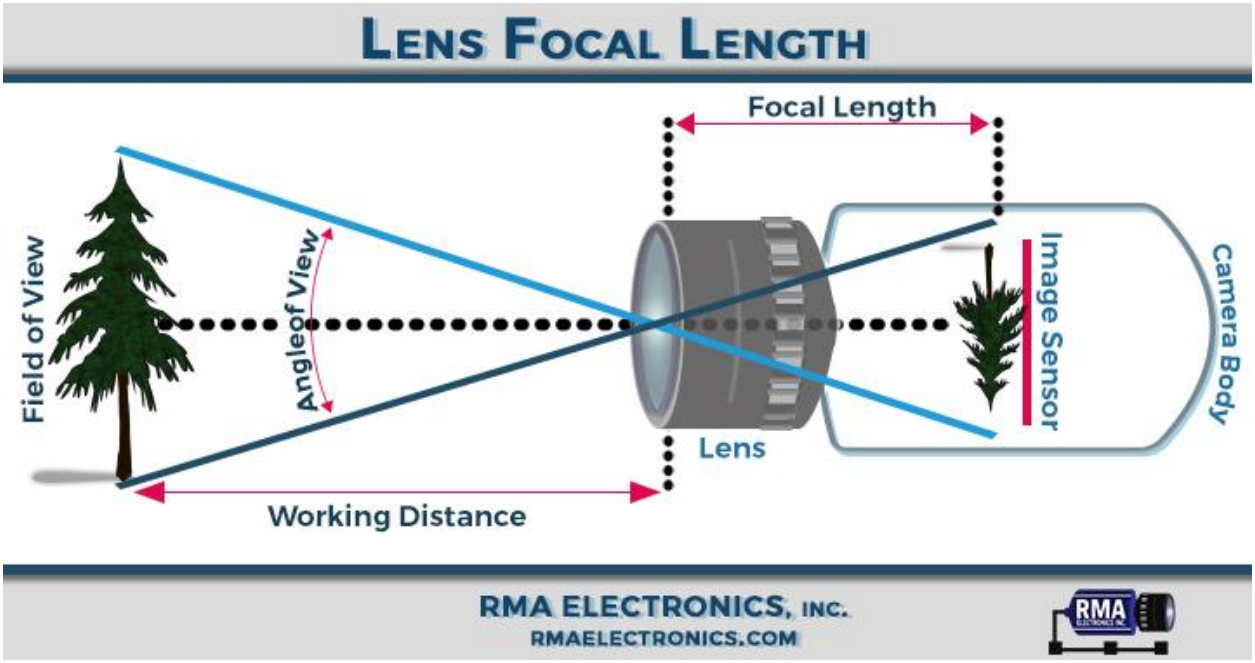

Рисунок 13 – Принцип вимірювання фокусної відстані

Для вимірювання фокусної відстані камери ми застосуємо формулу

$$f = \frac{A}{\tan(\propto)}, \qquad (2.2)$$

де A – це половина ширини зображення (у пікселях), $\propto$ - це параметр поля зору камери (у градусах).

Після обчислення фокусної відстані для кожного джерела відеопотоку ми зможемо запустити детектор Apriltag з параметрами, що дозволять обчислити положення тегу у кадрі. Для цього потрібно сформувати матрицю параметрів камери у вигляді {*f*,*f*,cx,cy}, де f – фокусна відстань, cx та cy – центр зображення по осі x та y відповідно.

2.2.2 Конвертування матриці положення об’єкту

Спочатку знаходимо центроїди – середнє значення точок, розраховуються за формулою

$$centroid_A = \frac{1}{N}\sum_{i=1}^{N} A^i \qquad (2.3)$$

$$centroid_B = \frac{1}{N}\sum_{i=1}^{N} B^i \ , \qquad (2.4)$$

де A^i та B^i - вектори 3x1 у вигляді $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$.

Наступним кроком є накопичення матриці H та виконання сингулярного розкладення для знаходження повороту наступним чином:

$$\mathrm{H} = (\mathrm{A} - centroid_{\mathrm{A}})(\mathrm{B} - centroid_{\mathrm{B}})^{\mathrm{T}} \qquad (2.5)$$

$$[U, S, V] = SVD(H) \qquad (2.6)$$

$$R = VU^T \ , \qquad (2.7)$$

де H – відома коваріаційна матриця.

Проте є особливий випадок, коли сингулярне розкладення повертає матрицю, що має хибні значення. Це вирішується перевіркою R<0 , якщо R>0, то ми повинні помножити 3й стовпчик V на -1.

Після знаходження параметру R потрібно знайти вектор трансформації t. Це можливо знайти підставляючи центроїди в рівняння (2.1)

$$R \times centroid_{\mathrm{A}} + t = \ centroid_B$$

$$t = \ centroid_B - R \times centroid_{\mathrm{A}} \qquad (2.8)$$

2.3 Фреймворки та бібліотеки

2.3.1 Фреймворк pupil_apriltags

Розроблена компанією Pupil Labs, pupil apriltags – це потужна бібліотека, написана на Python та CPython для взаємодії з технологією Apriltag. Компанія Pupil

Labs була заснована у 2014 у Берліні, Німеччина. Першим продуктом був Pupil Core - open source ПЗ та переносне обладнання для відстеження зору, що було зібране групою науковців. Через декілька років компанія стала лідером у сфері технологій відстежування зору. Бібліотеку можливо інсталювати на Linux, Windows чи MacOS. Для коректної роботи потрібно мати Python версії не менше ніж 3.6.

2.3.2 Фреймворк ThreeJS

Open source бібліотека з відкритим кодом, створена для відображення 3D графіки для веб технологій. Має дуже багато встроєних завантажувачів для усіх типів розповсюджених 3D моделей. Вона дозволяє використовувати графіку, що опрацьовує графічний процесор, використовуючи JS, як частину сайту, без під'єднання сторонніх бібліотек. Має різноманітні рендери, зокрема Canvas, SVG та WebGL. ThreeJS дозволяє додавати та видаляти об'єкти у реальному часі, та ефект туману. Фреймворк має дві вбудовані камери – ортогональну та перспективну та безліч різноманітних джерел світла.

Фреймворк постійно оновлюється та має величезну спільноту та публічний форум.

2.3.3 Бібліотека OpenCV

OpenCV (Open Source Computer Vision Library) — бібліотека комп'ютерного зору та машинного навчання з відкритим кодом. OpenCV було створено, щоб забезпечити загальну інфраструктуру для програм комп'ютерного зору та прискорити використання машинного сприйняття в комерційних продуктах. Будучи ліцензованим продуктом Apache 2, OpenCV дозволяє компаніям легко використовувати та змінювати код.

Бібліотека містить понад 2500 оптимізованих алгоритмів, що включає повний набір як класичних, так і найсучасніших алгоритмів комп'ютерного зору та машинного навчання. Ці алгоритми можна використовувати для виявлення та розпізнавання облич, ідентифікації об'єктів, класифікації дій людей у відео,

відстеження рухів камери, відстеження рухомих об’єктів, вилучення 3D-моделей об’єктів, створення 3D-хмар точок із стереокамер, з’єднання зображень для отримання високої роздільної здатності. зображення цілої сцени, знаходити подібні зображення в базі даних зображень, видаляти червоні очі із зображень, зроблених за допомогою спалаху, стежити за рухами очей, розпізнавати пейзаж і встановлювати маркери для накладання на нього доповненої реальності тощо. OpenCV має понад 47 тисяч користувачів спільнота та оціночна кількість завантажень перевищує 18 мільйонів. Бібліотека широко використовується компаніями, дослідницькими групами та державними структурами.

Поряд із такими відомими компаніями, як Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota, які використовують бібліотеку, є багато стартапів, таких як Applied Minds, VideoSurf і Zeitera, які широко використовують OpenCV. Застосування OpenCV охоплює широкий діапазон: від з’єднання зображень вуличного перегляду, виявлення вторгнень у відеоспостереження в Ізраїлі, моніторингу шахтного обладнання в Китаї, допомоги роботам у навігації та підбиранні об’єктів у Willow Garage, виявлення нещасних випадків утоплення в басейні в Європі, використання інтерактивного мистецтва в Іспанія та Нью-Йорк, перевірка злітно-посадкових смуг на наявність сміття в Туреччині, до перевірки етикеток на продуктах на фабриках по всьому світу та швидкого виявлення облич у Японії.

Бібліотека має інтерфейси C++, Python, Java і MATLAB і підтримує Windows, Linux, Android і Mac OS. OpenCV здебільшого орієнтується на програми бачення в реальному часі та використовує переваги інструкцій MMX і SSE, якщо вони доступні. Зараз активно розробляються повнофункціональні інтерфейси CUDA і OpenCL . Існує понад 500 алгоритмів і приблизно в 10 разів більше функцій, які створюють або підтримують ці алгоритми. OpenCV написаний на C++ і має шаблонний інтерфейс, який без проблем працює з контейнерами STL.

2.3.4 Інші фреймворки та бібліотеки

Для успішної роботи системи необхідно додати декілька інших бібліотек, зокрема Numpy – потужну бібліотеку для наукових розрахунків, написану на мовах C, Fortran та Python.

2.4 Висновки з розділу 2

1. Як вхідні дані можливо використати будь-яке джерело відеокадрів, у якого відомо параметр поля зору.

2. Після знаходження фокусної відстані потрібно заповнити матрицю параметрів камери для подальших розрахунків.

3. Використовуючи бібліотеку OpenCV можливо виконати усі операції пов’язані з камерою, оскільки ця бібліотека має значні можливості для роботи з відео та має різноманітні модулі для обробки відеокадрів.

4. Для визначення правильності вибору методів та засобів потрібно випробування, шляхом створення комп’ютерної системи, яка буде опрацьовувати відеокадри у реальному часі та буде працювати на пристроях з обмеженими ресурсами.

РОЗДІЛ 3 РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ ВИЗНАЧЕННЯ ПОЛОЖЕННЯ ОБ'ЄКТУ В ОБМЕЖЕНОМУ ПРОСТОРІ У РЕАЛЬНОМУ ЧАСІ

3.1 Використання Apriltags для визначення положення об'єкту

3.1.1 Налаштування середовища для розпізнавання тегів

Найпершим кроком, потрібним для розробки системи розпізнавання об'єктів є налаштування робочого середовища. Перевірка роботи системи була виконана на комп'ютері з характеристиками:

1. Процесор Intel Core i5-9600KF 3.7GHz/8GT/s/9MB;
2. Відеокарта NVIDIA GeForce GTX 1660;
3. 16 Гб оперативної пам'яті;
4. Операційна система Windows 10;

Для роботи потрібно встановити бібліотеку pupil apriltags, openCV, numpy та їх залежності. Для цього були виконані наступні кроки:

1. Встановлено мову програмування Python 3.11.3;
2. Встановлено numpy версії 1.24.3;
3. Встановлено opencv_python версії 4.7.0.72;
4. Встановлено pupil_apriltags версії 1.0.4.post10;

Для перевірки встановлених бібліотек завантажимо одне зображення з apriltag та виконаємо команду :

```
Python3 test.py
```

Після обробки ми отримаємо результат, зображений на рисунку 14.

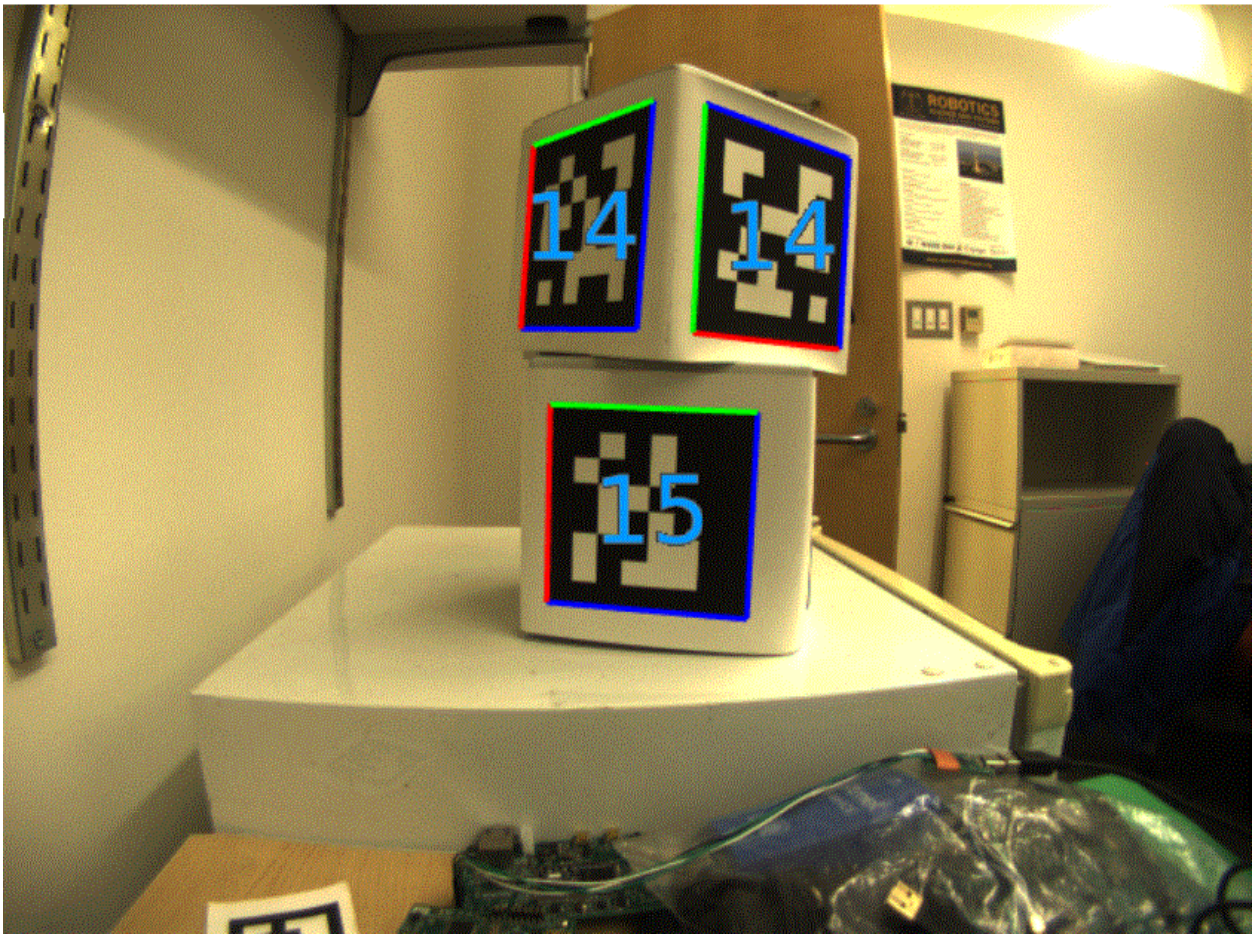

Рисунок 14 – Декодований тег

Після отримання результату та перевірки його на коректність можна зробити висновок, що система відпрацювала правильно.

3.1.2 Підготовка відеопотоків для роботи

Для того, щоб процес розпізнавання пройшов успішно, потрібно спочатку перевірити якість відеопотоку та коректність відображення тегів. Для цього потрібно перевірити розподільне значення відео, а також значення, яке передає NVR. Зазвичай найвищий показник у відео, що йде з відеопотоку main. Після перевірки розподільного значення, параметр центрального пікселя по вісі x та вісі y заноситься у матрицю параметрів камери як значення cx та cy.

3.1.3 Підготовка тегів

AprilTags доступні в кількох різних варіантах (сімействах), залежно від того, скільки бітів може представляти тег. Деякі сімейства тегів мають великі, грубі бітові блоки, які можна побачити здалеку, але можуть охоплювати лише невеликий номерний простір. Найпопулярнішим є сімейство «36x11», яке має розрядність 6x6. Він може охоплювати числа від 0 до 586 і досить стійкий до хибнопозитивних виявлень. Якщо не потрібен повний номерний простір і потрібно працювати з камерою з нижчою роздільною здатністю, треба розглянути сімейство «16h5», яке може представляти від 0 до 29. Однак під час використання цього тегу може бути більше помилкових позитивних виявлень.

Ідеальний AprilTag має бути:

- ідеально рівний, без перекосів;
- матовий;
- висока контрастність: чорний на білому;
- жорсткий і міцний;
- легка вага;

3.1.4 Запуск процесу розпізнавання

Після підготовки відеопотоку та шаблону тегу була виконана команда `python3 test.py`. Результатом виконання цієї команди був запуск тестової версії скрипта, що завантажує дані про камеру з файлу конфігурації, обраховує фокусну відстань камери та запускає процес розпізнавання. Кінцевим етапом є покадрове відображення відеопотоку з відміченими тегами, що були знайдені.

3.1.5 Перевірка якості розпізнавання

Після перегляду відеопотоку є можливість оцінити якість знаходження тегів за формулою:

$$Q = \frac{frames_f}{frames_t} * 100\% , \qquad (2.9)$$

де $frames_f$ – кількість фреймів відеопотоку зі знайденими тегами, а $frames_t$ – сумарна кількість фреймів у відеопотоці.

Для покращення цього показника ми можемо зробити наступне:

- Покращити розподільне значення відео.
- Збільшити розмір тегу.
- Зробити тег більш контрастним.
- Розмістити тег на більш зручному місці.

3.2 Розробка та функціонал комп'ютерної системи для визначення положення об'єкту в обмеженому просторі у реальному часі

3.2.1 Налаштування середовища для розробки системи

Для розробки системи відстежування об'єктів було використано наступний стек технологій:

1. Мова програмування Python 3.11.3 – для реалізації функціоналу процесу розпізнавання.
2. Фреймворк JQuery 1.10.2 – для візуальної частини проекту.
3. Фреймворк ThreeJS r158 – для роботи з 3D моделлю.
4. Бібліотека opencv_python версії 4.7.0.72 – для роботи з відео потоком, роботи з кадрами та пост-обробки відео потоку.
5. Фреймворк Flask 2.2.3 – для серверної частини проекту.
6. Бібліотека pupil_apriltags версії 1.0.4.post10 – для розпізнавання і подальшої роботи з тегами Apriltags.

Основною причиною використання цього стеку технологій на базі фреймворку Flask є необхідність кросплатформного рішення для систем, які будуть працювати як під ОС Linux так і під системою Windows. Важливим елементом усієї системи є бібліотека OpenCV так як вона підтримує безліч варіантів роботи з відео потоками. Для коректної роботи з технологією Apriltags використано бібліотеку pupil_apriltags, яка дозволяє читати інформацію з тега без зайвих зусиль.

Так як розробка системи відстежування об’єктів проводиться на тому ж комп’ютері, на якому було проведена перевірка розпізнавання тегів, то у встановленні мови програмування Python та бібліотек OpenCV та pupil_apriltags немає потреби. Для встановлення фреймворку Flask виконаємо команду:

```
pip install Flask
```

Для встановлення фреймворку ThreeJS завантажимо необхідну версію з офіційного репозиторію проекту https://github.com/mrdoob/three.js та розпакуємо її у каталог ./static/scripts/build

Після підготовки середовища для розробки системи потрібно виконати калібрування камери, щоб отримати найбільш точне положення об’єкту у системі моделі.

Необхідність калібрування обумовлено тим, що бібліотека pupil_apriltags передає положення тегу у системі камері. Процес калібрування потрібно почати зі створення дата сету зображень з координатами, які відповідають положенню тегу у системі камери.

Для створення датасету потрібно взяти відеоматеріал з орієнтирами, до яких можливо створити прив’язку на місцевості, або якщо процес виконується у реальному часі, то потрібно розташувати теги на предметах, які можуть бути явними орієнтирами (наприклад: кути стола, якісь окремо стоячі предмети і т.д).

Для виконання калібрування у цьому проекті використовувався відеоряд, який був знятий на попередньо обміряній місцевості. На рисунку 15 можна побачити об’єкт, який відстежується. За попередніми вимірюваннями початком створення дата сету вважається зелена лінія, до якої від камери відстань 8м. Камера знаходиться на відстані 2.8 м від полу, та рівно по центру. Відстань від камери до червоної лінії складає 4 м. Процес калібрування виконується один раз, під час налаштування системи. Після цього система стабільно буде працювати з налаштованими параметрами. Проте необхідно враховувати, що зміна положення камери, або зміна моделі камери, чи зміна будь-яким чином фокусної відстані камери призведе до необхідності нового калібрування, оскільки матриці, що будуть обчислені в результаті калібрування, будуть впливати на положення об’єкту на моделі.

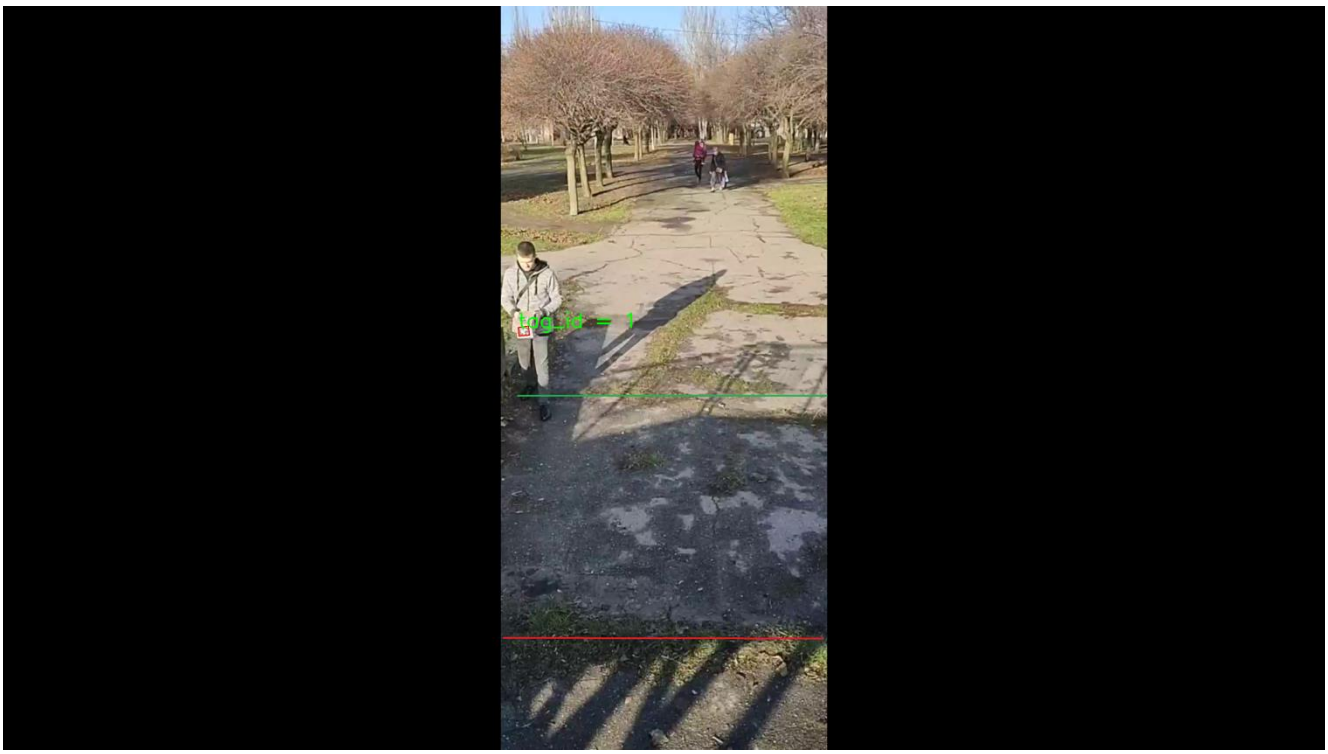

Рисунок 15 – Кадр з відзнятого датасету

Для створення датасету був створений окремий скрипт, код якого наведений у лістингу 1.

Лістинг 1 – Код створення датасету для калібрування

```
import cv2
import numpy as np
from pupil_apriltags import Detector
import imutils

def find_marker(ptA, ptB, ptC, ptD):
        L = [ptA, ptB, ptC, ptD]
        ctr = np.array(L).reshape((-1, 1, 2)).astype(np.int32)
        return cv2.minAreaRect(ctr)

def get_resolution(video):
        cap = cv2.VideoCapture(video)
```

```
        if not cap.isOpened():
            print("Cannot open camera")
            exit()
        ret1, frame1 = cap.read()
        fshape = frame1.shape
        fheight = fshape[0]
        fwidth = fshape[1]
        cap.release()
        return [fheight/2,fwidth/2]
def __init__():
    video_name = 'E://source/repos/Diploma/calib10_3.avi'
    tag_size = 0.1
    cy,cx = get_resolution(video_name)
    focalLength = cx/np.tan(69.7)
    camera_params = [focalLength,focalLength,cx,cy]
    cap = cv2.VideoCapture(video_name)
    detector = Detector(families='tag36h11',nthreads=8)
    if not cap.isOpened():
        print("Cannot open camera")
        exit()
    frame = 0
    foundedFrame = 0
    while True:
        frame+=1

        ret, img = cap.read()

        if ret == True:
                    data = []
                    gray = img[:,:,1]
                    try:
                        results =
detector.detect(gray,True,camera_params,tag_size=tag_size)
```

```
                    except BaseException as err:
                        results = None

                    for r in results:
                            center = r.center.astype(np.int32)
                            font = cv2.FONT_HERSHEY_SIMPLEX
                            (ptA, ptB, ptC, ptD) = r.corners
                            ptB = (int(ptB[0]), int(ptB[1]))
                            ptC = (int(ptC[0]), int(ptC[1]))
                            ptD = (int(ptD[0]), int(ptD[1]))
                            ptA = (int(ptA[0]), int(ptA[1]))
                            marker = find_marker(ptA, ptB, ptC,
ptD)
                            box = cv2.cv.BoxPoints(marker) if
imutils.is_cv2() else cv2.boxPoints(marker)
                            box = np.int0(box)
                            cv2.drawContours(img, [box], -1,
(0, 0, 255), 2)
                            cv2.putText(img, "tag_id = %s" %
r.tag_id,
                                        (ptD[0], ptD[1]),
cv2.FONT_HERSHEY_SIMPLEX,
                                        2.0, (0, 255, 0), 3)
                            record = {"tag_id": r.tag_id,
"coords": [float(r.pose_t[0]),
float(r.pose_t[1]),float(r.pose_t[2])]}
                            if r.tag_id!=0:
                                data.append(record)



                    resized = cv2.resize(img, (640,480),
interpolation = cv2.INTER_AREA)
```

```
                    cv2.imshow("frame",resized)
                    if cv2.waitKey(1) & 0xFF == ord('q'):
                        break
                    if len(data)>0:
                       print(data)

cv2.imwrite(f'E://source/repos/Diploma/dataset/{frame}.png',img
)
                       with
open(f'E://source/repos/Diploma/dataset/{frame}.txt',"w") as f:
                            f.write(f'{data}')
        else:
             break


__init__()
```

Мета цього коду відкрити попередньо відзнятий відеоряд та зберегти усі кадри, які будуть мати теги у своєму складі. Положення тегів у системі камери буде записано разом з їх id у окремий .txt файл, який буде мати номер ідентичний номеру кадру. Після створення дата сету необхідно обрати від 3 до 6 кадрів, які будуть мати характерну прив’язку на місцевості. Наприклад, на кадрі, що зображений на рисунку 16, об’єкт знаходиться на точці, яка символізує крайнє ліве положення та була попередньо виміряна. Результат програмного вимірювання положення тегу у системі камери дає результат:

```
[-1.3980656753518708, -0.463532663901426, 9.956740266303768],
```

де перше число – координата тегу по вісі “X”, друга по вісі “Y”, а третя – показує дистанцію від камери до обраного тегу. Обравши шість тегів з визначеними положеннями у системі камери потрібно перейти к наступному кроку – розмістити тег у системі моделі, відповідно до вимірів у реальному просторі.

Сама 3D модель виконана засобами ThreeJS, та підтримує будь-які стандартні маніпуляції з 3D моделлю. На 3D моделі знаходиться піктограма, яка є покажчиком камери, а також куб, який є покажчиком об’єкту, що відстежується. Окремої уваги заслуговує код створення 3D моделі, наведений у лістингу 2.

Лістинг 2 – Реалізація функціоналу створення сцени

```
container = document.getElementById('container');
const width = container.clientWidth;
const height = container.clientHeight;

scene = new THREE.Scene();
scene.background = new THREE.Color(0xf0f0f0);

camera = new THREE.PerspectiveCamera(70, width / height, 1,
10000);
camera.position.set(0, 250, 1000);
scene.add(camera);

scene.add(new THREE.AmbientLight(0xf0f0f0, 3));
const light = new THREE.SpotLight(0xffffff, 4.5);
light.position.set(0, 1500, 200);
light.angle = Math.PI * 0.2;
light.decay = 0;
light.castShadow = true;
light.shadow.camera.near = 200;
light.shadow.camera.far = 2000;
light.shadow.bias = - 0.000222;
light.shadow.mapSize.width = 1024;
light.shadow.mapSize.height = 1024;
scene.add(light);

const planeGeometry = new THREE.PlaneGeometry(2000, 2000);
```

```
planeGeometry.rotateX(- Math.PI / 2);
const planeMaterial = new THREE.ShadowMaterial({ color:
0x000000, opacity: 0.2 });

const plane = new THREE.Mesh(planeGeometry, planeMaterial);
plane.position.y = 0;
plane.position.x = 0;
plane.receiveShadow = true;
scene.add(plane);
renderer = new THREE.WebGLRenderer({ antialias: true });
renderer.setPixelRatio(window.devicePixelRatio);
renderer.setSize(width, height);
renderer.shadowMap.enabled = true;
container.appendChild(renderer.domElement);
labelRenderer = new CSS2DRenderer();
labelRenderer.setSize(window.innerWidth, window.innerHeight);
labelRenderer.domElement.style.position = 'absolute';
labelRenderer.domElement.style.top = '0px';
labelRenderer.domElement.style.pointerEvents = 'none';
document.getElementById('container').appendChild(labelRenderer.
domElement);
const groundMaterial = new THREE.MeshPhongMaterial({ color:
0xFF0000 });
// column

const column1 = new THREE.Mesh(new THREE.BoxGeometry(50, 100,
800), groundMaterial);
column1.position.x = -400;
column1.position.y = 0;
column1.position.z = -400;

column1.castShadow = true;
column1.receiveShadow = true;
```

```
scene.add(column1);



// column

const column2 = new THREE.Mesh(new THREE.BoxGeometry(50, 100,
800), groundMaterial);
column2.position.x = 400;
column2.position.y = 0;
column2.position.z = -400;

column2.castShadow = true;
column2.receiveShadow = true;
scene.add(column2);
```

Спочатку створюється сам об’єкт сцени, також вказується колір фону сцени. Після створення об’єкту сцени на саму сцену додаються усі інші складові 3D моделі. Зокрема, створюється об’єкт камери. ThreeJS підтримує два види камери – ортогональну та перспективну. Для оптимальної роботи системи була обрана перспективна камера. Наступним кроком додається світло для сцени. У проекті використовується світло оточення та точкове світло. Для відображення границь області використовується допоміжна поверхня з затемненого матеріалу. Стіни, які обмежують простір створюються програмним шляхом, як об’єкти `THREE.BoxGeometry`, розміри стін створюються відповідно реальному простору.

Оскільки ціль скрипту є правильне відображення моделі на головній сторінці, то потрібно передати параметри рендеру у контейнер. Цей процес виконується за допомогою методу `.setSize.`

Після підготовки сцени завантажуємо модель, що буде покажчиком камери. Цей процес наведений у лістингу 3.

Лістинг 3 – Завантаження моделі камери

```
const manager = new THREE.LoadingManager(loadModel);
function onProgress(xhr) {

    if (xhr.lengthComputable) {

        const percentComplete = xhr.loaded / xhr.total * 100;
        console.log('model ' + Math.round(percentComplete, 2)
+ '% downloaded');

    }

}
function loadModel() {

    object.position.x = 0;
    object.position.y = 0;
    object.scale.setScalar(0.1);
    scene.add(object);

    render();

}


new MTLLoader().load('/static/models/Security cameras
v3.1.mtl', function (materials) {

        materials.preload();

        new OBJLoader()
```

```
            .setMaterials(materials)
            .load('/static/models/Security cameras v3.1.obj',
function (object) {

                object.position.set(0,0,0);
                object.scale.setScalar(0.01);
                object.rotateY(Math.PI);
                const text = document.createElement('div');
                text.className = 'label';
                text.style.color = 'rgb(0,0,255)';
                text.textContent = "CAMERA";
                const label = new CSS2DObject(text);
                label.position.copy(object.position);
                label.position.y = 10;
                label.layers.set(0);
                object.castShadow = true;
                object.receiveShadow = true;
                root.add(label)
                root.add(object)
                scene.add(root)
                render()

            }, onProgress);

  });
```

Для коректного завантаження моделі використаємо завантажувач .mtl та .obj файлів. Встановлюємо позицію моделі як координату 0,0,0 та масштаб моделі 1% від оригінального розміру. Для впевненості у коректному завантажуванні додамо метод onProgress, який буде передавати хід процесу завантаження у консоль браузера. Серед параметрів також важливо передати параметри тіні та процесу передачі тіні на супутні об’єкти.

Після створення моделі потрібно розмістити об’єкт на моделі у потрібній точці. Для розміщення необхідно додати код, який дозволить отримати координати об’єкту, та переміщувати об’єкт за допомогою миші.

Код наведений у лістингу 4.

Лістинг 4 – код подій переміщення та графічних підказок

```
var gui = new GUI();
const objectFolder = gui.addFolder('Object');
objectFolder.add(detection_object.position, 'x', -20,
20).listen()
objectFolder.add(detection_object.position, 'y', -20,
20).listen()
objectFolder.add(detection_object.position, 'z', -20,
20).listen()

function onPointerDown(event) {

    onDownPosition.x = event.clientX;
    onDownPosition.y = event.clientY;

}

function onPointerUp(event) {

    onUpPosition.x = event.clientX;
    onUpPosition.y = event.clientY;

    if (onDownPosition.distanceTo(onUpPosition) === 0) {

        transformControl.detach();
        render();
```

```
    }

}

function onPointerMove(event) {

    pointer.x = (event.clientX / window.innerWidth) * 2 - 1;
    pointer.y = - (event.clientY / window.innerHeight) * 2 + 1;



    raycaster.setFromCamera(pointer, camera);

    const intersects =
raycaster.intersectObjects(splineHelperObjects, false);

    if (intersects.length > 0) {

        const object = intersects[0].object;

        if (object !== transformControl.object) {

            transformControl.attach(object);

        }}}
```

Функція переміщення об’єкту працює за допомогою технології raycaster, яка є складовою фреймворку ThreeJS. Після додавання цього коду модель буде мати вигляд як на рисунку 16.

У полях графічного підкажчика знаходяться координати x,y,z об’єкта, що переміщується. Також за допомогою підкажчика є можливість провести точне налаштування позиції об’єкта. На рисунку 17 зображений об’єкт, що був розташований

згідно вимірів положення тегу, що був зображений на рисунку 15. Координати об’єкту з моделі заносимо у окремий .txt файл.

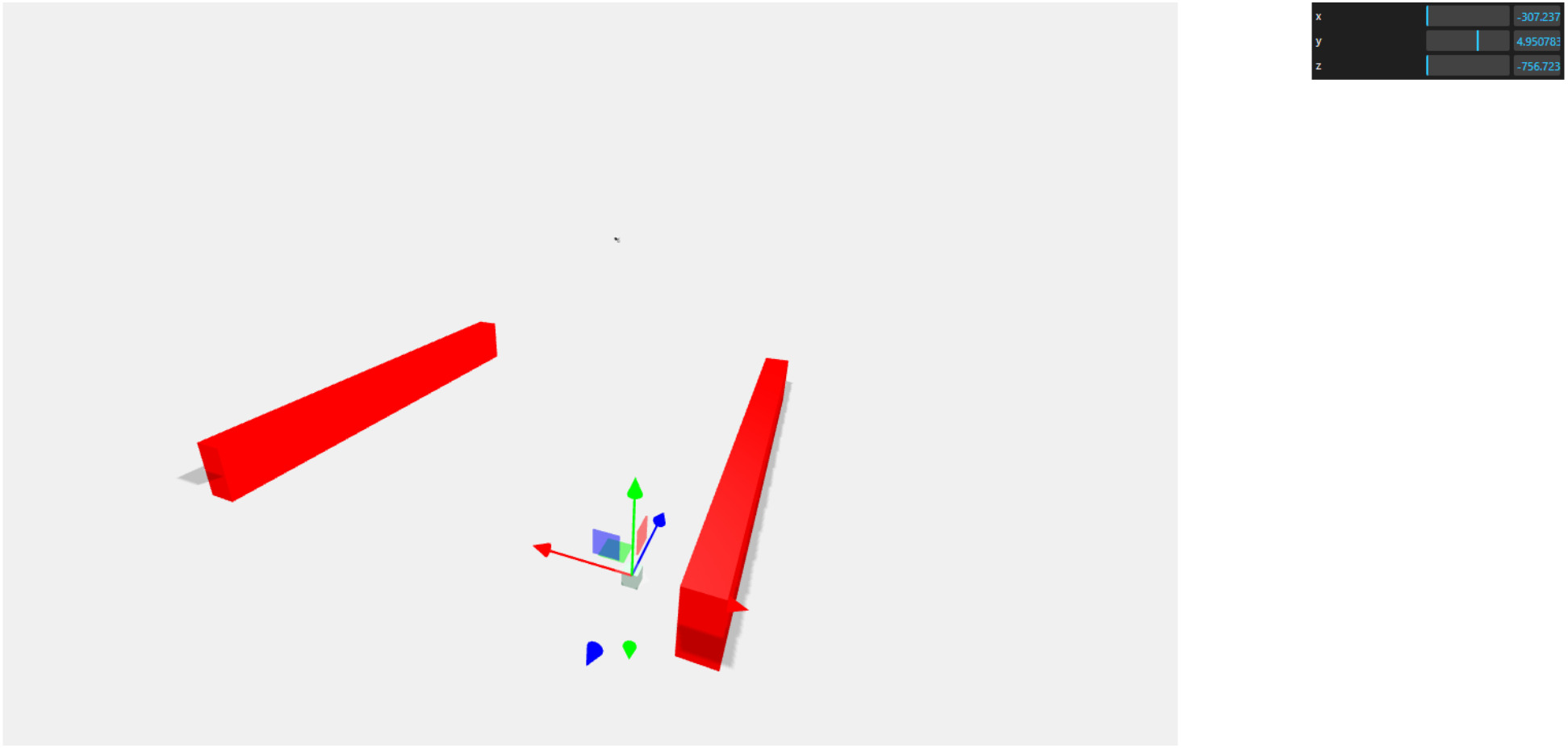

Рисунок 16 – Модель для калібрування

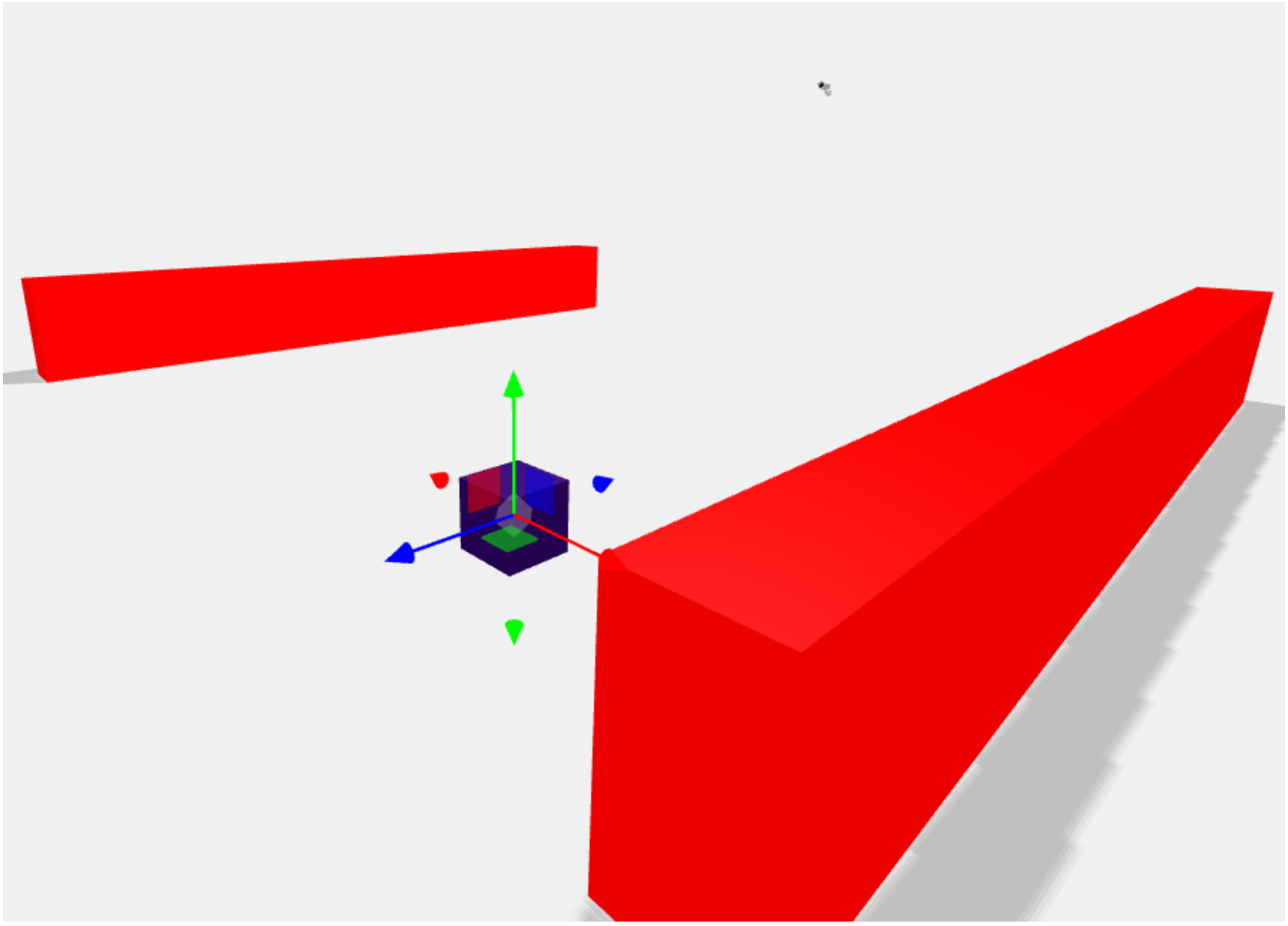

Рисунок 17 – Розташування першого тега

Для виконання операції калібрування був створений скрипт, який наведений у лістингу 5

Лістинг 5 – Скрипт калібрування

```
def rigid_transform_3D(A, B):
    assert len(A) == len(B)

    N = A.shape[0]  # total points

    centroid_A = mean(A, axis=0)
    centroid_B = mean(B, axis=0)

    # centre the points
```

```
        AA = A - tile(centroid_A, (N, 1))
        BB = B - tile(centroid_B, (N, 1))

        # dot is matrix multiplication for array
        H = transpose(AA) * BB

        U, S, Vt = linalg.svd(H)

        R = Vt.T * U.T

        # special reflection case
        if linalg.det(R) < 0:
            print("Reflection detected")
            Vt[2, :] *= -1
            R = Vt.T * U.T

        t = -R*centroid_A.T + centroid_B.T

        #print (t)
        return R, t


    R, t = rigid_transform_3D(apriltag_model, world_model)
    print(f'R:{R}')
    print('----------------------------')
    print(f't:{t}')
    rotation = pickle.dumps(R, protocol=2)
    translation = pickle.dumps(t, protocol=2)
    client = MongoClient('localhost', 27017)
    db = client.diploma_camera_database
    plot_data = db.diploma_camera_calibration
    retrieved = plot_data.count_documents({"camera_id":
camera_id})
```

```
if retrieved == 0:
    record = {"camera_id": camera_id,
              'R': rotation, 't': translation}
    post_id = plot_data.insert_one(record).inserted_id
else:
    for doc in plot_data.find({"camera_id": camera_id}):
        plot_data.update_one({
            '_id': doc['_id']
        }, {
            '$set': { 'R': rotation, 't': translation}
        }, upsert=False)

print("done")
sys.stdout.flush()
```

Принцип роботи скрипта детально був описаний у розділі 2.2.1, та був створений на базі дослідження [5]. Основним функціоналом цього скрипта є метод `rigid_transform_3D`, який приймає в якості аргументів масиви A та B. Масив A – це масив координат у системі камери, а масив B – у системі моделі. Далі виконуються обчислення згідно формул 2.3 та 2.4. Наступним кроком є накопичення матриці H з формули 2.5, та виконання сингулярного розкладення за формулою 2.6. Результати калібрування є матриця обертання R та вектор трансформації t. Для подальшої роботи ці дані ком пресуються за допомогою модуля pickle та заносятся у базу даних проекту.

3.2.2 Програмна архітектура

На рисунку 18 зображена діаграма компонентів системи. Система ділиться на дві частини, які передають дані через сокет.

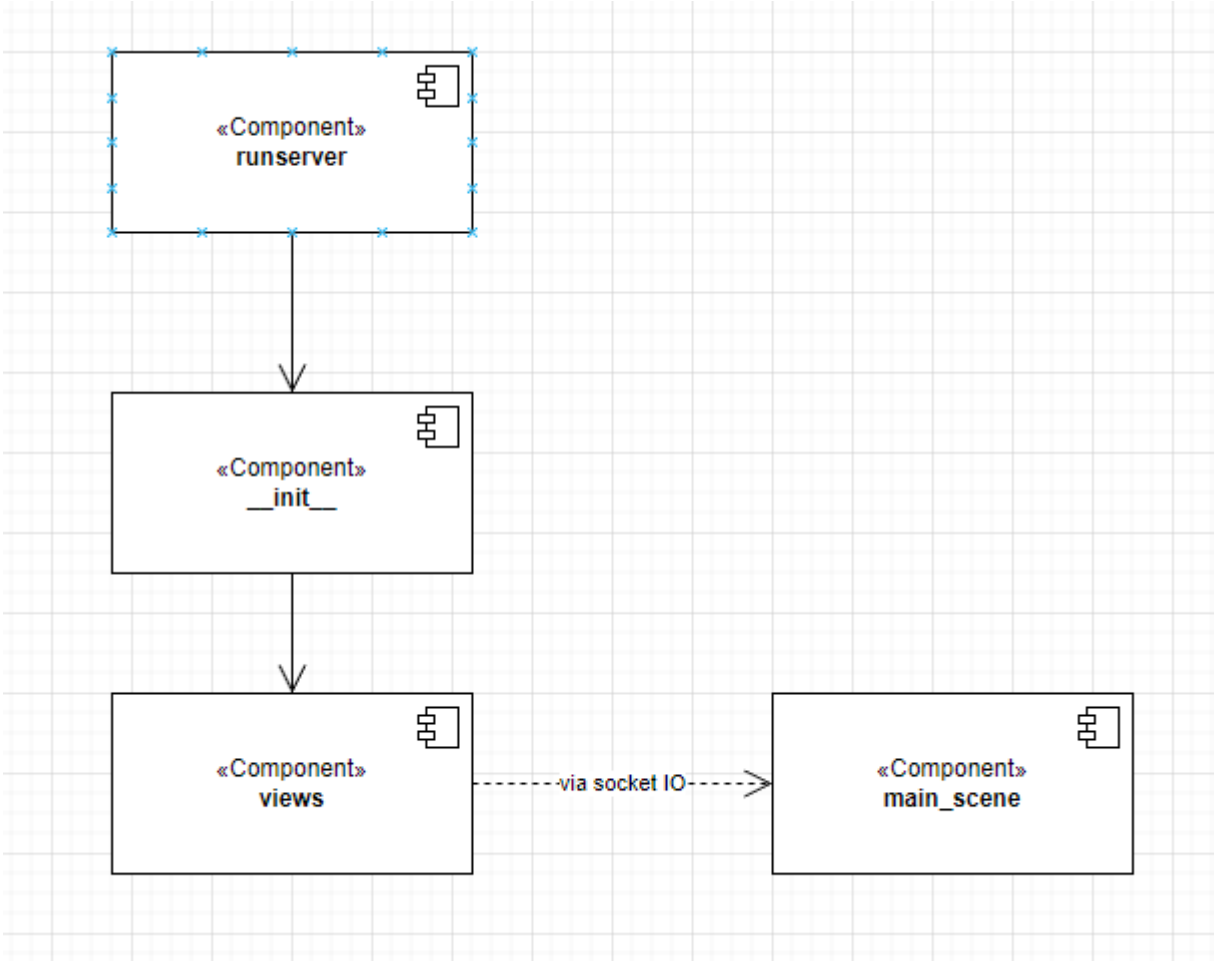

Рисунок 18 – Діаграма компонентів

У лівій частині діаграми знаходяться компоненти runserver, __init__ та views, що складають серверну частину застосунку. Кожен з цих компонентів є модулем Python.

Функціонал та принцип роботи серверної частини:

1. Модуль runserver створює об’єкт застосунку, визначає адресу хоста та порт на якому буде працювати застосунок.
2. Модуль __init__ визначає, що застосунок є об’єктом Flask.
3. Модуль views містить основний функціонал системи.

У правій частині діаграми знаходиться компонент main_scene, який є скриптом на мові javascript. Основною функцією цього скрипта є створення 3D моделі на головній сторінці застосунку.

Основними функціями програми є функції video_feed() та processing(). Метод video_feed() запускає метод processing() та отримує у результаті циклу фрейм відео потоку зі знайденими тегами. Метод processing() у свою чергу завантажує відеоряд з камери, отримує розподільну здатність кадру, запускає детектор тегів та виконує кроки алгоритму пошуку тегів згідно розділу 2.1 цієї роботи. Після знаходження тегів засобами бібліотеки OpenCV малює в кадрі рамку навколо знайдених тегів, виконує перетворення координат тегів у систему моделі та пересилає ці координати до модулю main_scene за допомогою SocketIO. Після завершення усіх операцій фрейм відео потоку перетворюється у масив байтів та повертається у циклі до методу video_feed().

3.2.3 Реалізація основного функціоналу

В лістингу 6 зображено метод `processing()`, який реалізує процес обробки відео потоку та роботу з окремими кадрами для визначення положення тегу у просторі.

Спочатку задаються необхідні константні значення. Так як ми працюємо з відео потоком, який був відзнятий попередньо, то ми повинні задати шлях до відео. Також ми задаємо розмір тегу, який бере участь у роботі. Наступним кроком буде вираховування розподільної здатності відео. Для цього завантажується метод `get_resolution(video_name)`, який приймає у аргументах повний шлях до відео. Принцип роботи цього метода є в тому, що метод засобами OpenCV завантажує один кадр з відео, отримує ширину та висоту кадру у пікселях та повертає результати у функцію `processing`(). Наступним кроком задаємо фокусну відстань камери, яку ми можемо вирахувати за допомогою формули 2.2. Інформація про розподільну здатність та фокусну відстань потрібна для оформлення матриці параметрів камери. Далі створюється об’єкт класу `VideoCapture`, який приймає в якості аргументу шлях до відео, та об’єкт класу `Detector` з бібліотеки pupil_apriltags.

Клас `Detector` потребує для ініціалізації параметри сім'ї тегів, які будемо шукати (за замовченням це буде сімейство 36h11) та число потоків на яких він може працювати.

Після завершення підготовки запускається цикл, у якому в кожну ітерацію завантажується новий кадр з відео, та якщо завантаження пройшло успішно, то виконується наступний алгоритм:

1. Робиться копія кадру та видаляються кольори у цій копії. Окрім цього ініціалізується масив для координат тегу.
2. Виконується метод `detect` з бібліотеки pupil_apriltag для аналізу зображення на наявність тегів. В якості аргументів передається кадр з кроку 1, булевий параметр для підтвердження аналізу позиції тегу в кадрі, матрицю параметрів камери та розмір тегу.
3. Далі для кожного тегу малюється рамка навколо тегу та додаються знайдені координати до масиву з кроку 1.
4. Перевіряється масив на наявність даних та у разі наявності перетворюється у формат JSON та відправляється через SocketIO
5. Наприкінці циклу кадр з рамками навколо тегів перетворюється у формат .jpg та конвертується у масив бітів. Після виконання цих операцій зображення повертається у метод `video_feed()` за допомогою ключового слова `yield`.

Лістинг 6 – Реалізація функціоналу обробки відео потоку та роботи з тегами

```
def find_marker(ptA, ptB, ptC, ptD):
L = [ptA, ptB, ptC, ptD]
ctr = np.array(L).reshape((-1, 1, 2)).astype(np.int32)
return cv2.minAreaRect(ctr)

def get_resolution(video):
        cap = cv2.VideoCapture(video)
        if not cap.isOpened():
            print("Cannot open camera")
            exit()
```

```
            ret1, frame1 = cap.read()
            fshape = frame1.shape
            fheight = fshape[0]
            fwidth = fshape[1]
            cap.release()
            return [fheight/2,fwidth/2]

    def processing():
        #camera_id = "10cm_2"
        video_name =
'E://source/repos/Diploma/DetectionAppServer/DetectionAppServer
/DetectionAppServer/static/videos/10cm_1.avi'
        tag_size = 0.1
        cy,cx = get_resolution(video_name)
        focalLength = cx/np.tan(69.7)
        camera_params = [focalLength,focalLength,cx,cy]
        cap = cv2.VideoCapture(video_name)
        detector = Detector(families='tag36h11',nthreads=8)
        if not cap.isOpened():
            print("Cannot open camera")
            exit()
        frame = 0
        while True:
            frame+=1
            ret, img = cap.read()
            if ret == True:
               if frame%5==0:
                 data = []
                 gray = img[:,:,1]
                 try:
                   results =
detector.detect(gray,True,camera_params,tag_size=tag_size)
                 except BaseException as err:
```

```
                    results = None

                for r in results:
                    center = r.center.astype(np.int32)
                    font = cv2.FONT_HERSHEY_SIMPLEX
                    (ptA, ptB, ptC, ptD) = r.corners
                    ptB = (int(ptB[0]), int(ptB[1]))
                    ptC = (int(ptC[0]), int(ptC[1]))
                    ptD = (int(ptD[0]), int(ptD[1]))
                    ptA = (int(ptA[0]), int(ptA[1]))
                    marker = find_marker(ptA, ptB, ptC, ptD)
                    box = cv2.cv.BoxPoints(marker) if
imutils.is_cv2() else cv2.boxPoints(marker)
                    box = np.int0(box)
                    cv2.drawContours(img, [box], -1, (0, 0,
255), 2)
                    cv2.putText(img, "tag_id = %s" %
r.tag_id,(ptD[0], ptD[1]), cv2.FONT_HERSHEY_SIMPLEX, 2.0, (0,
255, 0), 3)
    res = [r.pose_t[0],r.pose_t[1],r.pose_t[2]]
    q = R.dot(res)+t
    record = {"tag_id": r.tag_id, "coords":
[float(q[0]),float(q[1]),float(q[2])]}
    if r.tag_id!=0:
        data.append(record)
    if len(data)>0:
                        try:
                         bsondoc = json.dumps(data,
indent=4, sort_keys=True, default=str)
    socketio.emit('update_position', {'data': bsondoc})

                        except socket.error as er:
```

```
                                        print ("Error Occured:
",er)
                                        #s.close()
                                        break

                    (flag, encodedImage) = cv2.imencode(".jpg",
img)
                    yield(b'--frame\r\n' b'Content-Type:
image/jpeg\r\n\r\n' + bytearray(encodedImage) + b'\r\n')
            else:
                 break
```

Перетворення координат об’єкту у систему моделі виконується за допомогою формули 2.8, яка у коді виглядає наступним чином:

```
q = R.dot(res)+t
```

В результаті отримаємо матрицю координат об’єкту, яка передається на фронтенд, як вказано у п.4. Матрицю R та вектор t завантажуємо один раз під час завантаження головної сторінки. Код завантаження наведений у лістингу 7.

Лістинг 7 – код завантаження матриці R та вектора t

```
client = MongoClient('localhost', 27017)
db = client.diploma_camera_database
calib_data = db.diploma_camera_calibration
x = calib_data.find_one({"camera_id":1})
if x != None:
    R = pickle.loads(x["R"])
    t = pickle.loads(x["t"])

else:
    R = 0
  t = 0
```

3.2.3 Графічний інтерфейс користувача та функціональні можливості

На рисунку 19 зображено графічний інтерфейс застосунку з позначеними елементами.

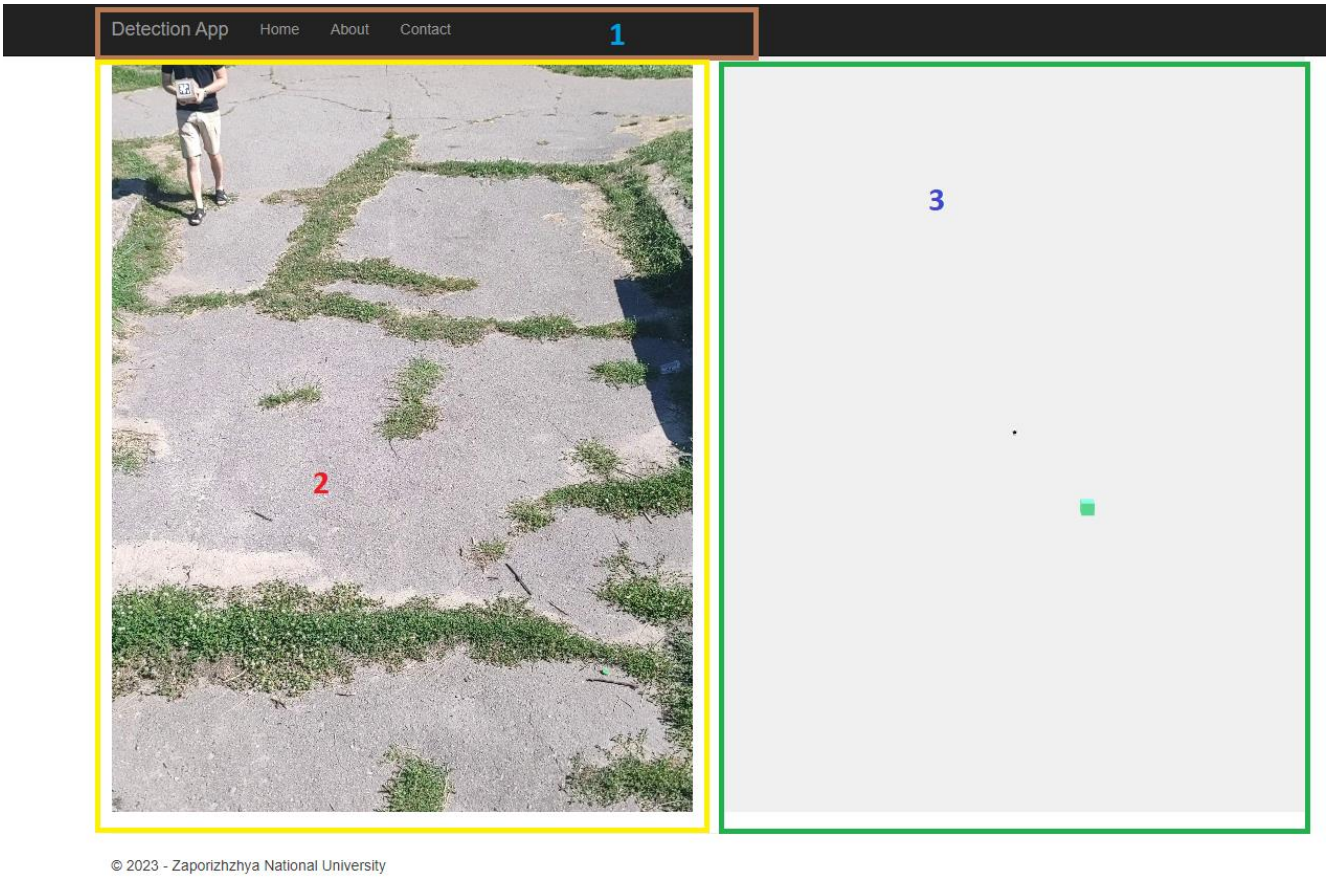

Рисунок 19 – Графічний інтерфейс користувача

Головна сторінка має в своєму складі наступні елементи:

1. Навігаційне меню.
2. Шар з відео.
3. Блок з 3D моделлю.

Навігаційне меню має у своєму складі посилання на сторінки: головна, про розробника та контакти розробника. Шар з відео потрібен для демонстрації відеоряду з промаркованими тегами. Якщо тег був знайдений, то він буде обведений зеленим контуром та буде відображено id тега.

Особливу увагу викликає блок 3, де знаходиться скрипт з 3D моделлю. Оскільки код створення 3D моделі був написаний під час підготовки системи, він не

потребує додаткових виправлень. Проте для успішної роботи системи необхідно додати код, який буде приймати дані з працюючого модуля обробки відео.

Прийом позицій об’єктів виконується за допомогою засобів socketIO. Створюється підключення до сокету та після отримання пакету, який символізує успішність підключення, починає приймати параметри розташування тегів. Цей процес наведений у лістингу 8.

Лістинг 8 – Отримання даних з сокету

```
var socket = io.connect('http://' + document.domain + ':' +
location.port);



socket.on('my response', function (result) {
     console.log('socket connected')



})
socket.on('update_position', function (result) {
     var docs = []
     var client_buffer = result.data.toString()
     try {
          client_buffer = JSON.parse(client_buffer)
          docs = client_buffer;
          detection_object.position.set(docs[0].coords[0] *
100*-1, docs[0].coords[1] * 100, docs[0].coords[2] * 100)
          render()
     }
     catch {
         console.log("Record failed. Start next record")
     }
     console.log(client_buffer)
})
```

Інформація про розробника представлена на сторінці About. Зміст сторінки наведений на рисунку 20.

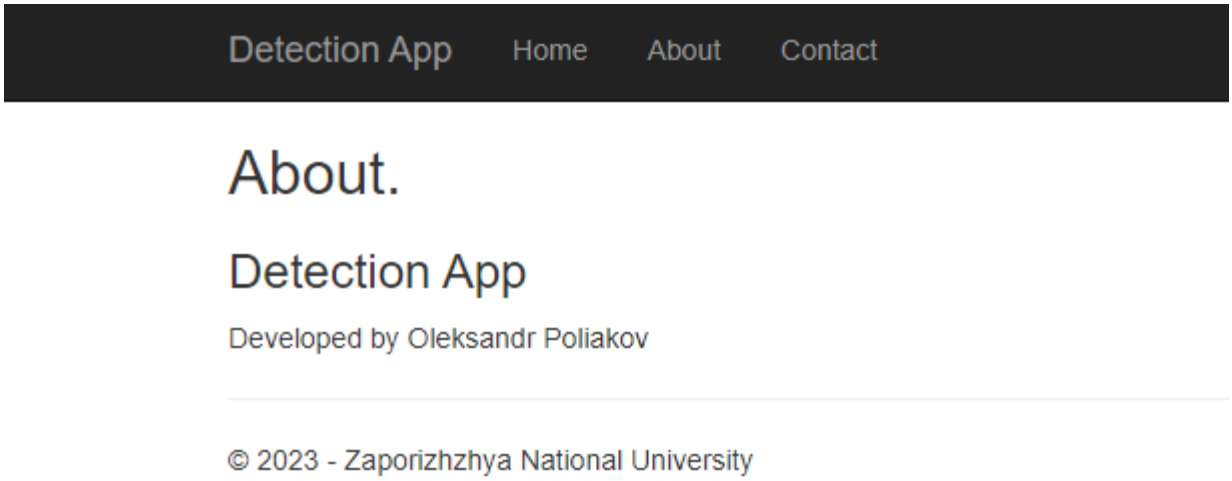

Рисунок 20 – Сторінка About

3.3 Висновки з розділу 3

1. Було налаштоване середовище для роботи з тегами Apriltag.
2. У налаштованому середовищі були написані скрипти для обробки тегів, скрипти для зображення даних на 3D моделі та обробка положення тегу відносно системи координат моделі.
3. Була протестована якість розпізнавання тегів на представленому відеоряді.
4. Була розроблена комп’ютерна система, що відстежує положення об’єктів у реальному часі на основі підготовленого відеоряду. В процесі роботи були описані її функції та програмна архітектура. Створені основний функціонал, графічний інтерфейс та усі необхідні методи для успішної роботи системи.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ РОБОТИ КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ ВИЗНАЧЕННЯ ПОЛОЖЕННЯ ОБ'ЄКТУ В ОБМЕЖЕНОМУ ПРОСТОРІ У РЕАЛЬНОМУ ЧАСІ

4.1 Результати роботи системи для розпізнавання об'єктів

4.1.1 Аналіз результатів розпізнавання тегів

Під час роботи над системою було протестовано 5 відео, відзнятих для цієї роботи. З них три відео були відзняті при нормальному освітленні, а останні два при дуже яскравому сонячному освітленні. Кожне відео мало в своєму складі більше тисячі відеокадрів. Якщо порівнювати якість тегів, то чотири відео мали теги, роздруковані з високою контрастністю, а одне відео мало тег, роздрукований з низькою контрастністю та не розташований на жорсткій основі. Це напряму вплинуло на якість розпізнавання об'єкту. Дані щодо результатів тестування системи наведені у таблиці 4.

Таблиця 4 – результати роботи системи

№	Усього кадрів	Кадри з тегами	Процент успішності
1	1165	883	75,79
2	1100	819	74,45
3	1010	801	79,30
4	1150	460	40
5	1075	270	25,11

Процент успішності роботи було вираховувано за допомогою формули 2.9. Як було виявлено у процесі роботи, освітлення дуже суттєво впливає на якість розпізнавання тегу. Також суттєво впливає основа, на якій був розміщений тег. Якщо тег був розміщений на жорсткій поверхні, то процент успішності буде значно вищий. Як наведено у таблиці 4, у порівнянні з відео 1, відео 5 має значно нижчий процент успішного розпізнавання тегів. Під час аналізу результатів розпізнавання

тегів було виявлено ще один критерій успішності розпізнавання – розподільна здатність відеопотоку. При використання відеоматеріалу з низькою розподільною здатністю (менше ніж 1280x562 пікселів) якість розпізнавання тегів при ідеальних умовах (висококонтрасний друк тега, жорстка основа, рівне освітлення) буде на 50% нижче. Оскільки середня розподільна здатність відеопотоку камери відео нагляду зазвичай вище цього параметра, то цим критерієм можна знехтувати.

Узагальнюючи результати роботи системи, можна виділити високий процент розпізнавання тегів при нормальному освітленні та при якісному роздрукуванні тегів.

4.1.2 Аналіз точності роботи системи на прикладі 3D моделі

На рисунку 21 наведено приклад роботи системи з підключеним графічним помічником, який дозволяє побачити координати об’єкту у системі моделі.

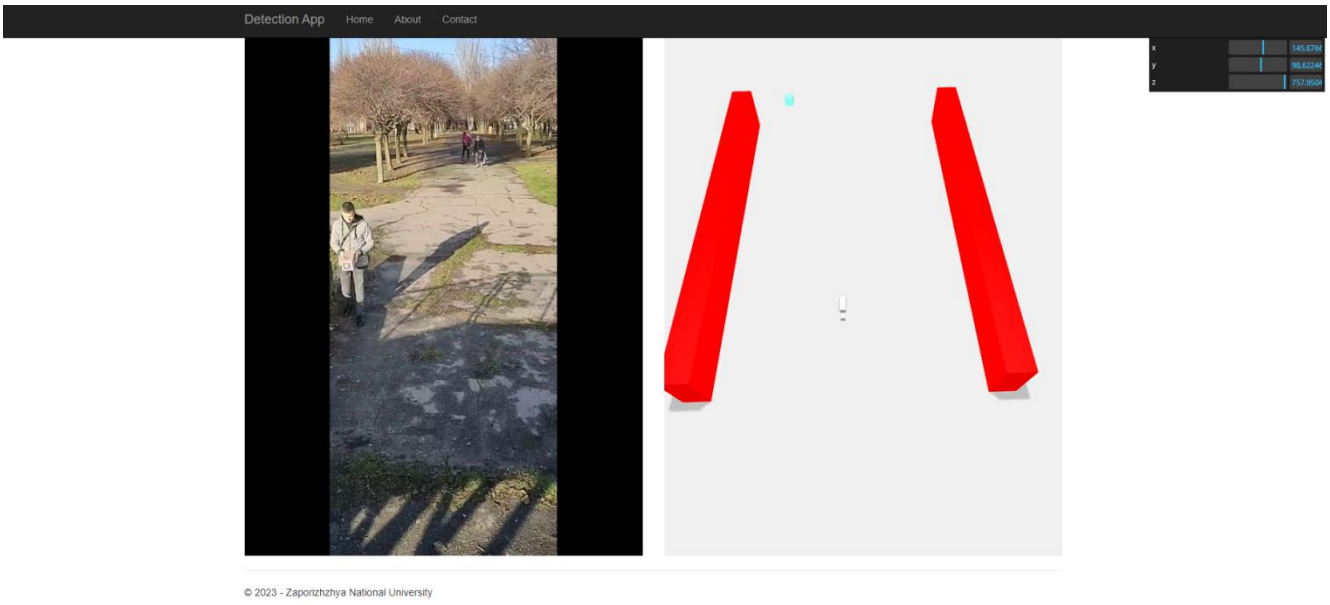

Рисунок 21 – Приклад працюючої системи

У наведеному прикладі можна побачити, що об’єкт знаходиться на відстані 757 см від камери, на висоті 98 см від поверхні та на відстані 145 см від центра камери. Виміри у реальному просторі показують, що саме у цій позиції об’єкт знаходиться на відстані 7.6 метрів (760 см) від камери, висота від поверхні складає 95

см, а розміщення по горизонталі складає 150 см від центру. Базуючись на цих вимірах можна зробити висновок, що найбільша похибка складає 5 см по горизонталі і не більше ніж 1 см при визначенні положення об’єкта по вісі Y та Z.

Найбільше на якість визначення положення об’єкта впливає процес калібрування камери. Так як алгоритм калібрування базується на двох матрицях: матриці положень об’єкта у системі камери та матриці положення об’єкта у системі моделі, то розглянемо окремо вірогідні недоліки при роботі з кожною з них.

При складанні матриці положень об’єкта у системі камери, тобто матриці положень тегу Apriltag, є висока вірогідність неправильно заповнити матрицю параметрів камери перед запуском алгоритму пошуку тега. По-перше, необхідно досить точно знати фокусну відстань камери. По-друге, перед початком роботи необхідно пересвідчитися, що правильно вказаний реальний розмір тега при створенні екземпляру класа Detector(). По-третє, при створенні датасету для калібрування особливу увагу слід приділити орієнтирам у просторі, тобто кадр з визначеним тегом повинен бути жорстко прив’язаний до чітко виміряної позиції у просторі.

При складанні матриці положень об’єкта у системі моделі особливу увагу потрібно приділити одиницям виміру при створенні 3D простору. Різні одиниці виміру при розташуванні об’єкту у просторі моделі призведе до критичної похибки при створенні матриці обертання та вектора трансформації для подальшої роботи.

При коректній роботі з системою достатньо усього трьох окремих позицій тега у просторі для того, щоб виконати калібрування, та отримати положення об’єкта у просторі з високою точністю.

4.1.3 Варіанти покращення точності роботи системи

Для покращення точності роботи системи, окрім усунення недоліків, описаних у розділі 4.1.2, можна віднести ще проблему фізичного характеру, а саме викривлення по типу «Риб’яче око» [11]. Спотворення присутні у всіх об’єктивах типу «риб’яче око» та більшості ширококутних об’єктивів. Крім того, спотворення є функцією кута поля зору і зазвичай зростає зі збільшенням поля зору камери. Для усунення цієї проблеми можна використати вбудований пакет функцій бібліотеки

OpenCV cv2.fisheye, який базується на алгоритмах з досліджень [11] та [12]. Виконання цього алгоритму потребує попередньо роздрукованого зображення шахової дошки. Принцип роботи алгоритму калібрування полягає у наступних кроках:

1. Отримується кадр з відео потоку;
2. З кадру видаляються кольори, як для алгоритму пошуку Apriltag;
3. Засобами бібліотеки OpenCV знаходяться кути шахової дошки.
4. Створюється матриця кутів шахової дошки;
5. Виконується калібрування зображення за допомогою функції cv2.fisheye.calibration
6. Повертаються параметри розподільної здатності зображення та матриці K і D. K і D – це матриці параметрів, котрі властиві тільки об’єктиву, на якому проводиться калібрування. Результати, отримані у цьому кроці потрібно занести в базу даних для подальшого використання.

Після виконання калібрування необхідно при обробці відео потоку використовувати функцію cv2.fisheye.initUndistortRectifyMap та функцію cv2.remap на кожному з кадрів для перетворення зображення у неспотворене «риб’ячим оком» зображення. Виконання цих рекомендацій, а також усунення недоліків, описаних у розділі 4.1.2 дозволить визначити положення об’єкта у просторі з найвищою точністю.

4.2 Результати роботи розробленої комп’ютерної системи для визначення положення об’єкту в обмеженому просторі у реальному часі

4.2.1 Аналіз використання пам’яті системою

Для вимірювання швидкості роботи системи було використано засоби бібліотеки memory_profiler. Ця бібліотека дозволяє за допомогою декораторів зробити

профілювання коду, написаного на мові Python та отримати основні показники використання пам’яті у системі під час роботи програми.

Для використання бібліотеки її потрібно встановити через менеджер пакетів pip, виконавши команду pip install memory_profiler та додати до коду програми за допомогою команди from memory_profiler import profile.

Наступним кроком потрібно додати декоратор @profile перед модулем, який потрібно профілювати. Результати профілювання наведені у таблиці 5

Щодо інших параметрів, то тести, виконані за допомогою бібліотеки psutil показали використання оперативної пам’яті у 0,14 Гб та завантаження центрального процесора не більш ніж на 3% від нормального стану, що дозволяє зробити висновок, що система працює успішно.

Таблиця 5 – Результати профілювання системи

Етапи програми	Розподільна здатність відео			
	1280x562		3840x2160	
	Збільшення витрат пам’яті на етапі, МіВ	Усього витрачено пам’яті, МіВ	Збільшення витрат пам’яті на етапі, МіВ	Усього витрачено пам’яті, МіВ

Початок роботи програми	0	72	0	67.4
Створення екземпляру класа Detector	35.3	107.3	35.3	108.6
Зчитування кадра відео	170.7	279.3	190.0	297.3
Запуск методу detect	5,0	284,3	1.9	299.2
Декодування зображення	5.1	289,4	5.1	341,5

4.2.2 Варіанти покращення точності роботи системи

Варіанти покращення точності роботи системи наведені у розділах 4.1.2 та 4.1.3. При виконанні цих рекомендацій можна суттєво покращити точність відстежування положень об’єкта у просторі. Особливу увагу слід приділити проблемі «риб’ячого ока», якщо для роботи з системою використовується ширококутний об’єктив.

При створенні системи відстежування об’єктів за допомогою технології Apriltag потрібно розуміти, що ця система потребує якісно роздрукованих тегів та чіткого зображення тегів у кадрі відео потоку. При низькому освітленні або навпаки при дуже яскравому освітленні можна побачити, що неякісно роздруковані теги на поганій основі можуть не визначатися, що у свою чергу призведе до втрати актуального положення об’єкту у просторі.

Незважаючи на цей недолік, у сферах, які були описані у розділі 1.3 цієї кваліфікаційної роботи, зазвичай рівне штучне освітлення, а тому підхід з використанням технології Apriltag є найбільш оптимальним для визначення положення об’єктів у реальному часі.

4.3 Висновки з розділу 4

1. Розподільну здатність відеопотоку, з яким працювала система, оптимально обрати не вище за 3840x2160 пікселів.

2. Покращення роботи системи напряму залежить від процесу калібрування системи. Рекомендації для покращення точності роботи системи були наведені у розділах 4.1.2 та 4.1.3.

3. Похибка при визначенні положення об’єкту склала 5 см по горизонталі і не більше ніж 1 см при визначенні положення об’єкта по вісі Y та Z.

4. При використанні якісно роздрукованих тегів та нормального освітлення процент розпізнавання складає більше 74% .

ВИСНОВКИ

1. У процесі роботи була досліджена проблема визначення положення об’єкту у реальному часі: були розглянуті переваги та недоліки існуючих засобів та методів її вирішення, розглянуті приклади сфер застосування систем відстежування положень об’єкту у реальному часі.

2. Були досліджені засоби для вирішення проблеми відстежування об’єктів. Для роботи з відео була обрана бібліотека OpenCV, для відстежування об’єктів було вирішено застосувати технологію Apriltag, для відображення положення об’єкту у просторі було обрано бібліотеку ThreeJS .

3. Проведено калібрування камери, була створена матриця обертання та вектор трансформації. Для проведення процесу калібрування був створений датасет з координатами об’єкту у системі камери та підготовлені вимірювання положення об’єкта у реальному просторі, створене середовище для калібрування, а також було успішно проведено сам процес калібрування камери.

4. Була розроблена готова до впровадження система відстежування об’єктів у реальному часі, яка працює на основі відео потоку з камери.

5. Були досліджені результати роботи системи, визначені недоліки у роботі системи та надані рекомендації по їх усуненню.

6. Досліджені результати роботи розробленої комп’ютерної системи мають найбільшу похибку у 5 см, що є дуже добрим результатом у сферах її використання. Було визначено, що процес калібрування напряму впливає на якість визначення положення об’єкту.

7. Поставлена проблема визначення положення об’єкта у реальному часі повністю вирішена з допустимою точністю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Woodman, Oliver & Harle, Robert. (2008). Pedestrian Localisation for Indoor Environments. UbiComp 2008 - Proceedings of the 10th International Conference on Ubiquitous Computing. 114-123. 10.1145/1409635.1409651.

2. Hightower, Jeffrey & Borriello, Gaetano. (2001). Location Systems for Ubiquitous Computing. Computer. 34. 57 - 66. 10.1109/2.940014.

3. Mautz R. Indoor positioning technologies. ETH Zurich Research Collection. 2012. URL:https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/54888/eth-5659-01.pdf (дата звернення: 14.12.2022).

4. Abbas, Syed Muhammad, Salman Aslam, Karsten Berns, and Abubakr Muhammad. 2019. "Analysis and Improvements in AprilTag Based State Estimation"*Sensors* 19, no. 24: 5480.

5. Sorkine-Hornung Olga, Rabinovich Michael. Least-Squares Rigid Motion Using SVD. — 2017. — January. — Technical note.

6. Baichuan Huang, Jun Zhao, Jingbin Liu. A Survey of Simultaneous Localization and Mapping with an Envision in 6G Wireless Networks. Nanyang Technological University, Singapore, Wuhan University, Wuhan, China. 2020. URL: https://arxiv.org/pdf/1909.05214.pdf (дата звернення: 30.04.2023).

7. Zicong J. Real-time object detection method based on improved YOLOv4-tiny. Journal of Network Intelligence. 2022. Vol.7, no.1. URL: https://arxiv.org/ftp/arxiv/papers/2011/2011.04244.pdf (дата звернення: 30.04.2023).

8. Ashkan Goharfar, Jaber Babaki, Mehdi Rasti, Pedro H. J. Nardelli. Indoor Positioning via Gradient Boosting Enhanced with Feature Augmentation using Deep Learning. Department of Computer Engineering, Amirkabir University of Technology, Tehran, Iran,. URL: https://arxiv.org/pdf/2211.08752.pdf (дата звернення: 30.04.2023).

9. Yuzhuo Ren, Feng Hu. CAMERA CALIBRATION WITH POSE GUIDANCE. NVIDIA, Autonomous Vehicle, Santa Clara, CA, USA. 2021. URL: https://arxiv.org/abs/2102.10202 (дата звернення: 30.04.2023).

10. Jin Wu, Ming Liu, Zebo Zhou and Rui Li. Fast Rigid 3D Registration Solution: A Simple Method Free of SVD and Eigen-Decomposition. IEEE.2018. URL: https://arxiv.org/pdf/1806.00627.pdf (дата звернення: 30.04.2023).

11. M. Lee, H. Kim and J. Paik, "Correction of Barrel Distortion in Fisheye Lens Images Using Image-Based Estimation of Distortion Parameters," in *IEEE Access*, vol. 7, pp. 45723-45733, 2019, doi: 10.1109/ACCESS.2019.2908451.

12. Cłapa, Jakub & Blasinski, Henryk & Grabowski, Kamil & Sekalski, Przemyslaw. (2014). A fisheye distortion correction algorithm optimized for hardware implementations. Proceedings of the 21st International Conference on Mixed Design of Integrated Circuits and Systems, MIXDES 2014. 415-419. 10.1109/MIXDES.2014.6872232.

13. Wei, Jin & Li, Chen-Feng & Hu, Shi-Min & Martin, Ralph & Tai, Chiew-Lan. (2011). Fisheye Video Correction. IEEE transactions on visualization and computer graphics. 18. 10.1109/TVCG.2011.130.

14. Liu, Qingyun & Wu, Tianyue & Zhu, Qing & Yan, Chunhuizi. (2023). Research on Target Pose Measurement Based on AprilTag Identification. 10.1007/978-981-19-7184-6_35.

15. Huang, Xin & Liu, Zuoshi & Cheng, Pengsheng. (2022). Design of Target Tracking System Based on Apriltag. 10.3233/ATDE220222.

16. Поляков О. ВІДСТЕЖУВАННЯ ОБ’ЄКТІВ У РЕАЛЬНОМУ ЧАСІ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЇ APRILTAG. ЗБІРНИК наукових праць студентів, аспірантів, докторантів і молодих вчених «МОЛОДА НАУКА-2023». Запоріжжя: Запорізький національний університет, 2023. С. 114–116.

17. Поляков О. АВТОМАТИЗОВАНА СИСТЕМА ВІДСТЕЖУВАННЯ ОБ’ЄКТІВ У РЕАЛЬНОМУ ЧАСІ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ APRILTAG. III Всеукраїнської науково-практичної конференції за участю молодих науковців «АКТУАЛЬНІ ПИТАННЯ СТАЛОГО НАУКОВО-ТЕХНІЧНОГО ТА СОЦІАЛЬНО-ЕКОНОМІЧНОГО РОЗВИТКУ РЕГІОНІВ УКРАЇНИ», Запоріжжя: Запорізький національний університет, 2023. С. 141–142.

Декларація

академічної доброчесності

здобувача вищої освіти ЗНУ

Я_____Поляков Олександр Олександрович , студент(ка)___2__курсу,

форми здобуття освіти ____денна_____,

Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ

Спеціальності121 Інженерія програмного забезпечення,

адреса електронної пошти __se22m-18@stu.zsea.edu.ua__,

підтверджую, що написана мною кваліфікаційна роб

ота на тему «**Автоматизована система відстежування об'єктів у реальному часі з використанням технології Apriltag** »

відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений/ознайомлена;

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

-згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою Інтернет-системи, а також на архівування роботи в базі даних цієї системи.

Дата_____ Підпис_____ Поляков О.О
(студент)

Дата_____ Підпис_____ Полякова Н.П.
(науковий керівник)