

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО

В.о. завідувача кафедри

_____ Олександр КОВАЛЬ

«___» _____ 2026 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Система визначення просторової орієнтації об'єктів для кінцевого
ефектора робота»

Виконав:

студент IV курсу, групи ТВ-23

Сербін Ілля Олександрович

(прізвище, ім'я, по батькові)

_____ (підпис)

Керівник:

Доцент, к.т.н., Варава Іван Андрійович

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

_____ (підпис)

Рецензент:

_____ (посада, науковий ступінь, вчене звання, прізвище та ініціали)

_____ (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень із праць інших авторів
без відповідних посилань.

Студент _____ (підпис)

Київ – 2026

**Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Олександр КОВАЛЬ

«___» _____ 2026р.

ЗАВДАННЯ

на дипломну роботу студенту

_____ Сербін Ілля Олександрович

(прізвище, ім'я, по батькові)

1. Тема роботи: Система визначення просторової орієнтації предметів для
кінцевого ефектора робота

керівник роботи Варава Іван Андрійович, к.т.н.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «28» травня 2026 р. №1992-с

2. Строк подання студентом роботи «10» червня 2026 р.

3. Вихідні дані до роботи: Вихідні зображення з RGB-камери, бібліотека OpenCV,
середовище ROS2 Jazzy, методи сегментації зображень, алгоритми виділення
контурів, метод Watershed, метод solvePnP для оцінки просторової орієнтації
об'єктів, роботизований маніпулятор Puma 560.

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): Провести аналіз існуючих систем комп'ютерного зору та методів
сегментації об'єктів. Розробити структуру системи на основі ROS2 із взаємодією
вузлів через ROS2 topics та custom message. Реалізувати алгоритм обробки
зображення з використанням OpenCV. Реалізувати сегментацію та виділення
контурів об'єктів на зображенні. Визначити геометричні параметри та просторову
орієнтацію об'єктів. Реалізувати алгоритм вибору оптимального способу
захоплення об'єкта. Інтегрувати систему комп'ютерного зору з системою

керування роботизованого маніпулятора у середовищі ROS2. Провести тестування системи та оцінити результати її роботи.

5. Перелік ілюстративного матеріалу: Загальна структура системи комп'ютерного зору; архітектура ROS2 системи; схема взаємодії ROS2 вузлів; схема алгоритму обробки зображення; схема сегментації об'єктів; схема визначення просторової орієнтації об'єкта; схема вибору способу захоплення; інтерфейс візуалізації результатів; приклади результатів сегментації; результати визначення координат та орієнтації об'єктів; результати тестування системи.

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	31.10.2025	
2	Дослідження предметної області	01.11.2025 – 31.11.2025	
3	Дослідження існуючих рішень	01.12.2025 – 31.12.2025	
4	Постановка вимог до проєктування системи	01.01.2026 – 31.01.2026	
5	Розробка програмного продукту	01.02.2026 – 03.05.2026	
6	Тестування	04.05.2026 – 12.05.2026	
7	Захист програмного продукту	13.05.2026	
8	Оформлення дипломної роботи	18.05.2026 – 27.05.2026	
9	Передзахист	03.06.2026	
10	Захист	15.06.2026 – 19.06.2026	

Студент

(підпис)

Сербін І.О.

(ім'я, ПРІЗВИЩЕ)

Керівник роботи

(підпис)

Варава І.А.

(ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 54 сторінки, 18 рисунків, 1 таблицю, 5 формул, 2 додатки та 20 посилань.

Метою роботи є розробка системи комп'ютерного зору для визначення просторової орієнтації об'єктів та вибору способу їх захоплення роботизованим маніпулятором у середовищі ROS2.

У роботі розглянуто сучасні підходи до побудови систем комп'ютерного зору для робототехнічних комплексів, методи сегментації зображень, виділення контурів та визначення просторового положення об'єктів. Проаналізовано можливості використання бібліотеки OpenCV та фреймворку ROS2 для створення розподілених програмних систем керування роботизованими комплексами.

Розроблена система забезпечує отримання зображень з камери, попередню обробку кадрів, сегментацію об'єктів, виділення контурів, визначення координат та кутової орієнтації об'єктів, оцінювання можливості їх захоплення різними типами захватних механізмів, вибір оптимального об'єкта та передачу результатів через ROS2 topics.

Практичне значення одержаних результатів полягає у створенні програмної системи, яка може бути використана як складова систем керування роботизованими маніпуляторами для автоматизованого виявлення об'єктів, оцінювання їх положення у просторі та планування операцій захоплення. Розроблене програмне забезпечення може бути використане в навчальних, дослідницьких та промислових робототехнічних системах.

Ключові слова: комп'ютерний зір, ROS2, OpenCV, роботизований маніпулятор, сегментація зображень, просторова орієнтація, визначення координат, захоплення об'єктів, контури, робототехніка.

ABSTRACT

Structure and scope of the thesis. The thesis consists of 54 pages, 18 figures, 1 table, 5 formulas, 2 appendices and 20 references.

The purpose of this work is to develop a computer vision system for determining the spatial orientation of objects and selecting an appropriate grasping method for a robotic manipulator in the ROS2 environment.

The thesis examines modern approaches to computer vision systems for robotic applications, image segmentation methods, contour extraction techniques, and object pose estimation algorithms. The capabilities of the OpenCV library and the ROS2 framework for developing distributed robotic software systems are analyzed.

The developed system provides image acquisition from a camera, image preprocessing, object segmentation, contour detection, estimation of object coordinates and orientation, evaluation of grasping feasibility for different end-effector types, selection of the most suitable object, and transmission of the obtained results through ROS2 topics.

The practical value of the obtained results lies in the development of a software system that can be used as a component of robotic manipulator control systems for automated object detection, spatial pose estimation, and grasp planning. The developed software can be applied in educational, research, and industrial robotic systems.

Keywords: computer vision, ROS2, OpenCV, robotic manipulator, image segmentation, spatial orientation, pose estimation, object grasping, contour detection, robotics.

ЗМІСТ

ВСТУП	7
1 ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ СИСТЕМИ КОМП'ЮТЕРНОГО ЗОРУ ДЛЯ РОБОТИЗОВАНОГО МАНІПУЛЯТОРА	8
1.1 Аналіз предметної області та постановка задачі	8
1.2 Основні вимоги до системи комп'ютерного зору	9
2 ПОРІВНЯННЯ ІСНУЮЧИХ РІШЕНЬ	11
2.1 Системи комп'ютерного зору на базі OpenCV	11
2.2 Нейромережеві системи визначення об'єктів	12
2.3 Обґрунтування вибраного методу	13
2.4 Висновки до розділу	15
3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	16
3.1 Мова програмування Python	16
3.2 Бібліотека OpenCV	17
3.3 Середовище Docker та операційна система Ubuntu	18
3.4 Платформа ROS2	19
3.5 Бібліотека NumPy	21
3.6 Висновки до розділу	22
4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ	23
4.1 Загальна архітектура системи	24
4.2 Реалізація модуля обробки зображення	27
4.3 Реалізація алгоритму сегментації об'єктів	29
4.4 Реалізація алгоритму оцінки можливості захоплення	34
4.5 Реалізація передачі даних між ROS-нодами	37
4.6 Визначення просторового положення та орієнтації об'єкта	38
4.7 Формування точки підходу до об'єкта	41
4.8 Висновки до розділу	43
5 РОБОТА СИСТЕМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ	44
5.1 Отримання зображення з камери	44
5.2 Визначення контурів та координат об'єктів	46
5.3 Результати визначення просторових параметрів об'єкта	47
5.4 Аналіз придатності об'єкта для захоплення	48
5.5 Формування фінальних результатів роботи системи	50
5.6 Висновки до розділу	52
ВИСНОВКИ	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	55
ДОДАТОК А Лістинг розробленої системи	57
ДОДАТОК Б Презент	76

ВСТУП

Комп'ютерний зір є одним із основних напрямків розвитку сучасних робототехнічних систем. Системи комп'ютерного зору використовуються для автоматичного аналізу сцени, визначення положення об'єктів та взаємодії роботизованих механізмів із навколишнім середовищем [1].

У межах дипломної роботи було розроблено систему комп'ютерного зору для визначення положення прямокутних об'єктів, а також оцінки можливості захоплення їх роботизованим маніпулятором. Система реалізована з використанням мови програмування Python, бібліотеки OpenCV та ROS2.

Основними задачами системи є:

- отримання зображення з реальної камери;
- сегментація об'єктів;
- визначення контурів та координат;
- визначення кута повороту;
- аналіз придатності для захоплення різними методами;
- передача результатів через ROS2 topics.

У роботі використовується суміш HSV-сегментації, алгоритму Canny Edge Detection та Watershed Segmentation для виділення об'єктів на зображенні.

Практична частина роботи включає розробку ROS2-вузлів, інтеграцію із реальною камерою, передачу даних між компонентами системи та візуалізацію результатів роботи алгоритмів.

1 ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ СИСТЕМИ КОМП'ЮТЕРНОГО ЗОРУ ДЛЯ РОБОТИЗОВАНОГО МАНІПУЛЯТОРА

1.1 Аналіз предметної області та постановка задачі

У сучасних автоматизованих виробничих системах роботизовані маніпулятори широко використовуються для виконання операцій переміщення, сортування та захоплення різних об'єктів. Одним із основних завдань подібних систем є можливість визначення положення предметів у робочій зоні перед виконанням механічної взаємодії. Для цього у більшості сучасних робототехнічних комплексів використовуються системи комп'ютерного зору.

Система комп'ютерного зору дозволяє отримувати інформацію про навколишнє середовище за допомогою камери та виконувати подальший аналіз отриманого зображення програмними методами. На основі аналізу зображення система може визначати контури об'єктів, їх координати, форму, орієнтацію та інші параметри, необхідні для роботи роботизованого маніпулятора.

У межах даної роботи розглядається задача визначення положення прямокутних об'єктів, які знаходяться у контейнері у випадковому положенні. Об'єкти можуть частково перекривати один одного, мати різний кут повороту відносно камери та знаходитися на різній відстані від меж контейнера. У зв'язку з цим виникає необхідність створення системи, здатної автоматично аналізувати зображення та визначати об'єкти, доступні для захоплення.

Особливістю задачі є необхідність не лише знаходження об'єкта на зображенні, але й визначення можливості його коректного захоплення роботизованим механізмом. У випадку, якщо об'єкт перекритий іншими предметами або його поверхня недостатньо відкрита, захоплення може бути неможливим або призвести до помилки роботи маніпулятора. Саме тому система

повинна виконувати додатковий аналіз видимості об'єкта та оцінювати придатність його поверхні для захоплення.

Загальна схема роботи системи комп'ютерного зору для вирішення поставленої задачі наведена на рисунку 1.1.

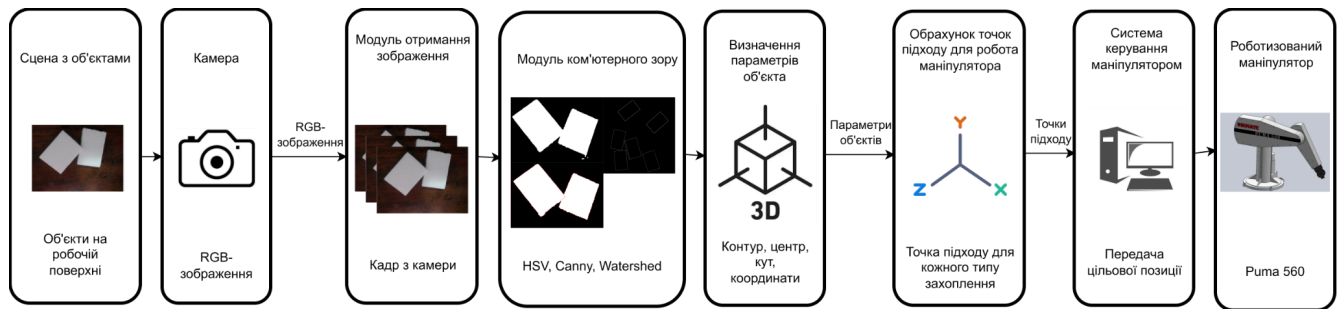


Рисунок 1.1 – Загальна схема роботи системи

Як показано на рисунку 1.1, робота системи починається з отримання зображення від камери. Далі виконується попередня обробка кадру та сегментація об'єктів для їх відокремлення від фону. Після цього здійснюється аналіз контурів і визначення геометричних параметрів виявлених об'єктів, зокрема їх координат та орієнтації. На основі отриманих даних система оцінює можливість захоплення кожного об'єкта, обирає найбільш придатний варіант та передає результати до системи керування роботизованим маніпулятором для виконання операції захоплення.

1.2 Основні вимоги до системи комп'ютерного зору

Для вирішення поставленої задачі система повинна виконувати:

- отримання зображення з камери;
- сегментацію об'єктів;
- пошук контурів;
- визначення координат та кута повороту;
- оцінку можливості захоплення;
- передачу результатів через ROS2 topics.

Після цього система повинна виконувати сегментацію об'єктів. Сегментація використовується для виділення окремих об'єктів на фоні зображення та формування їх контурів. Для цього можуть використовуватись методи бінаризації, морфологічної обробки, аналізу контурів та алгоритми розділення дотичних об'єктів.

Наступним етапом є визначення геометричних параметрів знайдених об'єктів. Система повинна визначати координати центру об'єкта, його ширину, висоту та кут повороту. Отримані параметри використовуються для подальшої передачі даних роботизованому маніпулятору.

Окремою задачею є визначення можливості захоплення об'єкта. Для цього необхідно виконати аналіз площі видимої поверхні, перевірити ступінь перекриття об'єкта іншими елементами та оцінити можливість розміщення механізму захоплення на поверхні предмета. У роботі розглядається можливість використання вакуумного захоплення та механічного захоплення за допомогою захвату.

У результаті виконання роботи повинна бути створена програмна система, здатна автоматично аналізувати зображення з камери, визначати положення об'єктів та формувати дані, необхідні для роботи роботизованого маніпулятора. Система повинна забезпечувати достатню точність визначення координат та стабільність роботи при різному положенні об'єктів у контейнері.

Основними задачами дипломної роботи є:

- аналіз існуючих методів комп'ютерного зору;
- реалізація сегментації об'єктів;
- визначення координат та кута повороту;
- оцінка можливості захоплення;
- реалізація передачі даних через ROS2;
- тестування роботи системи.

У результаті дипломної роботи було реалізовано систему комп'ютерного зору для визначення положення об'єктів та оцінки можливості їх захоплення роботизованим механізмом.

2 ПОРІВНЯННЯ ІСНУЮЧИХ РІШЕНЬ

2.1 Системи комп'ютерного зору на базі OpenCV

Для реалізації задач комп'ютерного зору широко використовується бібліотека OpenCV [1], [2], яка є одним із найбільш поширених програмних засобів для обробки цифрових зображень та відеопотоків. Бібліотека містить значну кількість готових алгоритмів для попередньої обробки зображень, сегментації об'єктів, пошуку контурів, визначення геометричних параметрів та оцінки просторового положення об'єктів.

Системи на базі OpenCV активно використовуються у робототехніці, автоматизованих виробничих комплексах та системах технічного зору, де необхідно забезпечити швидку обробку зображення у режимі реального часу. Важливою перевагою такого підходу є можливість отримання необхідної інформації про об'єкт без використання складних моделей машинного навчання. У більшості випадків аналіз виконується за допомогою класичних алгоритмів комп'ютерного зору, що дозволяє зменшити вимоги до обчислювальних ресурсів.

Для визначення та аналізу об'єктів OpenCV надає можливість використовувати різні методи обробки зображень, серед яких:

- сегментація зображення у кольоровому просторі HSV;
- пошук контурів об'єктів;
- виділення границь за допомогою детекторів ребер;
- морфологічна обробка зображення;
- аналіз геометричних характеристик об'єкта.

Застосування таких алгоритмів дозволяє визначати координати об'єкта на зображенні, його площу, форму, орієнтацію та інші параметри, необхідні для подальшого керування роботизованою системою. Особливо ефективним цей підхід є у випадках, коли об'єкти мають просту геометричну форму, а умови освітлення та фон залишаються відносно стабільними.

До переваг систем комп'ютерного зору на базі OpenCV можна віднести високу швидкість роботи, відсутність необхідності у навчанні моделей та можливість запуску навіть на малопотужних обчислювальних пристроях. Крім того, подібні системи відносно легко налаштовуються та інтегруються з робототехнічними платформами.

Основним недоліком таких рішень є залежність якості роботи від умов освітлення, кольору фону та правильності вибору параметрів сегментації. При значній зміні зовнішніх умов може знадобитися додаткове налаштування алгоритмів або корекція параметрів обробки зображення.

2.2 Нейромережеві системи визначення об'єктів

Окремим напрямком розвитку систем комп'ютерного зору є використання нейромережевих алгоритмів визначення об'єктів. Найбільш відомими представниками даного класу систем є YOLO, Mask R-CNN та інші сучасні моделі object detection. На відміну від класичних алгоритмів комп'ютерного зору, нейромережеві підходи дозволяють автоматично знаходити характерні ознаки об'єктів на основі попереднього навчання.

Подібні системи демонструють високу точність розпізнавання навіть у складних умовах експлуатації. Вони здатні працювати при нерівномірному освітленні, на складному фоні, а також у випадках часткового перекриття об'єктів. Крім того, одна модель може одночасно визначати велику кількість різних класів об'єктів без необхідності ручного налаштування правил сегментації.

Нейромережеві системи особливо ефективні у задачах, де сцена містить значну кількість різномірних об'єктів або коли неможливо забезпечити стабільні умови зйомки. Саме тому такі підходи широко застосовуються в автономних транспортних засобах, системах відеоспостереження та промислових роботизованих комплексах.

Разом із перевагами використання нейронних мереж має і певні недоліки. Насамперед необхідно сформувати набір навчальних даних, виконати процес

навчання моделі та регулярно проводити її перенавчання у випадку зміни умов експлуатації. Крім того, сучасні нейромережеві алгоритми потребують значних обчислювальних ресурсів та часто використовують графічні процесори для забезпечення необхідної швидкодії.

Також інтеграція та налагодження нейромережевих рішень зазвичай є більш складними порівняно з класичними алгоритмами комп'ютерного зору. Це збільшує час розробки та супроводу системи.

У межах даної роботи використання нейромережевих методів не є доцільним, оскільки система призначена для роботи з прямокутними об'єктами простої форми в контрольованих умовах. Для вирішення поставленої задачі застосування складних моделей машинного навчання не забезпечує суттєвих переваг порівняно з класичними алгоритмами обробки зображень.

2.3 Обґрунтування вибраного методу

Після аналізу існуючих підходів для реалізації системи було обрано класичний rule-based підхід на базі бібліотеки OpenCV та програмної платформи ROS2. Даний підхід дозволяє реалізувати всі необхідні функції визначення об'єктів та їх просторової орієнтації без використання складних моделей машинного навчання.

Основною задачею системи є визначення положення прямокутних об'єктів у робочій зоні маніпулятора, оцінка можливості їх захоплення та передача результатів до інших вузлів системи через механізм топіків ROS2. Для виконання цих функцій достатньо використання класичних алгоритмів комп'ютерного зору, які забезпечують необхідну швидкість та точність.

У розробленій системі використовується поєднання декількох алгоритмів обробки зображень:

- сегментація зображення у кольоровому просторі HSV;
- алгоритм виділення границь Canny Edge Detection [8];
- алгоритм вододільної сегментації Watershed Segmentation [9];
- пошук та аналіз контурів об'єктів;

– визначення геометричних характеристик та орієнтації об’єктів.

HSV-сегментація використовується для первинного виділення об’єктів на фоні. Алгоритм Canny дозволяє визначити границі об’єктів та покращити якість подальшого аналізу. Для розділення об’єктів, які дотикаються один до одного, застосовується алгоритм Watershed. Після цього виконується пошук контурів та розрахунок геометричних характеристик, необхідних для оцінки можливості захоплення.

Для прийняття рішення щодо способу захоплення використовується rule-based підхід, при якому оцінка виконується на основі форми об’єкта, його розмірів, орієнтації та інших геометричних параметрів. Такий підхід забезпечує прозорість логіки роботи системи та спрощує її налагодження.

До переваг обраного підходу належать можливість роботи у режимі реального часу, низькі вимоги до апаратних ресурсів, відсутність необхідності підготовки навчальних вибірок та простота інтеграції з ROS2. Крім того, алгоритми можуть бути легко модифіковані відповідно до особливостей конкретного виробничого середовища.

Недоліком обраного підходу є залежність результатів роботи від якості сегментації та умов освітлення. Однак для поставленої задачі ці обмеження не є критичними, оскільки система працює у контрольованому середовищі з відносно стабільними умовами отримання зображень.

Порівняння класичного та нейромережевого підходів наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння класичного та нейромережевого підходів

Критерії	Підхід на основі класичних алгоритмів	Нейромережевий підхід
Основний інструмент	OpenCV	YOLO
Швидкодія	Висока	Середня
Необхідність навчання	Відсутня	Необхідне
Потреба в датасеті	Відсутня	Необхідна
Робота в реальному часі	Так	Частково

Як видно з таблиці 2.1, неймережеві підходи забезпечують високу універсальність та точність розпізнавання, проте потребують значних обчислювальних ресурсів і попереднього навчання. Для задачі визначення просторової орієнтації прямокутних об'єктів у контрольованих умовах більш доцільним є використання rule-based підходу на базі OpenCV, який забезпечує необхідну швидкодію та простоту реалізації.

2.4 Висновки до розділу

У даному розділі було проведено аналіз існуючих підходів до реалізації систем комп'ютерного зору для задач визначення об'єктів та оцінки можливості їх захоплення роботизованими системами. Розглянуто класичні алгоритми комп'ютерного зору на базі бібліотеки OpenCV та сучасні неймережеві методи визначення об'єктів.

Проведений аналіз показав, що неймережеві підходи забезпечують високу точність розпізнавання та стійкість до змін умов зйомки, проте потребують навчання моделей, підготовки датасетів та значних обчислювальних ресурсів. Для задачі визначення просторового положення прямокутних об'єктів у контрольованому середовищі використання таких підходів є надлишковим.

На основі проведеного порівняння було обрано rule-based підхід із використанням бібліотеки OpenCV та платформи ROS2. Даний підхід забезпечує необхідну швидкодію, не потребує попереднього навчання та дозволяє реалізувати визначення геометричних параметрів, просторової орієнтації та можливості захоплення об'єктів у режимі реального часу.

Отримані результати аналізу стали основою для вибору програмних засобів та алгоритмів, використаних під час розробки системи комп'ютерного зору.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Мова програмування Python

Для реалізації програмної частини системи використовується мова програмування Python [4].

Python використовується для:

- отримання зображення з камери;
- обробки кадрів;
- сегментації об'єктів;
- пошуку контурів;
- визначення параметрів об'єктів;
- взаємодії між ROS2-нодами.

У роботі використовуються бібліотеки:

- OpenCV;
- NumPy;
- ROS2 Python API.

У межах роботи Python використовувався як основна мова реалізації алгоритмів комп'ютерного зору та взаємодії з ROS2. Вибір даної мови обумовлений наявністю великої кількості бібліотек для обробки зображень та робототехнічних застосувань, а також підтримкою ROS2 через пакет `roslru`.

Однією з причин вибору Python є простота інтеграції із бібліотеками комп'ютерного зору та платформою ROS2. Мова також дозволяє швидко тестувати алгоритми та змінювати параметри системи під час розробки.

У межах роботи окремі компоненти системи реалізовані у вигляді модулів, які відповідають за:

- обробку зображення;
- сегментацію;
- аналіз придатності захоплення;
- передачу результатів між ROS2-нодами.

Використання Python у межах даної роботи дозволяє реалізувати систему комп'ютерного зору та забезпечити інтеграцію із ROS2 і бібліотекою OpenCV.

3.2 Бібліотека OpenCV

Для реалізації алгоритмів обробки зображення у роботі використовується бібліотека OpenCV [1], [2], [14].

OpenCV використовується для:

- отримання кадрів із камери;
- перетворення кольорових моделей;
- HSV-сегментації;
- морфологічної обробки;
- пошуку контурів;
- визначення параметрів об'єктів.

На першому етапі система отримує зображення у форматі BGR та виконує перетворення у кольоровий простір HSV. Це дозволяє більш стабільно виділяти поверхні об'єктів при зміні освітлення.

Після формування бінарної маски система виконує морфологічну обробку для видалення шумів та покращення форми контурів.

Для пошуку меж об'єктів використовується алгоритм Canny Edge Detection [8]. Після цього система виконує пошук контурів та визначає:

- координати центру;
- площу об'єкта;
- ширину та висоту;
- кут повороту.

Для визначення орієнтації об'єкта використовується метод мінімального прямокутника.

У випадках часткового перекриття предметів використовується алгоритм вододільної сегментації (Watershed Segmentation) [9], який дозволяє розділяти дотичні об'єкти.

OpenCV також використовується для роботи із відеопотоком у режимі реального часу та формування debug image і best object image.

Використання OpenCV у межах роботи дозволяє реалізувати систему комп'ютерного зору без використання складних нейромережових алгоритмів.

3.3 Середовище Docker та операційна система Ubuntu

Для запуску робототехнічної системи у роботі використовується операційна система Ubuntu [6] та середовище контейнеризації Docker [5].

Ubuntu використовується як основна операційна система для запуску ROS2, OpenCV та допоміжних інструментів розробки. Система забезпечує підтримку роботи із камерами та компонентами робототехнічної платформи.

Для спрощення налаштування програмного середовища використовується Docker. Контейнер містить:

- Ubuntu;
- ROS2;
- OpenCV;
- Python та необхідні залежності.

Використання Docker дозволяє запускати систему у стабільному середовищі без необхідності повторного налаштування бібліотек на різних комп'ютерах.

У межах роботи Docker також використовується для інтеграції системи комп'ютерного зору із камерою та графічним інтерфейсом операційної системи.

Контейнеризація дозволяє забезпечити відтворюваність середовища розробки та уникнути проблем, пов'язаних із відмінностями версій бібліотек на різних комп'ютерах.

Використання Ubuntu та Docker дозволяє забезпечити стабільну роботу ROS2 та компонентів системи комп'ютерного зору у межах поставленої задачі.

3.4 Платформа ROS2

Для організації взаємодії між компонентами системи у роботі використовується платформа ROS2 [3].

ROS2 дозволяє будувати систему у вигляді окремих програмних вузлів — нод, які взаємодіють між собою через систему повідомлень та топіки [16].

У межах роботи ROS2 використовується для:

- отримання зображення з камери;
- передачі кадрів між модулями системи;
- передачі результатів обробки;
- взаємодії між компонентами системи комп’ютерного зору.

У системі використовуються такі основні ROS-ноди:

- camera_node;
- image_detection_node;
- best_object_node.

Нода camera_node відповідає за отримання відеопотоку з камери та передачу кадрів у topic /camera/image_raw [13].

Нода image_detection_node виконує сегментацію об’єктів, пошук контурів та визначення параметрів знайдених об’єктів.

Нода best_object_node виконує вибір найбільш придатного об’єкта для захоплення та формує результати для подальшої передачі іншим компонентам системи.

Структура взаємодії ROS-нод представлена на рис. 3.1.

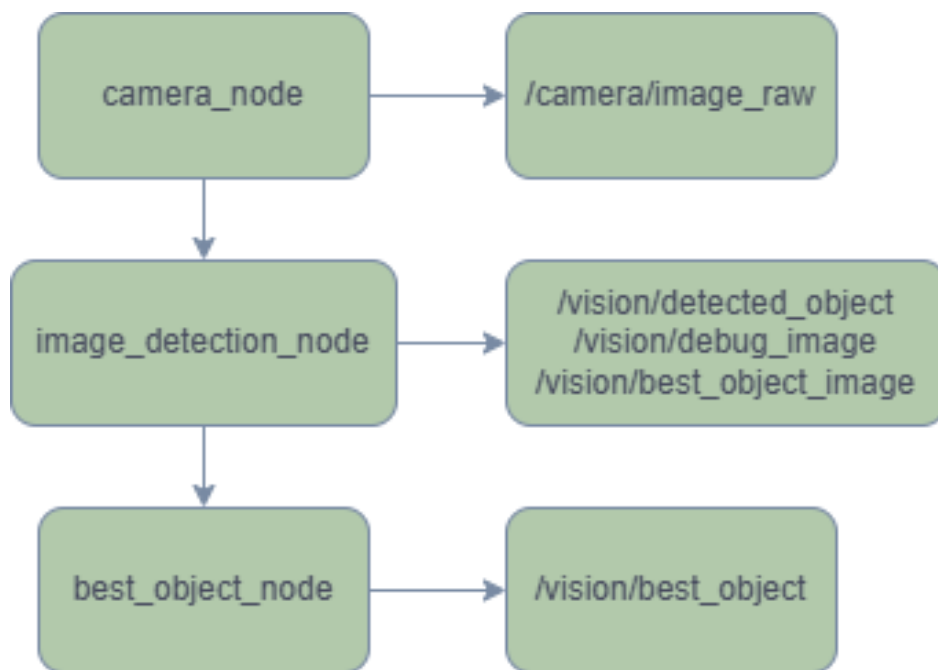


Рисунок 3.1 – Структура взаємодії ROS2-вузлів системи комп'ютерного зору

Для передачі структурованих даних між вузлами використовуються власні типи повідомлень (custom messages) [19], які містять інформацію про знайдені об'єкти, результати їх аналізу та параметри, необхідні для подальшої взаємодії з роботизованим маніпулятором.

У розробленій системі реалізовано повідомлення DetectedObject, DetectedObjects та BestObject, які використовуються для передачі результатів аналізу сцени між ROS2-нодами.

У межах роботи через топіки ROS2 передаються:

- зображення з камери;
- інформація про знайдені об'єкти;
- просторові координати та орієнтація об'єктів;
- результати вибору найбільш придатного об'єкта;
- debug image та best object image.

Для реалізації ROS2-нод використовується мова програмування Python. Запуск системи виконується у середовищі Docker на базі операційної системи Ubuntu.

Використання ROS2 дозволяє організувати передачу даних між компонентами системи та спрощує інтеграцію алгоритмів комп'ютерного зору із роботизованою платформою.

3.5 Бібліотека NumPy

Для виконання математичних обчислень та роботи з багатовимірними масивами у роботі використовується бібліотека NumPy [14].

NumPy є однією з основних бібліотек екосистеми Python для наукових та інженерних обчислень. Бібліотека забезпечує ефективну роботу з масивами даних, матрицями та векторами, а також містить засоби для виконання математичних операцій над ними.

У межах розробленої системи NumPy використовується для:

- зберігання та обробки зображень, отриманих із камери;
- формування матриці камери для алгоритму SolvePnP;
- роботи з координатами контурів та вершин об'єктів;
- виконання операцій над векторами та матрицями;
- підготовки даних для алгоритмів OpenCV.

Під час реалізації алгоритмів комп'ютерного зору зображення представляються у вигляді багатовимірних масивів NumPy, що дозволяє ефективно виконувати їх обробку та передавати між компонентами системи.

Бібліотека також використовується для формування структур даних, необхідних для визначення просторового положення та орієнтації об'єктів. Зокрема, за допомогою NumPy створюються масиви координат характерних точок, матриця внутрішніх параметрів камери та допоміжні матриці, які використовуються алгоритмом SolvePnP.

Використання NumPy дозволяє спростити реалізацію математичних обчислень та забезпечує ефективну взаємодію між власними модулями системи та бібліотекою OpenCV.

3.6 Висновки до розділу

У даному розділі було розглянуто основні програмні засоби та технології, використані під час розробки системи комп'ютерного зору. Як основну мову програмування обрано Python, який забезпечує зручну інтеграцію з бібліотеками комп'ютерного зору та платформою ROS2.

Для реалізації алгоритмів обробки зображень використовується бібліотека OpenCV, яка надає необхідні засоби для сегментації об'єктів, пошуку контурів та визначення їх геометричних характеристик. Для виконання математичних обчислень та роботи з матрицями застосовується бібліотека NumPy.

Організація взаємодії між окремими компонентами системи реалізована за допомогою платформи ROS2, яка забезпечує обмін даними між вузлами через механізм топіків та повідомлень. Для забезпечення стабільності програмного середовища та спрощення розгортання системи використовується контейнеризація Docker у середовищі операційної системи Ubuntu.

Обраний набір програмних засобів забезпечує реалізацію всіх необхідних функцій системи комп'ютерного зору та створює основу для подальшої розробки й інтеграції алгоритмів із роботизованим маніпулятором.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ

У межах дипломної роботи була розроблена програмна система комп'ютерного зору для визначення положення об'єктів та передачі отриманих даних роботизованій системі. Програмна реалізація побудована за модульним принципом із використанням платформи ROS 2, що дозволяє розділити систему на окремі функціональні компоненти.

Програмна система реалізована мовою Python із використанням бібліотеки OpenCV для обробки зображень. Для запуску системи використовується середовище Docker та операційна система Ubuntu.

Система розроблена відповідно до принципів модульної архітектури, що дозволяє розділити функції отримання даних, обробки зображення та прийняття рішень між окремими програмними компонентами. Використання ROS2 забезпечує слабе зв'язування між модулями та дозволяє організувати обмін даними через механізм топіків. Такий підхід спрощує масштабування системи, оскільки нові компоненти можуть бути додані без суттєвої зміни існуючої структури програмного забезпечення.

Контейнеризація за допомогою Docker дозволяє забезпечити однакове програмне середовище незалежно від апаратної платформи. Усі необхідні бібліотеки, пакети ROS2 та програмні залежності розміщуються всередині контейнера, що значно спрощує процес розгортання та тестування системи. Використання Ubuntu як базової операційної системи обумовлене її широкою підтримкою ROS2 та великою кількістю доступних засобів для робототехнічних застосувань.

Система побудована у вигляді набору ROS-нод, які взаємодіють між собою через механізм повідомлень [15]. Такий підхід дозволяє окремо запускати та тестувати компоненти програми, а також спрощує подальше розширення функціоналу.

4.1 Загальна архітектура системи

Архітектура програмної системи побудована за принципом послідовної обробки даних. Кожен програмний модуль виконує окрему задачу та передає результати наступному компоненту системи.

Основними компонентами системи є:

- модуль отримання зображення з камери;
- модуль обробки зображення;
- модуль визначення найкращого об'єкта для захоплення;
- модуль передачі результатів через ROS2.

Подібна структура дозволяє спростити процес налагодження системи та зробити програму більш масштабованою.

Кожен модуль виконує чітко визначений набір функцій та може розроблятися незалежно від інших компонентів системи. Такий підхід дозволяє спростити локалізацію помилок, скоротити час тестування та підвищити надійність програмного забезпечення. Крім того, модульна архітектура дозволяє замінювати окремі алгоритми без необхідності повної переробки всієї системи.

Структура взаємодії ROS-нод

У системі реалізовано три основні ROS-ноди: `camera_node`, `image_detection_node` та `best_object_node`. Нода `camera_node` відповідає за отримання зображення з реальної камери та публікує кадри у топик `/camera/image_raw`. Нода `image_detection_node` підписується на цей топик, отримує зображення та виконує його обробку за допомогою OpenCV.

Після обробки зображення `image_detection_node` публікує декілька результатів: список знайдених об'єктів у топик `/vision/detected_objects`, відлагоджувальне зображення у топик `/vision/debug_image` та зображення з виділеним найкращим об'єктом у топик `/vision/best_object_image`.

Нода `best_object_node` отримує інформацію про знайдені об'єкти, виконує rule-based оцінку та вибирає найбільш придатний об'єкт для захоплення. Результат

публікується у топик `/vision/best_object`.

Для передачі даних між вузлами використовуються механізми `publish-subscribe`, реалізовані в ROS2. На відміну від прямої взаємодії між програмними модулями, використання топиків дозволяє зменшити залежність між компонентами системи. Кожна нода може бути запущена окремо та обмінюватися повідомленнями незалежно від внутрішньої реалізації інших вузлів.

У розробленій системі для передачі інформації про знайдені об'єкти використовуються власні типи повідомлень (`custom messages`), які містять координати об'єктів, геометричні характеристики, кути орієнтації та інформацію про можливість захоплення. Це дозволяє передавати між вузлами структуровані дані без необхідності додаткового перетворення форматів.

На рисунку 4.1 наведено структуру взаємодії основних ROS-нод розробленої системи.

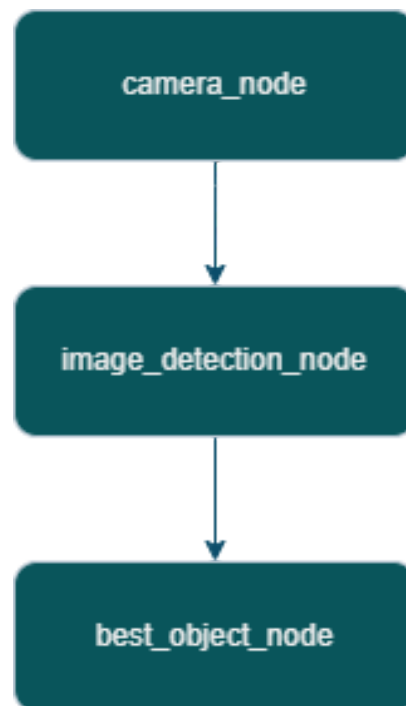


Рисунок 4.1 – Структура взаємодії нод

Представлена структура дозволяє розділити систему на незалежні програмні компоненти та забезпечує можливість окремого тестування кожного модуля. Завдяки використанню ROS2 можлива одночасна робота декількох вузлів на різних

обчислювальних пристроях, що є важливою перевагою при подальшій інтеграції системи технічного зору з роботизованим маніпулятором.

Схема обробки зображення

Обробка зображення у системі виконується послідовно у декілька етапів. На першому етапі система отримує кадр із камери та передає його до модуля комп'ютерного зору.

Після отримання кадру система виконує попередню обробку зображення, сегментацію об'єктів та пошук контурів.

Попередня обробка включає згладжування зображення для зменшення впливу шумів, корекцію контрасту та виділення області інтересу. Дані операції дозволяють підвищити якість сегментації та зменшити кількість помилкових спрацьовувань під час подальшого аналізу.

Після формування бінарної маски виконується аналіз контурів знайдених об'єктів. Для кожного контуру розраховуються геометричні характеристики, включаючи площу, координати центра мас, ширину, висоту та кут повороту. Отримані параметри використовуються для подальшого визначення просторової орієнтації об'єктів та оцінки можливості їх захоплення.

Після цього визначаються координати, площа та кут повороту знайдених об'єктів.

На наступному етапі система визначає найбільш придатний об'єкт для захоплення. Під час аналізу враховується положення об'єкта, ступінь перекриття та доступність поверхні для механізму захоплення.

Загальна послідовність роботи алгоритму представлена на рисунку 4.2.



Рисунок 4.2 – Схема обробки зображення у системі

Після завершення обробки результати передаються через ROS2 у вигляді повідомлень для подальшого використання роботизованим маніпулятором.

4.2 Реалізація модуля обробки зображення

Модуль обробки зображення є центральним компонентом розробленої системи комп'ютерного зору. Його основною задачею є отримання зображення від камери, запуск алгоритмів аналізу сцени, формування структурованої інформації про знайдені об'єкти та передача результатів іншим компонентам робототехнічної системи через механізми ROS2.

Модуль реалізований у вигляді окремої ROS-ноди `image_detection_node`, яка отримує кадри з камери через топік `/camera/image_raw`, виконує їх обробку та публікує результати у відповідні ROS2-топіки.

Після отримання нового кадру нода `image_detection_node` запускає послідовність програмних компонентів, кожен з яких відповідає за виконання окремої задачі. Такий підхід дозволяє розділити логіку роботи системи на незалежні функціональні модулі та спрощує подальшу підтримку програмного забезпечення.

У процесі розробки було використано модульний принцип побудови програмного коду. Основні етапи обробки винесені до окремих класів, що забезпечує зручність тестування, налагодження та подальшого розширення функціональних можливостей системи.

Основними програмними компонентами модуля є:

- `ImageProcessor` — виконує попередню обробку зображення;
- `SegmentationProcessor` — відповідає за сегментацію об'єктів та пошук контурів;
- `PoseEstimator` — визначає просторове положення та орієнтацію об'єктів;
- `PickingEvaluator` — оцінює можливість захоплення об'єктів;
- `BestObjectSelector` — виконує вибір найбільш придатного об'єкта для захоплення;

- PickPoseCalculator — розраховує параметри підходу маніпулятора до об'єкта;
- DetectionResultManager — формує результати та підготовлює їх до передачі через ROS2.

Структура взаємодії основних компонентів модуля обробки зображення наведена на рисунку 4.3.

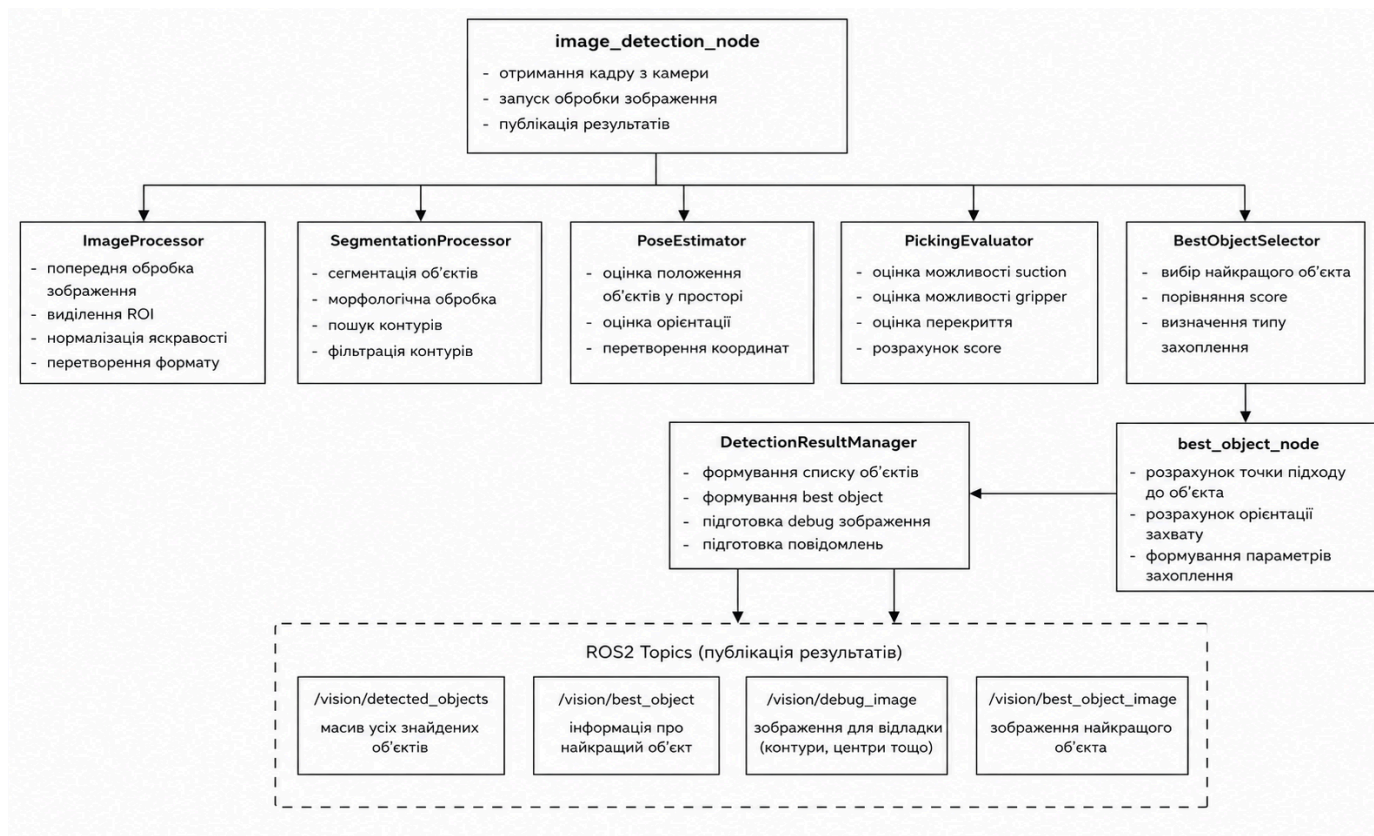


Рисунок 4.3 – Структура програмних компонентів модуля обробки зображення

Після отримання кадру зображення передається до класу ImageProcessor, який виконує початкову підготовку даних до подальшого аналізу. Після цього керування передається модулю SegmentationProcessor, який формує набір знайдених об'єктів та їх контурів.

На наступному етапі для кожного знайденого об'єкта виконується визначення його характеристик. Просторове положення та орієнтація об'єкта визначаються модулем PoseEstimator. Після цього модуль PickingEvaluator виконує оцінку можливості використання різних способів захоплення.

Отримані результати передаються до модуля BestObjectSelector, який аналізує всі знайдені об'єкти та вибирає найбільш придатний для подальшої взаємодії з роботизованим маніпулятором. Після вибору об'єкта модуль PickPoseCalculator виконує розрахунок координат точки підходу до об'єкта залежно від обраного способу захоплення.

Після завершення всіх етапів аналізу модуль DetectionResultManager формує структури даних, які використовуються для передачі результатів через ROS2. Система публікує інформацію про всі знайдені об'єкти, відлагоджувальне зображення та інформацію про об'єкт, який був обраний як найбільш придатний для захоплення.

Для спрощення процесу налагодження у системі реалізовано механізм візуалізації проміжних результатів роботи. Під час виконання алгоритму можуть відображатися проміжні та фінальні результати обробки, що дозволяє контролювати правильність роботи окремих програмних компонентів.

Використання модульної архітектури дозволило спростити процес розробки програмного забезпечення, забезпечити незалежність окремих компонентів та підвищити зручність подальшого розширення функціональних можливостей системи.

4.3 Реалізація алгоритму сегментації об'єктів

Для виділення об'єктів на зображенні використовується багатокроковий алгоритм сегментації, який включає перетворення кольорового простору, формування бінарної маски, морфологічну обробку та аналіз контурів.

Схема алгоритму сегментації об'єктів представлена на рис. 4.4

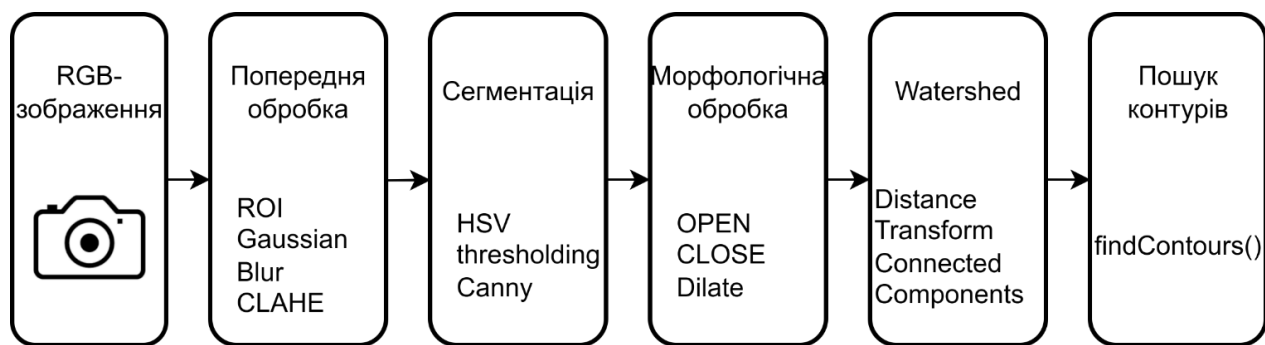


Рисунок 4.4 – Схема алгоритму сегментації об'єктів

Одним із ключових етапів роботи розробленої системи комп'ютерного зору є сегментація об'єктів. Метою даного етапу є виділення окремих предметів на зображенні та формування областей, які надалі використовуються для визначення геометричних характеристик і подальшого аналізу.

Алгоритм сегментації реалізовано із використанням бібліотеки OpenCV та базується на послідовному застосуванні методів попередньої обробки зображення, кольорової сегментації, морфологічної обробки та аналізу контурів.

Після отримання кадру від камери система виконує його перетворення з кольорового простору BGR у простір HSV. Використання моделі HSV дозволяє розділити інформацію про відтінок кольору, насиченість та яскравість, що спрощує виділення об'єктів за кольоровими ознаками та зменшує вплив змін освітлення на результати сегментації.

Для формування маски використовуються попередньо визначені діапазони значень HSV. Пікселі, що потрапляють до заданого діапазону, вважаються частиною об'єкта та включаються до бінарної маски. Усі інші області класифікуються як фон.

На рисунку 4.5 наведено приклад результату сегментації з використанням кольорового простору HSV.



Рисунок 4.5 – Результат HSV-сегментації об'єктів

Отримана маска може містити шумові області, дрібні дефекти та розриви контурів, які негативно впливають на подальший аналіз. Для усунення таких недоліків використовується морфологічна обробка зображення.

У системі застосовуються операції відкриття (Opening) та закриття (Closing). Операція відкриття використовується для видалення дрібних шумових областей, які не належать до об'єктів. Операція закриття дозволяє усунути невеликі розриви всередині сегментованих областей та покращити цілісність контурів.

Результат морфологічної обробки наведено на рисунку 4.6.

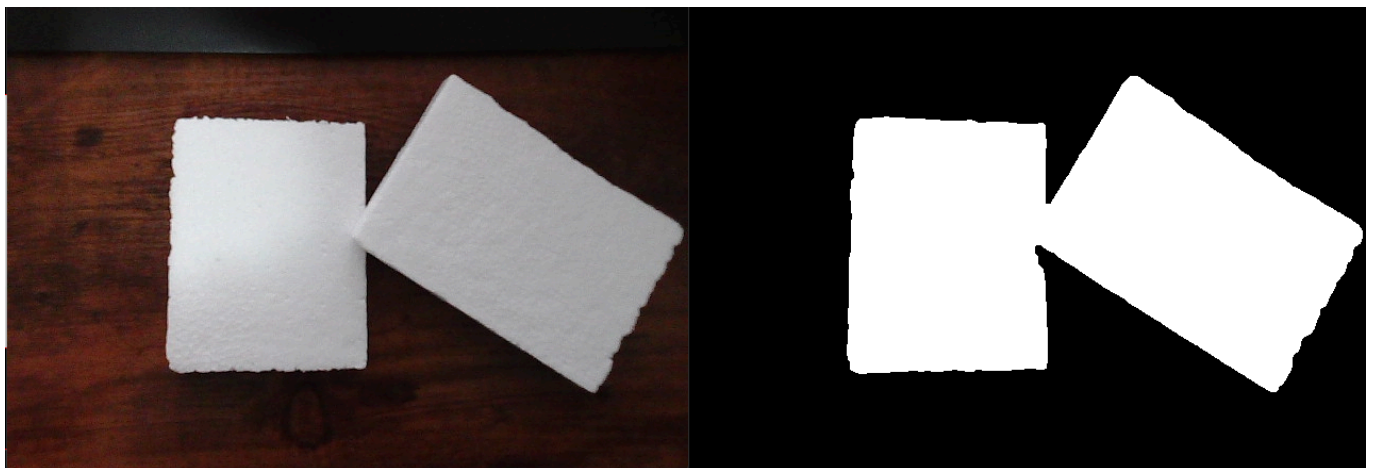


Рисунок 4.6 – Результат морфологічної обробки бінарної маски

Після формування якісної бінарної маски система виконує пошук меж об'єктів. Для цього використовується алгоритм Canny Edge Detection, який дозволяє визначити різкі зміни яскравості та сформувати карту ребер.

Використання алгоритму Canny дозволяє більш точно локалізувати межі об'єктів та підвищити якість подальшого пошуку контурів.

На рисунку 4.7 наведено результат виділення меж об'єктів за допомогою алгоритму Canny.

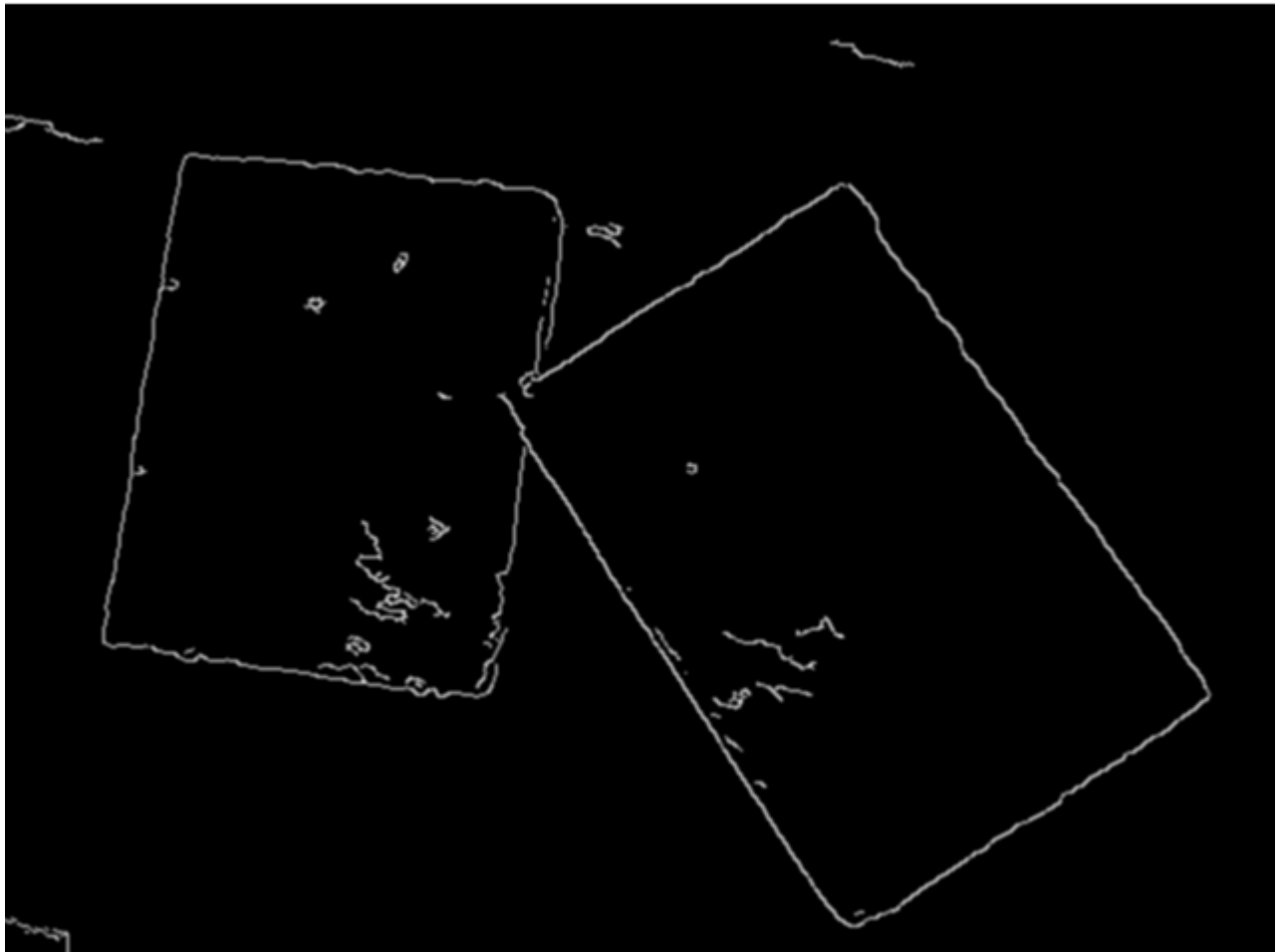


Рисунок 4.7 – Результат роботи алгоритму Canny Edge Detection

Однією з особливостей предметної області є можливість розташування об'єктів впритул один до одного або їх часткового перекриття. У подібних ситуаціях стандартний пошук контурів може помилково об'єднувати декілька предметів в один сегмент.

Для розв’язання даної задачі у системі використовується алгоритм вододільної сегментації Watershed. Перед його запуском виконується побудова карти відстаней (Distance Transform), яка дозволяє визначити найбільш імовірні центри окремих об’єктів.

На основі карти відстаней формуються маркери, які використовуються алгоритмом Watershed для розділення суміжних областей на окремі сегменти. Такий підхід дозволяє підвищити точність визначення кількості об’єктів у сцені та забезпечити коректне виділення їх контурів.

Результат роботи алгоритму Watershed наведено на рисунку 4.8.

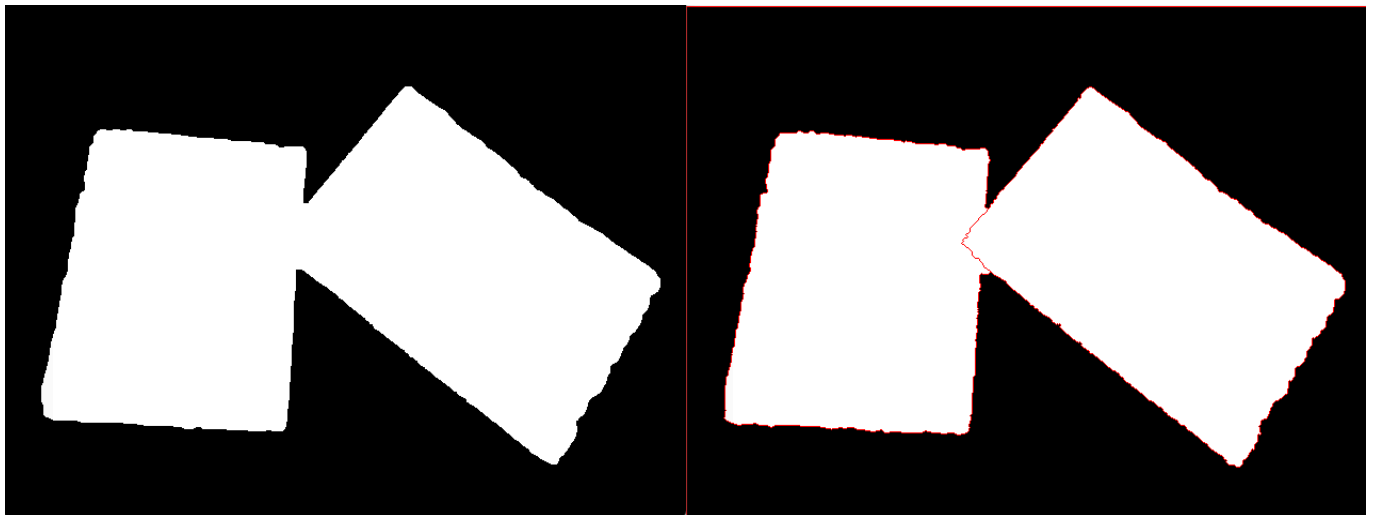


Рисунок 4.8 – Розділення дотичних об’єктів методом Watershed

Після завершення сегментації система виконує пошук контурів за допомогою функції `findContours` бібліотеки OpenCV [18]. Для кожного знайденого контуру виконується первинна фільтрація.

Фільтрація дозволяє відкинути шумові області та об’єкти, які не відповідають вимогам задачі. Під час перевірки враховуються площа контуру, співвідношення сторін, коефіцієнт прямокутності та коефіцієнт суцільності. Використання таких критеріїв дозволяє виключити випадкові області та залишити лише контури, які відповідають формі необхідних об’єктів.

На рисунку 4.9 наведено приклад результату пошуку та фільтрації контурів.

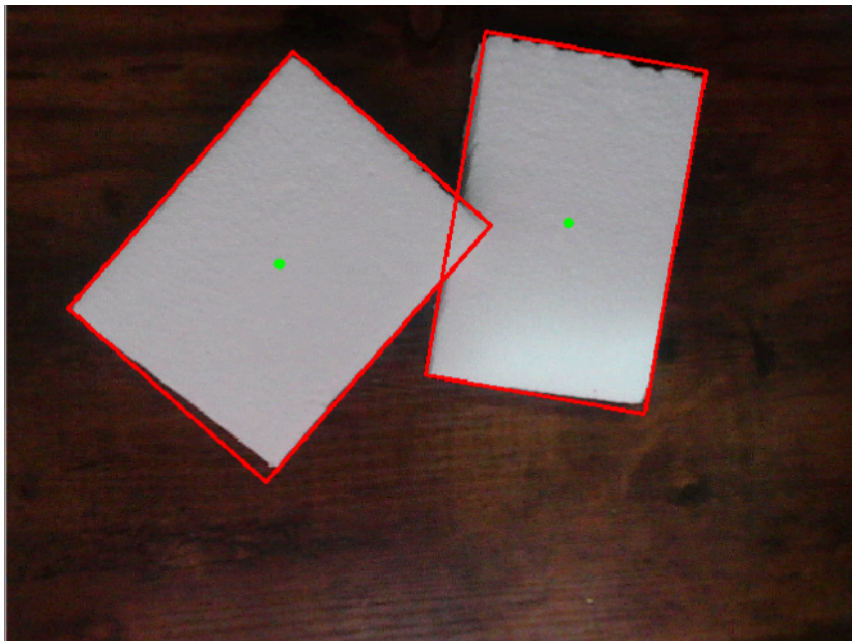


Рисунок 4.9 – Результат пошуку та фільтрації контурів

Після завершення всіх етапів сегментації система формує набір контурів, які передаються до наступних модулів для визначення геометричних параметрів, просторового положення та оцінки можливості захоплення об'єктів.

4.4 Реалізація алгоритму оцінки можливості захоплення

Після завершення сегментації та виділення контурів система виконує оцінку можливості захоплення знайдених об'єктів. Основною задачею даного етапу є визначення того, наскільки об'єкт придатний для подальшої взаємодії з роботизованим маніпулятором та який спосіб захоплення може бути використаний.

У межах роботи реалізовано rule-based алгоритм оцінки можливості захоплення. На відміну від підходів, що базуються на машинному навчанні, прийняття рішення виконується на основі набору заздалегідь визначених правил та геометричних характеристик об'єкта. Такий підхід дозволяє забезпечити прозорість роботи алгоритму та спрощує його налаштування.

На вхід алгоритму надходить інформація про знайдені об'єкти, сформована на попередніх етапах обробки зображення. Для кожного об'єкта аналізуються його

геометричні характеристики, форма контуру, ступінь перекриття та доступність поверхні для захоплення.

Першим етапом роботи алгоритму є перевірка геометричних параметрів об'єкта. Під час аналізу враховуються площа контуру, співвідношення сторін, коефіцієнт прямокутності та коефіцієнт суцільності. Об'єкти, параметри яких суттєво відрізняються від очікуваних значень, виключаються з подальшого аналізу.

Після перевірки геометричних характеристик виконується оцінка перекриття об'єкта іншими предметами. Для цього аналізуються контактні зони між контурами та ступінь деформації форми об'єкта. У випадку значного перекриття об'єкт отримує ознаку *occluded*, яка використовується під час подальшого розрахунку його рейтингу.

Наступним етапом є оцінка можливості використання вакуумного захоплення. Для такого типу захоплення важливою умовою є наявність достатньої відкритої верхньої поверхні, яка дозволяє надійно розмістити присоску. Якщо центральна область об'єкта залишається доступною та не перекривається сусідніми предметами, система визначає можливість використання вакуумного захоплення.

Окрім вакуумного захоплення виконується оцінка можливості використання механічного захвату. У цьому випадку аналізується доступність бокових поверхонь об'єкта та наявність достатнього вільного простору для розміщення губок захвату. Якщо бокові частини об'єкта перекриваються іншими предметами або відстань до сусідніх об'єктів є недостатньою, використання механічного захвату забороняється.

Після завершення всіх перевірок для кожного об'єкта формується набір характеристик, який включає інформацію про ступінь перекриття, доступні способи захоплення та числовий рейтинг придатності до захоплення. Рейтинг використовується для порівняння знайдених об'єктів між собою та вибору найбільш придатного кандидата для подальшої роботи маніпулятора.

Загальна послідовність роботи алгоритму оцінки можливості захоплення наведена на рисунку 4.10.



Рисунок 4.10 – Алгоритм оцінки можливості захоплення об'єкта

Після завершення аналізу результати передаються до модуля BestObjectSelector, який виконує порівняння всіх доступних об'єктів та визначає найбільш придатний варіант для подальшого захоплення роботизованим маніпулятором.

Використання rule-based підходу дозволило реалізувати алгоритм оцінки можливості захоплення без необхідності використання нейронних мереж або попереднього навчання моделей. При цьому система забезпечує достатню якість вибору об'єктів для задачі, що розглядається у межах даної роботи.

4.5 Реалізація передачі даних між ROS-нодами

Передача даних між окремими компонентами системи реалізована за допомогою механізмів платформи ROS2 [20]. Такий підхід дозволяє розділити програму на незалежні вузли, кожен з яких виконує власну задачу та обмінюється результатами з іншими частинами системи.

У межах розробленої системи основними ROS-нодами є:

- camera_node — отримання зображення з камери;
- image_processing_node — обробка зображення та визначення параметрів об'єктів;
- best_object_node — вибір найбільш придатного об'єкта для захоплення;
- нода керування або подальшої обробки — отримання фінальних даних для маніпулятора.

Передача зображення від камери виконується через окремий топик. Нода camera_node публікує кадри з камери, а нода обробки зображення підписується на цей топик та отримує дані для аналізу. Після отримання кадру виконується його перетворення у формат, придатний для роботи з бібліотекою OpenCV.

Після завершення обробки зображення нода image_processing_node формує повідомлення з результатами аналізу. У повідомленні передаються основні параметри знайдених об'єктів: координати центру, розміри, кут повороту, площа та ознаки придатності до захоплення.

Далі ці дані передаються до ноди best_object_node. Її задачею є аналіз усіх знайдених об'єктів та вибір найбільш придатного для захоплення. Під час вибору враховується доступність поверхні, відсутність перекриття, положення об'єкта та можливість використання відповідного механізму захоплення.

Після вибору об'єкта система формує фінальне повідомлення, яке може бути використане іншими компонентами робототехнічної системи рис. 4.11.

```
frame_width: 640
frame_height: 480
objects_count: 1
objects:
- id: 0
  center_x: 298.0
  center_y: 282.0
  x_mm: -11.279501914978027
  y_mm: -29.119417190551758
  z_mm: 388.3067321777344
  angle: 176.2359619140625
  raw_angle: 86.2359619140625
  alpha: 3.0957064628601074
  beta: -0.03976409509778023
  gamma: -3.6678528785705566
  width_px: 129.24693298339844
  height_px: 186.5572509765625
  area: 21780.0
  rectangularity: 0.9032864570617676
  solidity: 0.9271636009216309
  suction_pickable: true
  gripper_pickable: true
  occluded: false
---
```

Рисунок 4.11 – Передача даних між ROS-нодами

Таке повідомлення містить координати об'єкта, його орієнтацію та додаткову інформацію про тип захоплення.

У повідомленнях передаються координати об'єкта, кут повороту, геометричні параметри та інформація про можливість захоплення.

4.6 Визначення просторового положення та орієнтації об'єкта

Для подальшого використання результатів роботи системи комп'ютерного зору роботизованим маніпулятором недостатньо визначити лише положення об'єкта на площині зображення. Для формування траєкторії захоплення необхідно отримати просторові координати об'єкта відносно камери та оцінити його орієнтацію у тривимірному просторі.

У розробленій системі визначення просторового положення та орієнтації об'єктів реалізовано за допомогою алгоритму SolvePnP (Perspective-n-Point) [17], який входить до складу бібліотеки OpenCV [11], [12]. Даний алгоритм дозволяє

визначити положення та орієнтацію об'єкта відносно камери на основі відповідності між точками об'єкта у просторі та їх проєкціями на зображенні.

Для роботи алгоритму необхідно сформувати матрицю камери, яка містить параметри внутрішньої калібровки. У межах даної роботи використовується наближена модель камери, побудована на основі роздільної здатності зображення та відомих кутів огляду камери.

Фокусні відстані по горизонтальній та вертикальній осях визначаються за формулами:

$$f_x = \frac{R_x}{2 \tan(\frac{FOV_x}{2})} \quad f_y = \frac{R_y}{2 \tan(\frac{FOV_y}{2})}, \quad (4.1)$$

де R_x – ширина зображення у пікселях;

R_y – висота зображення у пікселях;

FOV_x – горизонтальний кут огляду камери;

FOV_y – вертикальний кут огляду камери.

f_x, f_y – фокусні відстані у пікселях.

Координати головної точки камери визначаються як центр зображення:

$$c_x = \frac{R_x}{2}, \quad c_y = \frac{R_y}{2}, \quad (4.2)$$

де c_x та c_y – координати оптичного центра камери.

На основі отриманих параметрів формується матриця камери:

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (4.3)$$

Матриця камери використовується алгоритмом SolvePnP для визначення просторового положення об'єкта.

Після завершення сегментації система визначає чотири характерні точки об'єкта. Для цього використовується орієнтований прямокутник, побудований навколо контуру об'єкта. Координати його вершин на зображенні використовуються як набір двовимірних точок.

Одночасно задаються відповідні тривимірні координати вершин верхньої поверхні об'єкта. Для цього використовується відома ширина та висота об'єкта у реальних одиницях вимірювання.

Після формування наборів двовимірних та тривимірних точок виконується алгоритм SolvePnP, результатом роботи якого є вектор положення ($tvec$) та вектор обертання ($rvec$).

У результаті виконання алгоритму SolvePnP система отримує два набори параметрів: вектор положення та вектор обертання об'єкта відносно камери.

Вектор положення містить просторові координати об'єкта по осях X , Y та Z і дозволяє визначити його розташування у системі координат камери. Отримані координати використовуються під час формування параметрів захоплення та подальшої взаємодії з роботизованим маніпулятором.

Окрім координат положення, алгоритм також визначає орієнтацію об'єкта у просторі. Для цього вектор обертання перетворюється у матрицю обертання, на основі якої обчислюються кути Ейлера. У розробленій системі визначаються три кути орієнтації: кут повороту навколо осі X (α), кут повороту навколо осі Y (β) та кут повороту навколо осі Z (γ).

Отримані координати та кути орієнтації додаються до структури даних об'єкта та передаються через ROS2 для подальшого використання іншими компонентами системи. Це дозволяє не лише визначати положення об'єкта на зображенні, але й оцінювати його просторову орієнтацію, що є важливим для планування траєкторії захоплення.

Перевагою використання алгоритму SolvePnP є можливість визначення просторового положення та орієнтації об'єкта за допомогою однієї RGB-камери без використання додаткових датчиків глибини або систем стереозору. При цьому точність отриманих результатів залежить від якості сегментації об'єкта, правильності визначення його характерних точок та точності параметрів моделі камери.

Використання алгоритму SolvePnP дозволило реалізувати визначення просторових координат та орієнтації об'єктів, необхідних для подальшої інтеграції системи комп'ютерного зору з роботизованим маніпулятором.

4.7 Формування точки підходу до об'єкта

Після визначення просторового положення та орієнтації об'єкта система повинна сформувати точку підходу, яка може бути використана роботизованим маніпулятором під час виконання операції захоплення. Основною задачею даного етапу є визначення безпечного положення інструмента перед безпосереднім контактом із об'єктом.

У розробленій системі реалізовано окремий модуль PickPoseCalculator, який виконує розрахунок точки підходу залежно від вибраного способу захоплення. Параметри підходу формуються після завершення аналізу об'єкта та вибору методу захоплення.

Для вакуумного захоплення використовується підхід уздовж нормалі до поверхні об'єкта. Такий спосіб дозволяє забезпечити правильне позиціонування присоски відносно поверхні та мінімізувати ризик зіткнення з об'єктом під час руху маніпулятора.

Координати точки підходу визначаються за формулою:

$$P_{approach} = P_{object} + d \cdot \overline{n}, \quad (4.4)$$

де $P_{approach}$ – координати точки підходу;

P_{object} – координати центра об'єкта;

n – одиничний вектор нормалі до поверхні;

d – відстань зміщення від поверхні об'єкта.

У розробленій системі нормаль до поверхні визначається на основі орієнтації об'єкта, отриманої за допомогою алгоритму SolvePnP. Для побудови точки підходу використовується зміщення вздовж нормалі на фіксовану відстань, що забезпечує безпечне наближення присоски до поверхні.

Для механічного захоплення використовується спрощений підхід. Оскільки захват виконується зверху, точка підходу формується шляхом зміщення координати по осі Z на задану висоту відносно центра об'єкта.

Координати точки підходу визначаються за формулою:

$$P_{approach} = (x, y, z + h), \quad (4.5)$$

де x, y, z – координати центра об'єкта;

h – висота підходу над об'єктом.

Такий підхід дозволяє розмістити захват над об'єктом перед початком операції захоплення та забезпечити додатковий запас безпеки під час руху маніпулятора.

Після завершення розрахунку координати точки підходу додаються до повідомлення про вибраний об'єкт та передаються через ROS2 для подальшого використання системою керування роботизованим маніпулятором.

Використання окремого етапу формування точки підходу дозволяє відокремити процес визначення параметрів об'єкта від процесу планування захоплення та спрощує подальшу інтеграцію системи комп'ютерного зору з роботизованим маніпулятором.

4.8 Висновки до розділу

У даному розділі було розглянуто програмну реалізацію системи комп'ютерного зору для визначення положення та орієнтації об'єктів у робочій зоні роботизованого маніпулятора. Розроблена система побудована на основі модульної архітектури із використанням платформи ROS2, що забезпечує розподіл функціональності між окремими програмними компонентами та спрощує подальше розширення системи.

Було реалізовано модуль обробки зображення, який виконує сегментацію об'єктів, пошук контурів, визначення геометричних характеристик та оцінку можливості захоплення. Для розділення дотичних об'єктів використано алгоритм Watershed, а для визначення просторового положення та орієнтації об'єктів — алгоритм SolvePnP.

У роботі реалізовано механізм передачі даних між ROS2-нодами із використанням власних типів повідомлень, що дозволяє передавати структуровану інформацію про знайдені об'єкти та результати їх аналізу. Також розроблено алгоритм формування точки підходу до об'єкта для різних способів захоплення.

У результаті було створено програмну систему, яка забезпечує визначення координат, просторової орієнтації та параметрів захоплення об'єктів і може бути використана як складова частина роботизованого комплексу.

5 РОБОТА СИСТЕМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Після реалізації основних компонентів системи було виконано тестування роботи алгоритмів на зображеннях із реальної камери. Основною задачею тестування була перевірка стабільності роботи системи, якості сегментації об'єктів, правильності визначення координат та оцінки можливості захоплення.

Під час роботи система отримує відеопотік із камери, виконує аналіз кожного кадру та передає результати через ROS2 топіки. Обробка кадрів виконується у режимі реального часу.

У процесі тестування перевірялись:

- стабільність отримання зображення;
- якість сегментації;
- правильність пошуку контурів;
- визначення кута повороту;
- робота алгоритму вибору об'єкта;
- передача повідомлень між ROS-нодами;
- формування debug image та best object image.

Для тестування використовувались прямокутні об'єкти, розміщені у контейнері у випадковому положенні. У деяких випадках об'єкти частково перекривали один одного, що дозволяло перевірити роботу алгоритму сегментації та оцінки можливості захоплення.

5.1 Отримання зображення з камери

Отримання зображення виконується за допомогою ROS-ноди camera_node. Дана нода відповідає за підключення до реальної камери та передачу кадрів через топік /camera/image_raw.

У межах роботи використовувалась USB-камера, підключена до системи Ubuntu. Для запуску системи камера передавалась у Docker-контейнер, після чого ставала доступною для ROS2 та OpenCV.

Для перевірки роботи камери використовувались стандартні інструменти ROS2 та OpenCV. Також виконувалось відображення відеопотоку у debug-вікні системи.

У процесі роботи було виявлено, що якість освітлення суттєво впливає на результат сегментації. При недостатньому освітленні збільшувалась кількість шумів та погіршувалась якість виділення поверхонь об'єктів. Для покращення стабільності роботи використовувалось більш рівномірне освітлення сцени.

Приклад отриманого зображення з камери під час роботи системи наведено на рисунку 5.1.



Рисунок 5.1 – Отримання зображення з камери у системі

Під час тестування не було зафіксовано втрати кадрів або критичних затримок передачі даних між ROS-нодами. Отриманий відеопотік використовувався для подальшого аналізу сцени та тестування алгоритмів комп'ютерного зору.

5.2 Визначення контурів та координат об'єктів

Для перевірки роботи алгоритму було проведено серію тестувань із різним розташуванням об'єктів у робочій зоні. Під час випробувань аналізувались окремі об'єкти, об'єкти з частковим перекриттям, а також предмети, розташовані під різними кутами відносно камери.

У більшості тестових сцен система коректно виконувала пошук контурів та визначення геометричних параметрів об'єктів. Для кожного знайденого предмета обчислювались координати центра, розміри та інші характеристики, необхідні для подальшого аналізу.

Під час тестування було перевірено роботу алгоритму на різному розташуванні об'єктів:

- окремі об'єкти;
- часткове перекриття;
- дотичні об'єкти;
- різний кут повороту.

Для налагодження системи використовувалось debug-зображення із відображенням знайдених контурів та центрів об'єктів.

На рисунку 5.2 наведено приклад роботи системи під час обробки сцени з двома об'єктами.

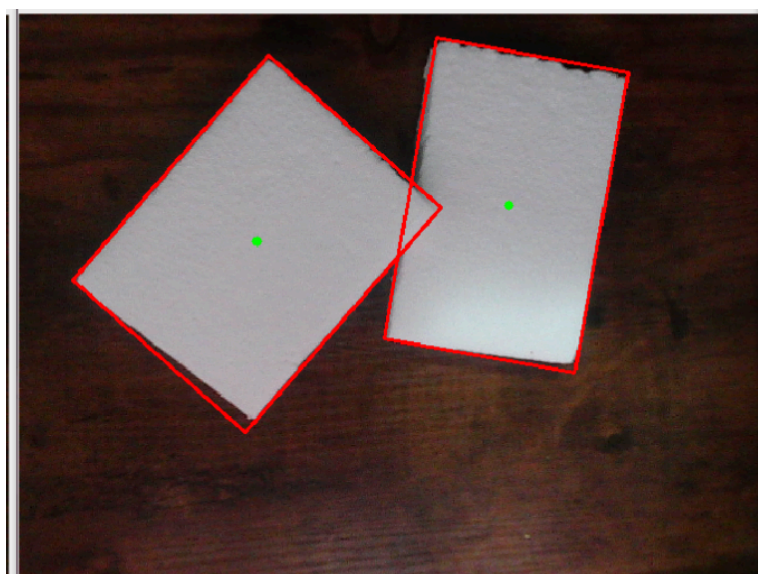


Рисунок 5.2 – Визначення контурів об'єктів

Червоним кольором відображені контури, визначені алгоритмом сегментації, а зеленими точками позначені центри знайдених об'єктів.

Під час тестування найбільші труднощі виникали у випадках значного перекриття об'єктів або нерівномірного освітлення сцени. У таких ситуаціях можливе незначне спотворення контурів, що впливає на точність визначення геометричних параметрів.

Результати тестування показали, що система коректно визначає межі об'єктів навіть у випадках їх часткового перекриття. Для кожного об'єкта формується набір геометричних параметрів, який використовується на наступних етапах аналізу.

5.3 Результати визначення просторових параметрів об'єкта

Після визначення контуру об'єкта система формує набір параметрів, які характеризують його положення, геометричні характеристики та просторову орієнтацію. Отримані дані використовуються для подальшого аналізу та формування параметрів захоплення.

Після визначення кута інформація додається до структури даних знайденого об'єкта та передається через топіки ROS2.

Під час тестування перевірялась коректність визначення координат об'єкта, кута його повороту та параметрів просторової орієнтації. Для перевірки коректності роботи алгоритмів визначення просторового положення об'єктів було проведено тестування формування та передачі повідомлень через ROS2 topic `/vision/detected_objects`. У сформованих повідомленнях міститься повна інформація про виявлені об'єкти, включаючи координати, кути орієнтації, геометричні характеристики та результати аналізу можливості захоплення.

Приклад такого повідомлення наведено на рисунку 5.3.

```
Every 0.5s: ros2 topic echo /vision/d... 5af0351b17b
frame_width: 640
frame_height: 480
objects_count: 1
objects:
- id: 0
  center_x: 285.0
  center_y: 200.0
  angle: 83.24642944335938
  raw_angle: 83.24642944335938
  width: 273.7973937988281
  height: 183.5072479248047
  area: 47711.5
  rectangularity: 0.9495996236801147
  solidity: 0.9657905101776123
  suction_pickable: true
  gripper_pickable: true
  occluded: false
--
```

Рисунок 5.3 – Приклад параметрів об'єкта, отриманих після аналізу сцени

На рисунку 5.3 наведено приклад повідомлення, сформованого після аналізу об'єкта. Повідомлення містить координати положення об'єкта відносно камери, кути його орієнтації, геометричні характеристики та результати оцінки можливості захоплення.

З отриманих даних видно, що система успішно визначає координати об'єкта по осях X, Y та Z, а також обчислює кути орієнтації α , β та γ . Додатково передаються параметри форми об'єкта, зокрема площа, розміри та коефіцієнти прямокутності й суцільності.

Під час тестування система стабільно визначала просторові параметри об'єктів та формувала повідомлення для подальшої передачі через ROS2. Найбільший вплив на точність результатів мали якість сегментації та правильність визначення меж об'єкта. У випадках значного перекриття предметів можливі незначні відхилення координат і кутів орієнтації.

5.4 Аналіз придатності об'єкта для захоплення

Після визначення контурів та геометричних параметрів система виконує аналіз придатності об'єкта для захоплення. Основною задачею даного етапу є визначення

того, чи може роботизований механізм коректно взаємодіяти із знайденим предметом.

Для перевірки роботи алгоритму було проведено тестування на сценах із різним взаємним розташуванням об'єктів. Особлива увага приділялась випадкам часткового перекриття предметів та їх розташуванню під різними кутами відносно камери.

У процесі тестування система виконувала оцінку всіх знайдених об'єктів та формувала рейтинг їх придатності до захоплення. На основі розрахованих характеристик виконувався вибір найбільш придатного кандидата для подальшої роботи маніпулятора.

Результатом роботи алгоритму є вибір об'єкта, який має найвищий рейтинг придатності для захоплення. Для візуального контролю правильності роботи системи формується спеціальне зображення, на якому відображається об'єкт, обраний як найкращий кандидат для взаємодії з роботизованим маніпулятором.

Приклад результату роботи алгоритму наведено на рисунку 5.4.

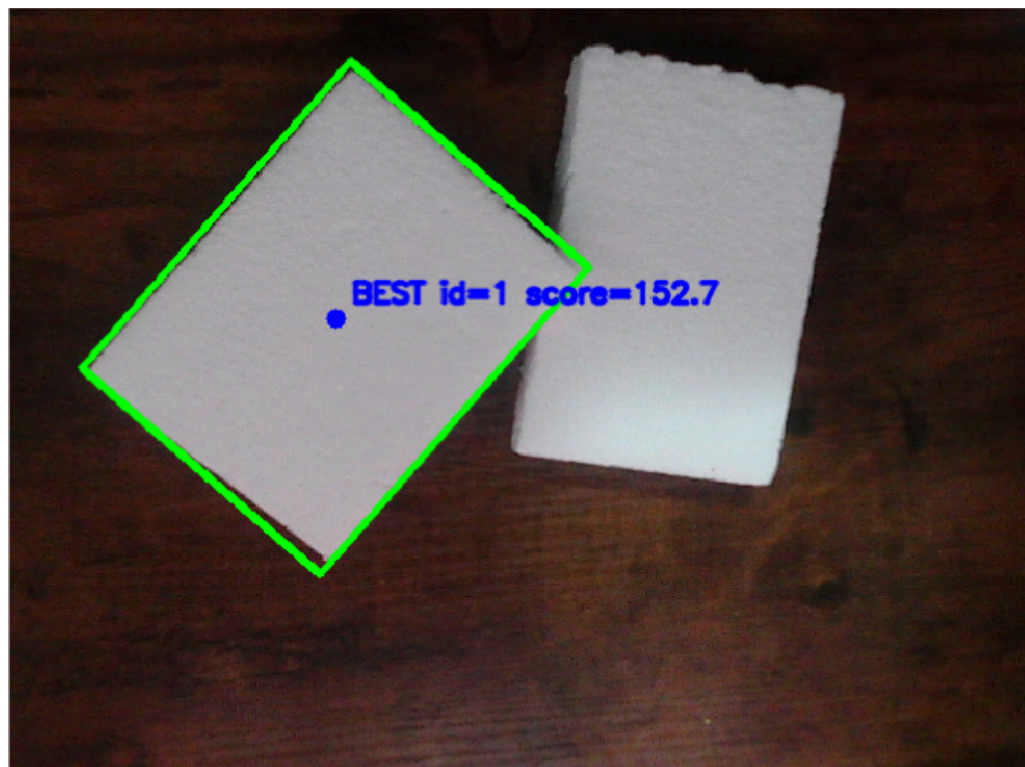


Рисунок 5.4 – Вибір найкращого об'єкта

Під час тестування система показала стабільну роботу у випадках:

- окремих об'єктів;
- часткового перекриття;
- різного кута повороту;
- різного розташування предметів у контейнері.

Найбільші труднощі виникали у випадках значного перекриття об'єктів або погіршення якості сегментації. У таких ситуаціях можливе зниження точності оцінки придатності об'єкта та його рейтингу. Проте у більшості тестових сцен система успішно виконувала вибір найбільш придатного об'єкта для захоплення.

Для налагодження алгоритму використовувались `debug image` та `best object image`, які дозволяли візуально перевіряти правильність вибору об'єкта та роботу алгоритму оцінки.

На зображенні відображається об'єкт, який система визначила як найбільш придатний для захоплення. Також можуть відображатись координати центру та додаткова інформація про тип доступного захоплення.

5.5 Формування фінальних результатів роботи системи

Для перевірки роботи алгоритмів та зручності налагодження у системі реалізовано візуалізацію результатів обробки зображення. Після завершення аналізу кадру система формує зображення із виділеним об'єктом, який був вибраний як найбільш придатний для захоплення.

На фінальному зображенні система відображає:

- контур вибраного об'єкта;
- координати центру;
- рейтинг об'єкта

Після формування результату зображення публікується через ROS2 topic `/vision/best_object_image`. Це дозволяє передавати результат між ROS-нодами та використовувати його для подальшого аналізу або візуального контролю роботи системи.

Окрім зображення, система також публікує параметри вибраного об'єкта через топіки ROS2. До основних параметрів належать:

- координати центру;
- ширина та висота;
- площа об'єкта;
- кут повороту;
- тип доступного захоплення.

Під час тестування візуалізація використовувалась для перевірки правильності роботи алгоритмів сегментації та вибору об'єкта. Відображення результатів дозволяло швидко виявляти помилки обробки або некоректне визначення контурів.

На зображенні відображено результат роботи системи після завершення аналізу сцени. Система виділяє об'єкт, який був визначений як найбільш придатний для захоплення роботизованим механізмом.

Окрім візуального представлення результатів, система формує спеціалізоване повідомлення, яке містить усі параметри обраного об'єкта та може використовуватися іншими компонентами робототехнічної системи. Передача цієї інформації здійснюється через ROS2-топик `/vision/best_object`.

Приклад сформованого повідомлення наведено на рисунку 5.5.

```
Every 0.5s: ros2 topic echo /vision/b... dfad2f60511
success: true
reason: ok
id: 0
x_mm: -68.56034088134766
y_mm: 0.0
z_mm: 0.0
angle_deg: 176.9644317626953
alpha: 4.943204879760742
beta: 46.31297302246094
gamma: 0.687065064907074
width_mm: 108.09445190429688
height_mm: 136.42164611816406
pick_method: any
score: 101.23155212402344
---
```

Рисунок 5.5 – Передача параметрів вибраного об'єкта через ROS2-топик `/vision/best_object`

На рисунку 5.5 наведено приклад повідомлення BestObject, сформованого після завершення аналізу сцени. Повідомлення містить інформацію про координати об'єкта відносно камери, його просторову орієнтацію, геометричні параметри та результати оцінки придатності до захоплення.

Отримані дані використовуються для подальшої взаємодії з роботизованим маніпулятором. Зокрема, передаються координати положення об'єкта по осях X, Y та Z, кути орієнтації α , β та γ , а також інформація про вибраний спосіб захоплення та рейтинг об'єкта.

Під час тестування система стабільно формувала повідомлення BestObject та передавала його через ROS2-топик без втрати даних. Отримані результати підтвердили коректність роботи механізму обміну повідомленнями між ROS-нодами та можливість подальшої інтеграції системи комп'ютерного зору із системою керування маніпулятором.

5.6 Висновки до розділу

У даному розділі було проведено тестування розробленої системи комп'ютерного зору та виконано аналіз результатів її роботи. Перевірка системи здійснювалась на зображеннях, отриманих із реальної камери, що дозволило оцінити працездатність усіх основних програмних компонентів у реальних умовах експлуатації.

Результати тестування показали, що система коректно виконує отримання зображень, сегментацію об'єктів, пошук контурів та визначення їх геометричних характеристик. Також підтверджено можливість визначення просторового положення та орієнтації об'єктів за допомогою алгоритму SolvePnP.

Під час випробувань система успішно виконувала оцінку придатності об'єктів до захоплення та вибір найбільш придатного кандидата на основі реалізованого rule-based алгоритму. Передача даних між ROS2-нодами здійснювалась без втрати

інформації, а використання власних типів повідомлень забезпечило зручний обмін структурованими даними між компонентами системи.

Аналіз результатів показав, що найбільший вплив на якість роботи системи мають умови освітлення та ступінь перекриття об'єктів у сцені. Незважаючи на це, у більшості тестових випадків система забезпечувала коректне визначення параметрів об'єктів та формування даних, необхідних для подальшої взаємодії з роботизованим маніпулятором.

Отримані результати підтвердили працездатність розробленої системи та можливість її використання як складової частини робототехнічного комплексу для задач автоматизованого захоплення об'єктів.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено систему технічного зору для визначення просторового положення та орієнтації об'єктів з подальшим формуванням параметрів захоплення роботизованим маніпулятором.

У процесі виконання роботи проведено аналіз існуючих систем комп'ютерного зору та методів сегментації об'єктів, а також виконано порівняння класичних алгоритмів комп'ютерного зору та сучасних нейромережевих підходів. За результатами аналізу для реалізації системи було обрано rule-based підхід на базі бібліотеки OpenCV та платформи ROS2.

Розроблено програмну архітектуру системи, яка складається з окремих ROS2-нод, що взаємодіють між собою за допомогою топіків та власних типів повідомлень. Реалізовано алгоритми попередньої обробки зображень, сегментації об'єктів, пошуку контурів, визначення геометричних характеристик та оцінки можливості захоплення об'єктів.

Для визначення просторового положення та орієнтації об'єктів використано алгоритм SolvePnP, що дозволяє отримувати координати та кути орієнтації об'єкта відносно камери за допомогою однієї RGB-камери. Також реалізовано алгоритми формування точки підходу для вакуумного та механічного способів захоплення.

Проведене тестування підтвердило працездатність розробленої системи у режимі реального часу. Система коректно виконує отримання зображень з камери, сегментацію об'єктів, визначення їх просторових параметрів, оцінку придатності до захоплення та передачу результатів між ROS2-нодами. Отримані результати можуть бути використані для подальшої інтеграції з системою керування роботизованим маніпулятором.

Таким чином, поставлена мета роботи досягнута, а всі визначені завдання виконані у повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bradski G., Kaehler A. Learning OpenCV: Computer Vision with the OpenCV Library. Sebastopol : O'Reilly Media, 2008. 555 p.
2. OpenCV Documentation. URL: <https://docs.opencv.org/> (дата звернення: 22.04.2026).
3. ROS2 Documentation. URL: <https://docs.ros.org/> (дата звернення: 24.04.2026).
4. Python Documentation. URL: <https://docs.python.org/3/> (дата звернення: 20.04.2026).
5. Docker Documentation. URL: <https://docs.docker.com/> (дата звернення: 30.04.2026).
6. Ubuntu Documentation. URL: <https://ubuntu.com/> (дата звернення: 04.05.2026).
7. Szeliski R. Computer Vision: Algorithms and Applications. London : Springer, 2011. 812 p.
8. Canny J. A Computational Approach to Edge Detection // IEEE Transactions on Pattern Analysis and Machine Intelligence. 1986. Vol. 8, No 6. P. 679–698.
9. Vincent L., Soille P. Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations // IEEE Transactions on Pattern Analysis and Machine Intelligence. 1991. Vol. 13, No 6. P. 583–598.
10. NumPy Documentation. URL: <https://numpy.org/doc/> (дата звернення: 25.04.2026).
11. OpenCV Camera Calibration and 3D Reconstruction. URL: https://docs.opencv.org/4.x/d9/d0c/group_calib3d.html (дата звернення: 11.05.2026)..
12. OpenCV solvePnP Documentation. URL: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html (дата звернення: 11.05.2026).
13. sensor_msgs/msg/Image Message. URL: https://docs.ros.org/en/iron/p/sensor_msgs/interfaces/msg/Image.html (дата звернення: 11.05.2026).
14. Bradski G., Kaehler A. *Learning OpenCV 4 Computer Vision with Python*. Birmingham: Packt Publishing, 2019. 768 p.

15. Quigley M., Conley K., Gerkey B., Faust J., Foote T., Leibs J., Wheeler R., Ng A. ROS: an open-source Robot Operating System. ICRA Workshop on Open Source Software, 2009.
16. Macenski S., Foote T., Gerkey B., Lalancette C., Woodall W. Robot Operating System 2: Design, architecture, and uses in the wild. Science Robotics. 2022. Vol. 7, No. 66.
17. OpenCV. Camera Calibration and 3D Reconstruction. URL: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html (дата звернення: 15.01.2026).
18. OpenCV. Structural Analysis and Shape Descriptors. URL: https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html (дата звернення: 22.03.2026).
19. ROS2 Documentation. Creating custom ROS 2 msg and srv files. URL: <https://docs.ros.org/en/rolling/Tutorials/Beginner-Client-Libraries/Custom-ROS2-Interfaces.html> (дата звернення: 18.04.2026).
20. ROS2 Documentation. About ROS 2 interfaces. URL: <https://docs.ros.org/en/foxy/Concepts/About-ROS-Interfaces.html> (дата звернення: 18.04.2026).

ДОДАТОК А Лістинг розробленої системи

A.1 `image_detection_pipeline.py` — координує повний цикл обробки зображення: попередня обробка, сегментація, оцінка об'єктів, вибір найкращого об'єкта та формування результатів.

```
import cv2
import numpy as np

from robot_vision.processor import ImageProcessor
from robot_vision.picking import PickingEvaluator
from robot_vision.segmentation_processor import SegmentationProcessor
from robot_vision.result_manager import DetectionResultManager
from robot_vision.config import CONFIG
from robot_vision.pose_estimator import PoseEstimator

class ImageDetectionPipeline:
    def __init__(self, config=CONFIG, debug=False):
        self.config = config
        self.debug = debug

        self.picking_evaluator = PickingEvaluator(config)
        self.segmentator = SegmentationProcessor(config, debug=debug)

        self.pose_estimator = None

    def process_frame(self, picture):
        processor = ImageProcessor(picture, self.config)

        hsv = self._prepare_hsv_mask(processor)
        refined_edges = self._prepare_edges(processor)

        hsv_blocked = self._remove_edges_from_mask(
            processor,
            hsv,
            refined_edges
        )

        contours, markers_vis = self.segmentator.split_touching_objects(
            hsv_blocked,
            processor.image
```

```

)

final_contours = self._resplit_contours(
    contours,
    hsv,
    refined_edges
)

filtered_contours = self._filter_normal_objects(final_contours)

if self.pose_estimator is None:
    self.pose_estimator = PoseEstimator(
        image_width_px=picture.shape[1],
        image_height_px=picture.shape[0],
        horizontal_fov_deg=self.config["camera"]["horizontal_fov_deg"],
        vertical_fov_deg=self.config["camera"]["vertical_fov_deg"],
        approx_epsilon_ratio=self.config["pose"]["approx_epsilon_ratio"]
    )

result_manager = self._build_result(filtered_contours, hsv.shape)

debug_image = self._draw_debug_image(
    picture.copy(),
    filtered_contours,
)

return {
    "detections": result_manager,
    "detections_dict": result_manager.to_dict(),
    "debug_image": debug_image,
    "markers_vis": markers_vis
}

def _prepare_hsv_mask(self, processor):
    hsv_mask = processor.hsv_mask()
    hsv = processor.top_surface_mask(hsv_mask)
    hsv = processor.clean_mask(hsv)

    return hsv

def _prepare_edges(self, processor):
    edges = processor.edge_map()

```

```

refined_edges = processor.refine_edges(edges)

return refined_edges

def _remove_edges_from_mask(self, processor, hsv, refined_edges):
    edges_barrier = cv2.threshold(
        refined_edges,
        127,
        255,
        cv2.THRESH_BINARY
    )[1]

    hsv_cut = cv2.bitwise_and(
        hsv,
        cv2.bitwise_not(edges_barrier)
    )

    hsv_blocked = processor.clean_mask(hsv_cut)

    return hsv_blocked

def _resplit_contours(self, contours, hsv, refined_edges):
    final_contours = []

    for cnt in contours:
        obj = self._get_single_object_info(cnt)

        if obj is None:
            continue

        if self.segmentator.needs_resplit(obj):
            split_cnts = self.segmentator.try_resplit_contour_with_edges(
                cnt,
                hsv,
                refined_edges,
                min_area=self.config["object_filter"]["min_area_px"] // 2
            )

            if len(split_cnts) >= 2:
                if self.debug:
                    print(
                        f"Contour {obj.id} was re-split "

```

```

        f"into {len(split_cnts)} parts"
    )

    final_contours.extend(split_cnts)
else:
    final_contours.append(cnt)
else:
    final_contours.append(cnt)

return final_contours

def _filter_normal_objects(self, contours):
    filtered_contours = []

    for cnt in contours:
        ok_shape, shape_debug = self.picking_evaluator.check_top_face_quality(cnt)

        if not ok_shape:
            if self.debug:
                print("Rejected contour:", shape_debug)
            continue

        filtered_contours.append(cnt)

    return filtered_contours

def _build_result(self, contours, mask_shape):
    result_manager = DetectionResultManager()

    objects = self.picking_evaluator.get_object_info(contours)

    for obj, cnt in zip(objects, contours):
        single_mask = self._contour_to_mask(cnt, mask_shape)

        suction_ok, suction_debug = self.picking_evaluator.can_pick_suction(
            obj,
            single_mask
        )

        gripper_ok, gripper_debug = self.picking_evaluator.can_pick_gripper(

```

```

    obj,
    single_mask
)

obj.suction_pickable = suction_ok
obj.gripper_pickable = gripper_ok

obj.reason = {
    "suction": suction_debug,
    "gripper": gripper_debug
}

pose = self.pose_estimator.estimate_pose_from_contour(
    cnt,
    real_width_mm=self.config["pose"]["object_real_width_mm"],
    real_height_mm=self.config["pose"]["object_real_height_mm"]
)

if pose is not None:
    obj.alpha = pose["alpha"]
    obj.beta = pose["beta"]
    obj.gamma = pose["gamma"]

    obj.x_mm = pose["x_mm"]
    obj.y_mm = pose["y_mm"]
    obj.z_mm = pose["z_mm"]
else:
    obj.alpha = 0.0
    obj.beta = 0.0
    obj.gamma = obj.angle

    obj.x_mm = 0.0
    obj.y_mm = 0.0
    obj.z_mm = 0.0

if suction_ok or gripper_ok:
    result_manager.add_object(obj)

return result_manager

```

```

def _contour_to_mask(self, contour, shape):
    mask = np.zeros(shape[:2], dtype=np.uint8)
    cv2.drawContours(mask, [contour], -1, 255, -1)
    return mask

def _draw_debug_image(self, image, contours):
    output = image.copy()

    for cnt in contours:
        M = cv2.moments(cnt)

        if M["m00"] == 0:
            continue

        cx = int(M["m10"] / M["m00"])
        cy = int(M["m01"] / M["m00"])

        rect = cv2.minAreaRect(cnt)
        box = cv2.boxPoints(rect)
        box = np.int32(box)

        cv2.drawContours(output, [box], 0, (0, 0, 255), 2)
        cv2.circle(output, (cx, cy), 4, (0, 255, 0), -1)

        cv2.putText(
            output,
            f"{{cx}}, {{cy}}",
            (cx + 10, cy - 10),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.6,
            (255, 0, 0),
            2,
            cv2.LINE_AA
        )

    return output

def _get_single_object_info(self, contour):
    objects = self.picking_evaluator.get_object_info([contour])

    if not objects:
        return None

```

```
return objects[0]
```

A.2 `segmentation_processor.py` — реалізує сегментацію об'єктів на зображенні, формування маски, морфологічну обробку, розділення дотичних об'єктів та пошук контурів.

```
import cv2
```

```
import numpy as np
```

```
class SegmentationProcessor:
```

```
    def __init__(self, config, debug=False):
```

```
        self.config = config
```

```
        self.debug = debug
```

```
    def split_touching_objects(self, binary_mask, original_image):
```

```
        seg_cfg = self.config["segmentation"]
```

```
        object_cfg = self.config["object_filter"]
```

```
        _, mask = cv2.threshold(binary_mask, 127, 255, cv2.THRESH_BINARY)
```

```
        kernel_size = seg_cfg["watershed_kernel_ks"]
```

```
        kernel = np.ones((kernel_size, kernel_size), np.uint8)
```

```
        opened = cv2.morphologyEx(
```

```
            mask,
```

```
            cv2.MORPH_OPEN,
```

```
            kernel,
```

```
            iterations=seg_cfg["watershed_open_iterations"]
```

```
        )
```

```
        dist = cv2.distanceTransform(opened, cv2.DIST_L2, 5)
```

```
        dist_norm = cv2.normalize(dist, None, 0, 1.0, cv2.NORM_MINMAX)
```

```
        _, sure_fg = cv2.threshold(
```

```
            dist_norm,
```

```
            seg_cfg["ws_fg_thresh"],
```

```
            1.0,
```

```
            cv2.THRESH_BINARY
```

```
        )
```

```

sure_fg = (sure_fg * 255).astype(np.uint8)

sure_bg = cv2.dilate(
    opened,
    kernel,
    iterations=seg_cfg["ws_bg_dilate_iter"]
)

unknown = cv2.subtract(sure_bg, sure_fg)

_, markers = cv2.connectedComponents(sure_fg)

markers = markers + 1
markers[unknown == 255] = 0

markers = cv2.watershed(original_image.copy(), markers)

contours = self._extract_contours_from_markers(
    markers,
    mask,
    min_area=object_cfg["min_area_px"]
)

markers_vis = self._create_markers_visualization(markers, mask.shape)

if self.debug:
    cv2.imshow("sure_fg", sure_fg)
    cv2.imshow("markers_vis", markers_vis)

return contours, markers_vis

def try_resplit_contour_with_edges(
    self,
    contour,
    object_mask,
    edge_mask,
    min_area=None
):
    object_cfg = self.config["object_filter"]
    seg_cfg = self.config["segmentation"]

    if min_area is None:

```

```

min_area = object_cfg["min_area_px"] // 2

x, y, w, h = cv2.boundingRect(contour)

roi_obj = object_mask[y:y + h, x:x + w]
roi_edges = edge_mask[y:y + h, x:x + w]

local_mask = np.zeros_like(roi_obj)

shifted = contour.copy()
shifted[:, 0, 0] -= x
shifted[:, 0, 1] -= y

cv2.drawContours(local_mask, [shifted], -1, 255, -1)

roi_obj_local = cv2.bitwise_and(roi_obj, local_mask)

edges_bin = cv2.threshold(
    roi_edges,
    1,
    255,
    cv2.THRESH_BINARY
)[1]

kernel_size = seg_cfg["edge_resplit_dilate_ks"]
kernel = np.ones((kernel_size, kernel_size), np.uint8)

edges_thick = cv2.dilate(
    edges_bin,
    kernel,
    iterations=seg_cfg["edge_resplit_dilate_iter"]
)

cut = roi_obj_local.copy()
cut[edges_thick > 0] = 0

_, labels, stats, _ = cv2.connectedComponentsWithStats(
    cut,
    connectivity=8
)

new_contours = []

```

```

for label in range(1, stats.shape[0]):
    area = stats[label, cv2.CC_STAT_AREA]

    if area < min_area:
        continue

    comp = np.zeros_like(cut)
    comp[labels == label] = 255

    cnts, _ = cv2.findContours(
        comp,
        cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE
    )

    if not cnts:
        continue

    cnt = max(cnts, key=cv2.contourArea)

    cnt[:, 0, 0] += x
    cnt[:, 0, 1] += y

    new_contours.append(cnt)

return new_contours

def needs_resplit(self, obj):
    object_cfg = self.config["object_filter"]
    seg_cfg = self.config["segmentation"]

    if obj.area > object_cfg["max_area_px"] * seg_cfg["resplit_area_multiplier"]:
        return True

    if obj.rectangularity < object_cfg["rectangularity_min"]:
        return True

    if obj.solidity < object_cfg["solidity_min"]:
        return True

    return False

```

```

def _extract_contours_from_markers(self, markers, mask, min_area):
    contours = []

    for label in np.unique(markers):
        if label <= 1:
            continue

        component_mask = np.zeros_like(mask, dtype=np.uint8)
        component_mask[markers == label] = 255

        cnts, _ = cv2.findContours(
            component_mask,
            cv2.RETR_EXTERNAL,
            cv2.CHAIN_APPROX_SIMPLE
        )

        for cnt in cnts:
            area = cv2.contourArea(cnt)

            if area > min_area:
                contours.append(cnt)

    return contours

def _create_markers_visualization(self, markers, shape):
    markers_vis = np.zeros(
        (shape[0], shape[1], 3),
        dtype=np.uint8
    )

    markers_vis[markers == -1] = (0, 0, 255)
    markers_vis[markers > 1] = (255, 255, 255)

    return markers_vis

```

A.3 `pose_estimator.py` — визначає просторове положення та орієнтацію об'єктів на основі їх геометричних параметрів і розраховує координати у реальних одиницях вимірювання.

```
import cv2
```

```

import numpy as np
import math

class PoseEstimator:
    def __init__(
        self,
        image_width_px,
        image_height_px,
        horizontal_fov_deg=60.0,
        vertical_fov_deg=45.0,
        approx_epsilon_ratio=0.03
    ):
        self.image_width_px = image_width_px
        self.image_height_px = image_height_px
        self.horizontal_fov_deg = horizontal_fov_deg
        self.vertical_fov_deg = vertical_fov_deg
        self.approx_epsilon_ratio=approx_epsilon_ratio

        self.camera_matrix = self._create_camera_matrix()
        self.dist_coeffs = np.zeros((4, 1), dtype=np.float32)

    def _create_camera_matrix(self):
        fov_x = math.radians(self.horizontal_fov_deg)
        fov_y = math.radians(self.vertical_fov_deg)

        fx = self.image_width_px / (2.0 * math.tan(fov_x / 2.0))
        fy = self.image_height_px / (2.0 * math.tan(fov_y / 2.0))

        cx = self.image_width_px / 2.0
        cy = self.image_height_px / 2.0

        return np.array([
            [fx, 0, cx],
            [0, fy, cy],
            [0, 0, 1]
        ], dtype=np.float32)

    def estimate_pose_from_contour(
        self,
        contour,
        real_width_mm,

```

```

    real_height_mm

):
    image_points = self._get_ordered_corners(contour)

    if image_points is None:
        return None

    half_w = real_width_mm / 2.0
    half_h = real_height_mm / 2.0

    object_points = np.array([
        [-half_w, -half_h, 0],
        [ half_w, -half_h, 0],
        [ half_w,  half_h, 0],
        [-half_w,  half_h, 0],
    ], dtype=np.float32)

    success, rvec, tvec = cv2.solvePnP(
        object_points,
        image_points,
        self.camera_matrix,
        self.dist_coeffs,
        flags=cv2.SOLVEPNP_ITERATIVE
    )

    if not success:
        return None

    rotation_matrix, _ = cv2.Rodrigues(rvec)

    alpha, beta, gamma = self._rotation_matrix_to_euler_angles(
        rotation_matrix
    )

    return {
        "alpha": alpha,
        "beta": beta,
        "gamma": gamma,
        "x_mm": float(tvec[0][0]),
        "y_mm": -float(tvec[1][0]),
        "z_mm": float(tvec[2][0]),
        "rvec": rvec,

```

```

        "tvec": tvec,
        "corners": image_points
    }

def _get_ordered_corners(self, contour):
    epsilon = self.approx_epsilon_ratio * cv2.arcLength(contour, True)
    approx = cv2.approxPolyDP(contour, epsilon, True)

    if len(approx) != 4:
        rect = cv2.minAreaRect(contour)
        box = cv2.boxPoints(rect)
        points = np.array(box, dtype=np.float32)
    else:
        points = approx.reshape(4, 2).astype(np.float32)

    return self._order_points(points)

def _order_points(self, points):
    rect = np.zeros((4, 2), dtype=np.float32)

    s = points.sum(axis=1)
    diff = np.diff(points, axis=1)

    rect[0] = points[np.argmin(s)]
    rect[2] = points[np.argmax(s)]
    rect[1] = points[np.argmin(diff)]
    rect[3] = points[np.argmax(diff)]

    return rect

def _rotation_matrix_to_euler_angles(self, R):
    sy = math.sqrt(R[0, 0] ** 2 + R[1, 0] ** 2)

    singular = sy < 1e-6

    if not singular:
        alpha = math.atan2(R[2, 1], R[2, 2])
        beta = math.atan2(-R[2, 0], sy)
        gamma = math.atan2(R[1, 0], R[0, 0])
    else:
        alpha = math.atan2(-R[1, 2], R[1, 1])
        beta = math.atan2(-R[2, 0], sy)

```

```

gamma = 0

return (
    math.degrees(alpha),
    math.degrees(beta),
    math.degrees(gamma)
)

```

A.4 best_object_selector.py — аналізує всі знайдені об'єкти та вибирає найбільш придатний для захоплення на основі розрахованих параметрів і оцінки.

```

from enum import Enum
import cv2

```

```

class PickMode(Enum):
    ANY = "any"
    SUCTION = "suction"
    GRIPPER = "gripper"

```

```

class BestObjectSelector:
    def __init__(
        self,
        pick_mode=PickMode.ANY,
        min_solidity=0.85,
        min_rectangularity=0.68,
        area_norm_px=30000
    ):
        self.pick_mode = pick_mode
        self.min_solidity = min_solidity
        self.min_rectangularity = min_rectangularity
        self.area_norm_px = area_norm_px

    def select_best(self, detection_result, pick_mode=None):
        if pick_mode is None:
            pick_mode = self.pick_mode

        objects = detection_result.get("objects", [])
        candidates = []

        for obj in objects:

```

```

        if not self._is_valid_candidate(obj, pick_mode):
            continue

        score = self._calculate_score(obj, pick_mode)
        candidates.append((score, obj))

    if not candidates:
        return None

    candidates.sort(key=lambda item: item[0], reverse=True)
    best_score, best_object = candidates[0]

    return {
        "pick_mode": pick_mode.value,
        "best_object": best_object,
        "score": best_score,
        "reason": self._explain_choice(best_object, best_score, pick_mode)
    }

def _is_valid_candidate(self, obj, pick_mode):
    if obj.get("occluded", True):
        return False

    if obj.get("solidity", 0) < self.min_solidity:
        return False

    if obj.get("rectangularity", 0) < self.min_rectangularity:
        return False

    suction_ok = obj.get("suction_pickable", False)
    gripper_ok = obj.get("gripper_pickable", False)

    if pick_mode == PickMode.SUCTION:
        return suction_ok

    if pick_mode == PickMode.GRIPPER:
        return gripper_ok

    if pick_mode == PickMode.ANY:
        return suction_ok or gripper_ok

```

```

return False

def _calculate_score(self, obj, pick_mode):
    score = 0.0

    solidity = obj.get("solidity", 0)
    rectangularity = obj.get("rectangularity", 0)
    area = obj.get("area", 0)

    score += solidity * 35
    score += rectangularity * 25

    normalized_area_score = min(area / 30000, 1.0)
    score += normalized_area_score * 15

    if pick_mode in [PickMode.ANY, PickMode.SUCTION]:
        if obj.get("suction_pickable", False):
            score += self._suction_score(obj)

    if pick_mode in [PickMode.ANY, PickMode.GRIPPER]:
        if obj.get("gripper_pickable", False):
            score += self._gripper_score(obj)

    return score

def _suction_score(self, obj):
    score = 0.0

    suction_info = obj.get("reason", {}).get("suction", {})
    inside_ratio = suction_info.get("inside_ratio", 0)

    score += 20
    score += inside_ratio * 20

    return score

def _gripper_score(self, obj):
    score = 0.0

    gripper_info = obj.get("reason", {}).get("gripper", {})

```

```

jaw_ratio_1 = gripper_info.get("jaw_inside_ratio_1", 0)
jaw_ratio_2 = gripper_info.get("jaw_inside_ratio_2", 0)
contact_len = gripper_info.get("contact_len", 0)

score += 15
score += min(jaw_ratio_1, jaw_ratio_2) * 15
score += min(contact_len / 250, 1.0) * 10

return score

def _explain_choice(self, obj, score, pick_mode):
    return {
        "object_id": obj.get("id"),
        "pick_mode": pick_mode.value,
        "score": score,
        "center_x": obj.get("center_x"),
        "center_y": obj.get("center_y"),
        "angle": obj.get("angle"),
        "solidity": obj.get("solidity"),
        "rectangularity": obj.get("rectangularity"),
        "area": obj.get("area"),
        "suction_pickable": obj.get("suction_pickable"),
        "gripper_pickable": obj.get("gripper_pickable"),
        "occluded": obj.get("occluded")
    }

@staticmethod
def _draw_best_object(image, best_result):
    if best_result is None:
        return image

    best_object = best_result["best_object"]

    cx = int(best_object.get("center_x", 0))
    cy = int(best_object.get("center_y", 0))

    width_px = float(best_object.get("width_px", 0))
    height_px = float(best_object.get("height_px", 0))
    angle = float(best_object.get("raw_angle", best_object.get("angle", 0)))

    rect = ((cx, cy), (width_px, height_px), angle)

```

```

box = cv2.boxPoints(rect)
box = box.astype("int32")

cv2.drawContours(image, [box], 0, (0, 255, 0), 3)
cv2.circle(image, (cx, cy), 6, (255, 0, 0), -1)

cv2.putText(
    image,
    f'{cx}, {cy} score={best_result.get('score', 0):.1f}',
    (cx + 10, cy - 10),
    cv2.FONT_HERSHEY_SIMPLEX,
    0.6,
    (255, 0, 0),
    2,
    cv2.LINE_AA
)

return image

```

ДОДАТОК Б Презент

Актуальність теми

- Автоматизація виробництва потребує машинного зору
- Визначення положення та орієнтації об'єктів
- Оцінка можливості захоплення
- ROS2 забезпечує взаємодію компонентів



Мета та завдання роботи

Мета: Розробка системи визначення просторової орієнтації предметів для роботизованого маніпулятора.

Завдання:

1. Провести аналіз існуючих методів комп'ютерного зору для виявлення та сегментації об'єктів.
2. Розробити загальний алгоритм роботи системи визначення просторової орієнтації об'єктів із використанням OpenCV.
3. Реалізувати взаємодію ROS2-вузлів через topics та custom messages.
4. Розробити алгоритм обробки зображення для виділення та аналізу об'єктів.
5. Реалізувати сегментацію зображення та виділення контурів об'єктів.
6. Розробити алгоритм оцінки можливості захоплення об'єктів та визначення їх параметрів.
7. Інтегрувати систему комп'ютерного зору з системою керування роботизованим маніпулятором.
8. Провести тестування системи та оцінити результати її роботи.



Аналіз існуючих рішень

Метод	Швидкість	Точність	Розділення дотичних об'єктів	Стійкість до освітлення	Складність реалізації
RGB-порогова сегментація	Висока	Низька	Низька	Низька	Низька
HSV-сегментація	Висока	Середня	Низька	Висока	Низька
Canny Edge Detection	Висока	Висока	Низька	Середня	Низька
Watershed	Середня	Висока	Висока	Середня	Середня
Contour Analysis	Висока	Висока	Низька	Середня	Низька
Hough Transform	Середня	Середня	Низька	Середня	Середня
YOLO	Низька	Дуже висока	Висока	Висока	Висока

Обраний підхід: HSV + Canny + Watershed + Contour Analysis:

через високу швидкість роботи, достатню точність сегментації та можливість обробки зображень у реальному часі без навчання моделі.

Алгоритм обробки зображення об'єктів за допомогою OpenCV

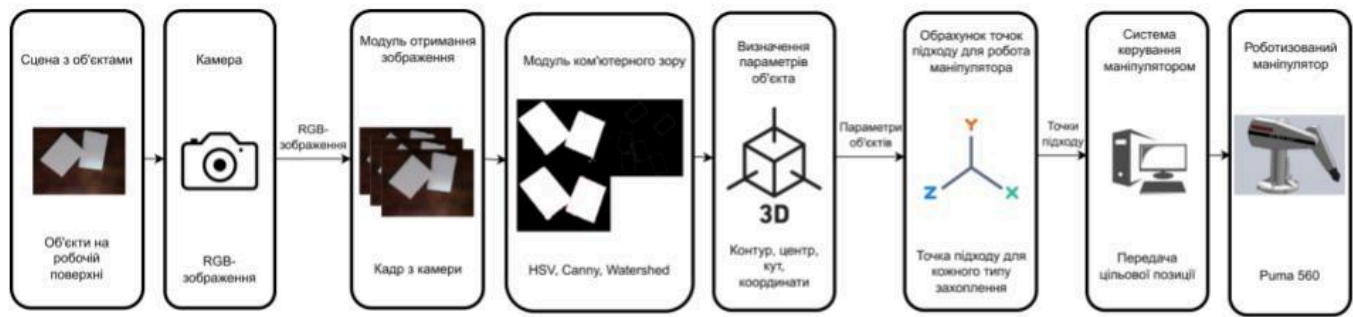
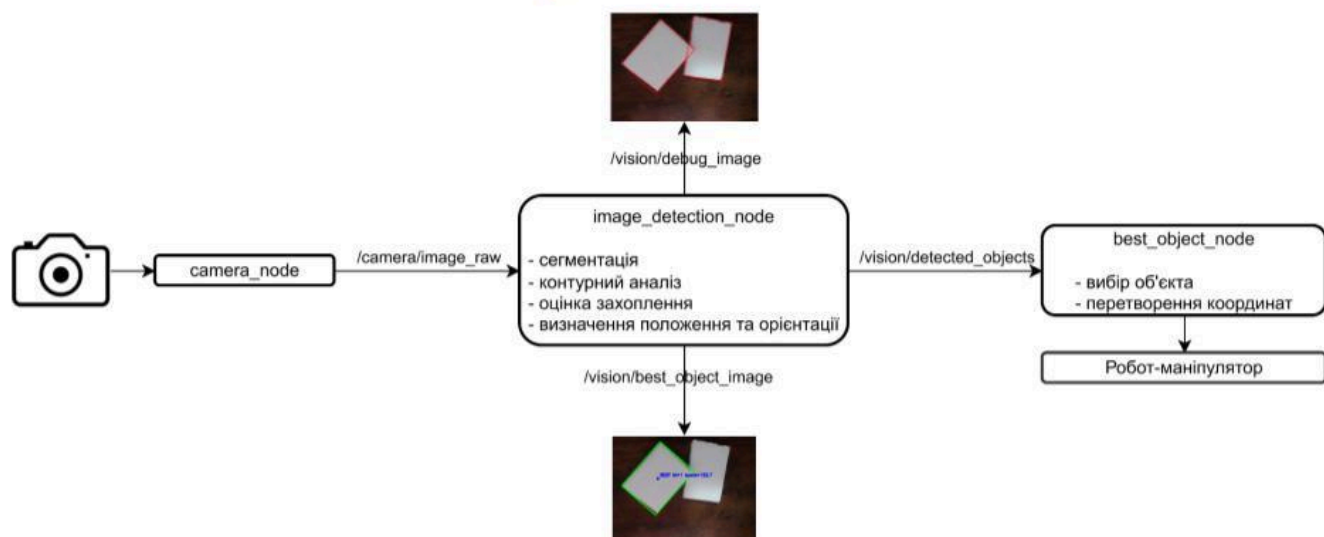


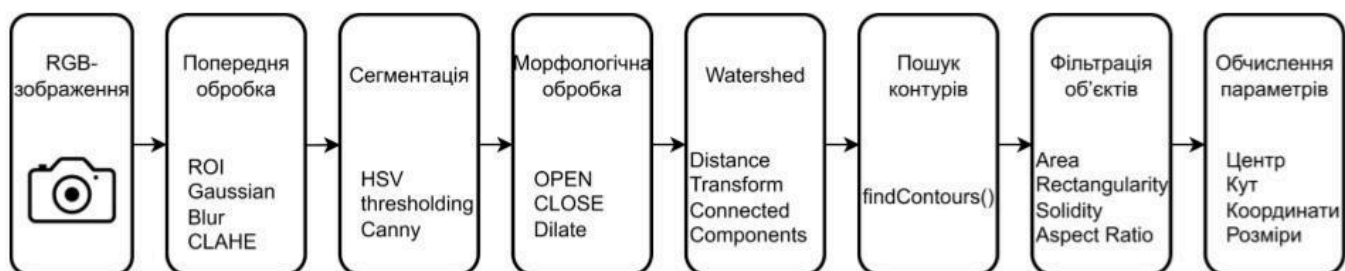
Схема взаємодії ROS2-вузлів



Система реалізована у вигляді ROS2-вузлів, що взаємодіють через topics та custom messages.



Алгоритм обробки зображення



Алгоритм забезпечує виділення об'єктів, їх фільтрацію та визначення геометричних параметрів для подальшого захоплення маніпулятором.



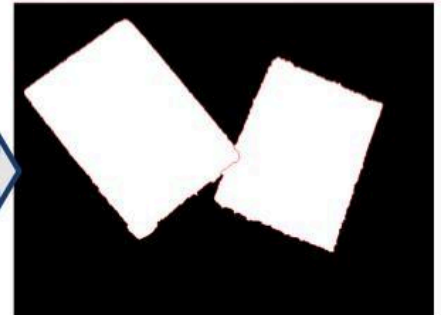
Сегментація та виділення контурів об'єктів на зображенні

- HSV-маска
- Видалення шуму
- Watershed-розділення об'єктів
- Розділення дотичних об'єктів
- Виділення контурів

HSV mask



Separated contours



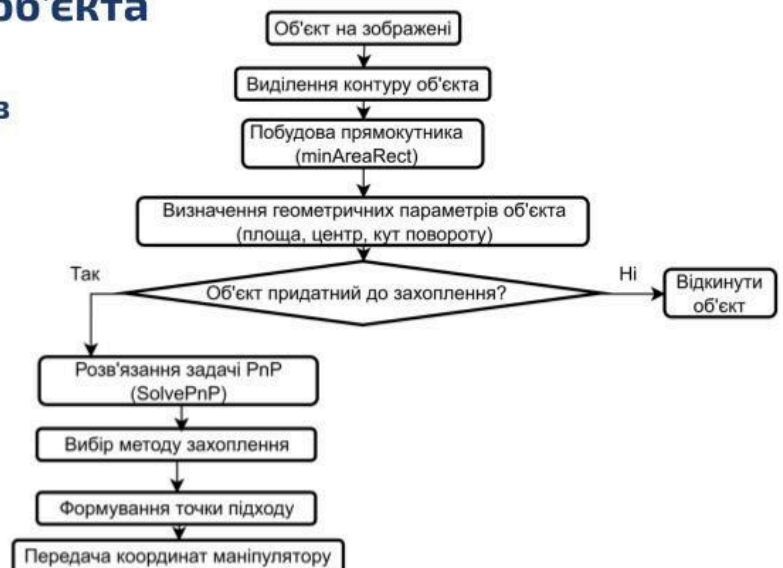
Алгоритм захоплення об'єкта

Визначення параметрів об'єктів

- minAreaRect
- Кут повороту
- Орієнтований прямокутник

Критерії оцінки захоплення

- Площа об'єкта
- Прямокутність
- Solidity
- Aspect Ratio



Інтеграція СТЗ з системою керування роботизованого маніпулятора

Для присоски:

$$P_{approach} = P_{object} + d \cdot \bar{n}$$

де:

P_{object} — координати центра об'єкта;

d — відстань, на яку маніпулятор відходить від поверхні вздовж нормалі перед початком захоплення;

$\bar{n}$ — одиничний вектор нормалі до поверхні об'єкта.

Для клешні:

$$P_{approach} = (x, y, z + h)$$

де:

x, y, z — координати центра об'єкта;

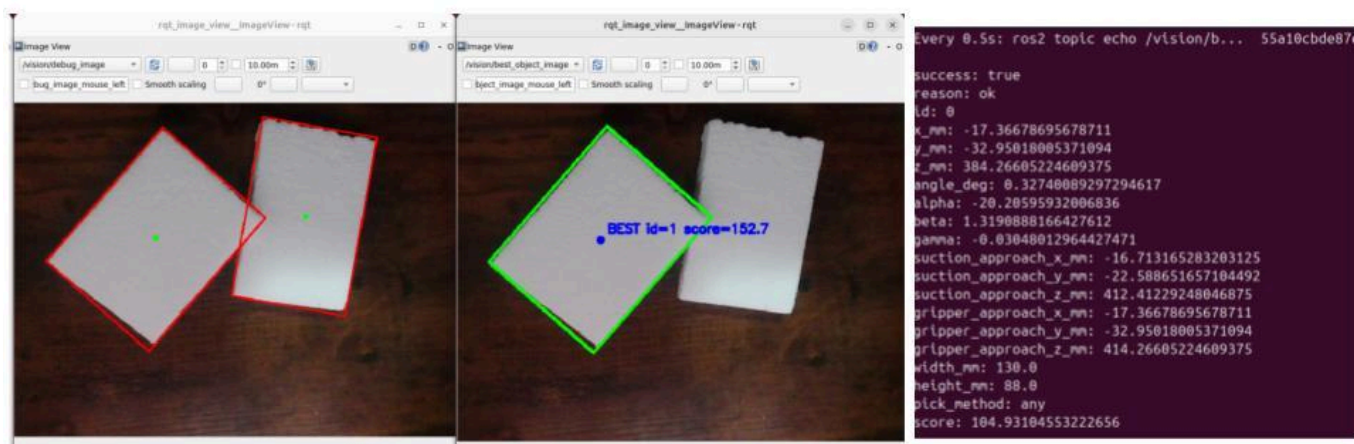
h — висота безпечного підходу над об'єктом по осі Z



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

10

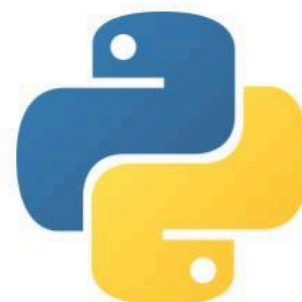
Тестування роботи інтегрованої системи



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

11

Використані інструменти



Висновки

В рамках цієї роботи було:

1. Проведено аналіз існуючих методів комп'ютерного зору для виявлення та сегментації об'єктів.
2. Розроблено загальний алгоритм роботи системи визначення просторової орієнтації об'єктів із використанням OpenCV.
3. Реалізовано взаємодію ROS2-вузлів через topics та custom messages.
4. Розроблено алгоритм обробки зображення для виділення та аналізу об'єктів.
5. Реалізовано сегментацію зображення та виділення контурів об'єктів.
6. Розроблено алгоритм оцінки можливості захоплення об'єктів та визначення їх параметрів.
7. Виконано інтеграцію системи комп'ютерного зору з системою керування роботизованим маніпулятором.
8. Проведено тестування системи, результати якого підтвердили працездатність розробленого програмного забезпечення та можливість його використання для автоматизованого захоплення об'єктів роботизованими системами.





Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"