Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та мехатроніки
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Розробка програмної бібліотеки для моделювання функцій сенсорної системи мобільного робота Festo Robotino
(тема)

Виконав: студент 2 курсу, гр. КТРСм-19-1
Рижов В.Б.
(прізвище, ініціали)

Спеціальність 151 – Автоматизація та комп'ютерно-інтегровані технології

освітньої програми Комп'ютеризовані та робототехнічні системи
(код і повна назва)

Тип програми освітньо-професійна
(повна назва освітньої програми)

Керівник проф. Сінотін А.М.
(посада, прізвище, ініціали)

Допускається до захисту
зав. кафедри _____ Невлюдов І.Ш.
(підпис) (прізвище, ініціали)

2020

Харківський національний університет радіоелектроніки

Факультет	Автоматики і комп'ютеризованих технологій
Кафедра	Комп'ютерно-інтегрованих технологій, автоматизації та мехатроніки
Рівень вищої освіти	другий (магістерський)
Спеціальність	151 – Автоматизація та комп'ютерно-інтегровані технології
Тип програми	освітньо-професійна
Освітня програма	Комп'ютеризовані та робототехнічні системи

(код і повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____ 2020 р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Рижову Віталію Борисовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмної бібліотеки для моделювання функцій сенсорної системи мобільного робота Festo Robotino
затверджена наказом по університету від 02.11. 2020 р., № 1509 Ст.
2. Термін подання студентом роботи до екзаменаційної комісії 09.12. 2020 р.
3. Вихідні дані до роботи: Мобільний робот Festo Robotino, середовище розробки Robotino View
4. Зміст пояснювальної записки (перелік питань, що потрібно розробити)
4.1 Вступ
4.2 Аналіз засобів моделювання та керування робототехнічними системами
4.3 Розробка архітектури програмної бібліотеки для моделювання функцій сенсорної системи
4.4 Розробка програмної бібліотеки для моделювання функції сенсорної системи
4.5 Безпека життєдіяльності та охорона праці
4.6 Висновки
4.7 Перелік джерел посилання

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Демонстраційний матеріал, представлений у форматі презентації PowerPoint (*.ppt) 15 с., формату А4.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз вихідних даних та літератури		
	за темою атестаційної роботи	05.09.2020	Виконано
2	Аналіз засобів моделювання та		
	керування робототехнічними		
	системами	12.09.2020	Виконано
3	Постановка мети та задач		
	дослідження	16.09.2020	Виконано
4	Розробка архітектури програмної		
	бібліотеки для моделювання		
	функцій сенсорної системи	14.10.2020	Виконано
5	Розробка програмної бібліотеки для		
	моделювання функції сенсорної	04.11.2020	Виконано
6	Оформлення пояснювальної записки		
	та презентації	22.11.2020	Виконано
7	Подання атестаційної роботи до		
	екзаменаційної комісії	23.12.2020	Виконано

Дата видачі завдання 01 вересня 2020 р.

Студент _____ Рижов В.Б.

(підпис) (прізвище, ініціали)

Керівник роботи _____ проф. Сінотін А.М.

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Атестаційна робота: 125 с., 0 табл., 35 рис., 2 дод., 22 джерел.

ОДОМЕТРІЯ, РОБОТ, ROBOTINO, ВІЗУЛІЗАЦІЯ, ЛОКАЛІЗАЦІЯ, C++, АЛГОРИТМ, БІБЛІОТЕКА, СИСТЕМА.

Об’єкт дослідження – сенсорна система мобільного робота.

Ціль роботи – підвищення ефективності сенсорної системи Robotino.

Методи дослідження й перелік апаратури – методи розрахунку розташування камери, використовуючи лише візуальні дані, що надаються, було протестовано на компьютері MSI 16 ОЗУ з процесором i7-K550.

Предмет дослідження – програмна бібліотека для моделювання функцій сенсорної системи мобільного робота Festo Robotino.

У магістерській атестаційній роботі виконано розробку програмної бібліотеки з удосконаленим керуванням сенсорною системою робота. А саме реалізація самолокалізації робота за допомогою алгоритму візуальної одометрії, який вдосконалюється за допомогою адаптивного вибору функцій. Впровадженя данної роботи є ефективним при її застосуванні у сучасних системах промислового виробництва.

Також було проведено аналіз можливостей сенсорної роботи системи Festo Robotino щодо магістерської атестаційної роботи, що викладено в статті на Міжнародній науково-технічній конференції студентів, аспірантів та молодих вчених.

Сфера застосування - сучасні системи промислового виробництва.

ABSTRACT

Attestation work contains: 125 pp., 0 tabs., 35 fig., 2 add., 22 sources.

ODOMETRY, ROBOT, ROBOTINO, VISUALIZATION, LOCALIZATION, C ++, ALGORITHM, LIBRARY, SYSTEM.

The object of research is the sensory system of a mobile robot.

The aim of the work is to develop software libraries for modeling the functions of the Festo Robotino mobile robot touch system.

Research methods and equipment fracture - methods of calculating the draw of the stone, using only verified data to be provided, were tested on a computer MSI 16 RAM with an i7-K550 processor.

The subject of the study is the evaluation of the efficiency of the Robotino sensor system.

In the master's attestation work the development of a software library with advanced control of the robot sensory system was performed. Namely, the implementation of self-localization of the robot using a computer odometry algorithm, which will be improved through adaptive selection of functions. The implementation of this work is effective in its application in modern industrial production systems.

An analysis of the possibilities of sensory work of the Festo Robotino system for master's attestation work was also carried out, it was presented in the article at the international scientific and technical conferences of students, graduate students and young scientists.

Scope - modern systems of industrial production.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ВВ – варіанти використання;

ОС – операційна система;

ПК – персональний комп’ютер;

ПЗ – програмне забезпечення;

GPS – global positioning system;

IMU – inertial measurement unit;

LAN – local area network;

OpenCV - open computer vision;

REST – representational state transfer;

ROS - robot operating system;

RPC – remote procedure call;

VO – visual odometry;

WLAN – wireless local area network;

UML – Unified Modeling Language.

ВСТУП

В останні роки автономні транспортні засоби ставали дедалі актуальнішими в науці, а також у нашому повсякденному житті. Є багато можливостей таких транспортних засобів. Одним із напрямків застосування є дослідження або робота в небезпечних середовищах, таких як палаючі конструкції, забруднені ядерними зонами або навіть інші планети. Інший приклад використання - це службові роботи, які можуть підтримувати інвалідів або людей похилого віку. Крім того, багато нещасних випадків на вулицях можна запобігти за допомогою самохідних автомобілів. Це лише декілька різних можливостей. Ось чому нам було б вигідно в майбутньому розробляти якісні автономні машини.

Важливим аспектом цієї роботи є міцна і точна само локалізація цих роботів. Для виконання всіх завдань для навігації потрібно правильне розташування. Сьогодні вже існує багато різних можливостей оцінки руху. Однією з найпоширеніших технік є система глобального позиціонування (GPS). Незважаючи на те, що зовні він працює досить добре, його не можна використовувати в приміщеннях. Більше того, він точний лише в межах декількох метрів, тому GPS може не підходити для кожного додатка. Ще два варіанти локалізації - це інерційні одиниці вимірювання (IMU) або датчики швидкості колеса. Однак надійні IMU є досить дорогими, і одометрія колеса працює лише в тому випадку, якщо немає пробуксовки колеса, що не завжди може бути забезпечено.

Оскільки людське око є одним з найважливіших датчиків для орієнтації людини, можливо також можна скористатися перевагами камери для локалізації. Процес визначення руху з послідовності зображень називається візуальною одометрією (VO). Протягом останніх кількох років камери стають більш точними та дешевими, що робить їх відповідним датчиком для автономних автомобілів. Отже, ця робота стосуватиметься застосування

алгоритму візуальної одометрії для самолокалізації. Для покращення результатів голосування буде застосовано адаптивну стратегію вибору функцій.

Об’єкт дослідження – сенсорна система мобільного робота.

Предмет дослідження це програмна бібліотека для моделювання функцій сенсорної системи мобільного робота Festo Robotino.

Новизна роботи полягає у розробці та реалізації адаптивного візуального алгоритму візуальної одометрії для самостійної локалізації мобільних роботів.

Метою випускної магістерської роботи є розробка програмної бібліотеки з удосконаленими програмними засобами керування сенсорною системою мобільного роботк Festo Robotino. Досягнення мети полягає у реалізації самолокалізації робота за допомогою алгоритму візуальної одометрії, який вдосконалюється за допомогою адаптивного вибору функцій.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати засоби моделювання та керування робототехнічними системами;

– проаналізувати особливості та структуру робота Robotino;

– проаналізувати операційні системи роботів та принципи програмного керування робототехнічними системами;

– розробити архітектуру програмної бібліотеки для моделювання функцій робота;

– розробити алгоритм адаптивної візуальної одометрії;

– розробити алгоритм адаптивного пошуку та стратегію вибору функцій;

– розробити UML-діаграми функціонування програмної бібліотеки;

– розробити програмну бібліотеку для моделювання функцій сенсорної системи, а саме прикладний інтерфейс розробника;

– оформити записку згідно з вимогами ДСТУ 3008:2015 [1].

1 АНАЛІЗ ЗАСОБІВ МОДЕЛЮВАННЯ ТА КЕРУВАННЯ РОБОТОТЕХНІЧНИМИ СИСТЕМАМИ

1.1 Сучасні типи мобільних роботів та їх застосування

Системи мобільних роботів останніми роками все більше і більше впроваджуються в сучасні системи промислового виробництва.

В Австрії 99,7% усіх компаній є малими та середніми підприємствами (SMEs), однак на них припадає лише 64% частки чистого доходу від продажу [2]. Більшість із них не могли дозволити собі технологій, необхідних для четвертої промислової революції ((Industry 4.0) відповідно до їх обмежених ресурсів.

Ця різниця між SMEs та великими підприємствами помітна в їх внутрішній складській логістиці та виробних лініях. Мобільні роботи, якими користуються великі підприємства, збільшують ефективність та гнучкість логістичних, складальних та виробничих процесів, завдяки чому людські ресурси будуть скорочені до монотонної роботи [3]. Коли мова йде про мобільний роботи - використовуються різні терміни. Найпоширеніші з них автономні мобільні роботи (AMR) та автоматизоване керування транспортних засобів (AGV) і до тепер немає абсолютно чітко встановленої відмінності. Загалом, AGV це внутрішні конвеєрні системи, пов’язані з підлогою, автоматично контрольовані транспортні засоби, основним завданням яких є обробка матеріалів.

Керівництво транспортним засобом реалізується за допомогою деяких апаратних засобів інфраструктури (проводів, чорних ліній тощо), розміщених поблизу оточення рухомого робота. На відміну від AMR може пересуватися автономно і виконує певне завдання (наприклад, побутовий робот-пилосос).

Навігація через його оточення дають датчики, встановлені на робота [4]. Мобільні роботи набагато гнучкіші в їх використанні та надають функції як динамічно-специфічні модифікації для клієнта, планування руху з урахуванням ситуації, спільні операції тощо. Отже, вони надзвичайно актуальні для сучасних

додатків Industry 4.0. Однак, використовуючи мобільну робототехніку в промислових додатках призводить до збільшення вимоги безпеки та добре організоване співіснування між рухомими машинами та людьми.

Існують різні типи мобільних роботів. У цьому розділі спочатку ми коротко опишемо два обрані важливі компоненти мобільних роботосистем щодо безпеки та другої уваги на два промислово важливі типи мобільних роботів (колісні роботи та мобільні маніпулятори) більш докладно.

Індустрія 4.0 - це частина четвертої промислової революції. Четверта промислова революція охоплює сфери, які, як правило, не класифікуються як галузі (рис. 1.1).

Хоча терміни "Індустрія 4.0" та "Четверта промислова революція" часто використовуються як взаємозамінні, "Індустрія 4.0" означає заводи, на яких машини доповнюються бездротовими з'єднаннями та датчиками, підключеними до системи, яка може контролювати всю систему [5].

Рисунок 1.1 – Зображення Industry 4.0

Коли комп'ютери були впроваджені в Індустрію 3.0, це було руйнівним завдяки доданню абсолютно нових технологій. Зараз і в майбутньому, коли Індустрія 4.0 розгортається, комп’ютери з’єднуються і взаємодіють між собою, щоб зрештою приймати рішення без втручання людини.

Поєднання кіберфізичних систем, інтернет-речей та інтернет-систем дає можливість використовувати Індустрію 4.0. За підтримки інтелектуальних машин, які стають розумнішими, коли отримують більше даних, рослини стають більш ефективними, продуктивними та менш марними. Зрештою, це мережа цифрових підключених машин, які створюють та обмінюються інформацією, що надає реальну потужність Індустрії 4.0.

По суті, Industry 4.0 описує тенденцію автоматизації та обміну даними у виробничих технологіях та процесах, які включають кіберфізичні системи (CPS), інтернет-речі (IoT) та штучний інтелект (рис. 1.2) [5].

Рисунок 1.2 – Використання IoT у промисловості

Концепція включає:

– розумне виробництво;

– розумна фабрика;

– промисловість виробництва.

Індустрія 4.0 сприяє тому, що називається "розумною фабрикою". На модульних та інтелектуальних заводах кіберфізичні системи контролюють фізичні процеси, створюють віртуальну копію фізичного світу та приймають

децентралізовані рішення. За допомогою IoT кіберфізичні системи взаємодіють між собою та з людьми в реальному часі, всередині організації та в межах організаційних служб.

Індустрія 4.0 - це виробнича сторона, еквівалентна орієнтації споживача, де предмети побуту, від автомобілів до тостерів, будуть підключені до Інтернету речей.

Впроваджуючись у цій галузі, IoT пропонує одночасно кілька переваг:

– гнучкість виробництва досягається за рахунок відхилення жорстких "конвеєрних" рішень, які дозволяють масово приймати та виконувати окремі замовлення, вільніше впроваджувати нові рішення у виробництво;

– регулювання виробництва досягається завдяки контролю на всіх рівнях та роботі на єдиній технологічній платформі;

– ефективність виробництва пов'язана з меншими витратами, пов'язаними з людським фактором: помилки, простої, висока вартість людської праці.

Ви не будете здивовані наявністю промислових роботів у виробничих компаніях. Рівень робототехніки в галузі зростає з кожним днем. Зі збільшенням рівня робототехніки зростає і різноманітність роботів. Розробники все частіше розробляють різновиди роботів для зменшення виробничих витрат, підвищення продуктивності та спрощення налаштування та запуску роботів. Для початку, щоб зрозуміти це різноманіття промислової робототехніки, давайте розглянемо основні типи роботів та їх функції.

Шарнірні роботи (рис. 1.3) у роботі імітують рух людської руки, вони складаються з обертових кінематичних пар і мають 4 - 6 осей управління. Ця конструкція дозволяє шарнірним роботам виконувати космічні рухи зі складною траєкторією. Прикладами завдань, що формулюють роботу, є: зварювання або контурне фрезерування та фарбування складних поверхонь, таких як корпус. Вони також використовуються для більшості завдань з відбору та розміщення. Однак вважається, що функціональність навісних роботів для цих завдань частіше вичерпується. Встановлення шарнірного робота, як правило, нерухоме

(на підлозі), але є варіанти кріплення на стіні або стелі [6]. Дальність навісних роботів може досягати декількох метрів, а вантажопідйомність - понад 1 тонну.

Рисунок 1.3 – Шарнірний робот

Декартові маніпулятори (рис. 1.4), як правило, мають три лінійні осі управління. Кожна з цих осей знаходиться під прямим кутом до двох інших. Якщо один із блоків, що виконує горизонтальний рух, підтримується на обох кінцях, декартовий робот називається порталом. Оскільки декартові маніпулятори рухаються лише лінійно, розробники просто пишуть програму для переміщення роботів кудись у космос, використовуючи прості тригонометричні функції. Характеристики портальних роботів можуть бути абсолютно різними і залежати від обраних лінійних сервоприводів та механічної частини [5].

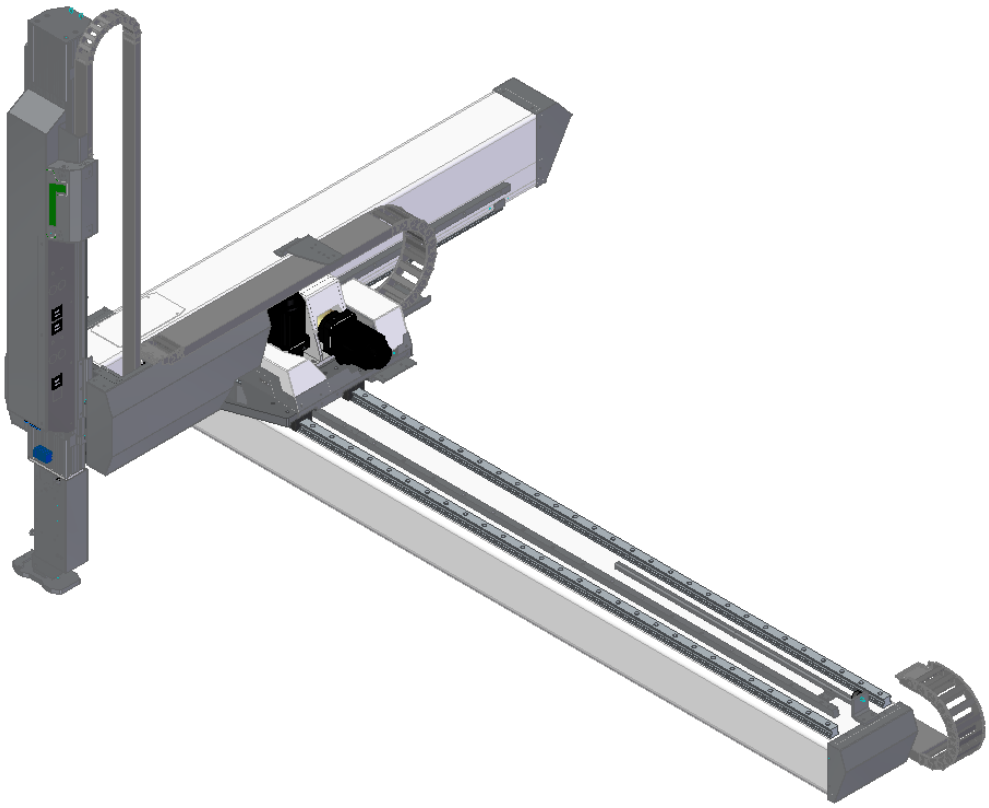

Рисунок 1.4 – Декартовий робот

Роботи Delta (рис. 1.5) - паралельний робот, відмінною рисою якого є трикутна платформа з трьома шарнірними важелями. Саме завдяки трикутній платформі робот отримав свою назву, оскільки візуально він виглядає як буква грецького алфавіту Δ - дельта. Особливістю є використання паралелограм у конструкції маніпулятора, що дозволяє зберегти просторову орієнтацію приводу. Головною перевагою дельта-роботів є їх висока швидкість завдяки мінімальній інерції.

Активно розвиваються мобільні роботи (роботи з транспортування матеріалів, зберігання, обслуговування верстатів) (рис. 1.6). Впровадження датчиків та навігаційних засобів у поєднанні з сучасним алгоритмічним програмним забезпеченням забезпечує високу швидкість та гнучкість. Вони можуть бути інтегровані з іншими мобільними системами та власною окремою навігаційною системою. Наприклад, час автономної роботи може перевозити

групу автомобілів і утримувати вантаж. Вони працюють з різними типами пневматичних тягових муфт.

Він оснащений функціями безпеки, що дозволяють мобільним роботам рухатися автономно та безпечно на виробничих поверхнях [6].

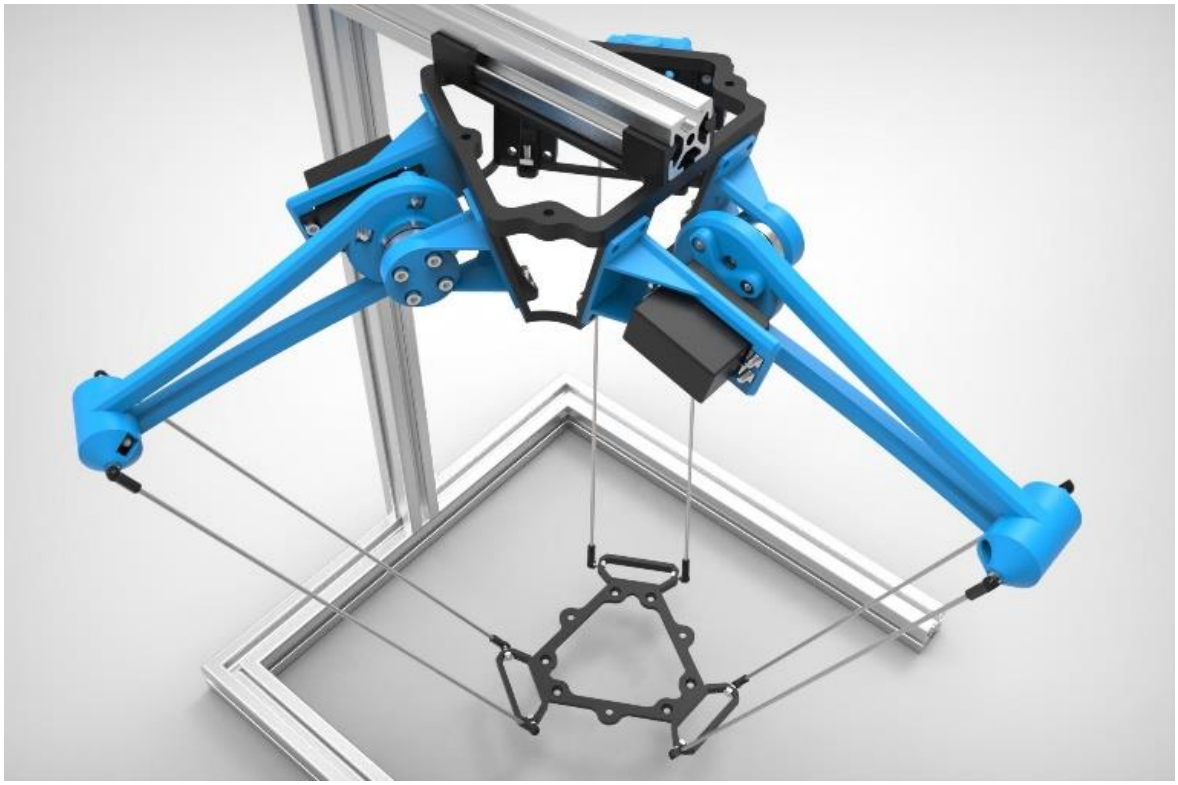

Рисунок 1.5 – Дельта-робот

Рисунок 1.6 – Мобільний робот транспортного типу

Колаборативні роботи (рис. 1.7) оснащені датчиками, які обмежують міцність або швидкість зв’язків і можуть, залежно від застосування, працювати в безпосередній близькості від людини та без встановлення захисної огорожі. Ця технологія розвивається швидше, ніж робот із сертифікацією безпеки. Деякі з цих роботів можуть бути двосторонніми, щоб краще копіювати навички маніпулювання людьми та сприяти інтеграції в існуючий виробничий процес без необхідності реконструкції. Використовуючи інтегровану систему машинного зору, адаптивна точність роботів, що працюють разом, дозволяє їм ефективно працювати в напівструктурованих середовищах.

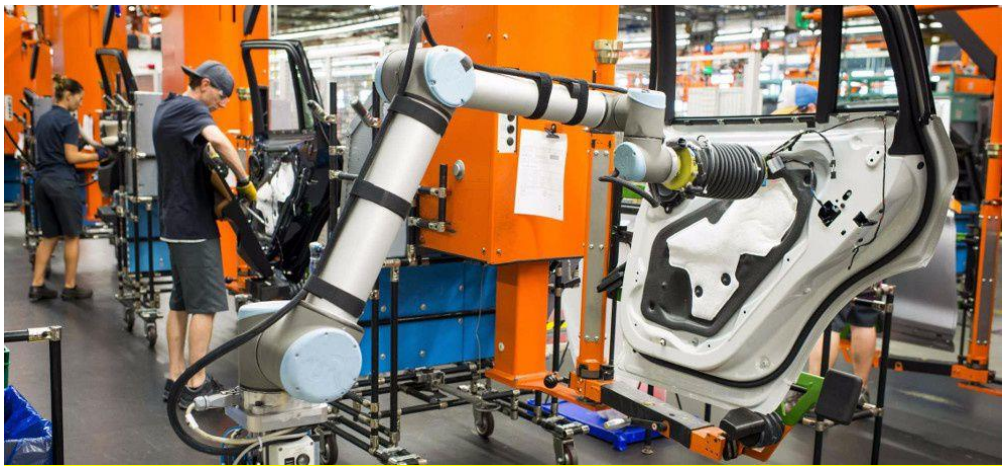

Рисунок 1.7 – Колаборативний робот

1.2 Формування зображення та калібрування камери

У цьому розділі будуть представлені деякі основні методи, які використовуються для обробки зображень загалом. Почнеться з моделі камери та способу створення зображень. Будуть також описані деякі основні математичні поняття. Наступна частина стосується калібрування камери та того, як її можна використовувати в подальших обчисленнях. Останній розділ проілюструє геометричні співвідношення між двома різними зображеннями камери.

Основою всіх підходів до комп'ютерного зору є зображення, зроблені камерою. Для того, щоб використовувати ці фотографії для різних програм, важливо знати, що саме являють собою зображення. Тож на початку буде пояснено процес формування реальності світу в образ. Тому введена так звана модель камери-обскури [7]. Рисунок 1.8 ілюструє, як це працює нижче.

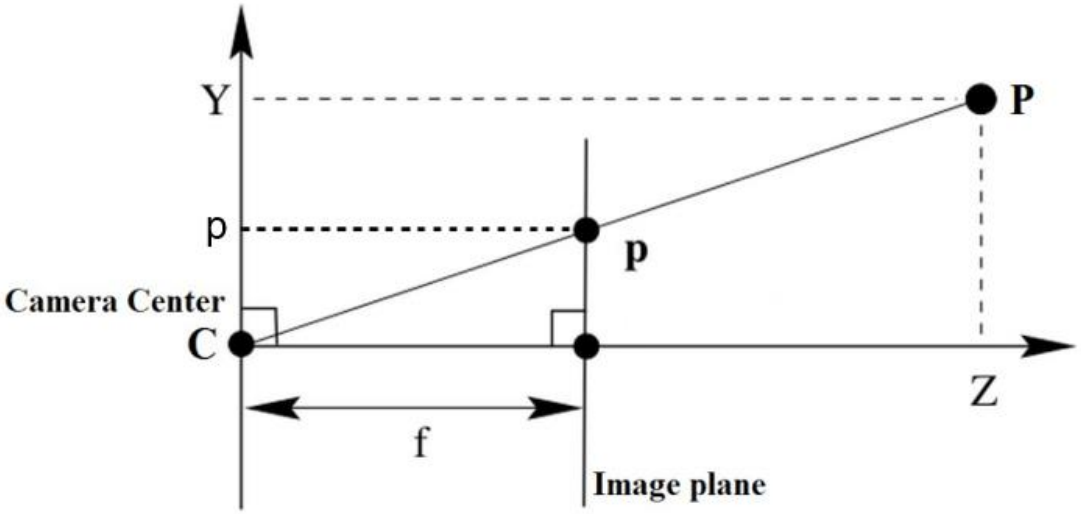

Рисунок 1.8 – Модель камери-отвору

Датчик камери вимірює промені світла, що надходять від 3D точки P. Образ цієї точки формується в площині зображення. Для простоти ми спочатку матємо подивитися на 2D приклад. Точка $P = (Y; Z)^T$ з відстанню Z до центру камери і положення Y відображається на точці p = (y) на зображенні. Фокусна відстань камери - f. Відповідно до теореми перехоплення, наступне співвідношення тримає [8], яке зображено на рисунку 1.9.

$$\frac{y}{Y} = \frac{f}{Z}.$$

Рисунок 1.9 – Зображення співвідношення відстані відносно її положення

Ця формула враховує, що фокусна відстань f дуже мала в порівнянні з відстань Z точки P до камери. Це припущення справедливо для більшості випадків у реальність, оскільки фокусна відстань становить лише кілька

сантиметрів. Отже, перетворення від форми (3D) до зображення (2D) буде записано, як на рисунку 1.10.

$$(X,Y,Z) \rightarrow (\frac{f}{Z}X, \frac{f}{Z}Y)$$

Рисунок 1.10 – Зображення рівняння перетворення від форми (3D) до зображення (2D)

Ця модель називається ідеальною моделлю камери-обскури. Завдяки цій роботі для простоти будуть використовуватися однорідні координати, які широко використовуються в проективній геометрії. Це означає, що координати доповнюються додатковою координатою, яка містить коефіцієнт масштабування для цієї точки [9]. Отже, двовимірною точкою в декартових координатах (x; y) буде (u; v; w) = (wx; wy; w) в однорідному поданні.

Крім того, рівняння теореми перехоплення пов'язують точку реального світу з точкою на зображенні камери, використовуючи систему координат камери. Для обробки зображень важливо скористатися координатами зображення в пікселях, які починаються з (0; 0) у верхньому лівому куті зображення. Для переведення центру оптичної осі в центр координатного кадру зображення буде встановлено набір о (o_x; o_p) для координат x та y.

Разом, рівняння, яке перетворює реальну точку 3D світу X у 2D точку зображення x, буде таким, як забражено на рисунку 1.11, де K - матриця камери, і це просто матриця ідентичності 3x4, яка є важливою для моделювання жорстких рухів тіла камери. є коефіцієнтом масштабування для точки зображення. Отже, наразі є рівняння, яке відображає точку реального світу до 2D-точки на зображенні камери.

Для того, щоб зробити це обчислення (точка зображення формується точкою в реальному світі), матриця камери K, яка містить фокусну відстань (f) та оптичну центральну точку (o_x; o_p) камери, потрібно. Більше того, потрібно

з’ясувати жорстке перетворення тіла = (Rjt), яке описує, як влаштована система камер щодо світової системи.

$$\lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & o_x \\ 0 & f & o_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \Leftrightarrow \lambda \cdot \mathbf{x} = \mathbf{K} \cdot \boldsymbol{\Upsilon} \cdot \mathbf{X}$$

Рисунок 1.11 – Зображення рівняння перетворення реальної точки 3D світу X у 2D точку зображення

У цьому контексті записи матриці камери називаються власними параметрами, оскільки вони описують специфічні значення всередині певної камери, тоді як значення (Rjt) називаються зовнішні параметри, оскільки вони визначають зовнішню позу камери щодо світового кадру.

Для оцінки цих внутрішніх та зовнішніх параметрів проводиться калібрування камери виконується [8]. Для цього робляться фотографії відомого площинного об’єкта, який називається калібрувальним обладнанням. У більшості випадків для калібрування використовується шахова дошка. Алгоритм виявлення кутів відбиває всі кути чорно-білих квадратів шашки на зображенні. Це генерує набір точок зображення x_i що відповідає куточкам шашки. Крім того, квадратний розмір шашки повинен бути відомий, тому набір реальних координат усіх кутів шашки X_iможна побудувати. Традиційно нижній лівий кут шашки визначається як центр світової системи координат. У цьому випадку всі кутові точки X_i буде зображено як (X_i; Y_i; 0).

Маючи два набори точок зображення x_i та відповідні точки реального світу X_i, як внутрішні, так і зовнішні параметри можна розрахувати [10]. Це робиться шляхом вставки цих значень на зображення з рівнянням на рисунку 1.11 і вирішити для всіх невідомих параметрів.

У всіх попередніх розрахунках була використана ідеальна модель камери-обскури. Насправді в центрі проекції камери не є отвір, а тонка лінза. Ця лінза

призводить до (головним чином радіального) спотворення зображення [11]. Існують різні типи спотворень, які можуть модифікувати зображення. Найважливішими ефектами називають спотворення бочок і подушечок через їх характерну форму. Рисунок 1.12 ілюструє, як це виглядає.

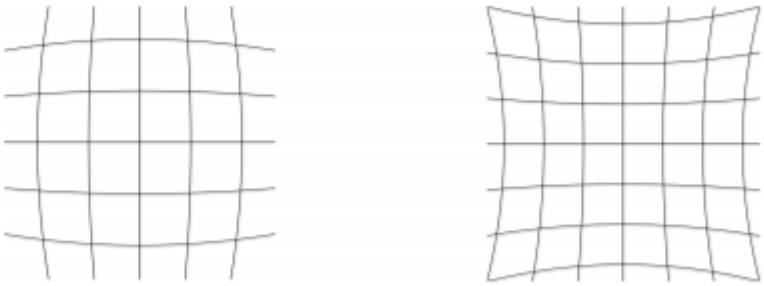

Рисунок 1.12 – Викривлення стовбура та подушечки

Особливо по краях зображення ефект спотворення досить великий. Таким чином, це також впливає на формування зображення, і тому його потрібно включати в розрахунки, що стосуються точок зображення та реального світу. Таким чином введено коефіцієнти спотворення $k = (k_1; k_2)^T$ що моделює ефект спотворення, як показано нижче на рисунку 1.13.

$$\breve{x} = x + x[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2]$$
$$\breve{y} = y + y[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2]$$

Рисунок 1.13 – Зображення рівнянь з розрахунками коефіцієнтів спотворення

У цьому рівнянні (x; y) - це реальні координати зображення зі спотвореннями, тоді як (x; y) - ідеальна точка зображення (без спотворень). Ці коефіцієнти (k_1; k_2) також належать до параметрів вбудованої камери і також обчислюються в процесі калібрування.

Нелінійна оптимізація за алгоритмом Левенберга-Маркварда знаходить матрицю K-епохи, коефіцієнти спотворення k і жорсткий рух тіла (Rjt), що мінімізує так звану помилку репроекції [12].

$$\sum_{i=0}^{n}\sum_{j=0}^{m}\|m_{ij}-\breve{m}(K,k,R_i,t_i,M_j)\|^2$$

Рисунок 1.14 – Зображення рівняння проекції

Отже, цей алгоритм оцінює ті внутрішні та зовнішні параметри, які найкраще відповідають усім зробленим зображенням. Чим більше зображень (з різних положень та кутів) буде використано для калібрувальної установки, тим кращі результати будуть отримані завдяки цій оптимізації. Крім того, коли коефіцієнти спотворення визначені, можна усунути спотворення знімків, зроблених камерою. Після цього ефект спотворення може бути усунутий для подальших розрахунків.

Також існує калібрування “Рука-Око”. Калібрування "Рука-Око" працює наступним чином: Калібрувальна установка (наприклад, шашка) розміщується в кімнаті та визначає центр світової системи координат. Тоді робот розміщений десь навколо цієї калібрувальної установки, так що камера спрямована до неї. Після цього вимірюється точна поза (відстань у (x; y; z) та орієнтація центру робота відносно світової координатної рамки. Це перетворення буде називатися $(Rjt)^{RW}$((Rjt) від світу до робота). Це вимірювання проводиться вручну і може спричинити деякі неточності протягом декількох міліметрів або сантиметрів. Тому ми виміряли багато різних пози роботів для поліпшення результатів шляхом обчислення середнього значення пізніше. Якщо і робот, і калібрувальна установка розміщені на рівному грунті, вимірювання, які потрібно зробити, зменшуються з шести параметрів (три переклади та три обертання), лише до трьох параметрів (перенесення x та y та обертання навколо осі z).

Вимірявши $(Rjt)^{RW}$, відносна поза від камери до світового кадру $(Rjt)^{WC}$ потрібен наступний. Це можна легко оцінити, сфотографувавши калібрувальну установку за допомогою цієї камери. Тоді програма калібрування може бути використана для обчислення зовнішніх параметрів (попередньо відкаліброваної) камери щодо світового кадру, який є калібрувальним обладнанням. Тепер маючи $(Rjt)^{RW}$ і $(Rjt)^{WC}$, зв'язок між камерою та роботом $(Rjt)^{RC}$ виводиться простим множенням, яке можна побачити на рисунку 1.15.

$$\begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}_C^R = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}_C^W \cdot \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}_W^R$$

Рисунок 1.15 – Зображення рівняння співвідношення звязку між камерою та роботом

Рисунок 1.16 показує, як працюють ці жорсткі перетворення тіла.

Як вже згадувалося раніше, ручне вимірювання пози робочого центру відносно світового кадру $(Rjt)^{RW}$ не дуже точно. Тоді як вимірювання пози камери, яке проводиться шляхом обчислення зовнішніх параметрів із зображення, є досить точним. Тому, щоб отримати кращі результати для калібрування «Рука-око», ми зробили різні вимірювання в різних позах робота. Для кожної пози було зроблено знімок калібрувальної установки, який веде до різної камери до перетворень роботів. Обчислення середнього віку цих ве $(Rjt)^{RC}$ значення, це може дати кращий результат щодо похибок вимірювання вручну.

Зовнішня камера робить зображення обох шашок. Потім поза зовнішньої камери щодо 2-ї шашки $(Rjt)^{2\ \text{координата}}$ і до світової каркасної шашки $(Rjt)^{Wext}$ обчислюється. З цими перетвореннями відома поза між двома шашками.

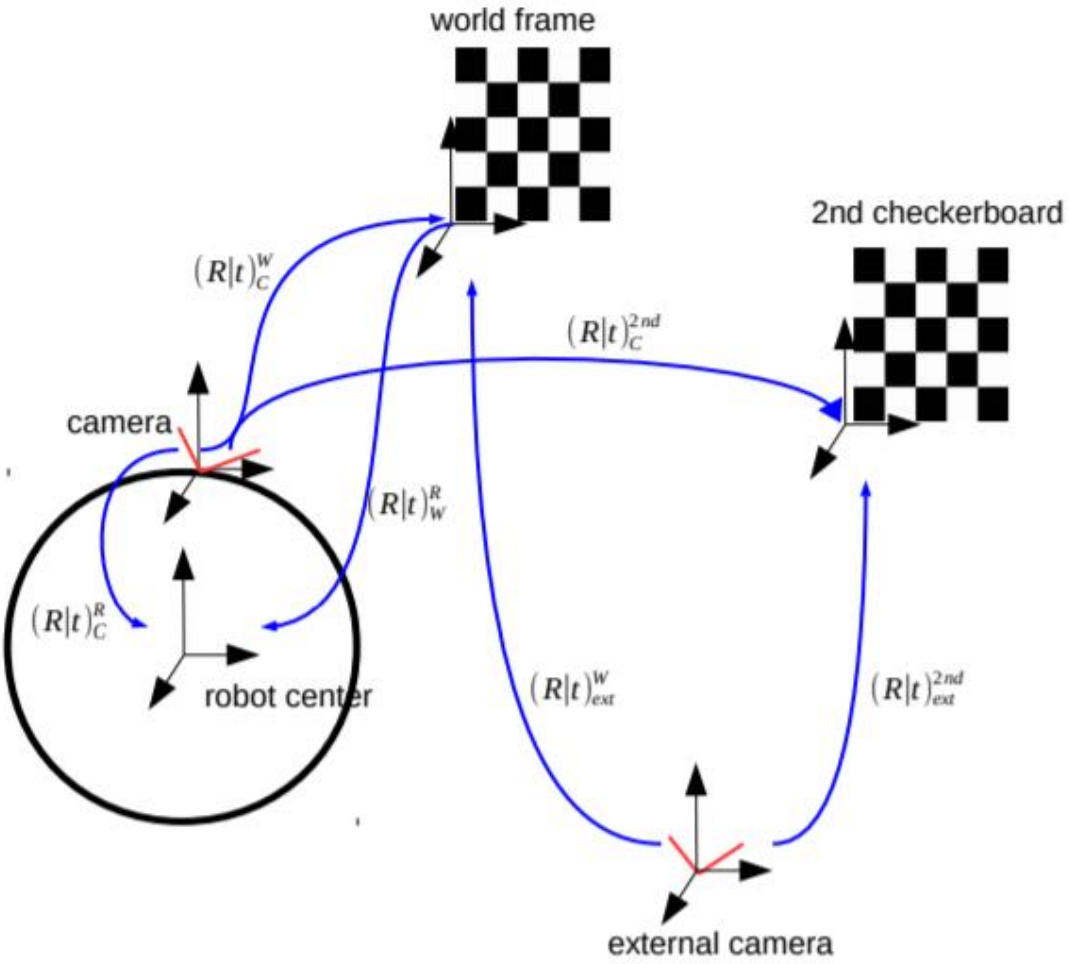

Рисунок 1.16 – Трансформації, які використовуються для калібрування "Рука-Око"

Для кожної з пози роботи робота камера-робот також робить знімок 2-ї шашки. Отже, перетворення $(Rjt)^{2Cnd}$ можна оцінити за допомогою цього зображення. Маючи всі ці взаємні позиції один до одного, можна розрахувати бажану позу між камерою та роботом $(Rjt)^{RC}$. У цьому випадку ця поза визначається за допомогою додаткової допомоги 2-ї шашки, а не лише безпосередньо над світовою рамковою шашкою. Відповідно до рисунка 1.15 цей розрахунок можна представити у вигляді, зображеному на рисунку 1.17.

Фотографуючи різні позиції 2-ї шашки для кожної з різних позицій робота, які були виміряні, є загалом 5x5 = 25 оцінок для $(Rjt)^{RC}$. Отримавши середнє значення всіх цих 25 значень, це має дати цілком хороший результат для калібрування «Рука-Око».

$$\begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}_C^R = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}_C^{2nd} \cdot \left(\begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}_{ext}^{2nd} \right)^{-1} \cdot \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}_{ext}^{W} \cdot \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}_W^R$$

Рисунок 1.17 – Зображення рівняння розрахунку пози між камерою та роботом

1.3 Геометрія стереовиглядів

При зйомці сцени, 3D-точка світу X_0 проектується в 2D точку x_0 на зображенні. Це відношення задано формулою, яка зображена на рисунку 1.18.

$$\lambda \cdot \mathbf{x}_0 = \mathbf{K} \cdot (\mathbf{R}|\mathbf{t}) \cdot \mathbf{X}_0.$$

Рисунок 1.18 – Зображення формули проекції 3D-точки світу X_0 в 2D точку x_0 на зображенні

Завдяки перетворенню з 3D у 2D інформація про глибину сцени втрачається. Крім того, лише з одним зображенням неможливо відновити масштабний коефіцієнт. Ось чому неможливо розрахувати реальну точку 3D світу, маючи лише одне зображення сцени. Однак важливо знати про структуру 3D сцени для локалізації робота в невідомому середовищі. На щастя, можна відновити цю інформацію про глибину з декількох зображень, які будуть описані в цьому розділі.

Як людина, ми можемо бачити в 3D, що означає, що ми можемо оцінити глибину навколишнього світу. Отже, око також створить два зображення однієї сцени з однаковими 3D-точками світу, але з різних поглядів. Знайшовши точку на зображенні сітківки, наш мозок може відновити глибину цієї точки у світі. Точно так само працює і при використанні камери (або стереокамер). На рисункк 1.19 – співвідношення двох геометричних оглядів.

Знаючи точну відстань та орієнтацію між двома знімками камери (жорстке перетворення тіла (Rjt)), зображення показує x_1 та x_2, які відповідають тому самому реальному світу.

У випадку, коли два розглянуті зображення зроблені камерою, (Rjt) завжди буде відомий. Інший випадок геометрії двох видів - це наявність двох зображень, зроблених однією і тією ж камерою. Під час переміщення однієї камери це також забезпечує два (або більше) зображення однієї сцени. Зараз проблема полягає в тому, що невідомо, як змінилася поза камери протягом двох знімків. Однак може бути корисним знати цей послідовний рух. Для відновлення цього (Rjt) з двох зображень береться до уваги епіполярна геометрія.

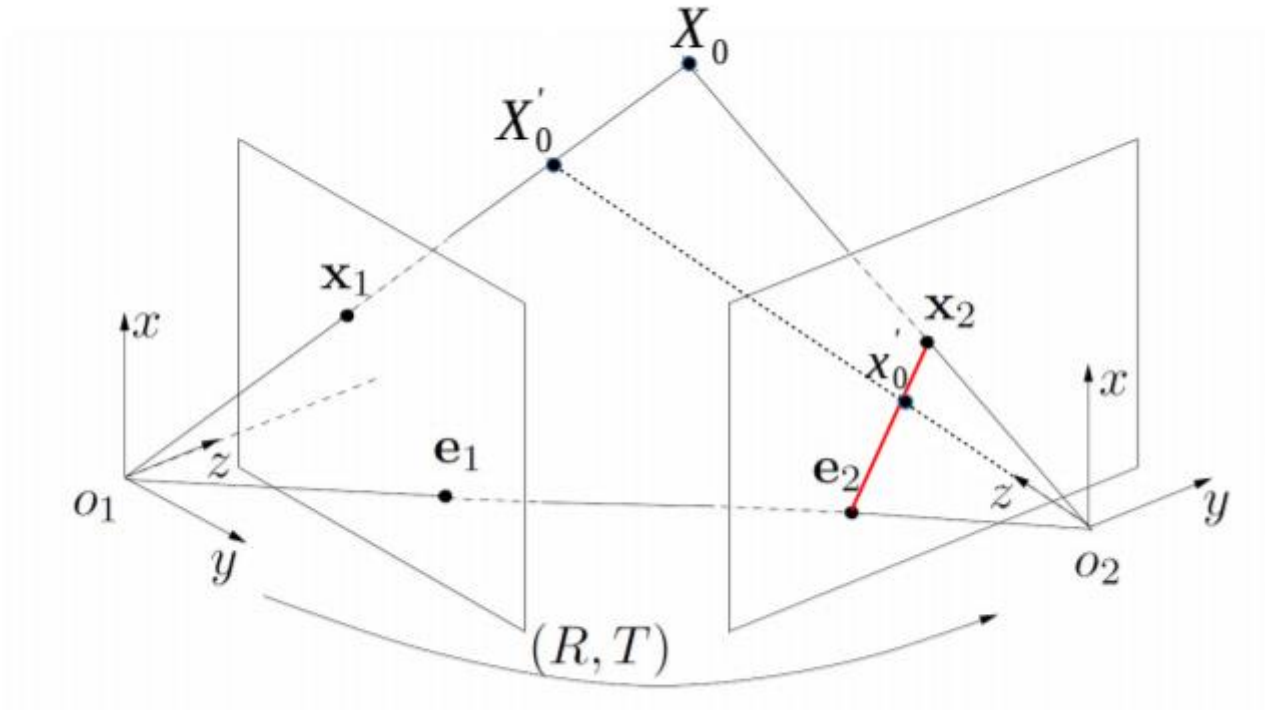

Рисунок 1.19 – Геометрія двох видів

Точку X_0 можна оцінити за допомогою тріангуляції з цією інформацією. Існує кілька способів триангуляції в комп'ютерному зорі. Хоча основний принцип полягає у вирішенні точки перетину двох ліній, які з'єднують центр

камери o_1 і o_2 з точками зображення x_1 відповідно x_2. Отже, за допомогою цієї техніки можна обчислити 3D координати точки X_0. Ось як можна отримати дані про глибину сцени, використовуючи камеру. Єдине, що потрібно знати раніше, це відносна поза (Rjt) камери та знімок. Це можна знайти, виконавши попереднє калібрування.

1.4 Операційні системи роботів

Для цієї роботи також буде використовуватися система ROS. Операційна система робота (ROS) є вигідною структурою, особливо для розробки програмного забезпечення для роботів. Він може бути використаний для промисловості роботів шляхом модульного програмування. Доступно багато пакетів, що містять певні функціональні можливості для роботів, а також драйвери обладнання. Ці пакети часто складаються з одного або декількох так званих вузлів (nodes).

Вузли це щось на зразок виконуваної програми, яка виконує конкретне завдання. Уже є пакет для Robotino, який містить багато корисних вузлів. Наприклад, ці вузли реалізують, що датчики робота можуть зчитуватися або що роботом можна керувати за допомогою клавіатури або джойстика. Структуру системи ROS, яка використана в цій роботі (рис 1.20).

VrMagic, imgProc, локалізація, управління та robotino_node - це вузли, які використовуються в нашій системі. Вони можуть спілкуватися між собою за допомогою ROS-повідомлень. Існують різні типи ROS-повідомлень. Наприклад, це можуть бути прості рядки, інформація про позицію (набір значень (R|t)) або також відеопотоки. Повідомлення публікуються через "Topic". Вузол, який надсилає інформацію, зображується як "Publisher", а одержувач позначається як "Subscriber". Вузли позначаються колом, а теми оформлюються прямокутником.

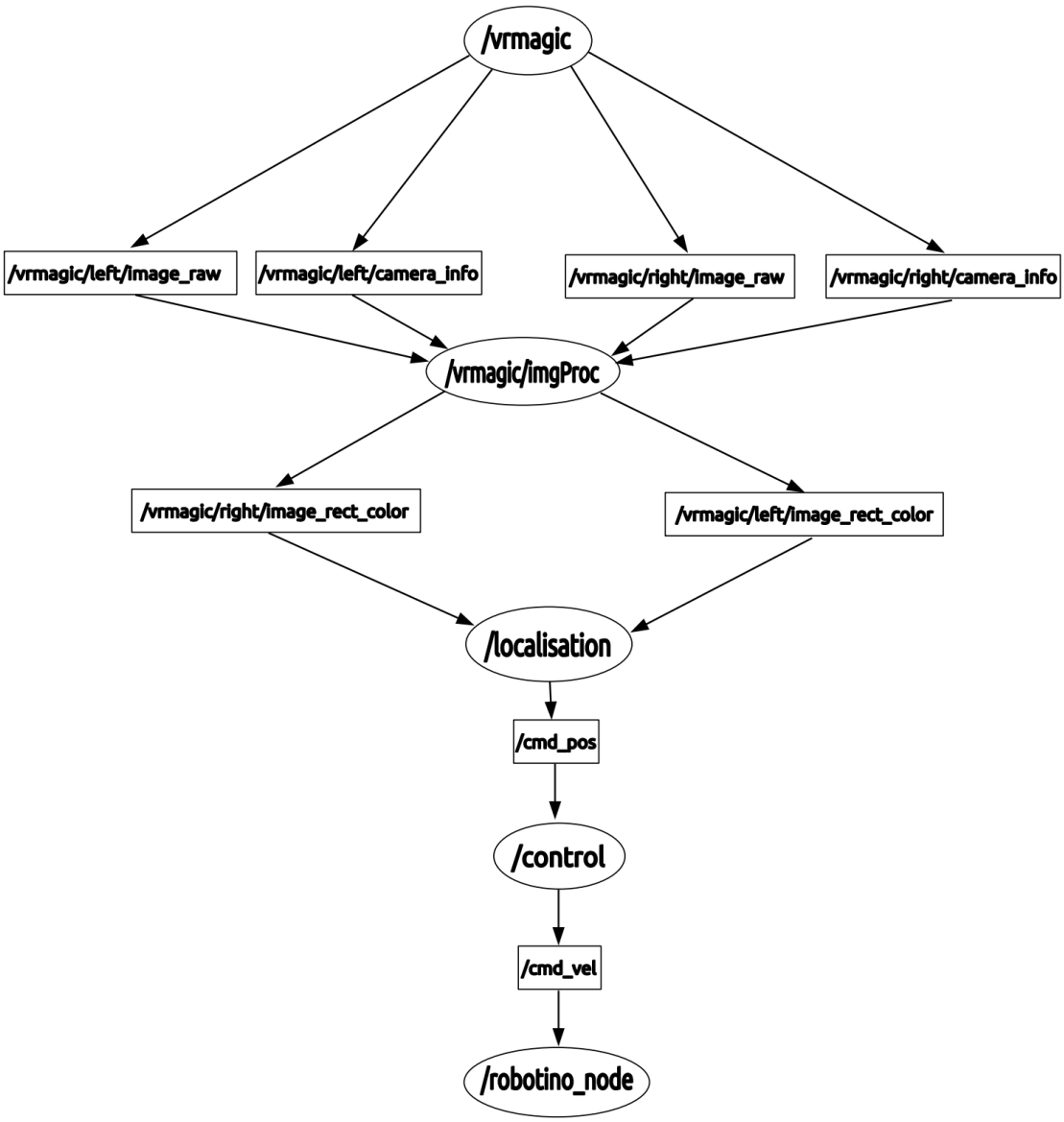

Рисунок 1.20 – Структура системи ROS

VrMagic, imgProc, локалізація, управління та robotino_node - це вузли, які використовуються в нашій системі. Вони можуть спілкуватися між собою за допомогою ROS-повідомлень. Існують різні типи ROS-повідомлень. Наприклад, це можуть бути прості рядки, інформація про позицію (набір значень (R|t)) або також відеопотоки. Повідомлення публікуються через "Topic". Вузол, який надсилає інформацію, зображується як "Publisher", а одержувач позначається як "Subscriber". Вузли позначаються колом, а теми оформлюються прямокутником.

Перший вузол програми це vrmagic. Це драйвер для стереокамери. Він отримує дані зображення та публікує необроблене зображення на вузлі imgProc.

Крім того, він надсилає camera_info, що містить дані про калібрування стереокамери. Отож imgProc отримує необроблений відеопотік, а потім виконує попередню обробку із зображеннями. Оскільки він також має дані калібрування, він спотворює зображення і після цього виправляє для подальших розрахунків. Вузол imgProc вже наданий спільнотою ROS. Після обробки вузол передає виправлені зображення (image_rect_colour) до локалізації. Цей вузол є центральною частиною цієї роботи, оскільки він містить імплементацію алгоритму зорової одометрії. Він отримує потік виправлених зображень, а потім обчислює поточне місцезнаходження робота з цією інформацією. Він також виконує адаптивний пошук функцій. Після визначення інформації про позу (cmd_pos), вона публікується в елементі управління вузлом. Таким чином, контроль знає про положення та орієнтацію робота. Більше того, пункт призначення робота встановлюється також у цьому вузлі. Отже, він обчислює бажані швидкості x та y для робота, який переміщує його до цільового місця. Потім ці команди швидкості (cmd_vel) публікуються в robotino_node, який є вузлом взаємодії з Robotino. Він підписується на команди швидкості та запускає робота із заданими швидкостями.

Поки Роботіно управляється за допомогою ROS, більшість справжніх процесів обробки зображень та алгоритмів комп'ютерного зору запрограмовані в коді C++. Пакет ROS "roscpp" реалізує код C++ у ROS-вузлах, тому його можна використовувати з іншими ROS-вузлами. Таким чином, через ROS можливо керувати роботом за допомогою програми C++. Причина, чому мова програмування C++ був обраний для реалізації в тому, що окрім хорошої швидкості обчислень, за цим стоїть велика спільнота, яка створює велику підтримку та пропонує корисні бібліотеки з відкритим кодом для комп'ютерного зору. Однією з таких бібліотек є OpenCV, що в основному використовується для цієї роботи.

OpenCV містить безліч функцій програмування та алгоритми для застосування комп'ютерного зору. Це включає завантаження зображень та

відеозаписів з фотокамер або камер, обробку зображень (наприклад, зміна колірного простору, вилучення цікавих областей тощо).

1.5 Принципи програмного керування робототехнічними системами

За останні два десятиліття дослідники дослідили кілька архітектурних парадигми їх роботизованих систем. Вони, як правило, класифікуються як дорадчі, реактивні, засновані на поведінці та гібридні архітектури. Кожна архітектурна парадигма визначає, як розділена загальна система управління роботом, набір послідовних або одночасних заходів управління, починаючи від управління на низькому рівні двигунів і датчики високого рівня можливостей, таких як планування та розпізнавання об’єктів, і як ці заходи об'єднуються та координуються.

Типовими видами діяльності, об’єднаними архітектурою управління роботами, є наступні:

– отримання та інтерпретація вхідних даних від датчиків;

– керування рухом усіх рухомих частин;

– відображення стану робота та середовища;

– управління всіма видами ресурсів (живлення, центральний процесор, пам’ять, пристрої);

– планування майбутньої діяльності;

– реагування на непередбачувані події.

На рисунках 1.21 і 1.22 зображені дві архітектури управління. Обидва вказують на наявність датчика модулі, що подають модулі управління сенсорними даними.

Під час розробки системи роботів архітектура управління роботом перетворюється на архітектуру програмного забезпечення робота.

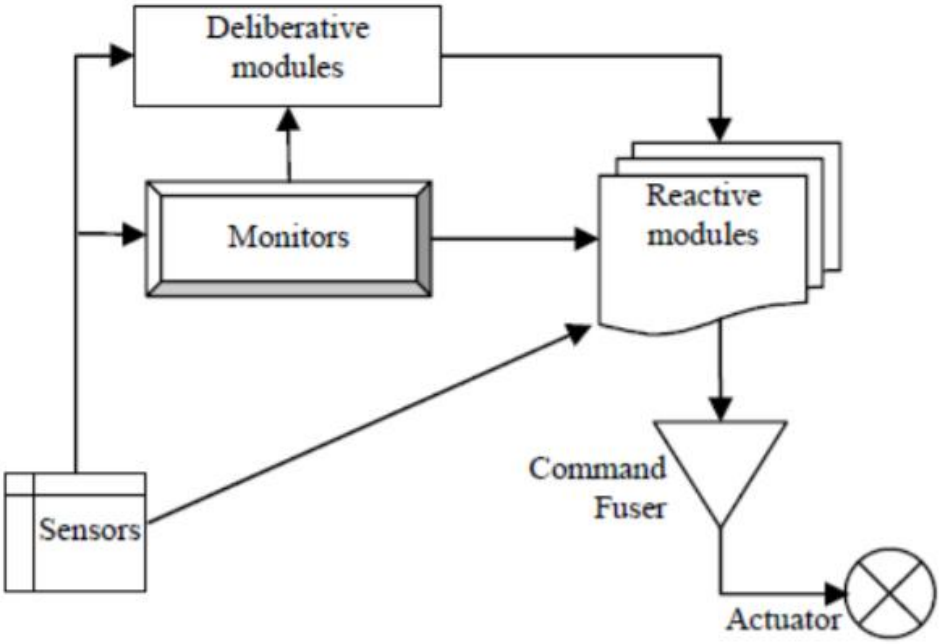

Рисунок 1.21 – Архітектура управління роботом (варіант 1)

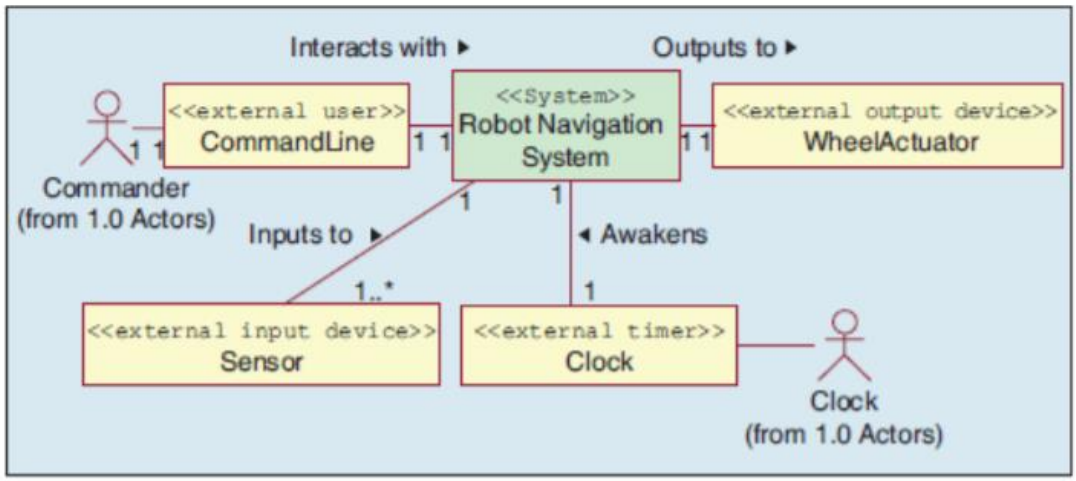

Рисунок 1.22 – Архітектура управління роботом (варіант 2)

В рамках програмної інженерії архітектура програмного забезпечення зазвичай визначається як структура або структури системи, яка містить програмні компоненти, зовні видимі властивості цих компонентів та взаємозв'язок між ними. Тут компоненти є одиницями реалізації та представляють кодовий засіб спосіб розгляду системи. Таким чином, архітектура програмного забезпечення робота описує розкладання системи управління роботом на сукупність програмних компонентів, інкапсуляція функціональних

можливостей та управління діяльністю в компоненти та потік даних та контрольної інформації між компонентами.

Дизайн або вибір архітектура програмного забезпечення спеціально враховує нефункціональні вимоги, а роботизована програмна система (ремонтопридатність, портативність, сумісність, масштабованість), тобто це ті вимоги, які характеризують якість програмного забезпечення та дозволяють повторне використання програмного забезпечення. Вони визначають організацію програмної системи як сукупність взаємопов’язаних компонентів, що реалізують певні алгоритми, наприклад, для планування завдань, навігація, пересування, оцінка пози тощо.

Відображення завдань робота на компоненти, тобто алгоритми, структури даних, синхронізація, а механізми зв'язку упаковані разом і визначення взаємодії компонентів є важливими етапами проектування, оскільки вони дуже важливі впливають на багаторазове використання цих компонентів. Таким чином, в той час як обидві архітектури управління вказують, що модулі датчиків забезпечують управління модулі із сенсорними даними, дві архітектури програмного забезпечення відрізняються взаємодією схеми (push / pull) між компонентом датчика та компонентами управління. Як наслідок, компонент датчика не можна використовувати багато разів додатків. Проблема походить від того, що два компоненти датчика поєднувати два різні аспекти: функціональність компонента (тобто надання сенсорних даних) і схема взаємодії (push / pull). Тоді як перший є стабільною власністю компонент датчика і, отже, повинен бути багаторазовим, останній є змінною вимогою програми і, отже, має бути гнучким.

Основними властивостями програмного керування є:

– багаторазове використання програмних компонентів;

– відкритість системи;

– гнучкість системи;

– гнучкість у теорії прийняття рішень;

– гнучкість у виробничих системах.

1.6 Висновки до першого розділу

У першому розділі був проведений аналіз засобів моделювання та керування роботехнічними системами, а саме було розглянуто сучасні типи мобільних роботів та їх застосування, формування зображення та калібрування камери, геометрія стереотипів, операційні системи роботів та були розглянуті принципи програмного керування робототехніми системами.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМНОЇ БІБЛІОТЕКИ ДЛЯ МОДЕЛЮВАННЯ ФУНКЦІЙ СЕНСОРНОЇ СИСТЕМИ

2.1 Розробка алгоритму адаптивної візуальної одометрії

Цей розділ вкаже на реалізацію власне алгоритму локалізації. Перший крок це спотворення та виправлення вихідних зображень. Це робиться ROS-вузлом imgProc. Потім ці попередньо оброблені зображення передаються у вузол локалізації, де обчислюється поза робота. Перевага цього розділу полягає в тому, що обидва вузли (imgProc та локалізація) можуть працювати паралельно. Це економить деякий час для алгоритму візуальної одометрії, оскільки йому більше не потрібно виправляти зображення. Подальшими кроками обчислення вузла локалізації є наступні:

а) беруться виправлені та неспотворені зображення з камери;

б) перетворення зображення у відтінки сірого для подальшої обробки;

в) визначення функції ORB у зображення та виконання дескрипторів функцій;

г) зіставлення очкових рис зі збігом грубої, використовуючи відстань удару та перехресну перевірку;

ґ) вибираються лише найкращі збіги, виконавши ці три кроки:

1) оскільки зображення виправляються, вибираються лише ті збіги, які лежать на одній координаті у (3 пікселі);

2) вибираються лише сірники з відстанню нижче порогового значення;

3) виконується RANSAC, щоб знайти основну матрицю зображення та видалити віддалені точки;

д) трикутник 3D-точок із набору відповідних точок об’єкта за допомогою лінійного методу трикутування;

е) відстежується вибраний набір точок об’єкта на наступному зображенні за допомогою трекера Lucas-Kanade;

є) оцінка (Rjt) камери з алгоритмом RANSAC;

ж) перевіряється, чи є цей (Rjt) достатньо великим, інакше відкиньте його (вибір ключового кадру);

з) обчислюється рух робота за рухом камери, застосувавши калібрування рук-очей;

и) об'єднується розрахований (Rjt), щоб отримати загальну позу;

і) створюються точки зображення на зображеннях для візуалізації;

ї) генерується інформація про позу та знову запускається цикл для наступного кадру зображення.

Алгоритм адаптивної візуальної одометрії виконується для самостійної локалізації мобільних роботів зображено на рисунку 2.1.

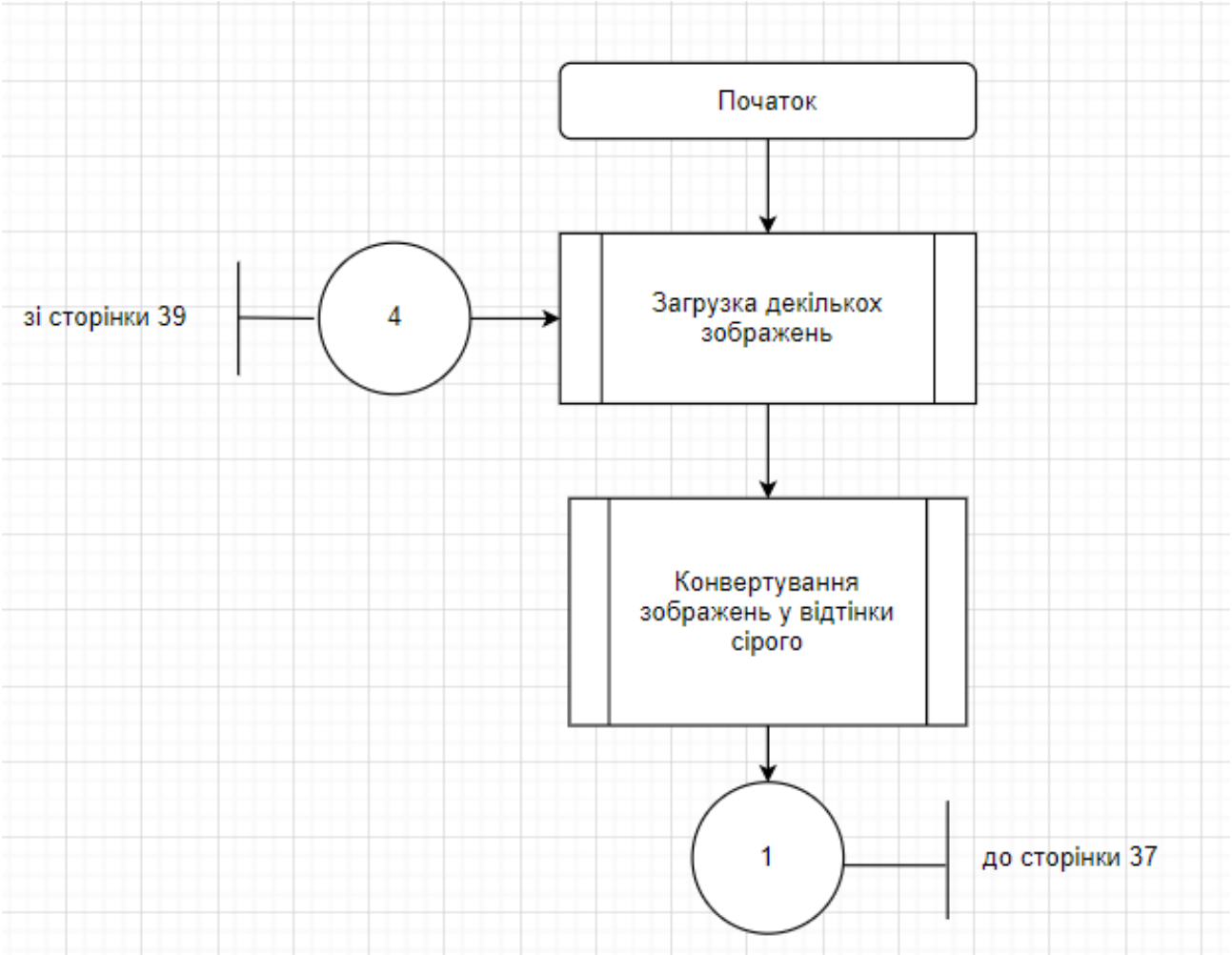

Рисунок 2.1 – Схема алгоритму адаптивної візуальної одометрії

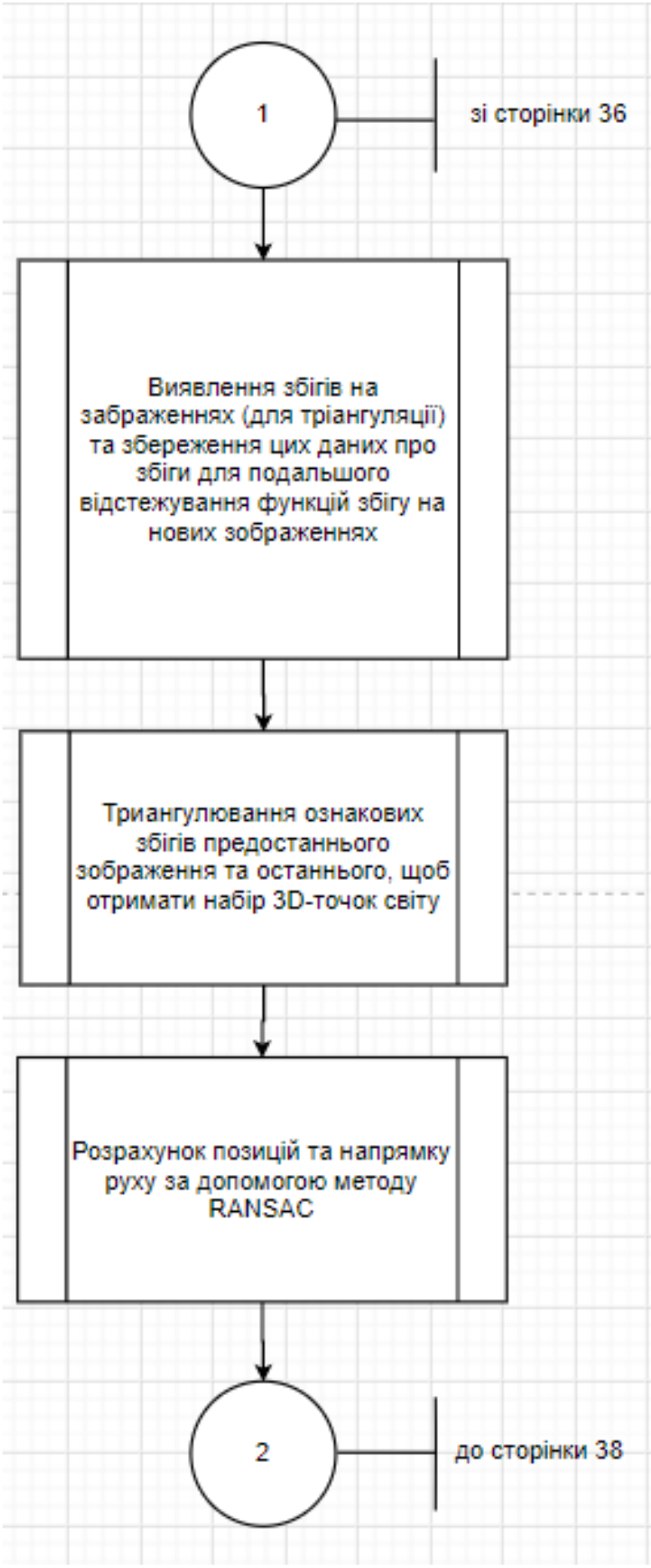

Рисунок 2.1, аркуш 2

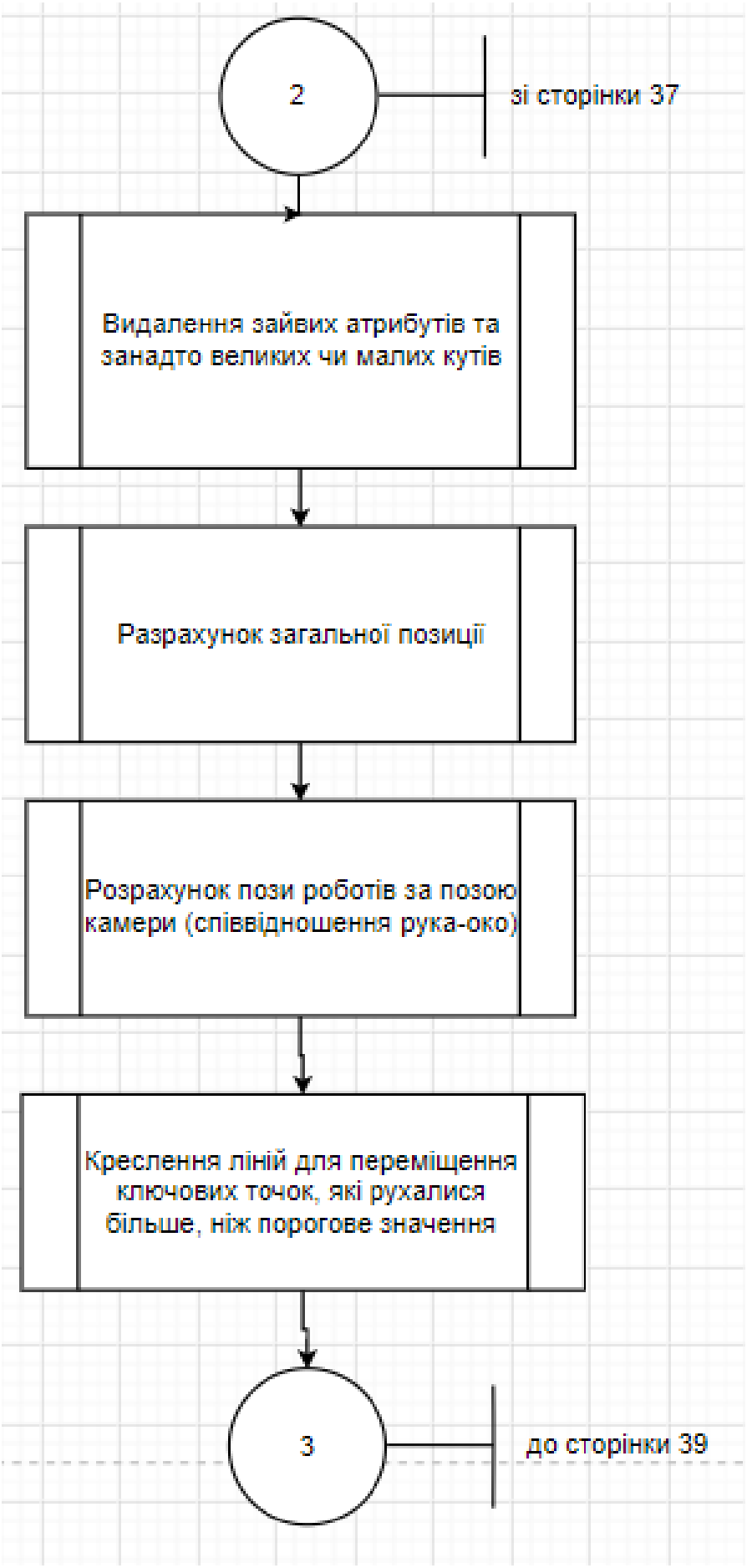

Рисунок 2.1, аркуш 3

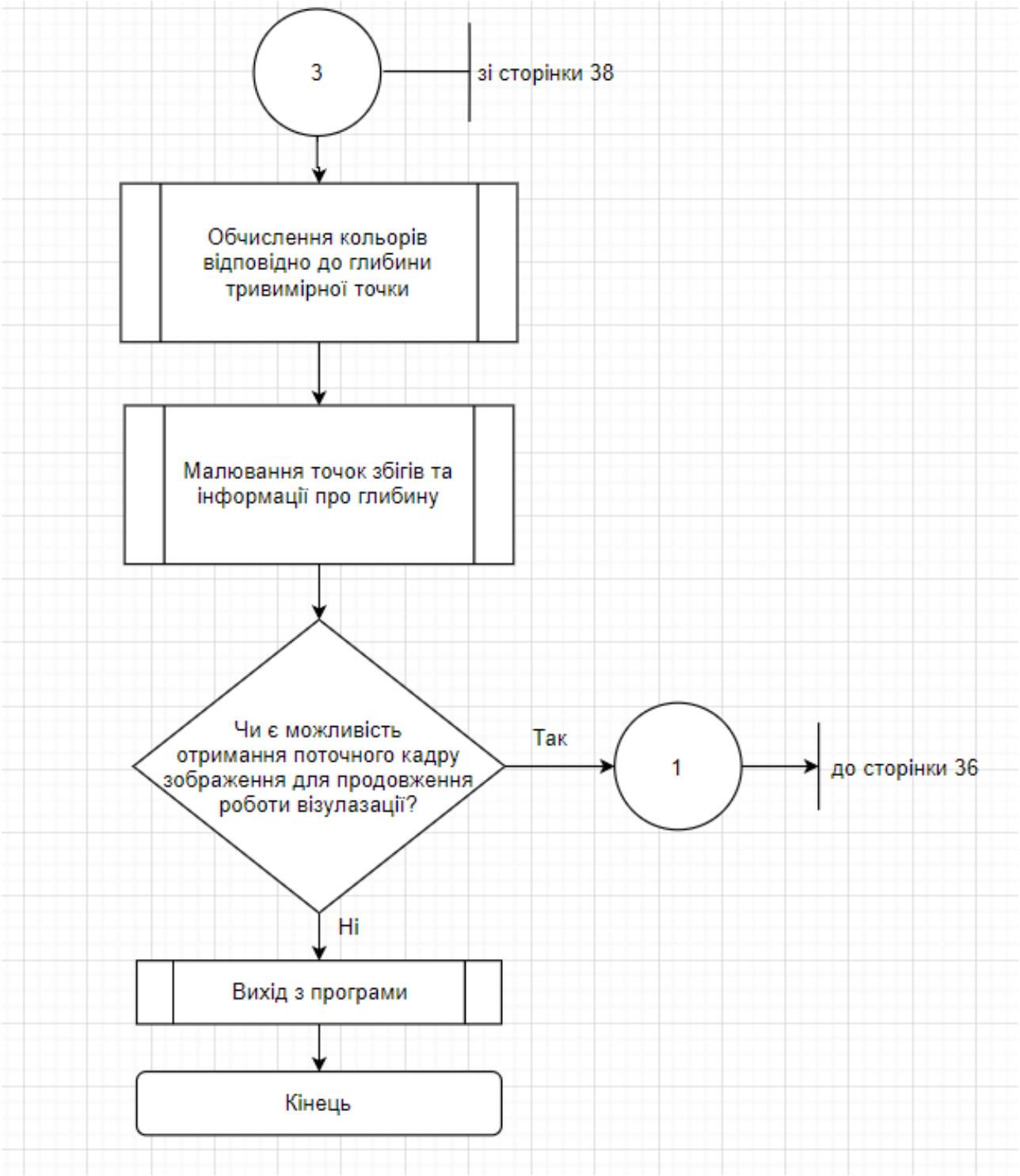

Рисунок 2.1, аркуш 4

Візуалізація цього вузла показує зображення камери. На першому зображенні намальовані всі точки функцій, які можна було б зіставити. Крім того, він показує лінію між розташуванням точки об’єкта в останньому та поточному кадрі. Це ілюструє рух камери. Наступне зображення камери відображається в вікні. Окремі точки також втягуються в зображення. У цьому

випадку вони забарвлені відповідно до їх глибини в сцені, що зображено на рисунку 2.2.

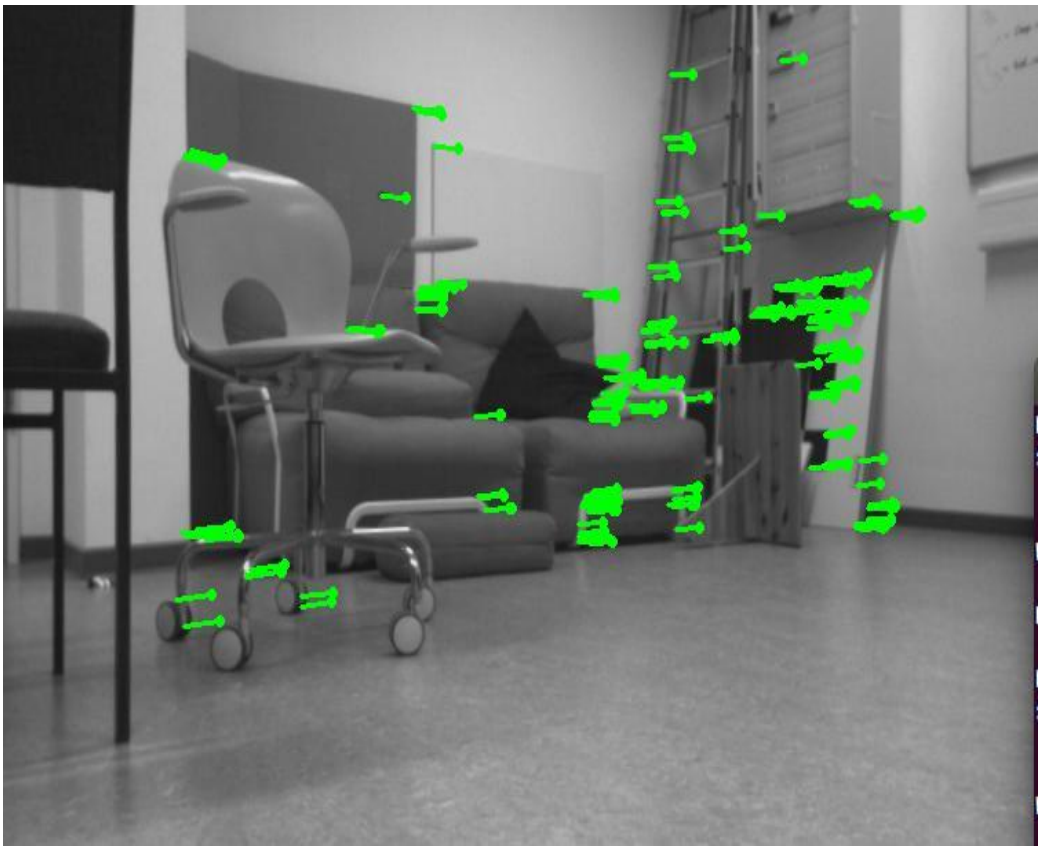

Рисунок 2.2 – Візуалізація зображення камери

Вузол локалізації також виконує адаптивний пошук функцій. Отже, він порівнює кількість відповідних точок об’єкта із заданим пороговим значенням. Якщо такі є на зображенні недостатньо точок об'єкта, вузол публікує команду обертати Robotino для додавання більше функцій. Адже опублікована інформація про позу обробляється вузлом управління. Цей вузол обчислює бажані швидкості x та y.

2.2 Розробка адаптивної стратегії вибору функцій

Попередні розділи проілюстрували, як оцінити положення робота за допомогою візуальної одометрії. Метою даної роботи є реалізація адаптивного

вибору функцій з метою покращення результатів алгоритму VO. Це означає, що робот повинен стикатися з різними секторами навколишнього середовища, залежно від його поточного положення. Деякі регіони надають більше функцій, а деякі - менше. Тож робот повинен шукати ті області з найбільшою кількістю точок, щоб отримати найкращу продуктивність.

Однією з найбільших проблем у всіх програмах комп'ютерного зору є те, що на зображенні не так багато характерних точок. Наприклад, дивлячись на звичайну білу стіну, алгоритм VO не може знаходити значущих точок, які збігаються та орієнтуються. Ось чому слід уникати ситуацій, коли поле зору не містить достатньо точок особливостей. Робот, який використовується для цієї роботи, Robotino, має всеспрямовану систему керування, що полегшує прямий рух у всіх напрямках та обертання на місці. Ці властивості використовуються для областей з найбільшою кількістю точок. Поки робот рухається до певного місця в кімнаті, він може обертатися навколо власної осі. Всякий раз, коли кількість корисних функцій, що перебувають у полі зору, опускається нижче певного порогу, робот обертається сам, поки в напрямку камери не буде достатньо точок функцій. Це простий і дуже динамічний метод, оскільки пошук функції виконується під час приводу. Тож на це завдання не витрачається часу.

Проблема цього динамічного методу і полягає в тому, що робот обертається лише тоді, коли точки об'єкта опускаються нижче порога. Це лише гарантує, що робот опиниться в розумній зоні, інакше він обертається. Це не гарантує, що робот звернений до найкращої області з найбільшою кількістю точок. Тому вводиться другий підхід. Цей більш статичний метод починається з початкового пошуку об’єктів на початку. Спочатку робот обертається на 90° ліворуч, а потім 90°праворуч, щоб просканувати всю сцену, яка знаходиться перед роботом. Він збирає дані про те, скільки точок об’єкта при якому обертанні кут. Після закінчення пошуку відомий напрямок із більшістю функцій. Поки цей початковий пошук особливостей не завершений, робот постійно стоїть на одному місці, просто обертаючись навколо власної осі. Це забороняє вносити помилки перекладу під час сканування. Після того, як робот

зіткнувся з областю з більшістю точок функцій, робот починає рухатися до цільового місця. Цей статичний пошук об’єкта слід повторити через деякий час, або якщо кількість точок об’єкта в поточному напрямку опускається нижче певного значення. Однією з проблем, яка може виникнути під час пошуку функцій, є те, що робот обертається в зону, в якій немає жодних функцій. Щоб уникнути такої ситуації, робот обертається лише настільки, що в його полі зору завжди залишається достатньо функцій.

Отже, ці два підходи адаптивного вибору функцій гарантують, що робот не буде втрачати сліди точок об’єкта під час руху. Це важливе розширення алгоритму VO, щоб зробити його більш надійним для роботи в невідомих середовищах.

2.3 Розробка UML-діаграм функціонування програмної бібліотеки

UML – уніфікована мова моделювання (Unified Modeling Language) – це система позначень, яку можна застосовувати для об'єктно-орієнтованого аналізу і проектування. Його можна використовувати для візуалізації, специфікації, конструювання та документування програмних систем [13].

Поведінку системи (тобто функціональність, яку вона забезпечує) описують за допомогою функціональної моделі, яка відображає системні прецеденти, системне оточення (дійових осіб) і зв'язку між ними.

Діаграма варіантів використання (ВВ) – це діаграма, на якій зображуються відносини між акторами і ВВ. За допомогою цієї діаграми можна:

– визначити загальні межі і контекст модельованої предметної області на початкових етапах проектування системи;

– сформулювати загальні вимоги до функціонального поведінки проектованої системи;

– розробити вихідну концептуальну модель системи для її подальшої деталізації у формі логічних і фізичних моделей;

– підготувати вихідну документацію для взаємодії розробників системи з її замовниками і користувачами.

Діаграма прецедентів містить у собі наступні елементи:

– актором (дійова особа) називається будь-який об'єкт, суб'єкт або система, що взаємодіє з моделюючою системою ззовні. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка служить джерелом впливу на моделюючу систему так, як визначить розробник;

– ВВ (прецедент) – опис безлічі послідовності дій (включаючи варіанти), що виконуються системою для того, щоб актор міг отримати певний результат [13];

– включення (include) говорить про те, що вихідний прецедент явно включає в себе поведінку цільового;

– розширення (extend) показує, що цільовий прецедент розширює поведінку вихідного.

Прецеденти та актори – це відображення вимог до системи, вони показують, хто і для чого буде використовувати майбутню систему [13].

Провівши дослідження в сфері предметної області, були визначені ВВ і основні дійові особи для автоматизованої системи резервного копіювання. Всі варіанти ВВ представлено на рисунку 2.3.

Одним з основних елементів мови моделювання є діаграми, які є графічним представленням набору елементів, найчастіше зображеного у вигляді зв'язного графа вершин (сутностей) і шляхів (зв'язків).

При розробці системи були створені наступні діаграми:

– діаграма моделі предметної області;

– діаграма послідовностей та кооперацій;

– діаграма класів проектування;

– діаграма станів.

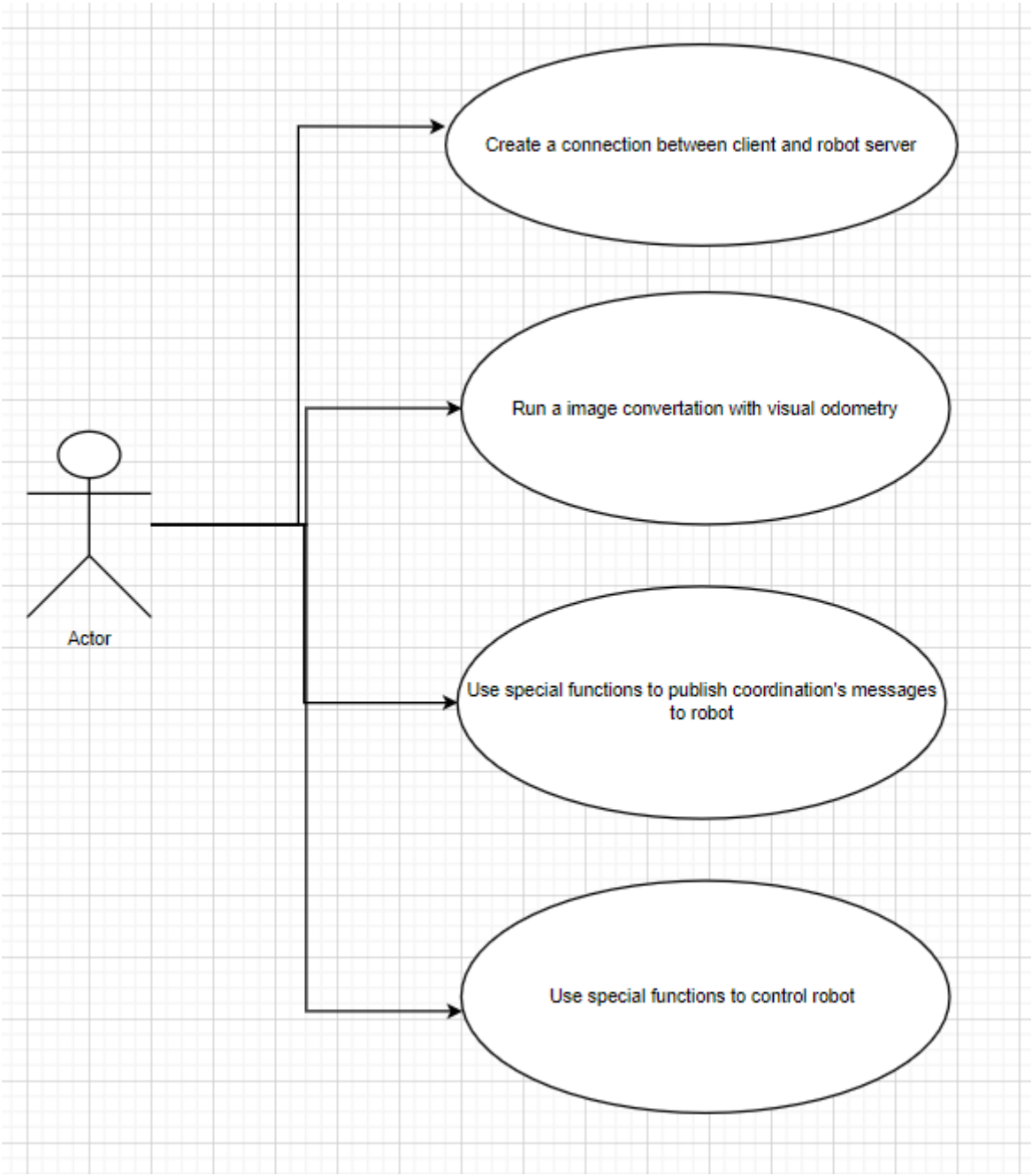

Рисунок 2.3 – Діаграма варіантів використання

Відповідно до методології об'єктно-орієнтованої розробки програмних систем, в основі якої в якості процесу в даній роботі використовується Уніфікований процес компанії Rational – Rational Unified Process (RUP), в технологічному процесі бізнес-моделювання можливі кілька сценаріїв моделювання виробництва, що зумовлює уявлення бізнес-моделі як над безліччю моделі предметної області. При розробці додатків типу комп'ютерних навчальних систем (КОС), основною метою яких є надання інформації та

управління нею, технологічний процес бізнес-моделювання на увазі моделювання предметної області [14]. Метою моделювання предметної області комп'ютерні технології навчання як альтернативної деталі технологічного процесу бізнес-моделювання є:

– ідентифікація класів понять або концептуальних класів комп'ютерної технології навчання;

– деталізація об'єктів предметної області;

– концептуальне уявлення понять предметної області.

Мовою Unified Modeling Language (UML) модель предметної області представляється у вигляді набору діаграм класів, на яких не визначені операції. Модель предметної області може відображати наступне:

– об'єкти предметної області або концептуальні класи;

– асоціацію між концептуальними класами;

– атрибути концептуальних класів.

Для створення моделі предметної області виконуються наступні етапи:

– виявляються концептуальні класи на основі списку категорій і методу аналізу текстового опису для поточної ітерації розробки;

– концептуальні класи відображаються в моделі предметної області;

– додаються необхідні асоціації, що відображають зв'язку, для яких потрібно виділення пам'яті;

Додаються атрибути, необхідні для виконання інформаційних вимог.

Модель предметної області комп'ютерних навчальних систем відображає основні (з точки зору моделює) класи понять (концептуальні класи) або словник предметної області.

Діаграма моделі предметної області для програмної бібліотеки відображено на рисунку 2.4.

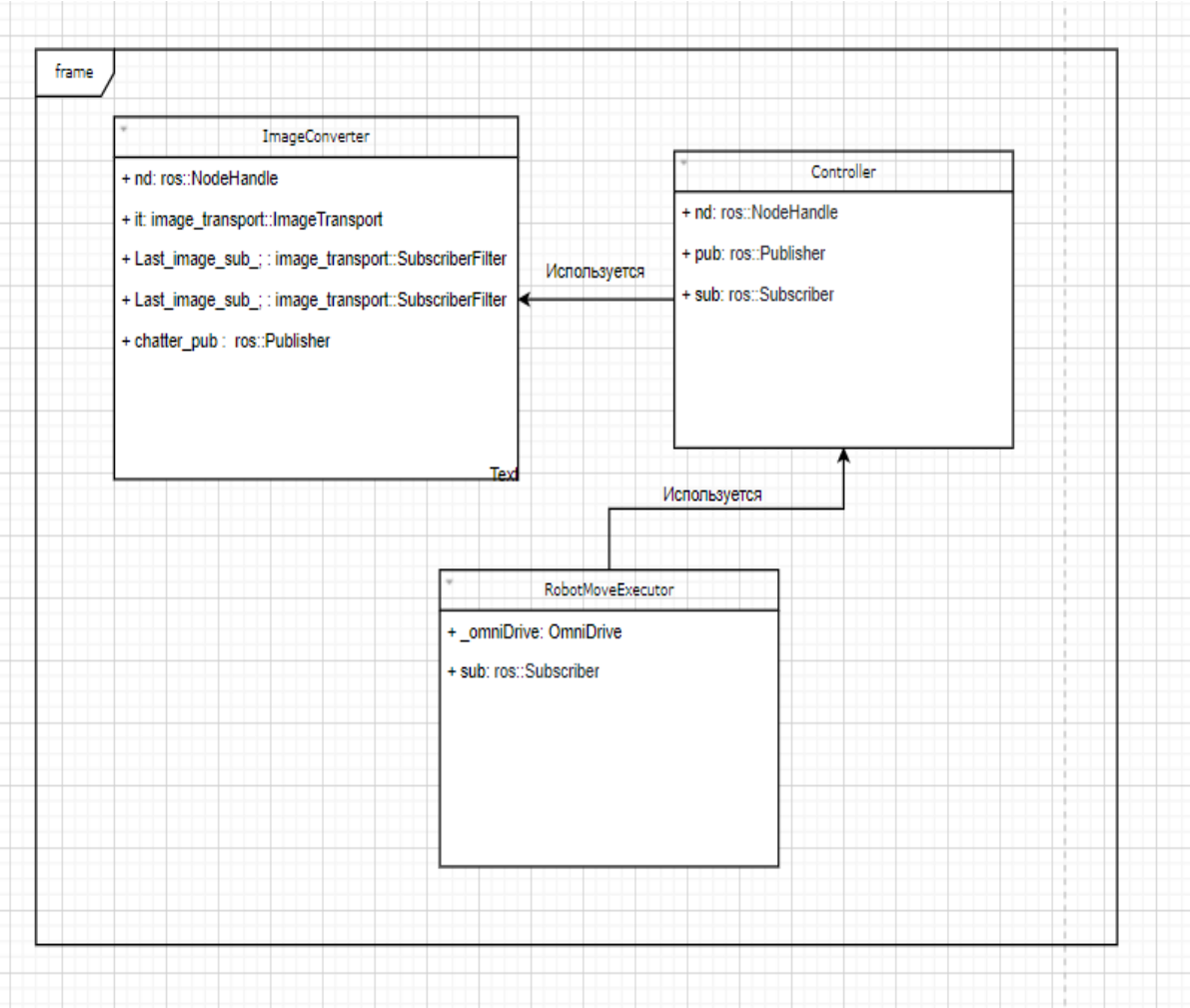

Рисунок 2.4 – Діаграма моделі предметної області

Діаграма комунікації – діаграма, на якій зображуються взаємодії між частинами композитної структури або ролями кооперації. На відміну від діаграми послідовності, на діаграмі комунікації явно вказуються відносини між об'єктами, а час як окремий вимір не використовується (застосовуються порядкові номери викликів).

Для основного успішного сценарію прецеденту була створена діаграма послідовностей системи, на якій були відображені основні зовнішні виконавці, система (як "чорний ящик"), а також системні події, ініційовані виконавцями. При цьому порядок подій повинен відповідати їх послідовності в описі прецеденту. Основним успішним сценарієм є отриманням результатів роботи функцій.

Були створені діаграми послідовностей та кооперацій для програмної біблаотеки, які відображені на рисунках 2.5 – 2.6.

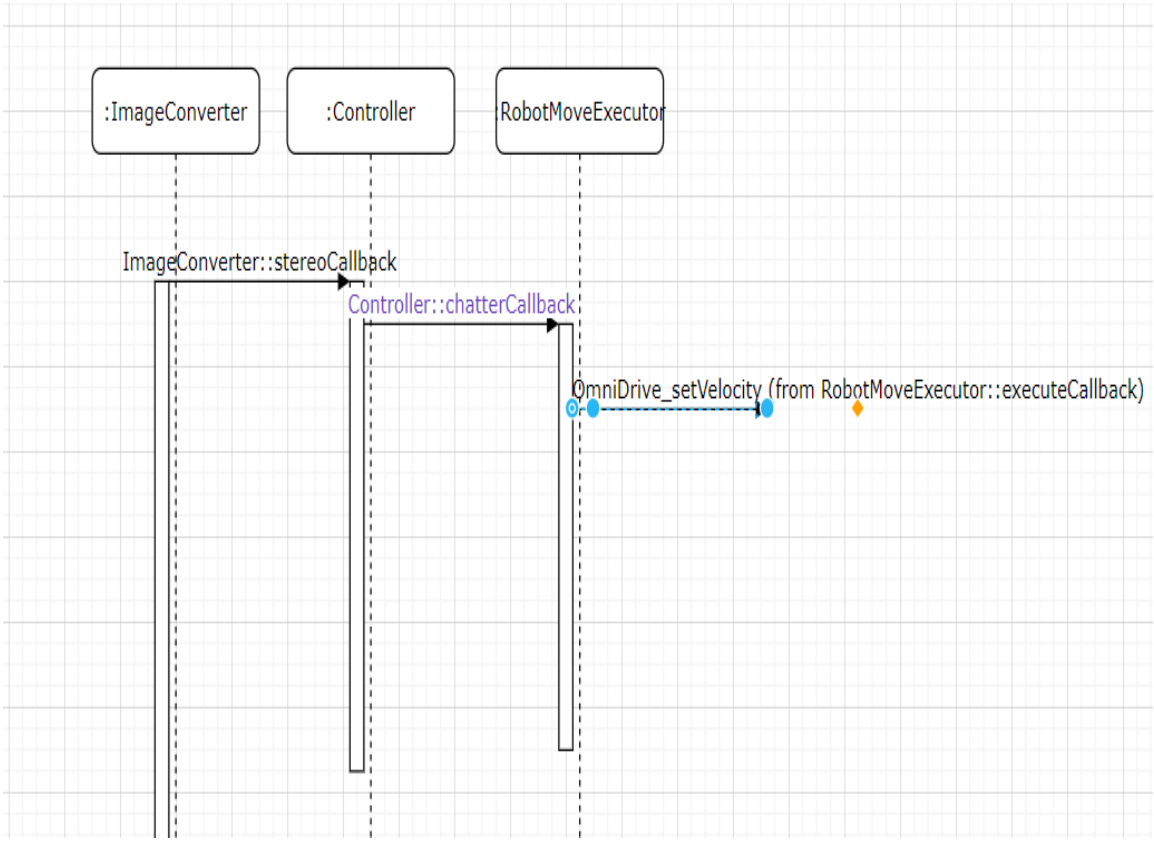

Рисунок 2.5 – Діаграма послідовностей

Діаграма класів – структурна діаграма мови моделювання UML, що демонструє загальну структуру ієрархії класів системи, їх кооперацій, атрибутів (полів), методів, інтерфейсів і взаємозв'язків між ними. Широко застосовується не тільки для документування та візуалізації, але також для конструювання за допомогою прямого або зворотного проектування.

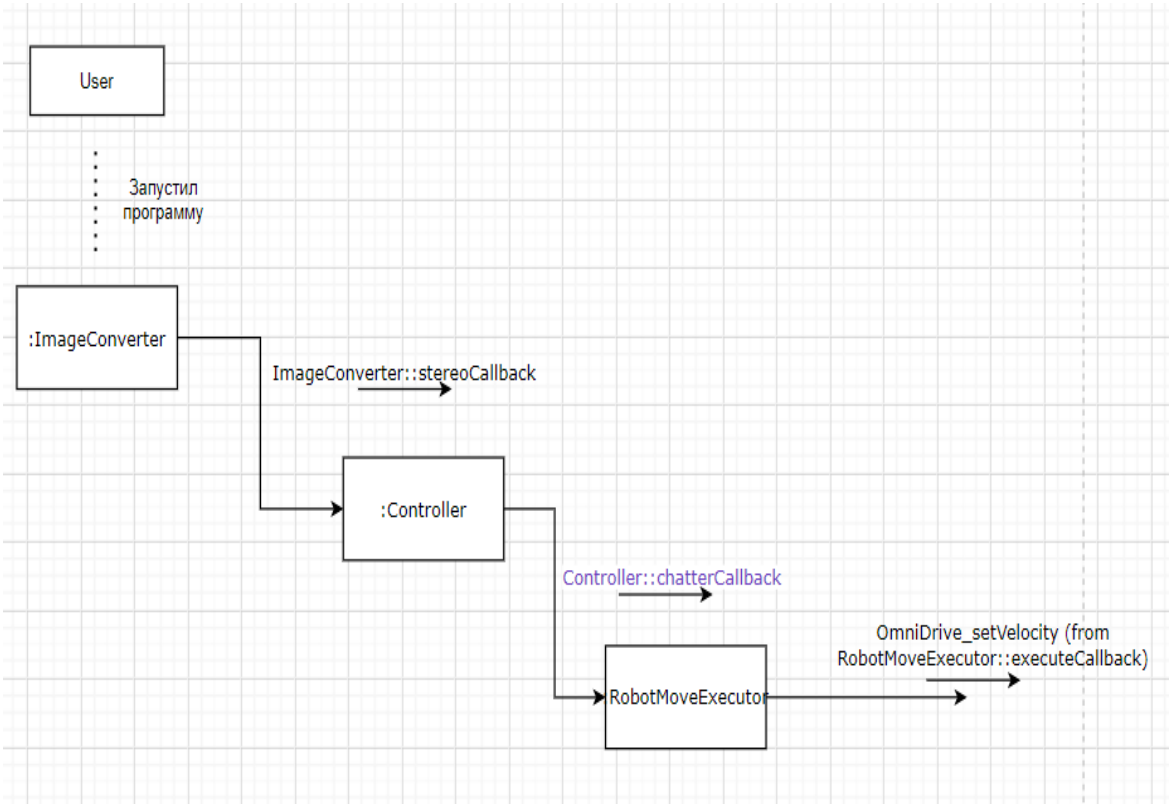

Рисунок 2.6 – Діаграма кооперацій

На цих діаграмах можна динаміку роботи програми у статичному та динамічному режимах за рахунок правильної послідовності.

Метою створення діаграми класів було графічне представлення статичної структури декларативних елементів системи (класів, типів і т. п.). Вона містить в собі також деякі елементи поведінки (наприклад – операції), проте їх динаміка повинна бути відображена на діаграмах інших видів (діаграмах комунікації, діаграмах станів). Для зручності сприйняття діаграму класів можна також доповнити поданням пакетів, включаючи вкладені [14].

При поданні сутностей реального світу потрібно було відобразити їх поточний стан, їх поведінку і їх взаємні відносини. На кожному етапі здійснювалось абстрагування від незначних деталей і концепцій, які не належать до реальності (продуктивність, інкапсуляція, видимість і т. п.). Таким чином, була створена діаграма класів для бібліотеки, яка відображена на рисунку 2.7.

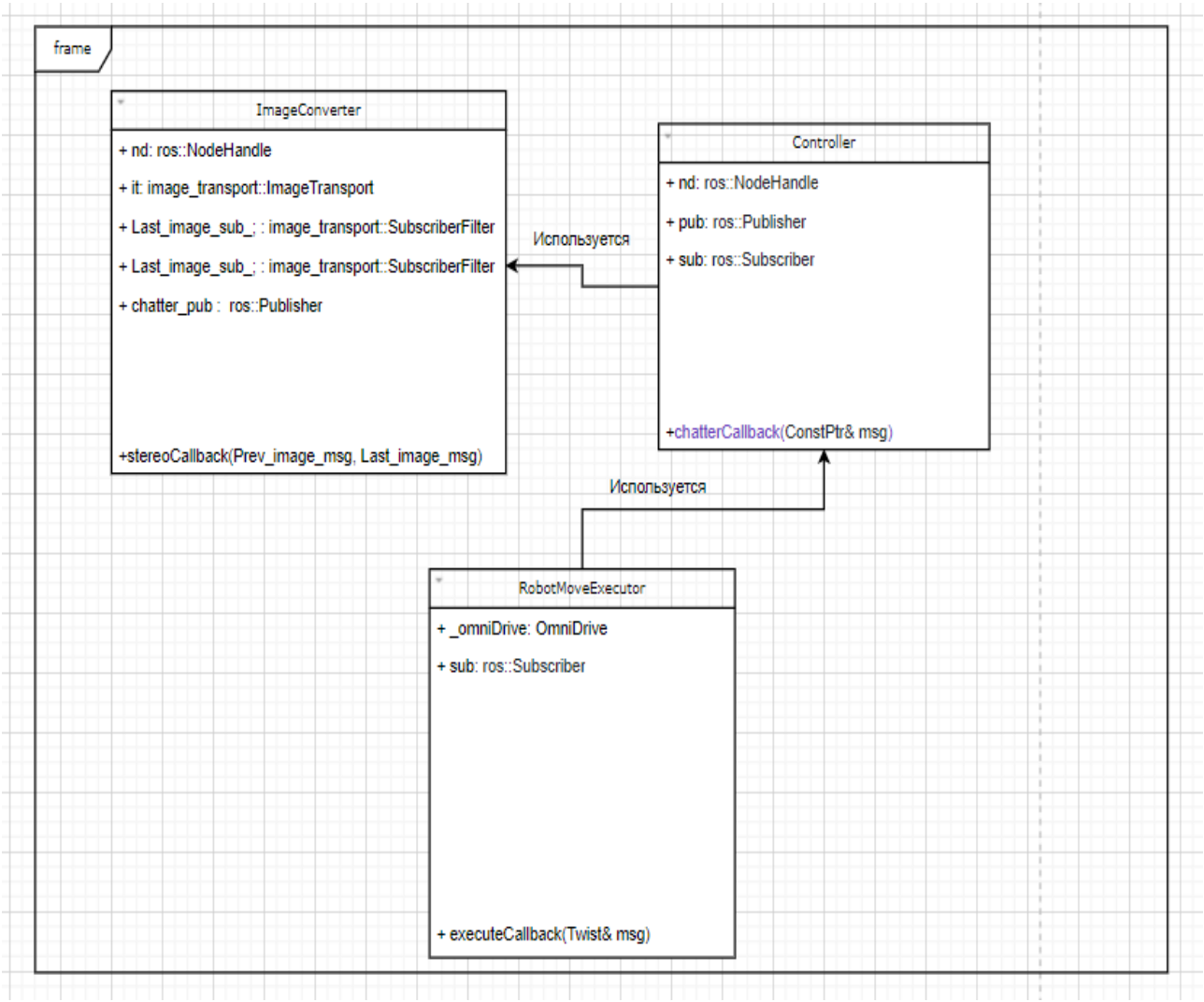

Рисунок 2.7 – Діаграма класів проектування

Діаграма станів може описати можливі послідовності станів і переходів, які в сукупності характеризують поведінку елемента моделі протягом його життєвого циклу. Діаграма станів представляє динамічну поведінку сутностей, на основі специфікації їх реакції на сприйняття деяких конкретних подій. Системи, які реагують на зовнішні впливи від інших систем або від користувачів, іноді називають реактивними [14]. Якщо такі дії ініціюються в довільні випадкові моменти часу, то говорять про асинхронному поведінці моделі.

Хоча діаграми станів найчастіше використовуються для опису поведінки окремих екземплярів класів (об'єктів), але вони також можуть бути застосовані для специфікації функціональності інших компонентів моделей, таких як варіанти використання, актори, підсистеми, операції і методи.

Основними елементами діаграми станів є «Стан» і «Перехід». Діаграма станів має схожу семантику з діаграмою діяльності, тільки діяльність тут замінена станом, переходи символізують дії [14]. У компоненті системи станами виступають режими програми, а переходами будуть функції від користувача, які відображено на діаграмі на рисунку 2.8.

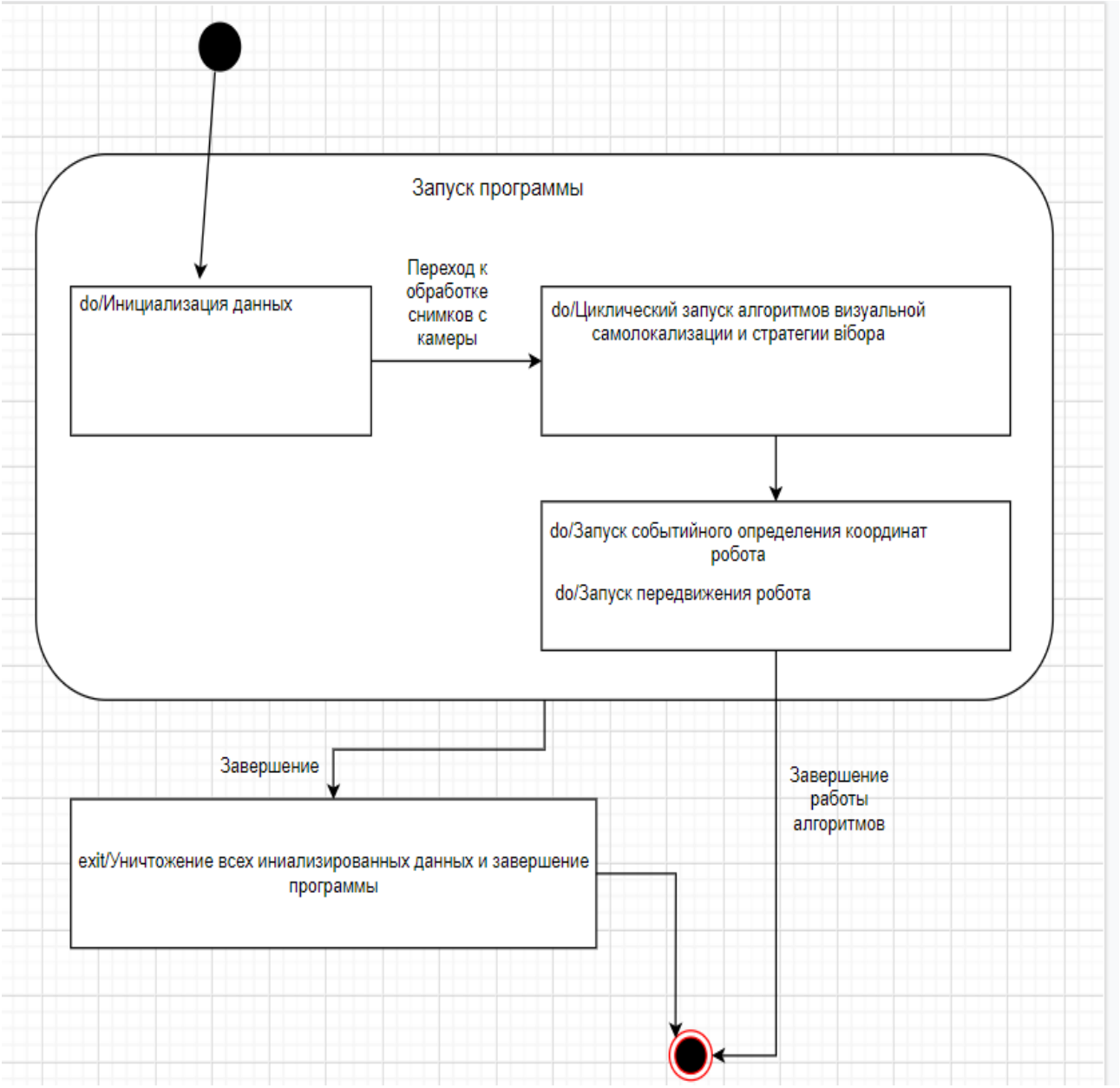

Рисунок 2.8 – Діаграма класів станів

2.4 Висновки до другого розділу

У другому розділі було розглянуто розробку архітектури програмної бібліотеки для моделювання функцій сенсорної системи, а саме розробку алгоритму адаптивної візуальної одометрії, розробка адаптивної стратегії вибору функцій, UML-діаграми функціонування програмної бібліотеки.

3 РОЗРОБКА ПРОГРАМНОЇ БІБЛІОТЕКИ ДЛЯ МОДЕЛЮВАННЯ ФУНКЦІЙ СЕНСОРНОЇ СИСТЕМИ

3.1 Використання існуючих технологій для рішення поставленої задачі

Технологіями реалізації програмного забезпечення (ПЗ) є інструменти розробки, як мови програмування, системи збірки проектів, компілятори, відладчики та інші. За допомогою мови програмування C++ був написаний код, який є реалізацією логіки роботи програмної бібліотеки для моделювання функцій сенсорної системи. Таким чином, імплементація програмного забезпечення бібліотеки була реалізована за допомогою високорівневої об’єктно-орієнтованої мови C++, а автоматична збірка виконувалася за допомогою утиліти CMake. Також прикладний інтерфейс програми був пов’язаний з використанням бібліотеки Robotino API2 та обгортка операційної системи ROS, використовувалось графічне середовище Robotino View для розробки та тестування робота Robotino та симуляційне середовище Robotino SIM для експериментів з Robotino.

Алгоритми розроблялись за допомогою мови програмування C++, тому що C++ – компільований, статично типізована мова програмування загального призначення [15].

Підтримує такі парадигми програмування, як процедурне програмування, об'єктно-орієнтоване програмування, узагальнене програмування. Мова має багату стандартну бібліотеку, яка включає в себе поширені контейнери і алгоритми, введення-виведення, регулярні вирази, підтримку багатопоточності і інші можливості. C++ поєднує властивості як високорівневих, так і низькорівневих мов. У порівнянні з його попередником - мовою C, - найбільшу увагу приділено підтримці об'єктно-орієнтованого і узагальненого програмування.

C++ широко використовується для розробки програмного забезпечення, будучи одним з найпопулярніших мов програмування. Область його застосування включає створення операційних систем, різноманітних прикладних програм, драйверів пристроїв, додатків для вбудованих систем, високопродуктивних серверів, а також розважальних програм (ігор). Існує безліч реалізацій мови C++, як безкоштовних, так і комерційних і для різних платформ. Наприклад, на платформі x86 це GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder і інші. C++ зробив величезний вплив на інші мови програмування, в першу чергу на Java і C#.

Синтаксис C++ успадкований від мови C. Одним з принципів розробки було збереження сумісності з C. Проте, C++ не є в строгому сенсі надбезліччю C; безліч програм, які можуть однаково успішно транслюватися як компіляторами C, так і компіляторами C++, досить велике, але не включає всі можливі програми на C.

Для збірки програми використовувалась утиліта CMake, яка є кроссплатформенною утилітою для автоматичного складання програми з вихідного коду [16]. При цьому сам CMake безпосередньо складанням не займається, а представляє з себе front-end. Як back-end можуть виступати різні версії make і Ninja. Так само CMake дозволяє створювати проекти для CodeBlocks, Eclipse, KDevelop3, MS VC++ та Xcode. Варто відзначити, що більшість проектів створюються не нативних, а все з тим back-end.

Безпосередньою збіркою CMake не займається, а тільки генерує Makefile, який потім буде виконаний утилітою make. Для того що б зібрати проект засобами CMake, необхідно в корені дерева початкових кодів розмістити файл CMakeLists.txt, який зберігає правила і цілі збірки.

CMake може перевіряти наявність необхідних бібліотек і підключати їх, збирати проекти під різними компіляторами і операційними системами. Тобто у вас є купа коду і файлик, що містить інформацію для cmake, і щоб скомпілювати це, вам потрібно просто запустити там cmake, який зробить все сам.

Вибір середовища програмування є невід’ємною частиною цієї роботи, Середовище програмування – це набір інструментів, які використовуються для перетворення символів в здійсненні обчислення.

Тобто середовище програмування (або середовище розробки) – це такі програми, в яких програмісти реалізовують свої коди з метою створення якогось окремого модуля або програми.

Простіше кажучи, середовище програмування служить для того, щоб розробити (написати) програму, і вона орієнтована на певну мову, сукупність мов програмування (ці машинні мови відносяться до однієї мовної групи, наприклад, C ++або C#).

Інтегроване середовище включає все необхідне для написання:

а) редактор синтаксису конкретного мови програмування;

б) компілятор – відповідає за трансляцію програми, переводить написану на наближеному до людському мови код в символи, зрозумілі комп'ютеру, машинний код (мова C++ відноситься до мов компіляції, тому для того, щоб обробити текст програми необхідний компілятор, часто замість компілятора використовується інтерпретатор);

в) відладчик – є інструментом для налагодження написаної програми. Відомо, що помилки в програмах допускають все поголовно: від синтаксичних (як правило, ще на стадії компіляції вони повинні бути виявлені) і, що набагато важливіше, логічними. Виявити і усунути останній тип помилок допомагає відладчик програм.

Як правило, для того, щоб успішно виконати написану програму на C++, треба пройти благополучно подолати п’ять етапів:

– редагування;

– попередня (препроцесорну) обробка;

– компоновка;

– компіляція;

– завантаження.

Редагування виконується за допомогою редактора програм, який, по суті, є звичайнісіньким редактором текстових файлів, такий як блокнот, word, але вбудований в ту чи іншу середу програмування. Програміст створює в цьому редакторі свою програму на C++ і вносить в неї різні поправки, коригування.

Для програмної бібліотеки для моделювання функцій сенсорної системи використовувалось інтегроване середовище розробки Visual Studio.

Microsoft Visual Studio – лінійка продуктів компанії Microsoft, що включають інтегроване середовище розробки програмного забезпечення і ряд інших інструментальних засобів [17]. Дані продукти дозволяють розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, підтримуваних Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework і Silverlight.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторінга коду. Вбудований відладчик може працювати як відладчик рівня вихідного коду, так і відладчик машинного рівня. Решта вбудовуються інструменти включають в себе редактор форм для спрощення створення графічного інтерфейсу додатку, веб-редактор, дизайнер класів і дизайнер схеми бази даних.

Visual Studio дозволяє створювати і підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду (як, наприклад, Subversion і Visual SourceSafe), додавання нових наборів інструментів (наприклад, для редагування і візуального проектування коду на предметно-орієнтованих мовах програмування) або інструментів для інших аспектів процесу розробки програмного забезпечення (наприклад, клієнт Team Explorer для роботи з Team Foundation Server).

Під час розробки бібліотеки використовувався список бібліотек мови С++, який містить перелік різних бібліотек шаблонів або класів, доступних для використання при написанні програми на мові програмування С++.

Основною бібліотекою при написанні алгоритмів та створення імплеменації логіки були STL.

Бібліотека стандартних шаблонів (STL – Standard Template Library) – - набір узгоджених узагальнених алгоритмів, контейнерів, засобів доступу до їх вмісту і різних допоміжних функцій в С++ [18].

Бібліотека стандартних шаблонів до включення в стандарт С++ була сторонньою розробкою, спочатку фірми HP, а потім SGI. Стандарт мови не називає її «STL», так як ця бібліотека стала невід'ємною частиною мови, проте багато людей до сих пір використовують цю назву, щоб відрізняти її від іншої частини стандартної бібліотеки (потоки введення-виведення – iostream), підрозділ Сі та ін.).

Проект під назвою STLPort, заснований на SGI STL, здійснює постійне оновлення STL, iostream і строкових класів. Деякі інші проекти також займаються розробкою приватних застосувань стандартної бібліотеки для різних конструкторських завдань. Кожен виробник компіляторів С++ обов'язково поставляє якусь реалізацію цієї бібліотеки, так як вона є дуже важливою частиною стандарту і широко використовується.

У бібліотеці виділяють п'ять основних компонентів:

– контейнер (Container) це зберігання набору об'єктів в пам'яті;

– ітератор (Iterator) це забезпечення засобів доступу до вмісту контейнера;

– алгоритм (Algorithm) це визначення обчислювальної процедури;

– адаптер (Adaptor) це адаптація компонентів для забезпечення різного інтерфейсу;

– функціональний об'єкт (Functor) це приховування функції в об'єкті для використання іншими компонентами.

Поділ дозволяє зменшити кількість компонентів. Наприклад, замість написання окремої функції пошуку елемента для кожного типу контейнера

забезпечується єдина версія, яка працює з кожним з них, поки дотримуються основні вимоги.

Також для об’єктного моделування використовувалася мова UML. UML (Unified Modeling Language – уніфікована мова моделювання) – мова графічного опису для об'єктного моделювання в області розробки програмного забезпечення, для моделювання бізнес-процесів, системного проектування та відображення організаційних структур.

UML є мовою широкого профілю, тобто відкритий стандарт, який використовує графічні позначення для створення абстрактної моделі системи, званої UML-моделлю. UML був створений для визначення, візуалізації, проектування та документування, в основному, програмних систем. UML не є мовою програмування, але на підставі UML-моделей можлива генерація коду.

UML дозволяє також розробникам програмного забезпечення досягти угоди в графічних позначеннях для представлення загальних понять (таких як клас, компонент, узагальнення (Generalization), агрегація (Aggregation) і поведінку) і більше сконцентруватися на проектуванні та архітектури.

В UML використовуються наступні види діаграм, які відображені у вигляді ієрархії на рисунку 3.1.

UML чудово зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки.

Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

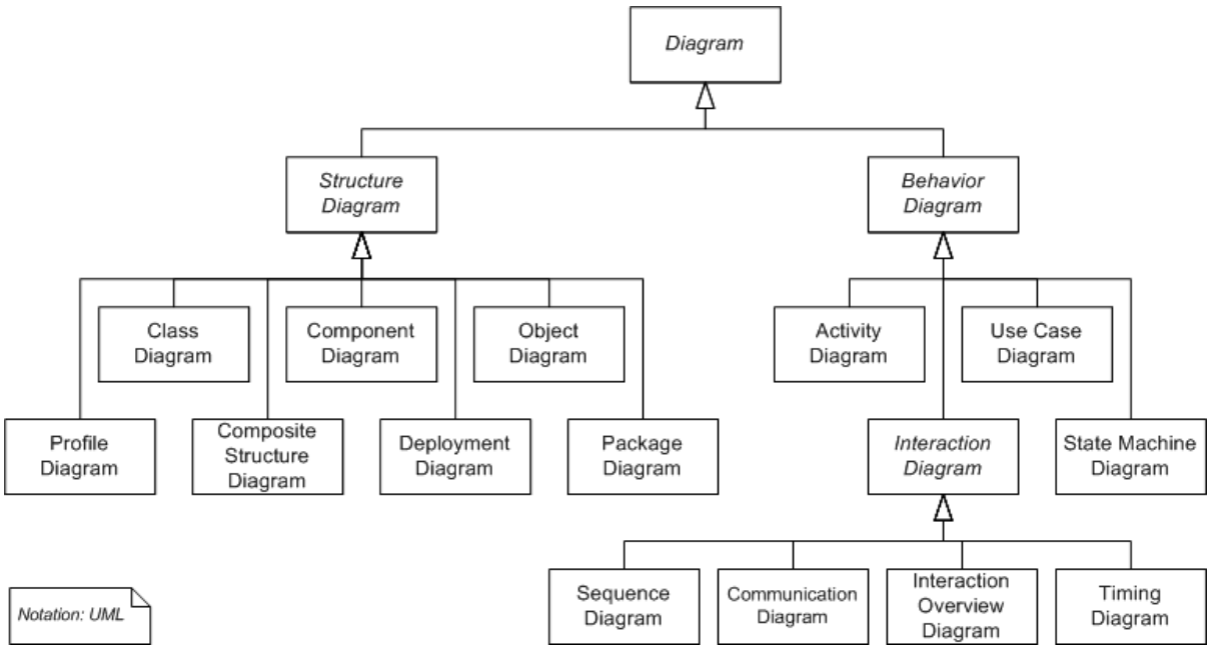

Рисунок 3.1 – Всі види UML-діаграм

3.2 Розробка алгоритмів функціонування сенсорної системи

Під час розробки алгоритмів в першу чергу підключалися необхідні бібліотеки.

```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <image_transport/subscriber_filter.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include "opencv2/opencv.hpp"
#include <message_filters/subscriber.h>
#include <message_filters/time_synchronizer.h>
#include <geometry_msgs/Twist.h>
#include <geometry_msgs/Pose.h>
```

ros/ros.h – це заголовки містить всі необхідні заголовки, які використовуються для роботи з системою ROS. Цей заголовки містить оголошення повідомлення типу std_msgs / String, який знаходиться в пакеті std_msgs.

image_transport/image_transport.h – завжди слід використовувати для підписки та публікації зображень. Він забезпечує прозору підтримку транспортування зображень у стислих форматах із низькою пропускною здатністю. Приклади (надаються окремими пакетами плагінів) включають стиснення JPEG / PNG та потокове відео Theora.

image_transport/subscriber_filter.h – цей клас обертає Subscriber як "фільтр", сумісний з пакетом message_filters. Він діє як фільтр найвищого рівня, просто передаючи повідомлення з передплати на транспортування зображень до фільтрів, які до нього підключені.

cv_bridge/cv_bridge.h – це ROS-бібліотека, яка забезпечує інтерфейс між ROS і OpenCV. CvBridge можна знайти в пакунку cv_bridge у стеці vision_opencv.

Деякі функції для розрізнення категорій кодування. Вони розміщені в sensor_msgs/image_encodings.h.

opencv2/opencv.hpp – включаються оголошення всіх взагалі функцій OpenCV.

message_filters/subscriber.h – заголовок бібліотеки повязаної з портом, який спеціалізований для зчитування даних постійного типу, опублікованих за темою.

time_synchronizer.h – бібліотека для синхронізування до 9 повідомлень за мітками часу.

TimeSynchronizer синхронізує до 9 вхідних каналів за позначками часу, що містяться в заголовках їх повідомлень. TimeSynchronizer приймає від 2 до 9 типів повідомлень як параметри шаблону і передає їх у зворотний виклик, який приймає загальний вказівник кожного.

Необхідний параметр розміру черги при побудові TimeSynchronizer повідомляє, скільки наборів повідомлень він повинен зберігати (за позначкою часу), очікуючи надходження повідомлень і завершення їх "набору".

geometry_msgs – geometry_msgs надає повідомлення для загальних геометричних приматів, таких як точки, вектори та пози. Ці примати призначені для забезпечення загального типу даних та полегшення взаємодії у всій системі.

Далі виконується ініціалізація необхідних перемінних для подальшого їх використання у алгоритмах.

```
Mat imgPrev, imgLast, imgPrevc, imgLastc, imgPrevNew, imgPrevOld, imgLastOld;
TermCriteria termcrit = TermCriteria(TermCriteria::COUNT + TermCriteria::EPS,
30, 0.01);
vector<float> err;
Size winSize(31, 31);
vector<uchar> statusLKT;
vector<KeyPoint> keypointsPrev, keypointsLast, keypointsPrevNew,
goodKeypointsStereoPrev, goodKeypointsStereoLast;
Mat descriptorsPrev, descriptorsLast, descriptorsPrevNew, goodDescriptorsStereo;
OrbDescriptorExtractor extractor(2000);
std::vector<DMatch> matches;
double avrgTime, tick;
std::vector<Point2f>  goodPointsNew, goodPointsTriPrev, goodPointsTriLast;
Mat K1, D1, P1, P2, HandEye, RTnew, Rnew, Rtotal, Ttotal, intrinsics, distortion,
rvec, tvec;
Mat RTtotal = (Mat_<double>(4, 4) << 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);
Mat tvecSaved = (Mat_<double>(3, 1) << 0, 0, 0);
int initFeatures = 0;
int maxFeat = 0;
double bestAngle = 0;
```

На цьому етапі поюяснюються призначення певних класів, а саме для чого вони використовувались, окрім тих типів данних, які не треба пояснювати, а саме їх назви та назви змінних мають зрозумілий контекст.

Клас Mat представляє n-вимірний щільний числовий одноканальний або багатоканальний масив. Він використовується для зберігання реальних векторів і матриць, відтінків сірого, обсягів вокселів, векторних полів, хмар точокгістограм (однак, дуже великі гістограми краще зберігати в SparseMat).

Клас TermCriteria для того, щоб визначати критерії завершення для ітераційних алгоритмів.

Змінну termcrit можна було ініціалізувати за замовчуванням, а потім перевизначити будь-які параметри, або структура може бути повністю ініціалізована за допомогою розширеного варіанту конструктора.

Використовується вектори з об’єктами структура даних KeyPoint, необхідних для виділених точкових детекторів. Такі екземпляри класу зберігають ключові точки в алгоритмі візульної одометрії, тобто функцію точки, знайдену одним із багатьох доступних детекторів ключових точок, таких як детектор кутів Харріса, FAST, тощо, які будут пояснені далі.

Такі ключові точки характеризуються двовимірним положенням, масштабом (пропорційним діаметру сусідства, який потрібно враховувати), орієнтацією та деякими іншими параметрами. Потім сусідство ключових точок аналізується за допомогою іншого алгоритму, який створює дескриптор (зазвичай представлений у вигляді вектора ознак). Потім ключові точки, що представляють один і той же об’єкт на різних зображеннях, можуть бути зіставлені за допомогою KDTree або іншого методу.

Використовується клас OrbDescriptorExtractor, що реалізує ORB детектор ключових точок та екстрактор дескрипторів.

Алгоритм використовує FAST для виявлення стабільних ключових точок, відбирає найсильніші риси за допомогою FAST або відповіді Харріса, знаходить їх орієнтацію, використовуючи моменти першого порядку, і обчислює дескриптори (де координати випадкових пар точок (або k-кортежів) повернуто відповідно до вимірюваної орієнтації).

Алгоритм починається з отримання загружених двох останніх зображень.

```
imgPrevc = cv_ptr1->image;
imgLastc = cv_ptr2->image;
```

Далі за допомогою функції cvtColor конвертуються зображення у відтінки сірого.

```
cvtColor(imgPrevc, imgPrev, COLOR_BGR2GRAY);
cvtColor(imgLastc, imgLast, COLOR_BGR2GRAY);
```

Наступним кроком виявляються точки збігів у попередньому та останньому зображенні (для тріангуляції) та відстежуються ці точки у новому попередньому зображенні за домогою функції calcFeatureSets.

```
calcFeatureSets(goodPointsNew, goodPointsTriPrev, goodPointsTriLast);
```

Далі розглядається функція calcFeatureSets для обчислення відповідних точок.

Виконується очищення всіх старих наборів даних.

```
goodPointsNew.clear();
goodPointsTriPrev.clear();
goodPointsTriLast.clear();
keypointsPrevNew.clear();
goodKeypointsStereoPrev.clear();
goodKeypointsStereoLast.clear();
keypointsPrev.clear();
keypointsLast.clear();
```

Далі виявляються точки-збіги за допомогою метода FAST на передостанньому та останньому зображенні.

FAST – це метод виявлення кутів, який можна використовувати для вилучення точок об’єкта, а згодом використовувати для відстеження та відображення об’єктів у багатьох завданнях комп’ютерного зору [22].

```
extractor.detect(imgPrevOld, keypointsPrev);
extractor.detect(imgLastOld, keypointsLast);
```

Виконується отримення дескрипторів ORB. ORB - це в основному поєднання детектора ключових точок FAST та короткого дескриптора з багатьма модифікаціями для підвищення продуктивності.

```
extractor.compute(imgPrevOld, keypointsPrev, descriptorsPrev);
extractor.compute(imgLastOld, keypointsLast, descriptorsLast);
```

Далі виконується збіг дескрипторів функцій із збігом грубої сили та перехресною перевіркою (з використанням дистанції Хеммінга).

```
BFMatcher matcher = BFMatcher(NORM_HAMMING, true);
matches.clear();
matcher.match(descriptorsPrev, descriptorsLast, matches);
```

Видаляються погані стереозбіги, які не знаходяться в одній і тій же координаті.

```
std::vector< DMatch > good_matches_stereo, veryGood;
for (int i = 0; i < matches.size(); i++)
{
	if (fabs(keypointsPrev[matches[i].queryIdx].pt.y -
	keypointsLast[matches[i].trainIdx].pt.y) <= 2)
	{
		good_matches_stereo.push_back(matches[i]);
	}
}
```

Видаляються збіги з відстанню, що перевищує поріг.

```
for (int i = 0; i < good_matches_stereo.size(); i++)
{
	if (good_matches_stereo[i].distance < 40)
	{
		veryGood.push_back(good_matches_stereo[i]);
	}
}
good_matches_stereo = veryGood;
for (int i = 0; i < good_matches_stereo.size(); i++)
{
	goodKeypointsStereoPrev.push_back(keypointsPrev[good_matches_stereo[i].q
ueryIdx]);
```

```
        goodKeypointsStereoLast.push_back(keypointsLast[good_matches_stereo[i].tr
ainIdx]);
}
vector<Point2f> goodPointsStereoPrev, goodPointsStereoLast, pointsNew;
for (int i = 0; i < goodKeypointsStereoPrev.size(); i++)
{
        goodPointsStereoPrev.push_back(goodKeypointsStereoPrev[i].pt);
        goodPointsStereoLast.push_back(goodKeypointsStereoLast[i].pt);
}
```

Далі видаляються значення вищі порогових за допомогою RANSAC. RANSAC це ітеративний метод, що використовується для оцінки параметрів математичної моделі для набору спостережуваних даних які містять викиди.

```
Mat statusStereo;
findFundamentalMat(goodPointsStereoPrev, goodPointsStereoLast,
CV_FM_RANSAC, 3, 0.999, statusStereo);
int stereoInliers = 0;
vector <KeyPoint> Prev, Last;
for (int i = 0; i < statusStereo.rows; i++)
{
        if (statusStereo.at<bool>(0, i))
        {
                Prev.push_back(goodKeypointsStereoPrev[i]);
                Last.push_back(goodKeypointsStereoLast[i]);
                stereoInliers++;
        }
}

goodKeypointsStereoPrev = Prev;
goodKeypointsStereoLast = Last;
goodPointsStereoPrev.clear(); goodPointsStereoLast.clear();
```

```
for (int i = 0; i < goodKeypointsStereoPrev.size(); i++)
{
	goodPointsStereoPrev.push_back(goodKeypointsStereoPrev[i].pt);
	goodPointsStereoLast.push_back(goodKeypointsStereoLast[i].pt);
}
```

Відстежуваються точки-збіги поточного кадру на основі точок старого кадру.

```
calcOpticalFlowPyrLK(imgPrevOld, imgPrev, goodPointsStereoPrev, pointsNew,
statusLKT, err, winSize, 3, termcrit, 0, 0.001);
```

Виконується останній етап функції calcFeatureSets, а саме генеруються набори відповідних точок старого зображення (для тріангуляції) та нового зображення.

```
int count = 0;
for (int i = 0; i < statusLKT.size(); i++)
{
	if (statusLKT[i])
	{
		goodPointsNew.push_back(pointsNew[i]);
		goodPointsTriPrev.push_back(goodKeypointsStereoPrev[i].pt);
		goodPointsTriLast.push_back(goodKeypointsStereoLast[i].pt);
		count++;
	}
}
cout << "matched features: " << count << endl;
```

Після виконання функції calcFeatureSets триангулюються точки-збіги передостаннього зображення та останнього зображення, щоб отримати набір 3D-точок світу за допомогою функції Triangulation, тріангулюються тривимірні точки лінійним методом (імплементацію функції Triangulation можно побачити у розділі з повним текстом програми) [23].

```
std::vector<Point3f>  worldPointsHart;
```

```
for (int i = 0; i < goodPointsTriPrev.size(); i++)
{
	Mat world = Triangulation(goodPointsTriPrev.at(i), P1, goodPointsTriLast.at(i), P2);
	worldPointsHart.push_back(Point3f(world.at<double>(0,0), world.at<double>(1, 0), world.at<double>(2, 0)));
}
```

Виконується розв’язання PnP за допомогою RANSAC (який описано вище). PnP – це проблема оцінки пози каліброваної камери з урахуванням набору з n 3D-точок у світі та відповідних їм 2D-проекцій на зображенні.

```
Mat inliers;
solvePnPRansac(worldPointsHart, goodPointsNew, K1, D1, rvec, tvec, false, 1000, 2.0, -1, inliers, CV_P3P);
tvec = -tvec;
rvec = -rvec;
```

Далі видаляються дуже малі або великі кути обертання та перетворення.

```
rvec.at<double>(0) = 0;
rvec.at<double>(2) = 0;
tvec.at<double>(1) = 0;
for (int i = 0; i < 3; i++)
{
	if ((fabs(rvec.at<double>(i)) < 0.2 / 180 * 3.14) || (fabs(rvec.at<double>(i)) > 1.5))
	{
		rvec.at<double>(i) = 0;
	}
}
for (int i = 0; i < 3; i++)
{
	if ((fabs(tvec.at<double>(i)) < 1) || (fabs(tvec.at<double>(i)) > 300))
```

```
        {
            tvec.at<double>(i) = 0;
        }
}
```

На майже останньому етапі обчислється загальна позиція камери робота.

```
Rodrigues(rvec, Rnew);
RTnew = (Mat_<double>(4, 4) << Rnew.at<double>(0, 0), Rnew.at<double>(0, 1),
Rnew.at<double>(0,2), tvec.at<double>(0), Rnew.at<double>(1,0),
Rnew.at<double>(1,1), Rnew.at<double>(1,2), tvec.at<double>(1),
Rnew.at<double>(2,0), Rnew.at<double>(2,1), Rnew.at<double>(2,2),
tvec.at<double>(2), 0, 0, 0, 1);
```

В результаті виконується розрахунок пози робота за позою камери (співвідношення рука-око).

```
RTnew = HandEye * RTnew * HandEye.inv();
RTtotal = RTtotal * RTnew;
```

Після виконання усіх вище описаних операцій виконується для зручності та подальшого використання розрахунок данних (можна побачити у розділі з повним текстом програми) для ліній переміщення ключових точок, які рухалися більше, ніж поріг, обчислення кольорів відповідно до глибини тривимірної точки, визначення інформації про глибину, збереження поточного кадру зображення для наступного кроку циклічних розрахунків та відправляється позиція робота до екзампляра класса Controller (який можна побачити у розділі з UML-діграмами).

3.3 Розробка функцій прикладного інтерфейсу розробника (API)

В цьому розділі буде описано етапи створення бібліотеки від етапу написання коду для збірки програми за допомогою утиліти CMake.

У розділі з розробкою UML-діграм можна було побачити, що основними классами бібліотеки є ImageConverter, Controller та RobotMoveExecutor. Кожен

класс бібліотеки має свій інтерфейс, вимагає підключення додаткових бібліотек та створення CMakeLists.txt файлів, що налаштовують процес створення оптимізованого для конкретної системи робочого файлу для автоматичної збірки.

Отже, для кожного класу були створені CMakeLists.txt файли з необхідними налаштування.

Для классу ImageConverter встановлюється мінімально необхідна версія cmake для проекту. Якщо запущена версія CMake нижча за <min> необхідну версію, вона припинить обробку проекту та повідомить про помилку.

```
cmake_minimum_required(VERSION 2.8.3)
```

Для налаштування імені результуючого файлу та імені проекту для подальшої роботи використана команда project.

```
project(ImageConverter)
```

Далі команда package знаходить зовнішні бібліотеки і завантажує їх та їх налаштування. Спеціальний список необхідних компонентів може бути перерахований після опції COMPONENTS (опції REQUIRED відповідає за необхідність обов’язкового заваження компонентів).

```
find_package(catkin REQUIRED COMPONENTS
  cv_bridge
  image_transport
  roscpp
  sensor_msgs
  std_msgs
  geometry_msgs
)
 find_package(OpenCV REQUIRED)
catkin_package()
```

Команда include_directories дозволяє додати вказані каталоги до тих, за допомогою яких компілятор шукає файли включення. Відносні шляхи інтерпретуються як відносно поточного вихідного каталогу. Каталоги

включення додаються до властивості каталогу INCLUDE_DIRECTORIES поточного файлу CMakeLists. Вони також додаються до властивості цілі INCLUDE_DIRECTORIES для кожної цілі у поточному файлі CMakeLists. Цільовими значеннями властивостей є ті, що використовуються генераторами.

include_directories(${catkin_INCLUDE_DIRS})

include_directories(${OpenCV_INCLUDE_DIRS})

Команда add_executable додає виконувану ціль під назвою <ImageConverter>, яка буде створена із вихідних файлів, перелічених у виклику команди. (Тут можна опустити вихідні файли, якщо вони будуть додані пізніше за допомогою target_sources ().) <ImageConverter> відповідає логічному імені цілі та має бути глобально унікальним у рамках проекту. Фактичне ім'я файлу побудованого виконуваного файлу будується на основі домовленостей власної платформи (наприклад, <ім'я> .exe або просто <ім'я>).

add_executable(ImageConverter src/main.cpp)

Команда target_link_libraries вказує бібліотеки аб, які будуть використовуватися при зв’язуванні даного цільового об’єкта та його залежних. Вимоги до використання залежностей цілі впливають на компіляцію власних джерел.

target_link_libraries(localisation ${catkin_LIBRARIES})

target_link_libraries(localisation ${OpenCV_LIBRARIES})

Так само, були створені CMakeLists файли з налаштування для автоматичної збірки для классів Controller та RobotMoveExecutor (код можна побачити у розділі з повним текстом програми).

Класс ImageConverter відповідає за роботу алгоритмів локалізації робота та стратегії вибору, імплементація яких у загальному випадку описано у підрозділі 3.2. У цьому розділі хочу додати описання того, як цей клас віддає фінальну позицію робота. Це відбується за допомогою Callback-functions у ROS-системі, а саме за кожен короткий проміжок часу діючі пакети (в данному випадку ImageConverter, Controller та RobotMoveExecutor) отримують та публікують (тобто відправляють) повідомлення за допомогою таких функцій,

тобто таким чином спілкуються між собою та отримають певну інформацію для подальшої їх автономної роботи (детально цей механізм описано у розділі 1.3).

Отже, фінальним результатом роботи ImageConverter є повідомлення з позицією робота geometry_msgs::Pose cmd_pos_msg, що можна побачити у розділі 3.2. І як раз за домомогою callback-функції ImageConverter::stereoCallback відправляється повідомлення про позицію, яку вже обробляє Controller.

chatter_pub.publish(cmd_pos_msg);

Далі описується імплементація класу Controller, який був створений для обробки повідомлення від ImageConverter та тривіальних розрахунків траєкторії подальшого руху робота для тестування алгоритмів.

Controller має власну callback-функцію, яка приймає вхідні дані у вигляді повідомлення від ImageConverter.

chatterCallback(const geometry_msgs::Pose::ConstPtr& msg)

По-перше виконується ініціалізація необхідних змінних.

```
// траєкторія призначення в мм
double destinationsX [4] ={0,1000,1000,0};
double destinationsY [4] ={1000,1000,0,0};
// максимальна швидкість роботино
double SPEED=0.2,speed;
//допоміжні змінні
double xr,yr,angle,xw,yw;
int k=0;
```

Далі описується основна логіка роботи Controller. Першою операцією оновлюється напрямок в матриці для тесту з отриманих координат.

```
xw=destinationsX[msg.location.x];
yw=destinationsY[msg.location.y];
```

Позиція msg->position відвовідає за сторону повороту робота. У цьому коді генеруємо дані для повороту вправо або вліво за бажанням, тобто тут використовуються згенеровані дані стратегії вибору.

```
// need to turn right
if(msg->position.y==1)
{
        geometry_msgs::Twist cmd_vel_msg;
        cmd_vel_msg.angular.z=-0.2;
        chatter_pub.publish(cmd_vel_msg);
} // need to turn left
else if (msg->position.y==2)
{
        geometry_msgs::Twist cmd_vel_msg;
         cmd_vel_msg.angular.z=0.2;
         chatter_pub.publish(cmd_vel_msg);
}
```

Якщо не потрібно повертати, то робот має рухатись по певній траєкторії, що разраховується нижче, а саме загружається позиція робота, розраховується дистанція від робота до наступної точки по траєкторії, створюються допоміжні змінні, виконується конвертування світової системи координат на систему координат роботів та адавтування системи координат, визначення наступної точки стосовно траєкторії, розрахунок необхідної швидкості робота для досягнення цієї точки.

```
else
{
        xr=msg->position.x;
        yr=msg->position.z;
        angle=-msg->orientation.y;

        double dist=fabs(xw-xr)+fabs(yw-yr);

         double xwn,ywn,xvel,yvel,xvelout,yvelout;
        geometry_msgs::Twist cmd_vel_msg;
```

```
xwn=cos(angle)*(xw-xr)+sin(angle)*(yw-yr);
ywn=-sin(angle)*(xw-xr)+cos(angle)*(yw-yr);

xvel=ywn;
yvel=-xwn;

 cmd_vel_msg.angular.z=0;

 if ((dist<=50)&&(k==3))
{
        speed=0;
}
 else
{
        speed=SPEED;
}
                if (fabs(xvel)>=fabs(yvel))
                {
                xvelout=xvel*speed/fabs(xvel);
                yvelout=yvel*speed/fabs(xvel);
                }
                else
                {
                xvelout=xvel*speed/fabs(yvel);
                yvelout=yvel*speed/fabs(yvel);
                }

                cmd_vel_msg.linear.x=xvelout;
                cmd_vel_msg.linear.y=yvelout;
```

```
                chatter_pub.publish(cmd_vel_msg);
}
```

У результаті Controller за допомою функції publish віддає повідомлення з данними для подальшого руху робота, за що вже відповідає клас RobotMoveExecutor, який в свою черзу також має callback-функцію executeCallback, яка приймає вхідні дані у вигляді повідомлення з координатами траєкторії руху роботу від Controller та виконує рух за допомогою методу OmniDrive_setVelocity, який приймає аргументами сам двигун у вигляді об’єкта (_omniDrive), координати (msg.linear.x, msg.linear.y) та кут (msg.linear.velAngle).

```
OmniDrive_setVelocity(_omniDrive, msg.linear.x, msg.linear.y,
msg.linear.velAngl);
```

3.4 Тестування розробленої програмної бібліотеки

Тестування розробленої бібліотеки можливо за допомогою розробленої програми, основні деталі розробки якої описуються нижче.

Для роботи фінального додатку та функціювання логіки усіх класів підключається бібліотека ROBOTINO API2.

```
#include <rec/robotino/api2/all.h>
using namespace rec::robotino::api2;
```

Були створені додаткові прості класи MyCom (відповідає за з’єднання з портом робота) та MyCamera (за отримання знімків камери робота).

```
// Класс MyCom
class MyCom : public Com
{
public:
MyCom()
: Com( "example_camera" )
{
}

void errorEvent( const char* errorString )
```

```
{
	std::cerr << "Error: " << errorString << std::endl;
}

void connectedEvent()
{
	std::cout << "Connected." << std::endl;
}

void connectionClosedEvent()
{
	std::cout << "Connection closed." << std::endl;
}

void logEvent( const char* message, int level )
{
	std::cout << message << std::endl;
}

void pingEvent( float timeMs )
{
	std::cout << "Ping: " << timeMs << "ms" << std::endl;
}
};


// Класс MyCamera
class MyCamera : public Camera
{
public:
MyCamera(OmniDrive* omniDrive) : _omniDrive(omniDrive)
{
}

void imageReceivedEvent( const unsigned char* data,
unsigned int dataSize,
unsigned int width,
unsigned int height,
unsigned int step )
{
//look logic in the part of the dyploma with full code of the program
}

	OmniDrive* _omniDrive = nullptr;
};
```

Точкою входу є функція main, де знаходиться старт програми. Додаток (програма) розроблема у вигляді бінарного файлу, тобто являє собою консольну программу, де все починається з роботи функції printHelp, яка вкажує користовачу, що можливо зробити (дивитись рисунок 3.2).

```
void printHelp()
{
	std::cout << "options:" << std::endl;
	std::cout << "--hostname=ipaddress     : set host to connect to" << std::endl;
	std::cout << "--ppm-binary             : output PPM binary images (default)" << 
std::endl;
	std::cout << "--ppm-plain              : output PPM plain images" << std::endl;
	std::cout << "--jpg                    : output jpg images (if available)" << std::endl;
	std::cout << "-help | --help | -h | /? : print this help page" << std::endl;
}
```

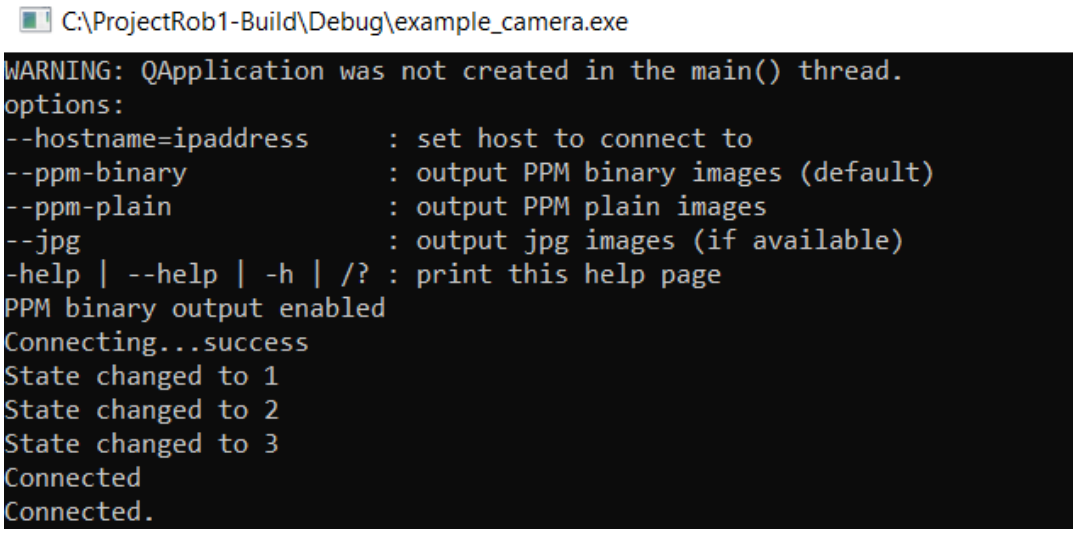

Рисунок 3.2 – Повідомлення з опціями для користувача

Далі використовуються три основні функції init(), drive() та destroy(). Функція init використовується першою, вона встановлює тип зображень отриманих об’єктом камери setJPGDecodingEnabled, встановлює з’єднання з сервером, на якому запущено симулятор з роботом.

```
void init( const std::string& hostname )
{
	// Initialize the actors
```

```
	if (JPG_IMAGE_OUTPUT == imageOutputFormat)
	{
		camera.setJPGDecodingEnabled(false);
	}

	// Connect
	std::cout << "Connecting...";
	com.setAddress( hostname.c_str() );

	com.connectToServer( true );

	if( !com.isConnected() )
	{
		std::cout << std::endl << "Could not connect to " << com.address() <<
std::endl;
#ifdef WIN32
		std::cout << "Press any key to exit..." << std::endl;
		rec::robotino::api2::waitForKey();
#endif
		rec::robotino::api2::shutdown();
		exit( 1 );
	}
	else
	{
		std::cout << "success" << std::endl;
	}
}
```

Наступна функція є найголовнішою, бо запускає в процес усю логіку, описану вище у підрозділах 3.1 - 3.2, а саме функція drive(). Вона виконує створення екземплярів класів нашої бібліотеки та запускає механізм асинхронного обміну повідомлення між цими екзмеплярами за допомою callback-функцій, який було описано раніше у декалькох розділах. Значною строчкою коду є виклик функції ros::spin(), яка запускає цикл публікацій та отримання даних у вигляді повідомлень між ImageConverter, Controller та RobotMoveExecutor.

```
void drive()
{
	while( com.isConnected() )
```

```
        {
                com.processEvents();
                ros::init(argc, argv, "localisation");

                ImageConverter ic;
                Controller controller;
                RobotMoveExecutor(com._omniDrive);

                ros::spin();

                rec::robotino::api2::msleep( 1000 );
        }
}
```

Додаток працює у циклі, поки не буде розриву зв’язку або неможливо з’єднатись (у такому випадку буде виведено про помилку з’єднання, яку зображено на рисунку 3.3) з сервером роботу або користувач самостійно не завершить роботу додатку. Під час завершення роботи додатку виконується логіка функції destoy(), яка знищує усі створені змінні для роботи додатку та звільняє всю пам'ять, яку використовую программа.

Весь код функції main() зображено нижче.

```
int main( int argc, char **argv )
{
        printHelp();
        std::string hostname = "127.0.0.1";

        for (int i = 1; i<argc; ++i)
        {
                std::string arg = argv[i];

                if ("--hostname" == arg.substr(0, 10))
                {
                        hostname = arg.substr(11,std::string::npos);
                }
                else if ("--ppm-binary" == arg.substr(0, 12))
                {
```

```
                imageOutputFormat = PPM_BINARY_IMAGE_OUTPUT;
            }
            else if ("--ppm-plain" == arg.substr(0, 11))
            {
                imageOutputFormat = PPM_PLAIN_IMAGE_OUTPUT;
            }
            else if( "--jpg" == arg.substr(0, 5) )
            {
                imageOutputFormat = JPG_IMAGE_OUTPUT;
            }
            else
            {
                printHelp();
                exit(0);
            }
        }

        switch (imageOutputFormat)
        {
        case PPM_BINARY_IMAGE_OUTPUT:
            std::cout << "PPM binary output enabled" << std::endl;
            break;

        case PPM_PLAIN_IMAGE_OUTPUT:
            std::cout << "PPM plain output enabled" << std::endl;
            break;

        default:
            std::cout << "JPG output enabled" << std::endl;
            break;
        }

#ifdef WIN32

        ::SetConsoleCtrlHandler( (PHANDLER_ROUTINE) sigint_handler, TRUE );

#else

        struct sigaction act;
        memset( &act, 0, sizeof( act ) );
        act.sa_handler = sigint_handler;

        sigaction( SIGINT, &act, NULL );

#endif
```

```
	try
	{
		init( hostname );

		drive();

		destroy();
	}
	catch( const rec::robotino::api2::RobotinoException& e )
	{
		std::cerr << "Com Error: " << e.what() << std::endl;
	}
	catch( const std::exception& e )
	{
		std::cerr << "Error: " << e.what() << std::endl;
	}
	catch( ... )
	{
		std::cerr << "Unknow Error" << std::endl;
	}

	rec::robotino::api2::shutdown();

#ifdef WIN32

	std::cout << "Press any key to exit..." << std::endl;
	rec::robotino::api2::waitForKey();

#endif
}
```

Тестування програми виконувалось у домашніх умовах з використанням програмних середовищ, як Robotino View та Robotino Sim. Для зв’язку з роботом використовувалась IP-адреса 127.0.0.1, де є два доступні порти 8080 та 12080. Програмним шляхом можливо отримати доступ до API робота тільки за допомогою другого порта 12080, бо через 8080 зв'язок неможливо було встановити.

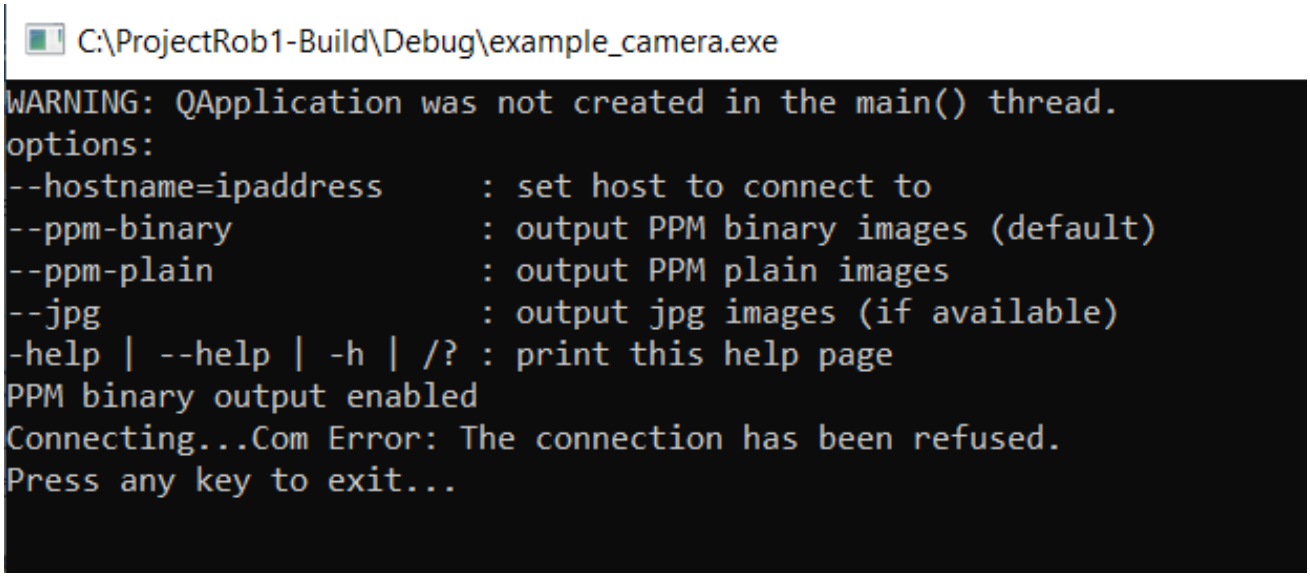

Рисунок 3.3 – Повідомлення неможливість з’єднання

Таким чином, зв'язок виходило встановити з API2 сервером, що давало змогу для керування роботом та моделюванням його дій у симуляторі. Але під чам спроб тестування виникла проблема, а саме під час підключення до робота за допомогою порта 12080 та спроби отримати зображення камери.

Доступ до камери був успішний, але по невідомій причині під час отримання зображень з другого серверу знімки, отримані програмним шляхом, були нульового розміру, що означає, що їх неможливо було використати для перевірки роботи додатку. У результаті, додаток неможливо протестувати повноцінно.

Хочу відмітити, що були спроби підключитись до камери роботу за допомогою Robotino View. По-перше, не відбувалось підключення до машини з портом 12080, результат чого можна побачити на рисунку 3.4. По-друге, підключення через 8080 порт було успішне і навіть вдалося отримати доступ до камери, результат відображено на рисунку 3.5.

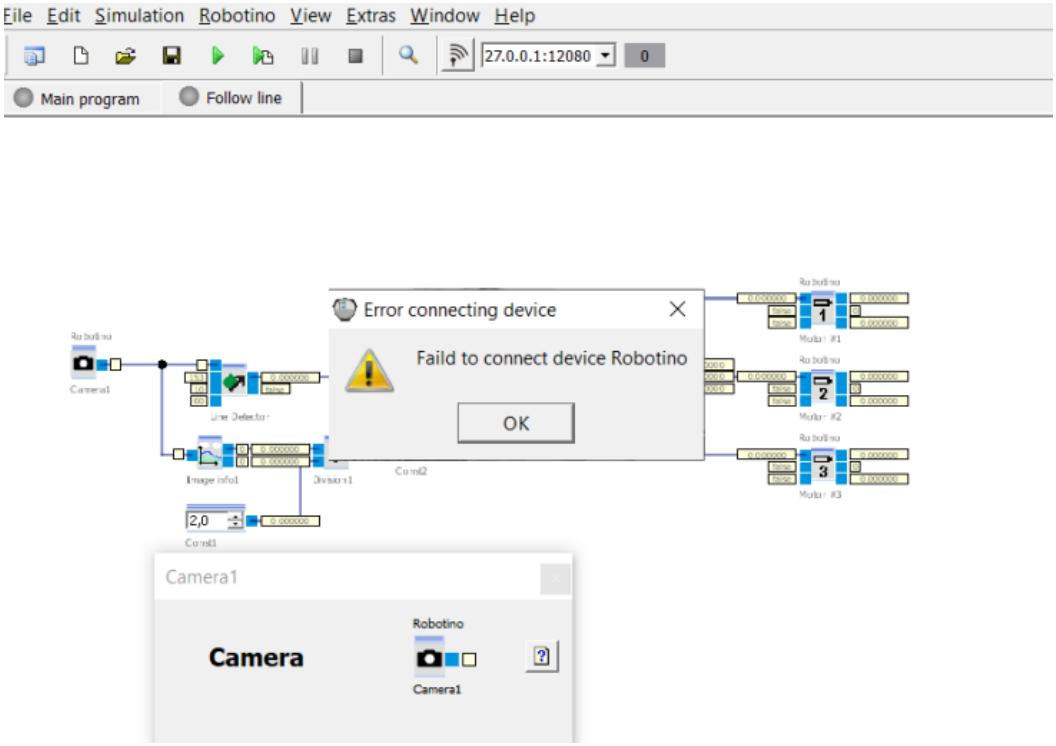

Рисунок 3.4 – Повідомлення про провал підключення до роботу з портом 12080

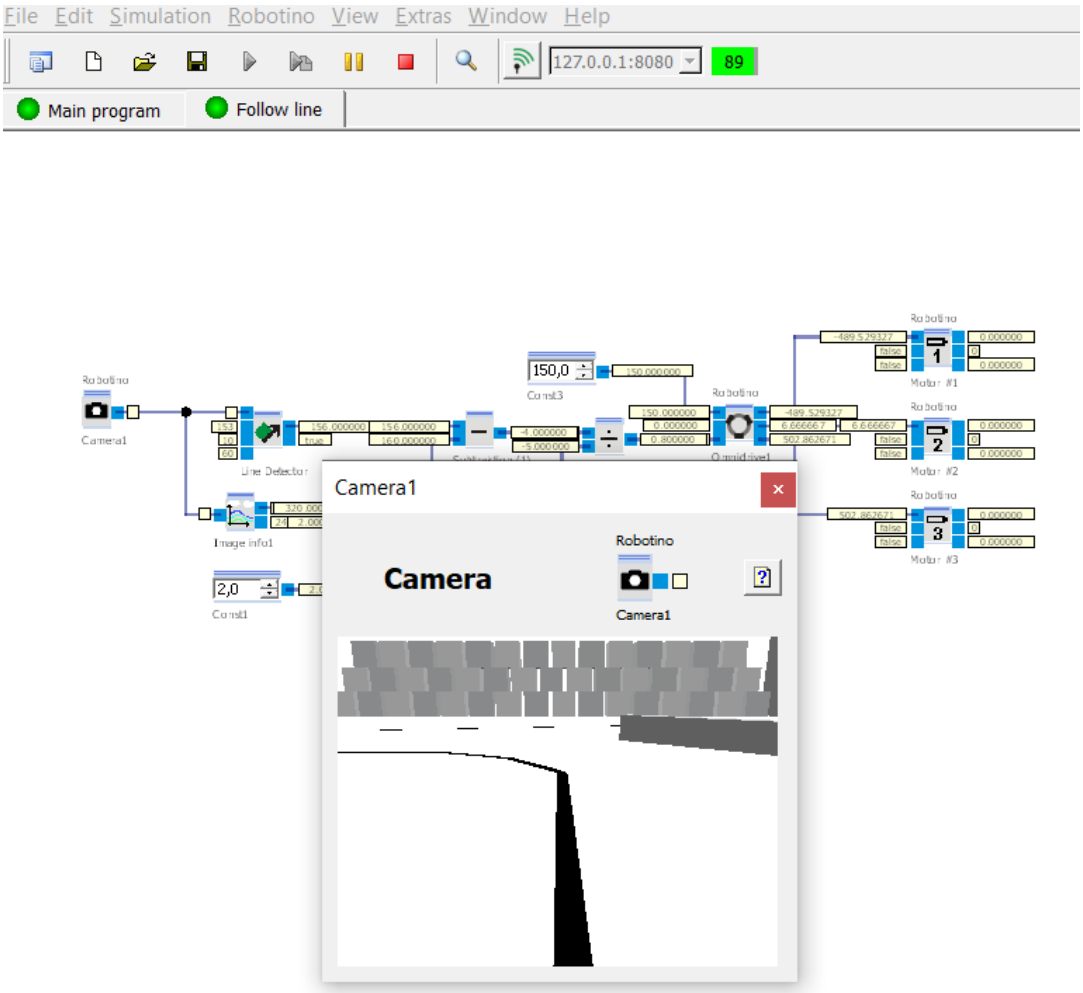

Рисунок 3.5 – Повідомлення про успішне підключення до роботу з портом 8080

3.4 Висновки до третього розділу

У третьому розділі було розглянуто розробку програмної бібліотеки для моделювання функцій сенсорної системи, а саме використання існуючих техноологій для рішення поставленої задачі, розробку алгоритму функціонування сенсорної системи, розробку функцій прикладного інтерфейсу розробника (API) та тестування розробленої програмної бібліотеки.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОХОРОНА ПРАЦІ

Приміщення із робочими місцями користувачів комп'ютерів для забезпечення електробезпеки обладнання, а також для захисту від ураження електричним струмом самих користувачів ПК повинні мати достатні технічні засоби НПАОП 40 – Правила експлуатації електрозахисних засобів, НПАОП 40 – Правила безпечної експлуатації електроустановок споживачів, НПАОП 40 – Правила будови електроустановок. Електрообладнання спеціальних установок».

З метою запобігання ушкодженням, що можуть статися через ураження електричним струмом, загоряння, коротке замикання тощо, розроблено загальний стандарт безпеки IEC 950. Загальним стандартом електробезпечності для країн Європейської співдружності є Cemark.

Сучасний рівень технічного прогресу неможливий без широкого впровадження електроустаткування, що у свою чергу викликає необхідність постійного вдосконалювання вимог до його безпечного обслуговування й засобів захисту.

Робота в області електробезпеки повинна ґрунтуватися на продуманій, чіткій, конкретній системі заходів, що забезпечує повне й точне виконання «Правил технічної експлуатації електроустановок споживачів» і «Правил безпечної експлуатації електроустановок споживачів». Особливу увагу керівники електрогосподарства повинні приділяти найсуворішому виконанню вимог зазначених Правил щодо утримування й експлуатації електричних мереж і станцій, включаючи розподільні пристрої, де за даними статистики найчастіше відбуваються нещасні випадки. Велика кількість нещасних випадків буває при обслуговуванні й ремонтах електроприводів, пускорегулюючої апаратури, електричного освітлення, зварювальних апаратів, електрифікованого транспорту, електроустаткування, піднімально-транспортних механізмів, ручного переносного електрифікованого інструменту, а також високочастотних установок.

Більша частина нещасних випадків відбувається через низький рівень організації робіт, грубих порушень Правил, у тому числі:

– безпосереднього дотику до відкритих струмоведучих частин і проводам.

– дотику до струмоведучих частин, ізоляція яких ушкоджена;

– дотику до металевих частин устаткування, що випадково під напругою;

– торкання до струмоведучих частин за допомогою предметів з низьким опором ізоляції;

– відсутності або порушення захисного заземлення;

– помилкової подачі напруги під час ремонтів або оглядів;

– впливу електричного струму через дугу;

– впливу крокової напруги й ін.

При ураженні електрострумом треба швидко вимкнути рубильник, обережно звільнити потерпілого від проводів, щоб не поширити дію струму на осіб, які беруть участь у наданні допомоги (звільнити потерпілого в гумових рукавицях або обгорнути руку сухою тканиною, стати на суху дошку чи килимок тощо).

Як установлено численними дослідженнями, небезпека електричного струму полягає в тому, що внаслідок проходження через тіло людини фібріляційного струму, зумовленого прикладанням різниць потенціалів, відбувається судорожне скорочення м'язів, у тому числі м'язів, що здійснюють дихальний рух грудної клітки, забезпечують роботу серця.

Фібріляційним струмом, що безумовно, призводить до смертельного ураження людини, вважається струм силою 0,1 А. Сила струму визначається не тільки значенням напруги, а й опором тіла людини в момент доторкання до струмоведучої частини.

Ні в якому разі не можна натискувати нижче від краю грудини на м'які тканини, цим можна пошкодити розміщені в черевній порожнині органи. Слід також уникати натискання на кінці ребер, бо це може призвести до перелому.

В оживленні беруть участь дві особи, у крайньому разі допомогу може надати й одна людина, яка по черзі проводить штучне дихання й масаж серця.

У разі ураження електричним струмом треба негайно подати потерпілому першу допомогу, бо це може закінчитись трагічно для нього. Негайно повідомити про нещасний випадок учителя або майстра, які знаходяться поблизу.

Потерпілого насамперед слід відірвати від електричних проводів. При цьому треба бути дуже обережним, бо від доторкання до потерпілого без додержання застережних заходів можна самому опинитися під дією струму.

Для відокремлення людини від електричних проводів, які перебувають під напругою, необхідно вжити таких заходів:

– вимкнути струм рубильником або викрутити запобіжну пробку, чи перерубати струможивильні проводи, при цьому кожний провід треба розрізувати окремо. Інструмент, яким розрізуєш або розрубуєш проводи, повинен бути з ручкою з ізоляційного матеріалу (сухого дерева, пластмаси, гуми та ін.);

– відірвати потерпілого від проводу, схопивши його за одяг. Перш ніж торкнутися до потерпілого, слід ізолювати себе від нього, надівши гумові рукавиці, калоші або стати на гумовий килимок, лист фанери або на сухі дерев'яні дошки. Ні в якому разі не можна ставати на вогку землю;

– якщо неможливо ізолювати себе від потерпілого, то слід відірвати його від проводу за допомогою дерев'яної дошки або палиці, діючи ними як важелями, якщо можливо, то слід вибити провід з рук потерпілого;

– можна також замкнути на коротко проводи, внаслідок чого перегорять запобіжники і в мережі зникне струм. Замикання також відбудеться, якщо накинути на голі проводи дріт або вогку ганчірку.

Останні способи можна застосовувати лише тоді, коли немає змоги вимкнути рубильник або викрутити запобіжну пробку.

Для цього слід:

– покласти потерпілого на спину, голову відхилити трохи назад, розстебнути комір, зняти пасок, одяг, що заважає рухам, а також взуття;

– за допомогою тупої викрутки, ложки або іншого подібного предмета розтиснути потерпілому зуби, видалити з рота й носа слиз та кров;

– для кращого проходження повітря до легенів потерпілого бинтом або носовою хусточкою слід притиснути йому язик до нижньої щелепи, вставивши між зуби дерев'яну паличку, або тримати язик у витягненому положенні пальцями, обгорнутими хустинкою;

– потім, стоячи позаду голови потерпілого, взяти його за зігнуті руки нижче ліктя й відвести їх від грудної клітки у сторони вгору так, щоб плечові частини рук лягли поряд з головою (вдих), тримаючи в такому положенні близько трьох секунд , відвести руки вниз, обережно притискаючи їх до грудної клітки (лічачи чотири, п'ять, шість).

Надаючи першу медичну допомогу при ураженні електрострумом. слід звернути увагу на дихання, серцево-судинну систему потерпілого.

У разі припинення дихання, серцевої діяльності слід негайно розпочати непрямий масаж серця (натискувати долонями частими поштовхами в ділянці середини грудної кістки, трохи лівіше з ритмом 40-60 поштовхів за хвилину), штучне дихання рот у рот (попередньо закрити потерпілому ніс, вдихнувши повними грудьми, видихнути крізь марлю або носовичок у рот, крізь марлю або носовичок аналогічно робити видих у ніс). Штучне дихання роблять з частотою 16-18 раз на хвилину. Одночасно викликають спеціалізовану бригаду швидкої допомоги. Непрямий масаж серця, штучне дихання роблять до повного відновлення або до надання спеціалізованої допомоги.

Пошкоджену поверхню шкіри навколо опіку треба обробити зеленкою чи рожевим розчином марганцівки, повинна бути аптечка першої лікарської допомоги.

Перша допомога буде справді корисною для потерпілого при виконанні таких правил:

– подавати допомогу тільки в нещасних випадках при хворобливому стані, що загрожує життю, а не займатися лікуванням хворого;

– з'ясовуючи характер ушкоджень, не доторкатися до ушкоджених частин тіла, не обмацувати їх, а обмежитись лише розпитуванням, оглядом;

– якщо неможливо визначити характер ушкодження, слід вважати його найтяжчим;

– піклуватися не тільки про те, щоб допомогти потерпілому, а й про те, щоб не пошкодити йому: вибирати заходи допомоги безболісні, безпечні;

– вживати тільки найнеобхідніших заходів;

– твердо пам'ятати, що перша допомога не може замінити допомогу лікаря.

Пожежа – неконтрольоване горіння поза спеціальним вогнищем, що розповсюджується у часі і просторі.

Залежно від розмірів матеріальних збитків пожежі поділяються на особливо великі (коли збитки становлять від 10000 і більше розмірів мінімальної заробітної плати) і великі (збитки сягають від 1000 до 10 000 розмірів мінімальної заробітної плати) та інші. Проте наслідки пожеж не обмежуються суто матеріальними втратами, пов'язаними зі знищенням або пошкодженням основних виробничих та невиробничих фондів, товарно-матеріальних цінностей, особистого майна населення, витратами на ліквідацію пожежі та її наслідків, на компенсацію постраждалим.

Найвідчутнішими, безперечно, є соціальні наслідки, які, передусім, пов'язуються з загибеллю і травмуванням людей, а також порушенням їх фізичного та психологічного стану, зростанням захворюваності населення, підвищенням соціальної напруги у суспільстві внаслідок втрати житлового фонду, позбавленням робочих місць тощо.

Не слід забувати й про екологічні наслідки пожеж, до яких, у першу чергу, можна віднести забруднення навколишнього середовища продуктами горіння, засобами пожежогасіння та пошкодженими матеріалами, руйнування озонового шару, втрати атмосферою кисню, теплове забруднення, посилення парникового ефекту тощо.

Цілком закономірно, що існує безпосередня зацікавленість у зниженні вірогідності виникнення пожеж і зменшенні шкоди від них. Досягнення цієї мети

є досить актуальним і складним соціально-економічним завданням, вирішенню якого повинні сприяти системи пожежної безпеки.

Пожежна безпека об'єкта – стан об'єкта, за якого з регламентованою імовірністю виключається можливість виникнення і розвитку пожежі та впливу на людей її небезпечних факторів, а також забезпечується захист матеріальних цінностей.

Основними напрямками забезпечення пожежної безпеки є усунення умов виникнення пожежі та мінімізація її наслідків.

Об'єкти повинні мати системи пожежної безпеки, спрямовані на запобігання пожежі, дії на людей та матеріальні цінності небезпечних факторів пожежі, в тому числі їх вторинних проявів. До таких факторів належать: полум'я та іскри, підвищена температура навколишнього середовища, токсичні продукти горіння й термічного розкладу матеріалів і речовин, дим, знижена концентрація кисню.

Вторинними проявами небезпечних факторів пожежі вважаються: уламки, частини зруйнованих апаратів, агрегатів, установок, конструкцій; радіоактивні та токсичні речовини і матеріали, викинуті зі зруйнованих апаратів та установок; електричний струм, пов'язаний з переходом напруги на струмопровідні елементи будівельних конструкцій, апаратів, агрегатів внаслідок пошкодження ізоляції під дією високих температур; небезпечні фактори вибухів, пов'язаних з пожежами; вогнегасні речовини.

Системи пожежної безпеки – це комплекс організаційних заходів і технічних засобів, спрямованих на запобігання пожежі та збитків від неї.

Пожежна безпека об'єкта повинна забезпечуватися системою запобігання пожежі, системою протипожежного захисту і системою організаційно-технічних заходів.

Системи пожежної безпеки мають запобігти виникненню пожежі і впливу на людей небезпечних факторів пожежі на необхідному рівні. Об'єкти, пожежі на яких можуть призвести до загибелі або масового ураження людей небезпечними факторами пожежі та їх вторинними проявами, а також до

значного пошкодження матеріальних цінностей, повинні мати системи пожежної безпеки, що забезпечують мінімально можливу імовірність виникнення пожежі.

Метою пожежної безпеки об'єкта є попередження виникнення пожежі на визначеному чинними нормативами рівні, а у випадку виникнення пожежі – обмеження її розповсюдження, своєчасне виявлення, гасіння пожежі, захист людей і матеріальних цінностей.

Основними вихідними даними при розробці комплексу технічних і організаційних рішень щодо забезпечення потрібного рівня пожежної безпеки в кожному конкретному випадку є чинна законодавча і нормативно-технічна база з питань пожежної безпеки, вибухопожежонебезпечні властивості матеріалів і речовин, що застосовуються у виробничому циклі, кількість вибухопожежонебезпечних матеріалів і речовин і особливості виробництва. На основі цих вихідних даних визначаються такі критерії вибухопожежонебезпеки об'єкта, як категорії приміщень і будівель за вибуховою і пожежною небезпекою, а також класи вибухонебезпечних і пожежонебезпечних зон. Саме залежно від категорії приміщень та будівель і класу зон за вибуховою і пожежною небезпекою, відповідно до вимог чинних нормативів, розробляються технічні і організаційні заходи і засоби забезпечення вибухопожежної безпеки.

ВИСНОВКИ

В результаті виконання атестаційної роботи було розроблено бібліотеку з вдосконаленими функціями для керування сенсорною системою робота Robotino, яка вирішує поставлені задачі.

В ході роботи також був створений бінарний додаток, за допомогою якого відбувається керування роботом, який використовує розроблену програмну бібліотеку з удосконаленими програмними засобами керування сенсорною системою мобільного роботк Festo Robotino. Було реалізовано самолокалізацію робота за допомогою алгоритму візуальної одометрії, який вдосконалюється за допомогою стратегії вибору функцій збігу.

Для досягнення поставленої мети було вирішено такі завдання:

– проаналізувано засоби моделювання та керування робототехнічними системами;

– проаналізувано особливості та структуру робота Robotino;

– проаналізувано операційні системи роботів та принципи програмного керування робототехнічними системами;

– розроблено архітектуру програмної бібліотеки для моделювання функцій робота;

– розроблено алгоритм адаптивної візуальної одометрії;

– розроблено алгоритм адаптивного пошуку та стратегію вибору функцій;

– розроблено UML-діаграми функціонування програмної бібліотеки;

– розроблено програмну бібліотеку для моделювання функцій сенсорної системи, а саме прикладний інтерфейс розробника.

Для запобігання ушкоджень та критичних ситуаций, які можуть виникати при експлуатації мобільного робота Festo Robotino було відображено правила та умови безпеки у розділа про безпеку життєдіяльності та охорону праці.

Одним із важливих аспектів є поява рухомих предметів на сцені. Для всіх розрахунків передбачалося, що вся сцена є статичною. Однак насправді часто

трапляються ситуації, коли рухомі об'єкти (наприклад, люди, машини, інші роботи тощо) є частиною зображення. Тож було б вигідно впровадити алгоритм, який може впоратися з іншими рухами у своєму оточенні. Оскільки RANSAC використовується для оцінки пози, цей алгоритм вже трохи надійний проти рухомих частин. Хоча ці частини повинні бути досить малими порівняно з рештою сцени, тому RANSAC може видалити їх як викиди.

Іншою основною проблемою зорової одометрії є зміщення розрахункового положення. Для отримання більш точних значень важливо максимально зменшити цей дрейф. Вибір лише тих кадрів, де відбувається великий рух, – це просто дуже проста техніка, і це буде працювати не у всіх ситуаціях. Кращим рішенням є отримання результатів за допомогою методу Кальмана або частинок. На основі попередніх станів (наприклад, положення, швидкість) робота, роблять припущення, де робот повинен знаходитись у певний час. Це можна порівняти з обчисленим значенням алгоритму VO, посилаючись на загальні результати. Більше того, ці літери дозволяють використовувати подальші дані сенсорів та злити їх між собою. Наприклад, у нашому випадку одометрія колеса Robotino може бути врахована також для завдання локалізації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008: 2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення [Текст]. - К.: ДП «УкрНДНЦ» , 2016. – 31 с.

2. Forschung Austria. (2016) KMU-DATEN [Електронний ресурс]. – Режим доступу: https://www.kmuforschung.ac.at/zahlen-fakten/kmu-daten// – (дата звернення: 16.10.2016). – Назва з екрану.

3. Automations Praxis. (2017) Mobile Robotik lost ¨ langwierigen Transport [Електронний ресурс]. – Режим доступу: //automationspraxis.industrie.de/servicerobotik/mobilerobotik-loest-langwierigen-transport/ – (дата звернення: 10.02.2017). – Назва з екрану.

4. Nevliudov, I. Intelligent Decision-Making Support for Flexible Integrated manufacturing [Текст] / I. Nevliudov, O. Tsymbal, A. Andrusevitch, V. Gopejenko.¬¬¬– Riga: ISMA, 2020. – 390 p.

5. Nevlyudov, I.S. Acoustic model application to mobile robot guidance [Текст] / Nevlyudov I.S., Tsimbal A.M., Milyutina S.S., Sharkovsky V.Y.– Telecommunications And Radio Engineering, 2012, v 71, No 17, P. 1589-1597.

6. Bronnikov, Artem. Flexible manufacturing tendencies and improvements with visual sensoring [Текст] / Artem Bronnikov, Nevliudov Igor, Oleksandr Tsymbal. – Eskisehir Technical University Journal of Science and Technology. Applied Sciences and Engineering, 2019. Vol. 20, ICONAT issue, P. 77-83.

7. Cross robotics and machine automation. The Difference Between AGVs and Mobile Robots [Електронний ресурс]. – Режим доступу: www.crossco.com/blog/difference-between-agvs-and-mobile-robots – (дата звернення: 5.07.2003). – Назва з екрану.

8. Бурдаков, С.Ф. Проектирование манипуляторов промышленных роботов и роботизированных комплексов [Текст]: учеб.пособие / С.Ф. Бурдаков, В.А. Дьяченко, А.Н. Тимофеев. – М.: Высшая школа, 2001. – 264 с.

9. Ковальчук, А.К. Основы теории исполнительных механизмов шагающих роботов [Текст]: учеб. / А.К. Ковальчук, Д.Б. Кулаков, Б.Б. Кулаков. – М.: Рудомино, 2010. – 170 с.

10. Волер, Крістіан. Комп’ютерне бачення 3D: ефективні методи та застосування / К. Волер [Текст]. – Лондон: Спрінгер, 2013. – 531 с.

11. Раман, С. Kamerakalibrierung / С. Раман [Текст]. – Берлин: Fabian Sinz, 2017. – 65 с.

12. Бадіно, Ернан. Бінокулярна оцінка его-руху для автомобільних застосувань. Кандидатська дисертація, Університет Гете, Франкфурт-на-Майні, 2009. [Електронний ресурс]. – Режим доступу: http://d-nb.info/992453542 – (дата звернення: 23.11.2009). – Назва з екрану.

13. Чжан, Чженю. Нова зручна техніка калібрування камери / Ч. Чжан [Текст]. – М.: Колосок, 2012. – 234 с.

14. Чжан, Чжендун. Калібрування камери з викривленням об'єктива із текстур низького рангу // Матеріали конференції IEEE 2011 р. з комп’ютерного зору та розпізнавання образів, CVPR '11, Washington, DC, США, 2011. IEEE Computer Society / Ч. Чжан [Текст]. – Лондон: Спрінгер, 2010. - 135 с.

15. Анант, Ранганатан. Алгоритм Левенберга-Маркварда. Ранганатан Анант [Текст]. – Сингапур: World Scientistic, 2004. – 387 с.

16. Буч, Г. Язык UML. Руководство пользователя / Г. Буч [Текст]. – М.: ДМК Пресс, 2007. - 489 с.

17. Каюмова, А.В. Визуальное моделирование систем в StarUML / А.В. Каюмова [Текст]. – Казань: Казанский федеральный университет, 2013. - 104 с.

18. Мова програмування С++. [Електронний ресурс]. – Режим доступу: http://progopedia.ru/language/c-plus-plus/ – (дата звернення: 10.05.2017). – Назва з екрану.

19. CMake. [Електронний ресурс]. – Режим доступу: https://cmake.org/ – (дата звернення: 16.02.2013). – Назва з екрану.

20. Середовище програмування Visual Studio. [Електронний ресурс]. – Режим доступу: https://it.ua/product/visual-studio-enterprise-MSDN/?tab=description – (дата звернення: 11.08.2015). – Назва з екрану.

21. Мова програмування С++. [Електронний ресурс]. – Режим доступу: http://progopedia.ru/language/c-plus-plus/ – (дата звернення: 02.03.2014). – Назва з екрану.

22. Основи наукових досліджень: Навч посібник / І. Ш. Невлюдов, Ю.М. Олександров, А.О. Андрусевич, О.О. Чала. – Кривий Ріг: Криворізький коледж НАУ, 2009. – 243.

23. Техніко-економічне обгрунтування інженерних рішень в автоматизованому виробництві: Підручник І.Ш. Невлюдов. – Кривий Ріг: Криворізький коледж НАУ, 2009. – 405.