Міністерство освіти і науки України
Львівський національний університет ветеринарної медицини та біотехнологій ім. С.З. Ґжицького
Факультет механіки, енергетики та інформаційних технологій
Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

другого (магістерського) рівня вищої освіти

на тему:

«Розробка віртуального маніпулятора з інтелектуальною системою зору для захоплення об’єктів»

Виконав: здобувач групи Іт-62
спеціальності 126
«Інформаційні системи та технології» Бекар Д.Р.

Керівник: Запорожцев С.Ю.

Рецензент: Семерак В.М.

ЛЬВІВ-2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Другий (магістерський) рівень вищої освіти
Спеціальність 126 – „Інформаційні системи та технології”

“ЗАТВЕРДЖУЮ”
Завідувач кафедри_____
д.т.н., проф. А.М. Тригуба
“____”_____202_ р.

ЗАВДАННЯ

на кваліфікаційну роботу здобувачу

_____Бекар Дмитро Романович _____

1. Тема роботи: __Розробка віртуального маніпулятора з інтелектуальною системою зору для захоплення об’єктів _____

Керівник роботи Запорожцев Сергій Юрійович, к.т.н., доцент.

Затверджені наказом по університету від 28 лютого 2025 року № 140/к-с

2. Строк подання здобувачем роботи 05.12.2025 р.

3. Вихідні дані до роботи: _____Загальні відомості про методи розробки віртуальних роботів-маніпуляторів, а також про симуляційні середовища розробки та системи і методи комп’ютерного зору _____

4. Зміст розрахунково-пояснювальної записки:

Вступ

1. Аналіз стану питання та постановка завдання. Основні сфери застосування роботів і тенденції розвитку. Маніпулятори: типізація, основні завдання, проблеми галузі. Використання комп’ютерного зору в робототехніці. Системи віртуального моделювання роботів. Аналіз існуючих рішень, проблеми та недоліки сучасних систем. Постановка задачі дослідження.

2. Обґрунтування та вибір інструментарію вирішення задачі. Вибір мови програмування та архітектури програмного рішення. Критерії вибору симуляційного середовища. Обґрунтування вибору PyBullet як середовища симуляції. Обґрунтування вибору підходів до керування маніпулятором. Інструменти комп’ютерного зору. Масштабування системи для реального маніпулятора.

3. Результати вирішення задачі. Підготовка середовища та моделі. Базове керування суглобами. Зворотна кінематика. Взаємодія з об’єктами у середовищі PyBullet. Використання комп’ютерного зору для визначення положення об’єктів. Можливості подальшої інтеграції комп’ютерного зору.

4. Охорона праці та безпека у надзвичайних ситуаціях.

5. Визначення ефективності інформаційної системи.
Висновки та пропозиції.
Список використаних джерел.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):___
Мета роботи та аналіз предметної області. Постановка задачі та вимоги до системи. Вибір інструментарію. Структура та архітектура програмного рішення. Базове керування маніпулятором та зворотна кінематика. Взаємодія маніпулятора з об'єктами. Використання комп'ютерного зору у симуляції. Оцінка ефективності та можливості розвитку системи. Висновки _____

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3, 5	*Запорожцев С.Ю., доцент кафедри інформаційних технологій*		
4	*Городецький І.М., доцент кафедри управління проектами та безпеки виробництва*		

7. Дата видачі завдання 28 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Приміт-ка
1	*Написання першого розділу*	28.02 - 31.03.25	
2	*Виконання другого розділу та аркушів ілюстраційного матеріалу до нього*	1.04 - 31.05.25	
3.	*Виконання третього розділу та аркушів ілюстраційного матеріалу до нього*	1.06 - 31.07.25	
4.	*Написання розділу «Охорона праці та безпека у надзвичайних ситуаціях»*	1.08 - 31.08.25	
5.	*Написання розділу «Визначення ефективності...»*	1.09 - 30.09.25	
6.	*Завершення оформлення розрахунково-пояснювальної записки та аркушів ілюстраційного матеріалу*	1.10 - 15.11.25	
7.	*Завершення роботи в цілому*	15.11 - 5.12.25	

Студент _____ Бекар Д.Р.
(підпис)

Керівник роботи _____ Запорожцев С.Ю.
(підпис)

РЕФЕРАТ

УДК 004.896:004.94

Розробка віртуального маніпулятора з інтелектуальною системою зору для захоплення об’єктів
Бекар Д.Р. Кафедра ІТ – Дубляни, ЛНУВМБ ім.С.З. Гжицького, 2025.
Кваліфікаційна робота: 85 с. текст. част., 12 рис., 11 арк. ілюстраційного матеріалу, 29 джерел.

Об’єкт дослідження – процес керування маніпулятором у симуляційному середовищі з використанням візуальної інформації.

Мета роботи – розробка віртуального маніпулятора з елементами комп’ютерного зору для задачі захоплення об’єктів у симуляційному середовищі.

Проведено аналіз предметної області, розглянуто роль комп’ютерного зору в задачах маніпулювання об’єктами, проаналізовано можливості його використання у віртуальних середовищах моделювання. Виконано обґрунтування вибору інструментарію вирішення задачі. Реалізовано модель віртуального маніпулятора в симуляційному середовищі, виконано налаштування фізичних параметрів та базових режимів роботи. Розроблено та продемонстровано сценарії взаємодії маніпулятора з об’єктами. Розглянуто можливості використання комп’ютерного зору для визначення положення об’єктів у сцені, а також описано перспективи подальшої інтеграції бібліотек комп’ютерного зору. Проаналізовані питання охорони праці та виконано оцінку економічної ефективності розробки.

Ключові слова: віртуальний робот-маніпулятор, симуляційне середовище розробки, комп’ютерний зір.

ABSTARCT

UDC 004.896:004.94

Development of a virtual manipulator with an intelligent vision system for capturing objects
Bekar D.R. Department of IT – Dublyany, Lviv NUVMB, 2025.
Qualification work: 85 p. text, 12 pict., 11 p. illustrative material, 29 sources.

The object of research is the process of controlling a manipulator in a simulation environment using visual information.

The purpose of the work is to develop a virtual manipulator with computer vision elements for the task of capturing objects in a simulation environment.

The subject area has been analyzed, the role of computer vision in object manipulation tasks has been considered, the possibilities of its use in virtual modeling environments have been analyzed. The choice of tools for solving the problem has been justified. A virtual manipulator model was implemented in a simulation environment, physical parameters and basic operating modes were configured. Scenarios of interaction of the manipulator with objects were developed and demonstrated. The possibilities of using computer vision to determine the position of objects in a scene are considered, and the prospects for further integration of computer vision libraries are described. Occupational safety issues are analyzed and an assessment of the economic efficiency of the development is performed.

Keywords: virtual robot manipulator, simulation development environment, computer vision.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface, прикладний програмний інтерфейс;

CNN – Convolutional Neural Network, згорткові нейронні мережи;

CPU - Central Processing Unit, центральний процесор;

GPU - Graphics Processing Unit, графічний процесор;

GPS – Global Positioning System, глобальна система позиціонування;

RGB – Red-Green-Blue Color System, кольорова система Червоний-Синій-Зелений;

ML – Machine Learning, машинне навчання;

AI – Artificial Intelligence, штучний інтелект;

GUI – Graphic User Interface, графічний інтерфейс користувача;

IK - Inverse Kinematics, зворотна кінематика.

ЗМІСТ

ВСТУП

Сучасний етап розвитку робототехніки характеризується стрімким зростанням ролі інтелектуальних систем, здатних взаємодіяти з навколишнім середовищем у напівавтоматичному або повністю автономному режимі. Одним із ключових напрямів у цій галузі є розробка роботизованих маніпуляторів, призначених для виконання операцій захоплення, переміщення та маніпулювання об’єктами різної форми та властивостей. Ефективність таких систем значною мірою визначається рівнем їх адаптивності та здатністю сприймати просторову інформацію, що зумовлює зростаючий інтерес до інтеграції технологій комп’ютерного зору у структуру керування маніпуляторами.

Традиційно тестування алгоритмів керування роботами здійснюється на фізичних прототипах, однак такий підхід супроводжується значними витратами ресурсів, підвищеними ризиками пошкодження обладнання та обмеженими можливостями масштабування експериментів. У зв’язку з цим актуалізується використання симуляційних середовищ, що дозволяють створювати віртуальні моделі роботів, відтворювати фізичні процеси та моделювати різноманітні сценарії взаємодії з об’єктами у контрольованих умовах. Віртуальні маніпулятори стають важливим інструментом для дослідження алгоритмів керування, кінематики, планування руху та інтеграції сенсорних підсистем без необхідності безпосереднього використання реального устаткування.

Особливу увагу в даній роботі приділено поєднанню симуляційного моделювання з елементами комп’ютерного зору як засобу визначення положення об’єктів у середовищі та формування основ для автономного захоплення. Такий підхід дозволяє не лише підвищити реалізм моделі, але й наблизити її до умов, у яких функціонують реальні роботизовані системи. Використання візуальної інформації для орієнтації маніпулятора в просторі

відкриває можливості для розробки більш гнучких та інтелектуальних алгоритмів взаємодії з об’єктами.

Актуальність теми зумовлена необхідністю створення універсальних навчально-дослідницьких платформ, які дозволяють моделювати процеси керування маніпуляторами, тестувати різні підходи до обробки візуальних даних та аналізувати поведінку системи у динамічному середовищі. Розробка віртуального маніпулятора з системою зору є важливим кроком у напрямку формування комплексних робототехнічних рішень, які в перспективі можуть бути адаптовані до реальних промислових, логістичних або сервісних застосувань.

Метою даної роботи є розробка віртуального маніпулятора з елементами комп’ютерного зору для задачі захоплення об’єктів у симуляційному середовищі. Об’єктом дослідження є процес керування маніпулятором у симуляційному середовищі з використанням візуальної інформації. Предметом дослідження є методи і засоби реалізації віртуальної системи керування маніпулятором із елементами комп’ютерного зору для задачі захоплення об’єктів.

Практична значущість роботи полягає у створенні прототипу програмної системи, яка може бути використана як основа для подальших досліджень у галузі робототехніки, навчальних цілей або майбутньої інтеграції з реальними апаратними платформами.

Таким чином, робота спрямована на комплексне дослідження процесу створення віртуального маніпулятора з системою зору, що поєднує сучасні підходи до симуляційного моделювання та інтелектуальної обробки даних, і має на меті продемонструвати можливості такого підходу для вирішення задач захоплення об’єктів у віртуальному середовищі.

РОЗДІЛ 1
АНАЛІЗ СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Основні сфери застосування роботів і тенденції розвитку

Сучасна робототехніка охоплює широкий спектр напрямів і стрімко розвивається під впливом досягнень у галузях інформаційних технологій, штучного інтелекту, сенсорики та мехатроніки. [1-5] Роботи вже давно перестали бути лише частиною промислового виробництва - вони дедалі активніше інтегруються у повсякденне життя, науку, медицину, транспортну та сервісну сфери. Зростання обчислювальних потужностей, поява доступних сенсорів і розвиток алгоритмів навчання дозволили створювати інтелектуальні системи, здатні аналізувати навколишнє середовище, приймати рішення та виконувати завдання з високим ступенем автономності.

Промислові роботи залишаються основою світового ринку робототехніки. Вони виконують завдання, пов’язані зі зварюванням, складанням, пакуванням, фарбуванням, транспортуванням та контролем якості. Використання таких систем значно підвищує ефективність виробничих процесів, знижує витрати на ручну працю та забезпечує стабільну якість продукції. Сучасні промислові маніпулятори можуть працювати цілодобово, відрізняються високою точністю позиціонування, мають сенсори для контролю сили дотику та вібрацій, а також оснащуються адаптивними алгоритмами для роботи з різними об’єктами.

Важливою тенденцією є розвиток колаборативних роботів, або коботів, які здатні безпечно взаємодіяти з людиною без жорстких огороджень. Завдяки сенсорним системам, вбудованому комп’ютерному зору та алгоритмам уникнення зіткнень коботи можуть працювати пліч-о-пліч із працівниками, виконуючи завдання, що потребують гнучкості й інтелектуальної підтримки. Це відкриває нові можливості для малих і середніх підприємств, де раніше впровадження робототехніки було економічно невигідним.

Розвиток мобільної робототехніки дозволив створити автономні системи, які орієнтуються в просторі, будують карти приміщень, планують маршрути та реагують на перешкоди. Автономні мобільні роботи використовуються у складських комплексах, лікарнях, аеропортах, логістичних центрах. Вони виконують перевезення вантажів, доставку медикаментів, моніторинг середовища, зменшуючи потребу в людських ресурсах. Такі системи часто базуються на комбінації комп'ютерного зору, лазерних далекомірів і глибоких нейронних мереж для прийняття рішень у реальному часі.

Медична робототехніка також демонструє стрімкий прогрес. Роботизовані хірургічні комплекси, як-от «da Vinci», забезпечують високоточні операції, мінімізуючи людський фактор і покращуючи відновлення пацієнтів. У протезуванні використовуються біомеханічні маніпулятори з елементами машинного навчання, здатні адаптуватися до рухів користувача. Роботи-реабілітанти допомагають відновлювати рухові функції, підтримуючи фізичну активність пацієнтів. У майбутньому очікується ще тісніше поєднання біоінженерії, сенсорики та штучного інтелекту, що дозволить створювати персоналізовані медичні рішення.

Окремий напрям становить аграрна робототехніка. У сільському господарстві роботи виконують завдання зі збору врожаю, обприскування, моніторингу стану ґрунтів і рослин. Використання безпілотних літальних апаратів і наземних роботів дозволяє проводити точне землеробство, зменшувати кількість використаних ресурсів і мінімізувати екологічний вплив. Такі системи часто інтегруються з геоінформаційними технологіями, що забезпечує точне позиціонування і контроль за кожною ділянкою поля.

У сфері транспорту відбувається поступовий перехід до автономних транспортних засобів. Автомобільні компанії активно впроваджують роботизовані системи керування, які використовують комбінацію сенсорів, камер, лідарів і алгоритмів глибинного навчання для розпізнавання об'єктів, прогнозування траєкторій та ухвалення рішень у складних дорожніх ситуаціях.

Такі технології поступово переходять із експериментальної стадії у практичну, зокрема у сфері логістики та міського транспорту.

Важливу роль у розвитку робототехніки відіграє сфера освіти та науки. Навчальні комплекси, симулятори та відкриті платформи дозволяють студентам і дослідникам експериментувати з алгоритмами керування, вивчати поведінку автономних систем, тестувати методи машинного зору та оптимізації. Університети активно використовують середовища, такі як Gazebo, V-REP (CoppeliaSim), Webots, а також бібліотеки Python для навчання моделей і симуляції руху без фізичного обладнання. Це робить робототехніку доступною для широкого кола дослідників і сприяє швидкому обміну знаннями.

Глобальні тенденції розвитку робототехніки визначаються трьома ключовими напрямами: підвищення автономності, розвиток інтелектуальних систем керування та інтеграція з технологіями Інтернету речей. Роботи поступово переходять від виконання програмованих дій до самонавчальних систем, здатних аналізувати досвід і приймати рішення на основі отриманих даних. Хмарні сервіси забезпечують об’єднання роботів у мережі, що дозволяє колективне навчання й обмін інформацією між різними пристроями.

Водночас відбувається перехід до використання штучного інтелекту в поєднанні з адаптивними механічними системами. Це дає можливість створювати роботів, які не просто діють за заданими алгоритмами, а й розуміють контекст, прогнозують поведінку об’єктів і навчаються у процесі взаємодії з навколишнім середовищем. Розвиток глибокого навчання, систем візуального сприйняття та нейромережевих контролерів ставить нові вимоги до апаратного забезпечення, проте водночас робить робототехніку ближчою до природної поведінки живих організмів.

Отже, сучасна робототехніка - це міждисциплінарна галузь, у якій поєднано знання з механіки, кібернетики, штучного інтелекту, програмування та сенсорики. Її розвиток спрямований на створення гнучких, надійних і розумних систем, здатних виконувати складні завдання без постійного

втручання людини. Головними тенденціями є автономність, співпраця між людиною і машиною, а також розширення можливостей за рахунок інтеграції з віртуальними моделями та системами штучного інтелекту. Ці процеси формують основу для подальших досліджень у напрямі інтелектуальних маніпуляторів із комп'ютерним зором, що стають важливою частиною сучасної науково-технічної еволюції.

1.2. Маніпулятори: типізація, основні завдання, проблеми галузі

Маніпулятори становлять один із найдавніших і водночас найважливіших типів роботів, які заклали основу розвитку всієї робототехніки. [6-9] Їхнє головне призначення полягає у виконанні дій, подібних до рухів людської руки: захоплення, переміщення, позиціонування, складання та обробка об'єктів у просторі. На відміну від автономних мобільних систем, маніпулятори працюють у фіксованій або обмеженій робочій зоні, проте забезпечують високу точність і повторюваність рухів, що робить їх незамінними у промисловому виробництві, наукових лабораторіях та сервісних сферах.

Типова конструкція маніпулятора складається з бази, ланок, з'єднаних шарнірними або поступальними зчленуваннями, та виконавчого органу, який може бути оснащений захоплювачем, інструментом чи сенсорним модулем. Кожне зчленування забезпечує один ступінь вільності, і загальна кількість ступенів визначає можливості маніпулятора у просторі. Поширеними є моделі з шістьма ступенями, які орієнтують робочий інструмент у будь-якому напрямі.

За конструктивними особливостями маніпулятори поділяють на кілька основних типів. Картазіанські мають три поступальні осі, що дозволяє виконувати рухи у прямокутній системі координат; вони прості в керуванні, але обмежені у маневреності. Циліндричні та сферичні моделі використовуються там, де необхідна робота навколо центральної осі або в замкнутому об'ємі. Найбільш універсальними вважаються шарнірно-зчленовані (або

антропоморфні) маніпулятори, які імітують рухи людської руки. Такі роботи можуть виконувати складні просторові маніпуляції, зокрема в умовах тісного простору. Окремо виділяють SCARA-маніпулятори, що поєднують високу швидкість і жорсткість конструкції, які застосовуються для швидкісного пакування, сортування та складання дрібних об’єктів.

Маніпулятори розрізняються також за типом приводу. Електричні системи найбільш поширені через простоту інтеграції з цифровими контролерами та відносно невисоку вартість обслуговування. Гідравлічні приводи забезпечують велику силу та момент, тому використовуються у важких промислових і будівельних роботах. Пневматичні системи характеризуються швидкодією, але меншою точністю, що обмежує сферу їхнього застосування. Останнім часом зростає інтерес до “м’яких” роботів, де рухи створюються за рахунок еластичних матеріалів і пневматичних камер, що дозволяє безпечно взаємодіяти з людиною та делікатними об’єктами.

Основні завдання маніпуляторів визначаються потребами виробництва або досліджень. У промисловості вони виконують транспортні операції, складання компонентів, зварювання, фарбування, полірування, обробку поверхонь, контроль якості продукції. У лабораторних умовах застосовуються для роботи з токсичними або радіоактивними матеріалами, точного дозування речовин, збирання мікросхем і біологічних зразків. У сервісній сфері маніпулятори входять до складу систем доставки, обслуговування, а також роботів-помічників, здатних виконувати побутові дії.

У наукових і навчальних цілях маніпулятори відіграють ключову роль у вивченні кінематики, динаміки, зворотного зв’язку та систем керування. Саме через них студенти і дослідники знайомляться з принципами робототехніки, моделюють різні сценарії руху та навчаються проектувати алгоритми позиціонування. Завдяки відкритим бібліотекам, як-от ROS (Robot Operating System), PyRobot, MoveIt або RoboDK API, можна створювати симуляції руху,

налаштовувати траєкторії та тестувати системи керування у віртуальному середовищі без фізичного обладнання.

Попри широке застосування, галузь має низку проблем, які гальмують подальший розвиток. Серед них - складність програмування та налаштування, висока вартість промислових систем і недостатня гнучкість при зміні виробничих завдань. Багато роботів потребують ручного калібрування, точного налаштування траєкторій і ретельного визначення обмежень робочого простору. Це робить процес інтеграції у виробництво тривалим і дорогим. Крім того, механічні системи часто мають обмежені можливості сприйняття середовища, що ускладнює їхню взаємодію з іншими об’єктами або людьми.

Важливою проблемою залишається адаптація роботів до змінних умов. Без системи зору чи тактильного зворотного зв’язку маніпулятор не здатен оцінювати, наскільки успішно він виконав дію. Тому розробка інтегрованих сенсорних систем, здатних забезпечити машинне сприйняття, є одним із головних напрямів сучасних досліджень. У поєднанні з методами штучного інтелекту це відкриває шлях до створення адаптивних маніпуляторів, що можуть навчатися на власному досвіді.

Ще одна тенденція - перехід до віртуального моделювання та симуляційного навчання. Замість того щоб працювати з дорогими фізичними пристроями, дослідники створюють цифрові копії роботів, які дозволяють відпрацьовувати алгоритми руху, керування та візуального сприйняття в безпечному середовищі. Це не лише здешевлює розробку, а й прискорює тестування, даючи змогу відтворювати тисячі сценаріїв за короткий час.

Таким чином, маніпулятори залишаються центральним елементом робототехніки, що поєднує інженерію, кібернетику та штучний інтелект. Їх розвиток поступово переходить від механічної точності до інтелектуальної адаптивності, коли система не просто виконує команди, а розуміє контекст дій, оцінює результати і здатна навчатися. Саме інтеграція комп’ютерного зору,

сенсорних технологій і машинного навчання визначає майбутнє роботів та формує основу для створення віртуальних маніпуляторів нового покоління.

1.3. Використання комп'ютерного зору в робототехніці

Розвиток комп'ютерного зору став одним із ключових чинників, що визначив сучасний рівень автономності робототехнічних систем. Якщо на початкових етапах роботи роботів базувалися переважно на простих сенсорах положення та дистанційного контролю, то сьогодні візуальне сприйняття стало центральною складовою їх функціонування. Комп'ютерний зір дозволяє машині не лише сприймати навколишнє середовище, а й аналізувати об'єкти, розпізнавати сцени, просторові структури, оцінювати відстань, визначати рух і навіть прогнозувати подальші зміни в оточенні. [10-13] Для роботів це означає можливість працювати в динамічних умовах без постійного контролю людини.

У промисловості системи машинного зору застосовуються для перевірки якості виробів, орієнтації деталей на конвеєрі, контролю збірних процесів та калібрування маніпуляторів. У сервісній та мобільній робототехніці вони дозволяють орієнтуватися в просторі, уникати перешкод, розпізнавати користувачів і навіть реагувати на жести або емоції. Важливим напрямом стала інтеграція комп'ютерного зору з роботами, що працюють у середовищах, небезпечних або недоступних для людини, наприклад у підводних або космічних умовах. Завдяки розвитку доступних відеокамер, сенсорів глибини та відкритих бібліотек для обробки зображень такі системи стали значно ближчими до широкого використання.

Класичні методи комп'ютерного зору базувалися на геометричних і статистичних алгоритмах. Вони передбачали попередню обробку зображення, виділення контурів, пошук ключових точок, зіставлення шаблонів і подальше визначення об'єктів за формою або кольором. Типовими прикладами є методи виявлення контурів Кенні, алгоритм Хафа для визначення ліній і кіл, дескриптори SIFT або SURF для виявлення характерних ознак. Ці методи

дозволяли створювати достатньо ефективні системи, проте вони були чутливими до освітлення, шумів і змін ракурсу. Для їх успішної роботи необхідне ретельне налаштування параметрів, а точність часто обмежувалась складністю сцени.

З появою глибоких нейронних мереж комп'ютерний зір зазнав якісного стрибка. Моделі на основі згорткових нейронних мереж (CNN) продемонстрували здатність самостійно виділяти ознаки зображення без ручного втручання. Такі архітектури, як AlexNet, VGG, ResNet, MobileNet, стали основою сучасних систем розпізнавання. Завдяки великим обсягам навчальних даних та використанню GPU вдалося досягти точності, що перевищує можливості традиційних методів. Нейромережеві моделі здатні не лише класифікувати об'єкти, а й локалізувати їх на зображенні, створюючи теплові карти й обмежувальні рамки, що є критично важливим для застосування у робототехніці.

Для задач локалізації та сегментації об'єктів у кадрі використовуються мережі YOLO, SSD та Faster R-CNN, які поєднують швидкість обробки з високою точністю. Для роботів, що працюють у реальному часі, особливо важливими є моделі, оптимізовані для швидкої обробки зображень на слабких пристроях, наприклад YOLOv5n або MobileNet-SSD. Саме такі рішення дозволяють впроваджувати системи зору навіть на вбудованих мікроконтролерах або одноплатних комп'ютерах. У поєднанні з алгоритмами відстеження рухомих об'єктів (наприклад, SORT або DeepSORT) вони забезпечують можливість динамічного аналізу сцени.

Важливу роль відіграють і засоби збору та підготовки даних. Для навчання систем зору використовуються великі відкриті набори зображень - COCO, ImageNet, Pascal VOC, Open Images, які містять сотні тисяч об'єктів із різними класами, ракурсами та умовами зйомки. Для робототехніки існують і більш спеціалізовані датасети, такі як KITTI або RoboNet, що враховують параметри просторового положення об'єктів. В умовах відсутності фізичного обладнання

навчання часто проводять у віртуальному середовищі, де можливо генерувати синтетичні зображення з точно відомими координатами об’єктів, що спрощує побудову моделей зору.

Завдяки комп’ютерному зору роботи отримують здатність не лише бачити, а й приймати рішення на основі зорових даних. Це дає змогу реалізовувати адаптивні системи керування, де рух маніпулятора чи мобільної платформи залежить від результатів аналізу відеопотоку. Такий підхід формує основу інтерактивної робототехніки, у якій візуальні спостереження безпосередньо впливають на поведінку пристрою. Зокрема, система може визначати положення цілі, оцінювати траєкторію її руху та в реальному часі коригувати власну дію.

Подальший розвиток галузі пов’язаний із поєднанням нейромережевого зору з іншими сенсорними системами - лідаром, ультразвуковими датчиками, GPS і системами інерціального вимірювання. Така мультисенсорна інтеграція дозволяє отримати більш надійну та повну інформацію про навколишнє середовище. Одночасно триває активне вдосконалення архітектур, що дозволяють зменшити обчислювальну складність моделей і розміщувати їх безпосередньо на борту роботів. Сучасні тенденції спрямовані також на розвиток алгоритмів самоадаптації, які можуть покращувати власну роботу без повторного навчання, використовуючи накопичений досвід взаємодії з середовищем.

Таким чином, комп’ютерний зір перетворився на один із ключових інструментів інтелектуальної робототехніки. Він поєднує методи штучного інтелекту, аналітики зображень і керування рухом, створюючи основу для нових поколінь автономних систем, здатних сприймати, аналізувати та взаємодіяти зі світом подібно до людини.

1.4. Системи віртуального моделювання роботів

Віртуальне моделювання стало одним із найважливіших інструментів сучасної робототехніки, що забезпечує можливість дослідження, проектування та тестування роботів без необхідності використання реального обладнання.

Воно дозволяє проводити експерименти у безпечному цифровому середовищі, відтворюючи фізику руху, властивості матеріалів, взаємодію між об’єктами та поведінку сенсорів. Такий підхід значно скорочує витрати на розробку, прискорює перевірку гіпотез і відкриває доступ до робототехнічних досліджень навіть за відсутності фізичних пристроїв. [14-16]

Суть симуляційного підходу полягає у створенні цифрової копії робота, його робочого простору та середовища, з яким він взаємодіє. Моделювання включає опис кінематики, динаміки, систем керування, датчиків і виконавчих механізмів. Це дає змогу не лише перевірити правильність алгоритмів, а й дослідити ефективність рухів, оптимізувати траєкторії або оцінити роботу систем зору. Віртуальне середовище дозволяє змінювати параметри експерименту, вводити нові перешкоди чи завдання, що було б складно або дорого реалізувати в реальності.

Одним із найпоширеніших середовищ для моделювання роботів є Gazebo, який часто використовується разом із фреймворком ROS (Robot Operating System). [17] Gazebo надає реалістичну фізичну симуляцію, підтримує динаміку твердих тіл, тертя, зіткнення та візуалізацію в режимі реального часу. Він дозволяє підключати моделі роботів, задавати сценарії руху, вбудовувати віртуальні камери, лідари, датчики глибини та інші сенсори. Інтеграція з ROS робить можливим перевірку алгоритмів навігації, маніпуляції або зорового керування ще до використання на фізичному пристрої.

Іншим важливим середовищем є Webots [18], орієнтований на освітні та дослідницькі цілі. Його перевага полягає у простоті створення сцен, візуалізації та доступності готових моделей роботів. Webots підтримує Python як основну мову сценаріїв, що робить його зручним для користувачів, знайомих із цією мовою. В середовищі можна моделювати роботів із різною кількістю ступенів свободи, підключати камери, симулювати освітлення, створювати сценарії взаємодії з об’єктами. Розробники активно підтримують відкриту спільноту, а

результати експериментів можна виводити у форматах, сумісних з TensorFlow або PyTorch, що полегшує інтеграцію з нейромережевими системами.

Значну роль відіграють і сучасні графічні рушії, адаптовані під робототехнічне моделювання. Наприклад, NVIDIA Isaac Sim, побудований на базі рушія Omniverse, забезпечує фотореалістичну графіку, підтримку фізики, сенсорів і алгоритмів машинного навчання. [19] Він дозволяє створювати синтетичні дані для тренування моделей комп'ютерного зору, що є надзвичайно цінним для глибоких нейронних мереж. Завдяки GPU-прискоренню та інтеграції з Python цей симулятор дає можливість швидко генерувати великі обсяги даних із точними мітками, що в реальному середовищі часто є неможливим.

Для завдань навчання роботів взаємодії з об'єктами активно використовується середовище PyBullet [20] - легке, відкрите й орієнтоване на фізично достовірні симуляції. PyBullet дозволяє створювати роботів, моделювати зчеплення, зіткнення, гравітацію, а також проводити навчання з підкріпленням (reinforcement learning). Його часто застосовують у поєднанні з TensorFlow або Stable Baselines для тренування агентів у різних сценаріях. Завдяки простоті й сумісності з Google Colab PyBullet став одним із найзручніших інструментів для досліджень у робототехніці без використання реального обладнання.

Віртуальні середовища також виконують роль навчальних лабораторій. Вони дозволяють студентам і дослідникам без ризику пошкодження техніки відпрацьовувати принципи керування, алгоритми руху та методи зорової локалізації. Це особливо важливо для освітніх програм, де кількість фізичних роботів обмежена або взагалі відсутня. За допомогою таких платформ можна відтворити експерименти, змінювати параметри середовища та одразу спостерігати наслідки. У поєднанні з Python та бібліотеками машинного навчання віртуальні симуляції дають змогу створювати комплексні експерименти, у яких алгоритм сприймає зображення, аналізує його й управляє віртуальним маніпулятором.

Окрему увагу слід приділити перевагам використання Google Colab як середовища для експериментів. [21] Його безкоштовна інфраструктура дозволяє запускати обчислення на GPU, працювати з Python, бібліотеками OpenCV, PyTorch, TensorFlow, а також підключати віртуальні симулятори через API. У такому форматі дослідження може проводитися навіть без потужного локального комп'ютера, що робить технологію доступною для більшості студентів і дослідників. Колаб дозволяє інтегрувати код, результати симуляцій і візуалізації в одному робочому документі, що зручно для створення наукових звітів або публікацій.

Віртуальне моделювання також має важливе значення для перевірки алгоритмів машинного навчання. Оскільки у реальному середовищі процес збору даних може бути дорогим або тривалим, симулятори дозволяють генерувати дані автоматично, варіюючи умови освітлення, положення об'єктів або шум у сенсорах. Це робить можливим попереднє навчання моделей у симуляції з подальшою адаптацією в реальному світі - підхід, відомий як sim-to-real transfer. Його застосування дозволяє скоротити час і витрати на підготовку роботів до практичного використання.

Таким чином, системи віртуального моделювання стали невід'ємною частиною досліджень і розробки в робототехніці. Вони поєднують фізичну достовірність, гнучкість програмного налаштування й можливості інтеграції з інструментами машинного навчання. Використання таких середовищ, як Gazebo, Webots, PyBullet чи Isaac Sim, відкриває широкі можливості для дослідження, навчання та створення інтелектуальних роботів без необхідності мати фізичне обладнання, що особливо важливо для академічних і експериментальних проектів.

1.5. Аналіз існуючих рішень, проблеми та недоліки сучасних систем

Сучасний ландшафт рішень у сфері маніпуляторів із системами комп'ютерного зору характеризується великою різноманітністю підходів, від

індустріальних комерційних продуктів до відкритих академічних проєктів. У комерційній галузі представлені готові промислові маніпулятори з інтегрованими системами зору та алгоритмами контролю, що пропонують надійність і сертифікацію для застосування у виробництві. Такі рішення часто поєднують апаратні модулі високої якості, пропрієтарні середовища для налаштування і служби підтримки, проте вони часто є дорогими та закритими, що обмежує їхню адаптацію для дослідницьких або освітніх задач. Паралельно з цим розвиваються відкриті ініціативи та фреймворки, які роблять робототехніку більш доступною: ROS і супутні пакети, відкриті симулятори, бібліотеки для зору й навчання, що дозволяють дослідникам швидко прототипувати й обмінюватися результатами. Академічні роботи пропонують нові алгоритми для локалізації, сегментації та планування захвату, часто демонструючи суттєві досягнення в контролі маніпуляцій на синтетичних та реальних датасетах. Проте між теоретичними результатами та надійною промисловою експлуатацією залишається прогалина, яка обумовлена низкою системних обмежень. [22-23]

Одна з ключових проблем полягає у складності адаптації моделей зору до реальних умов. Багато наукових досліджень показують високу точність алгоритмів на відмічених датасетах або у контрольованих експериментах, тимчасом як при перенесенні в нові умови, з іншим освітленням, фоном або типом об’єктів точність значно знижується. Це явище зумовлене як обмеженістю тренувальних даних, так і великою варіативністю зовнішніх факторів. Рішення на базі глибокого навчання залежні від великих та різноманітних наборів зображень з якісними розмітками, а збір реальних даних з точними мітками для маніпуляцій є дорогою та трудомісткою задачею. Хоча синтетичні дані, згенеровані в симуляторах, частково вирішують цю проблему, питання sim-to-real transfer лишається відкритим: моделі, натреновані на синтетиці, часто потребують додаткової адаптації або донавчання на реальних прикладах, аби досягти стабільної роботи.

Інша суттєва проблема - це затримки й обчислювальна складність. Для забезпечення надійного візуального контролю в реальному часі необхідні моделі, що поєднують високу швидкість обробки з надійністю локалізації. Багато сучасних архітектур забезпечують високу точність, але потребують значних обчислювальних ресурсів, що не завжди сумісно з обмеженими обчислювальними платформами на борту робота. Хмарні обчислення можуть частково вирішити це питання, однак залучають додаткові ризики у вигляді затримок мережі, проблем з безпекою даних і залежності від зв'язку. У результаті індустрія прагне знайти компроміс між легкістю моделей та їхньою ефективністю, що призвело до появи оптимізованих архітектур і методів квантизації, проте це також часто супроводжується втратою точності.

Ще однією проблемою є інтеграція з існуючими системами керування й безпеки. Виробничі середовища мають жорсткі вимоги до надійності, сертифікації та гарантій безпечної роботи поруч із людьми. Рішення на основі машинного зору повинні працювати узгоджено з контролерами руху, сенсорними мережами і механізмами аварійного відключення. В реальних умовах відмови сенсорів, некоректні детекції або непередбачувані відхилення траєкторій можуть призводити до аварійних ситуацій. Тому багато практичних рішень включають багаторівневі перевірки та резервні механізми, що підвищує складність системи й вимагає додаткових ресурсів на інтеграцію та валідацію.

Проблеми зі стандартизацією та сумісністю також значущі. Різні виробники апаратури пропонують власні протоколи, формати даних та інтерфейси, що ускладнює створення універсальних рішень. Хоча ROS значно сприяв уніфікації, існує велика кількість версій, пакетів і суміжних інструментів, що потребують синхронізації. Це ускладнює перенесення розробок між проєктами і вимагає додаткового часу на адаптацію.

В науковому середовищі активно досліджуються підходи до підвищення надійності та автономності. Одним із напрямів є розробка методів активного зору, коли робот програмно змінює положення сенсора або освітлення для

покращення спостереження, а також комбінування зору з іншими сенсорами для мультисенсорної оцінки. Іншою важливою лінією досліджень є використання методів підкріплення для навчання стратегій захвату і руху, що дає змогу агентам самостійно відкривати ефективні поведінкові механізми. Хоча ці підходи демонструють значний потенціал у лабораторних умовах, вони потребують великої кількості тренувальних епізодів і ретельного налаштування винагород, що часто робить їх важкими для прямого застосування у виробництві.

Масштабною проблемою виступає також верифікація і валідація складних AI-систем. Переконатися в коректності поведінки нейромережевих контролерів у всіх можливих сценаріях практично неможливо, що породжує питання відповідальності і безпеки. Інструменти формальної верифікації активно розвиваються, але вони ще не досягли ступеня зрілості, необхідного для широкого використання в індустрії. Це створює бар'єри для впровадження нових алгоритмів у критичних галузях, таких як медицина або авіація.

Нарешті, варто відзначити питання людського фактора: адаптація робітників до співпраці з роботами вимагає навчання та зміни процесів організації праці. Недостатня увага до ергономіки взаємодії та інтерфейсів може знизити ефективність впровадження навіть технічно вдосконалених рішень. Соціальні та етичні аспекти автоматизації також впливають на прийняття технологій у суспільстві та на рівень інвестицій.

Підсумовуючи, існуючі рішення і наукові розробки у сфері маніпуляторів з комп’ютерним зором демонструють значний прогрес, але стикаються з низкою практичних обмежень. Недостатня адаптивність до реальних умов, обчислювальні обмеження, складність інтеграції, проблеми верифікації й людський фактор залишаються важливими перешкодами. У зв’язку з цим перспективними є міждисциплінарні підходи, які поєднують оптимізовані алгоритми з надійною апаратурою, симуляційними методами для підготовки

даних і ретельною інженерією інтеграції, що дозволить наблизити наукові досягнення до реальної експлуатації.

Постановка задачі дослідження

Проведений аналіз показав, що сучасна робототехніка активно інтегрує методи комп’ютерного зору та машинного навчання для підвищення автономності й точності дій роботів. Проте, попри значний прогрес у цій сфері, залишається проблема доступності та простоти систем, які можна використовувати для навчання, досліджень або первинного проектування роботизованих маніпуляторів із функцією візуального сприйняття. Більшість промислових рішень є закритими, а апаратні комплекси потребують значних фінансових і технічних ресурсів. Це ускладнює процес підготовки майбутніх фахівців і стримує експериментальні дослідження в навчальних умовах.

Тому виникає необхідність створення віртуальної системи, яка б дозволяла досліджувати взаємодію роботизованого маніпулятора з навколишнім середовищем, реалізовувати базові алгоритми управління і використовувати нейронні мережі для розпізнавання об’єктів без потреби у фізичному обладнанні. Така система має поєднувати симуляційне середовище та алгоритми комп’ютерного зору, що забезпечить гнучкість, наочність і зручність у використанні.

Метою дослідження є створення прототипу віртуального маніпулятора з елементами комп’ютерного зору, здатного ідентифікувати об’єкти та виконувати базові маніпуляційні дії. Для досягнення цієї мети передбачається розробка моделі маніпулятора, реалізація візуального модуля на основі готових нейронних мереж і створення програмного середовища для взаємодії між ними. Практична значимість полягає у можливості використання отриманої системи як навчально-дослідницької платформи, що сприяє кращому розумінню процесів інтеграції штучного інтелекту в робототехніку.

РОЗДІЛ 2
ОБҐРУНТУВАННЯ ТА ВИБІР ІНСТРУМЕНТАРІЮ ВИРІШЕННЯ ЗАДАЧІ

2.1. Вибір мови програмування та архітектури програмного рішення

Вибір мови програмування та архітектури програмного рішення є ключовим етапом при проєктуванні системи віртуального маніпулятора з елементами комп’ютерного зору, оскільки саме ці компоненти визначають гнучкість, масштабованість, зручність розробки та подальший потенціал розвитку створеної системи. З урахуванням навчально-дослідного характеру роботи, орієнтації на симуляторне середовище та необхідності інтеграції різнорівневих алгоритмів керування, моделювання та аналізу, критично важливим стає забезпечення балансу між функціональністю, доступністю та зрозумілістю програмної реалізації.

Мова Python була обрана як основна платформа реалізації програмного комплексу завдяки її універсальності, широкій підтримці у сфері робототехніки та розвинутій екосистемі бібліотек. [24-25] Важливим фактором стала її тісна інтеграція з PyBullet, що дозволяє безпосередньо керувати симуляційним середовищем, отримувати дані про стан маніпулятора, координати об’єктів та параметри фізичної взаємодії в режимі реального часу. Python забезпечує зручний синтаксис, високу читабельність коду та можливість швидкого прототипування, що особливо актуально для експериментальної розробки, де алгоритмічні рішення часто уточнюються й адаптуються у процесі тестування.

Застосування Python також сприяє реалізації модульного підходу, при якому окремі функціональні блоки системи можуть розроблятися та вдосконалюватися незалежно. Це створює умови для логічного поділу програмного комплексу на рівні керування рухом, взаємодії з середовищем, обробки сенсорних даних та логіки прийняття рішень. Такий підхід забезпечує

прозору структуру коду, полегшує процес налагодження та дозволяє поступово ускладнювати систему без необхідності радикальної перебудови існуючої архітектури.

Архітектура програмного рішення будується за принципом функціонально розподіленої структури, в якій симулятор відіграє роль інтерактивного середовища, тоді як логіка керування реалізується окремими програмними модулями. Центральним елементом виступає керуючий модуль, який відповідає за ініціалізацію симуляції, завантаження моделей, конфігурацію параметрів сцени та запуск основного циклу взаємодії. У межах цього циклу здійснюється зчитування поточного стану маніпулятора, обробка інформації щодо положення об’єктів і формування команд управління, що передаються до симулятора для виконання відповідних рухів.

Така структура дозволяє чітко розмежувати симуляційний шар і рівень логіки керування, що є важливим з точки зору масштабування системи. У перспективі це відкриває можливість заміни або доповнення симулятора реальним апаратним інтерфейсом без суттєвих змін загальної концепції програмної побудови. Відповідно, розроблена архітектура не лише задовольняє поточні потреби віртуального середовища, а й формує основу для подальшої інтеграції з фізичними робототехнічними системами.

Значну увагу приділено побудові послідовного механізму обробки подій та керування часовими аспектами симуляції. Архітектура враховує необхідність синхронізації між візуалізацією, фізичними обчисленнями та виконанням алгоритмів керування, що дозволяє уникнути некоректної поведінки моделі та забезпечити стабільність експериментів. Логіка циклічного виконання програмних процедур забезпечує відтворюваність результатів та контроль над перебігом симуляції, що має важливе значення для аналізу ефективності реалізованих підходів.

Окремою перевагою обраної мовної платформи є її сумісність із засобами наукових обчислень і візуалізації, що дозволяє органічно поєднувати процес

моделювання з аналітичними інструментами. Це створює умови для глибшого дослідження поведінки маніпулятора, аналізу траєкторій, оцінки точності позиціонування та виявлення закономірностей у процесах взаємодії з об’єктами.

Таким чином, вибір Python як мови програмування та модульної архітектури як основи програмного рішення зумовлений прагненням забезпечити прозору, гнучку та масштабовану структуру системи, здатну ефективно підтримувати симуляцію процесів захоплення об’єктів у віртуальному середовищі. Такий підхід відповідає сучасним тенденціям у сфері робототехнічного моделювання та створює передумови для подальшого розвитку програмного комплексу з урахуванням нових функціональних вимог.

2.2. Критерії вибору симуляційного середовища

Проєктування віртуального маніпулятора з елементами комп’ютерного зору передбачає використання симуляційного середовища як базової платформи для відпрацювання алгоритмів керування, перевірки коректності математичних моделей та дослідження поведінки системи в умовах, максимально наближених до реальних. [26] Саме симуляційне середовище виступає тією ланкою, що поєднує теоретичну модель із практичною реалізацією, забезпечує можливість багаторазового повторення експериментів та дозволяє варіювати параметри системи без ризику пошкодження апаратної частини. Тому обґрунтування вибору такого середовища повинно базуватися не на популярності або зручності конкретного інструменту, а на комплексі функціональних, технічних та науково-методичних критеріїв.

Одним з ключових факторів є адекватність фізичної моделі. Симуляційне середовище має забезпечувати коректне моделювання динаміки твердих тіл, урахування сил гравітації, інерції, тертя, зіткнень та інших фізичних процесів, що безпосередньо впливають на поведінку маніпулятора при взаємодії з

об’єктами. Від якості фізичного рушія залежить достовірність отриманих результатів і можливість екстраполяції поведінки системи на реальний робототехнічний комплекс у майбутньому. При цьому важливою є стабільність чисельних розрахунків і можливість тонкого налаштування параметрів фізичної моделі для досягнення балансу між точністю та обчислювальною швидкістю.

Не менш вагомим критерієм виступає підтримка кінематичних моделей і засобів розрахунку прямої та зворотної кінематики. Для задач керування маніпулятором принципове значення має можливість задавати положення кінцевого ефектора в робочому просторі з подальшим обчисленням відповідних кутів повороту суглобів. Наявність вбудованих механізмів інверсної кінематики дозволяє істотно спростити архітектуру програмного рішення та зосередитися на логіці керування, а не на складності математичних перетворень. Водночас симулятор повинен забезпечувати доступ до низькорівневих параметрів суглобів, таких як положення, швидкість і момент, що відкриває можливості для реалізації різних стратегій керування.

Важливу роль відіграє інтегрованість із мовами програмування. Оскільки більшість сучасних алгоритмів управління та обробки даних реалізуються із застосуванням високорівневих мов, симуляційне середовище повинно мати зручний програмний інтерфейс для взаємодії з користувацьким кодом. Особливо цінною є підтримка мови Python як де-факто стандарту в галузі прототипування роботизованих систем та наукових досліджень. Простота написання коду, доступність бібліотек, а також можливість швидкого налагодження та тестування суттєво підвищують ефективність розробки і зменшують час на реалізацію алгоритмів.

Окремим аспектом є продуктивність симуляції та її масштабованість. Середовище має забезпечувати достатню швидкість оновлення сцени, щоб у реальному часі відображати рухи маніпулятора та результат взаємодії з об’єктами. Це особливо актуально при реалізації інтерактивних експериментів, де затримки у відображенні можуть спотворювати сприйняття процесу та

ускладнювати аналіз. Можливість роботи як у режимі реального часу, так і в прискореному режимі також є перевагою, оскільки дозволяє оперативно проводити серії експериментів і накопичувати статистичні дані.

Суттєвим критерієм є гнучкість симуляційного середовища щодо налаштування сцени. Система повинна дозволяти створювати та модифікувати віртуальне оточення, додавати об’єкти, змінювати їх фізичні властивості та параметри взаємодії. Це дає змогу відтворювати різні сценарії експлуатації маніпулятора та аналізувати поведінку системи в умовах, що варіюються за складністю. Така гнучкість сприяє побудові універсальної моделі, здатної адаптуватися до нових задач без суттєвого перепроєктування.

З точки зору наукових досліджень важливою є також відкритість і документованість платформи. Доступ до технічної документації, прикладів використання та активної спільноти розробників значно спрощує процес освоєння середовища й знижує поріг входження. Це забезпечує можливість глибшого розуміння внутрішніх механізмів симуляції та сприяє коректній інтерпретації результатів експериментів.

Не менш важливою є стабільність програмного середовища та передбачуваність його поведінки. При багаторазовому повторенні експериментів результати повинні бути відтворюваними за незмінних початкових умов. Це принципово для наукової роботи, де кожен етап повинен мати можливість перевірки та повторення.

Таким чином, вибір симуляційного середовища для задач керування маніпулятором має ґрунтуватися на сукупності критеріїв, що охоплюють фізичну достовірність моделі, програмну інтегрованість, продуктивність, гнучкість налаштування та науково-методичну і практичну доцільність. Саме комплексний підхід до оцінювання таких параметрів дозволяє створити ефективну основу для подальшого проєктування системи та реалізації алгоритмів керування у межах віртуального моделювання.

2.3. Обґрунтування вибору PyBullet як середовища симуляції

Обґрунтування вибору PyBullet як основного середовища симуляції для реалізації віртуального маніпулятора базується на поєднанні функціональних можливостей, технічної доступності та відповідності поставленій дослідницькій меті, що полягає у моделюванні процесів захоплення об’єктів із використанням алгоритмів керування та елементів комп’ютерного зору. У даному проєкті пріоритет надається не фотореалістичності середовища, а точності фізичної взаємодії, коректності кінематичних обчислень і можливості інтеграції з алгоритмічними модулями, орієнтованими на аналіз сцен та прийняття рішень. Саме в цьому контексті PyBullet демонструє оптимальний баланс між складністю, продуктивністю та функціональною достатністю.

PyBullet є легковаговим фізичним рушієм із відкритим вихідним кодом, який підтримує моделювання твердих тіл, обмежень, контактної взаємодії, гравітації, тертя та інерційних характеристик об’єктів. Його висока адаптивність до задач робототехнічного моделювання зумовлена наявністю вбудованих механізмів для роботи з кінематичними ланцюгами, підтримкою опису роботів у форматі URDF та реалізацією зворотної кінематики на рівні середовища. Це дозволяє зосередити основну увагу дослідження на алгоритмах керування, логіці взаємодії з об’єктами та інтеграції візуального аналізу без необхідності глибокого втручання у низькорівневі фізичні моделі.

Важливою перевагою PyBullet є його тісна інтеграція з мовою Python, що забезпечує зручний доступ до функцій симуляції, гнучкість у налаштуванні параметрів та можливість швидкого прототипування. Це особливо актуально в межах навчально-дослідної роботи, де важливо не лише досягти працездатного результату, а й забезпечити прозорість алгоритмічних рішень, можливість експериментування та корекції програмної логіки без значних витрат часу. Інтерпретована природа Python спрощує процес відлагодження, аналізу

помилок та поступового нарощування функціональності системи, що є критично важливим при побудові складних інтерактивних моделей.

З точки зору візуалізації PyBullet надає достатній інструментарій для контролю перебігу симуляції, спостереження за положенням об’єктів, траєкторією руху маніпулятора та результатами взаємодії з середовищем. Використання вбудованого графічного інтерфейсу дозволяє оперативно оцінювати коректність поведінки системи на різних етапах розробки, що сприяє формуванню інтуїтивного уявлення про роботу алгоритмів. Одночасно середовище підтримує режими без візуалізації, що відкриває можливість використання симуляції для масових експериментів, автоматизованого тестування або подальшого впровадження методів машинного навчання.

Ще одним аргументом на користь PyBullet є його широка популярність у науковій спільноті та наявність значної кількості прикладів, документації і дослідницьких проєктів, у яких це середовище використовується для моделювання роботизованих систем різного рівня складності. Завдяки цьому створюється методологічна база, що дозволяє орієнтуватися на вже апробовані підходи, адаптувати їх до конкретної задачі та забезпечити узгодженість результатів із сучасними тенденціями у сфері симуляційної робототехніки. Крім того, відкритість платформи сприяє можливості її модифікації, аналізу внутрішніх механізмів та розширення функціональності відповідно до потреб дослідження.

У порівнянні з альтернативними середовищами, які часто вимагають складної інсталяції, великих обчислювальних ресурсів або орієнтовані переважно на ігрову індустрію, PyBullet демонструє практичну доцільність для академічних задач, де першочергову роль відіграють точність фізичного моделювання та алгоритмічна прозорість. Його використання не потребує спеціалізованого обладнання, що розширює доступність програмного комплексу та дозволяє відтворювати експерименти на стандартних персональних комп’ютерах.

Таким чином, вибір PyBullet як основного середовища симуляції у даній роботі є логічно обґрунтованим, оскільки воно забезпечує необхідний рівень фізичної достовірності, простоту інтеграції з програмними засобами керування, підтримку базових кінематичних операцій та сприятливі умови для подальшого розвитку системи. Це створює надійну основу для дослідження процесів захоплення об’єктів у віртуальному просторі та формування підходів, які в перспективі можуть бути адаптовані для використання у реальних робототехнічних комплексах.

2.4. Обґрунтування вибору підходів до керування маніпулятором

Вибір підходів до керування маніпулятором є ключовим етапом проєктування системи, оскільки від нього безпосередньо залежить точність, плавність та надійність рухів кінцівок у симуляційному середовищі. [27] Для вирішення цієї задачі було проаналізовано існуючі методи керування, що широко застосовуються в робототехніці, зокрема класичні алгоритми позиційного та швидкісного контролю, підходи з використанням зворотної кінематики та сучасні методи оптимізації траєкторій.

У межах симуляційного середовища PyBullet ефективним виявився режим POSITION_CONTROL, який забезпечує управління кутами суглобів маніпулятора, дозволяючи точно задавати бажану позицію кінцівок і водночас враховувати фізичні обмеження, такі як межі обертання суглобів та максимальні сили приводу. Цей підхід дозволяє реалізовувати плавні, контрольовані рухи, що особливо важливо при демонстрації маніпуляцій з об’єктами та оцінці коректності моделі фізичної взаємодії у віртуальному просторі.

Для забезпечення інтуїтивного та гнучкого керування кінцевим ефектором використовувався метод обчислення зворотної кінематики. Даний підхід дозволяє розраховувати необхідні кути суглобів для досягнення заданої позиції кінцівки у тривимірному просторі без потреби ручного підбору параметрів. Це

суттєво спрощує реалізацію сценаріїв переміщення та маніпуляцій із об’єктами, оскільки алгоритм автоматично підбирає оптимальні значення для всіх суглобів, враховуючи задані обмеження та фізичні параметри моделі. Застосування зворотної кінематики сприяє зменшенню часу на підготовку експериментальних сценаріїв та підвищує відтворюваність результатів симуляції.

Особлива увага приділялася також інтеграції простих алгоритмів контролю кінцевого ефектора з циклічною обробкою команд у головному циклі симуляції. Це забезпечує синхронізацію рухів маніпулятора з обробкою даних про положення об’єктів і дозволяє уникнути конфліктів між різними алгоритмами керування. Також така організація коду дає змогу підключати додаткові алгоритми, наприклад, автоматичне позиціонування об’єктів або використання простих стратегій “захоплення-відпускання”, без необхідності кардинальної перебудови архітектури програми.

Вибір саме позиційного керування у поєднанні з зворотною кінематикою обумовлений прагненням забезпечити баланс між простотою реалізації та гнучкістю управління. Це дозволяє ефективно демонструвати роботу маніпулятора в симуляторі, а також формує основу для подальшого ускладнення сценаріїв, таких як багатокрокові маніпуляції, переміщення декількох об’єктів одночасно або інтеграція з алгоритмами комп’ютерного зору.

Крім того, застосування даного підходу дозволяє проводити візуальну оцінку точності позиціонування, що є важливим критерієм для науково-дослідного характеру роботи. Користувач може спостерігати за рухом кінцівки, оцінювати плавність рухів, перевіряти дотримання обмежень та коректність виконання команд, що є необхідним для формування достовірних висновків про поведінку системи.

В підсумку можна сказати, що обґрунтований вибір підходів до керування маніпулятором забезпечує поєднання точності, надійності та гнучкості,

дозволяючи реалізовувати основні завдання роботи: управління рухом суглобів, переміщення кінцевого ефектора у просторі та взаємодію з об’єктами. Такий підхід закладає основу для подальшого розвитку системи, інтеграції більш складних алгоритмів контролю та комп’ютерного зору, а також потенційної адаптації до реального апаратного маніпулятора.

2.5. Інструменти комп’ютерного зору

Інтеграція комп’ютерного зору у системи керування роботизованими маніпуляторами є важливим аспектом, що дозволяє підвищити автономність, точність та адаптивність робота у взаємодії з об’єктами у тривимірному просторі. У межах даної роботи використання комп’ютерного зору розглядається переважно як перспектива розвитку, що може бути реалізована як у віртуальному симуляторі, так і у подальшому на реальному маніпуляторі. Основна ціль полягає у забезпеченні можливості визначення положення об’єктів у просторі та подальшого керування рухом кінцевого ефектора без ручного завдання координат.

Комп’ютерний зір в контексті симуляторного середовища можна розглядати на кількох рівнях. По-перше, це отримання зображень сцени з віртуальної камери, яка може бути розташована у довільній точці простору або на самій кінцівці маніпулятора. Віртуальні камери, що підтримуються у PyBullet, дозволяють генерувати RGB-зображення, глибину та маски об’єктів, що відкриває широкі можливості для моделювання алгоритмів обробки даних, типових для комп’ютерного зору. Використання таких зображень дозволяє тренувати алгоритми визначення координат об’єктів та оцінювати їх точність у контрольованих умовах, не залучаючи фізичний апарат, що особливо важливо на ранніх етапах дослідження.

По-друге, серед ключових інструментів комп’ютерного зору, що можуть бути застосовані у симуляції, слід виділити класичні методи обробки зображень, включаючи сегментацію за кольором, порогування, виділення

контурів та використання морфологічних операцій для покращення точності визначення об’єктів. Наприклад, виділення куба певного кольору може бути реалізовано через переведення RGB-зображення у колірний простір HSV та визначення пікселів, що відповідають заданому діапазону відтінків. Такі методи дозволяють створювати базові функції визначення положення об’єкта у пікселях, що у поєднанні з відомими параметрами камери або геометрією сцени дає змогу приблизно реконструювати координати у тривимірному просторі.

По-третє, більш просунуті підходи передбачають використання алгоритмів глибокого навчання, зокрема детекторів об’єктів та сегментаційних нейронних мереж. У віртуальному середовищі це може бути реалізовано через рендеринг сцени та генерацію навчальних датасетів з контрольованими мітками об’єктів. Такий підхід дозволяє навчати моделі визначати положення об’єктів за зображеннями без фізичного доступу до сенсорів реального робота, забезпечуючи високу точність та стабільність алгоритмів перед інтеграцією на апаратний маніпулятор.

Важливим аспектом інтеграції комп’ютерного зору у симулятор є синхронізація з основним циклом керування маніпулятором. Це дозволяє здійснювати обробку кадрів у режимі реального часу та видавати команди руху кінцевому ефектору на основі актуальних даних про положення об’єкта. У віртуальному середовищі така синхронізація є значно простішою, оскільки можна одержувати ідеальні дані з глибини та масок, проте принципи організації циклу обробки та керування залишаються такими ж, як і для реальної системи.

Крім того, у симуляторі реалізовано можливість тестування алгоритмів комп’ютерного зору за різних умов освітлення, перспективи камери та положення об’єктів. Це дозволяє оцінити стійкість алгоритмів, ідентифікувати слабкі місця та визначити оптимальні параметри обробки зображень, що особливо важливо для майбутньої інтеграції на реальному маніпуляторі. Наприклад, можна варіювати положення камери, відстань до об’єкта,

орієнтацію сцени та проводити порівняльний аналіз результатів роботи алгоритму з відомими координатами об’єктів.

Загалом, інструменти комп’ютерного зору у контексті симуляторного середовища дозволяють створювати основу для автономного керування, забезпечують можливість тестування та налаштування алгоритмів без ризику пошкодження фізичного обладнання і формують платформу для подальшого розширення системи. Їх використання у PyBullet може бути обмежене лише програмними ресурсами та швидкістю обробки кадрів, однак навіть базові алгоритми сегментації та визначення координат об’єктів відкривають широкі перспективи для розвитку автономних маніпуляцій та інтеграції складніших сценаріїв захоплення та переміщення об’єктів.

Таким чином, розгляд інструментів комп’ютерного зору у віртуальному середовищі формує основу для подальшої інтеграції повноцінних сенсорних систем у роботизовану платформу та дозволяє оцінювати потенційну ефективність алгоритмів перед їх використанням на реальному апараті, що є важливим етапом підготовки до масштабування та підвищення автономності системи.

2.6. Масштабування системи для реального маніпулятора

Перехід від віртуального симуляційного середовища до впровадження системи керування на реальному роботизованому маніпуляторі є закономірним етапом розвитку досліджуваної технології та вимагає комплексного осмислення як технічних, так і методологічних аспектів масштабування. Симуляція розглядається як базова платформа для апробації алгоритмів, однак практична реалізація передбачає врахування значно ширшого спектра факторів, що пов’язані з фізичними обмеженнями, нестабільністю реального середовища та необхідністю інтеграції апаратних компонентів. [28]

Однією з ключових особливостей масштабування є відмінність між ідеалізованими умовами симуляції та реальною поведінкою механічних систем.

У симуляторі динаміка руху маніпулятора визначається математичними моделями, що лише наближено відображають реальність, тоді як у фізичному пристрої проявляються ефекти тертя, люфтів, інерційності, затримок у керуванні та неточностей датчиків. Тому алгоритми, що демонструють стабільну роботу у віртуальному середовищі, потребують додаткової адаптації та калібрування під конкретні апаратні характеристики реального маніпулятора.

Важливою складовою масштабування є інтеграція сенсорних систем, зокрема камер, енкодерів, датчиків сили та положення, які забезпечують зворотний зв’язок у режимі реального часу. На відміну від симуляції, де дані про положення об’єктів та стан маніпулятора можуть бути отримані безпосередньо з моделі, у реальній системі ці дані формуються шляхом обробки сигналів з фізичних сенсорів, що супроводжується похибками та шумами. Відтак виникає необхідність застосування алгоритмів фільтрації, компенсації похибок та прогнозування стану системи, що ускладнює загальну архітектуру програмного рішення.

Масштабування також передбачає зміну вимог до швидкодії та надійності системи. У віртуальному середовищі виконання алгоритмів може бути менш критичним до часових затримок, тоді як у реальному маніпуляторі будь-яка затримка в обробці даних або передачі команд безпосередньо впливає на точність позиціонування та безпечність роботи. Це вимагає оптимізації програмного коду, використання ефективних алгоритмів планування траєкторій та впровадження систем реального часу, здатних гарантувати передбачувану поведінку маніпулятора.

Окрему увагу слід приділити питанням безпеки при взаємодії робота з навколишнім середовищем. Якщо у симуляції помилки в керуванні не несуть фізичних наслідків, то у реальній системі вони можуть призводити до пошкодження обладнання або створення небезпеки для оператора. Тому масштабування передбачає впровадження механізмів аварійного зупинення,

обмеження швидкості руху, контроль граничних положень та систем моніторингу стану маніпулятора, що істотно підвищує надійність функціонування системи.

Перспективним напрямом масштабування є поетапний перехід від симуляції до реального робота через так званий підхід sim-to-real, коли програмні алгоритми спочатку тестуються у віртуальному середовищі, після чого поступово адаптуються до фізичного пристрою. Такий підхід дозволяє мінімізувати ризики та скоротити витрати на розробку, оскільки більшість помилок виявляються ще на етапі моделювання. Крім того, використання доменової рандомізації у симуляції, зміна параметрів середовища та варіація умов дозволяють підвищити узагальнювальну здатність алгоритмів та підготувати їх до непередбачуваних ситуацій у реальному середовищі.

Масштабування системи також відкриває можливості для розширення функціональності маніпулятора, зокрема реалізації складніших сценаріїв взаємодії з об’єктами, автономного захоплення, сортування або виконання технологічних операцій. Інтеграція комп’ютерного зору з системами керування дає змогу створити гнучкі рішення, здатні адаптуватися до змін у робочому просторі та виконувати завдання без жорстко заданих координат. Це, у свою чергу, сприяє переходу від експериментальних прототипів до практичних індустріальних застосувань.

Таким чином, перспективи масштабування системи для реального маніпулятора полягають не лише у перенесенні програмного рішення з симуляції на фізичний пристрій, а й у комплексному переосмисленні архітектури, методів керування та способів взаємодії з середовищем. Реалізація такого переходу забезпечує підвищення практичної цінності дослідження, формує основу для подальшої комерціалізації та впровадження роботизованих систем у виробничі та сервісні процеси, а також відкриває нові горизонти для подальших наукових і прикладних розробок у сфері робототехніки та комп’ютерного зору.

Висновки до розділу

В результаті виконання даного розділу вирішені наступні питання:

- розглянуто мову програмування та архітектуру програмного рішення, зупиненося на Python як зручному інструменті для прототипування та швидкої розробки, визначено загальні принципи побудови модульної та масштабованої структури програми;

- проведено аналіз критеріїв вибору симуляційного середовища для задач керування маніпулятором, визначено основні параметри та вимоги до такого середовища, зокрема підтримку фізичних моделей, можливості реалізації зворотної кінематики та інтеграції з інструментами комп'ютерного зору;

- обґрунтовано вибір PyBullet як основного середовища симуляції, підкреслено його переваги в контексті відкритого коду, наявності готових моделей маніпуляторів, можливостей розширення та інтеграції з Python;

- обґрунтовано підходи до керування маніпулятором, включно з позиційним керуванням, використанням зворотної кінематики та основних алгоритмів планування траєкторій у віртуальному середовищі;

- здійснено огляд інструментів комп'ютерного зору, оцінено можливості їх інтеграції у симуляційне середовище, а також визначено потенціал для подальшого розвитку системи з урахуванням обробки зображень і визначення положення об'єктів;

- визначено перспективи масштабування системи для роботи на реальному маніпуляторі, підкреслено необхідність адаптації алгоритмів, інтеграції сенсорних систем та забезпечення безпеки та надійності при роботі з фізичним обладнанням.

РОЗДІЛ 3
РЕЗУЛЬТАТИ ВИРІШЕННЯ ЗАДАЧІ

3.1. Підготовка середовища та моделі

Підготовка середовища та моделі є першим і найважливішим етапом практичної частини, оскільки від коректності її реалізації залежить подальше тестування та керування маніпулятором. На початковому етапі було встановлено середовище Python та бібліотеку PyBullet у середовищі розробки PyCharm на операційній системі Windows. PyBullet обрано завдяки її відкритості, доступності та широкому набору функцій для роботи з фізичною симуляцією роботів і взаємодії з об’єктами у віртуальному середовищі.

Після встановлення бібліотеки здійснено підключення до симуляції за допомогою функції p.connect(p.GUI), що дозволило отримати графічне вікно для візуалізації сцени (рис.3.1). Відповідний код наданий в Додатку А.

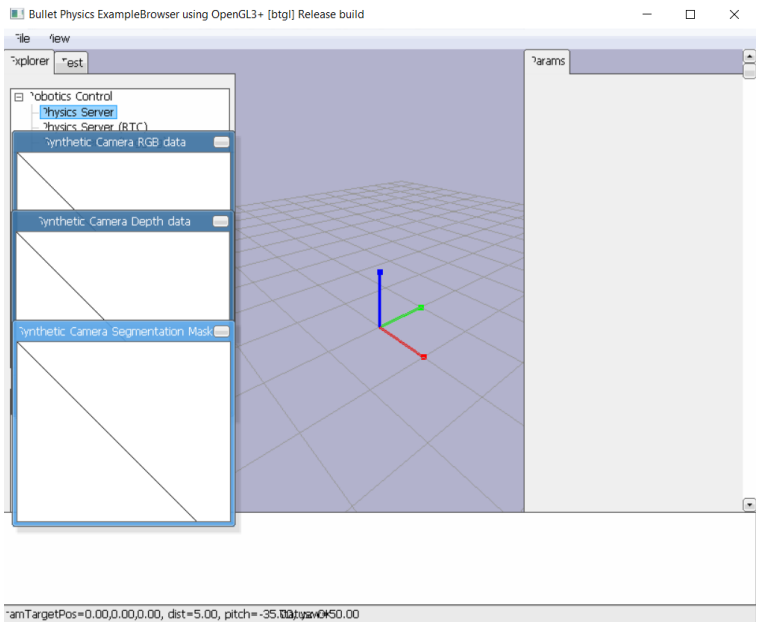

Рисунок 3.1 - GUI PyBullet з порожньою сценою

Встановлення гравітації у напрямку осі Z здійснювалось через p.setGravity(0, 0, -9.81), що забезпечує фізично коректне падіння об’єктів та їх взаємодію. Далі додано стандартну підлогу за допомогою URDF-файлу plane.urdf, завантаженого через додатковий шлях pybullet_data.getDataPath(). Це створило базову поверхню для розміщення маніпулятора та інших об’єктів (рис.3.2). Відповідний код - в Додатку Б.

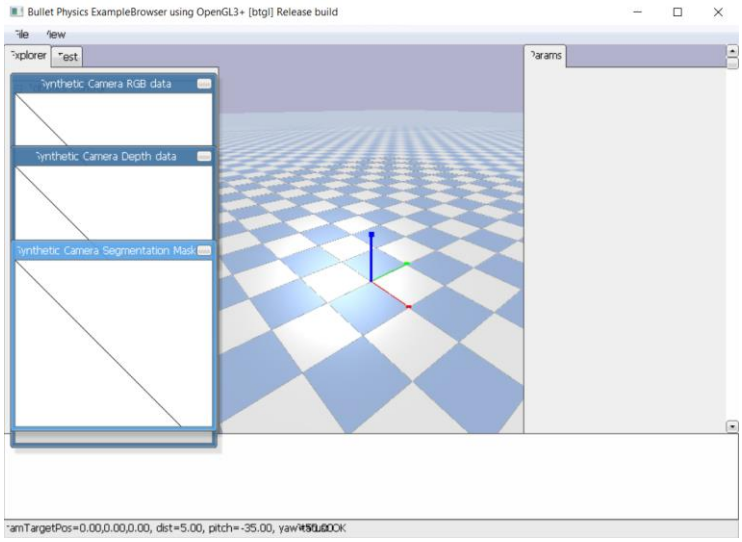

Рисунок 3.2 - Створення підлоги в сцені

Для демонстрації функціоналу було обрано маніпулятор KUKA IIWA. Його URDF-модель завантажувалась у сцену за допомогою функції p.loadURDF, де вказувались стартова позиція [0, 0, 0] та початкова орієнтація у вигляді кватерніона, обчисленого з кутів Ейлера [0, 0, 0]. Після завантаження моделі було перевірено кількість суглобів через p.getNumJoints(robotId), що підтвердило наявність семи керованих ланок (рис.3.3). Ця інформація важлива для подальшої реалізації керування та зворотної кінематики. Відповідний код для реалізації надано в Додатку В.

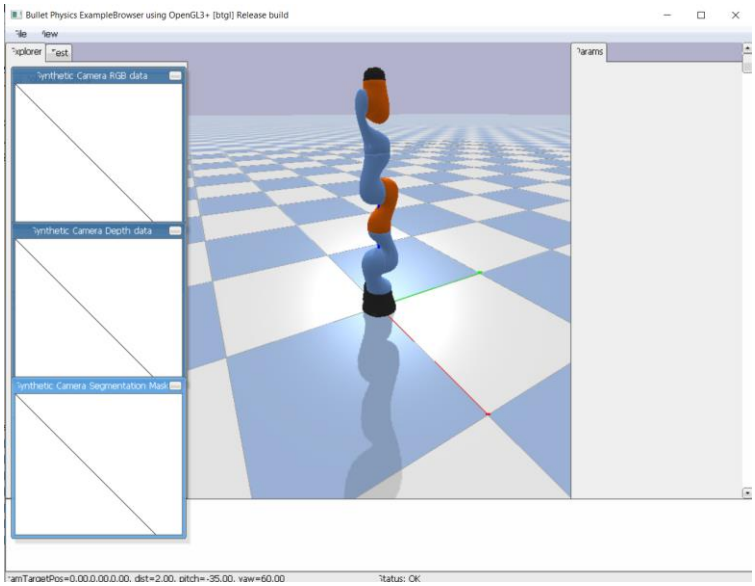

Рисунок 3.3 - Віртуальна модель маніпулятора

На цьому етапі також проведено первинну перевірку стабільності симуляції (рис.3.4). Відповідний код наведений в Додатку Г.

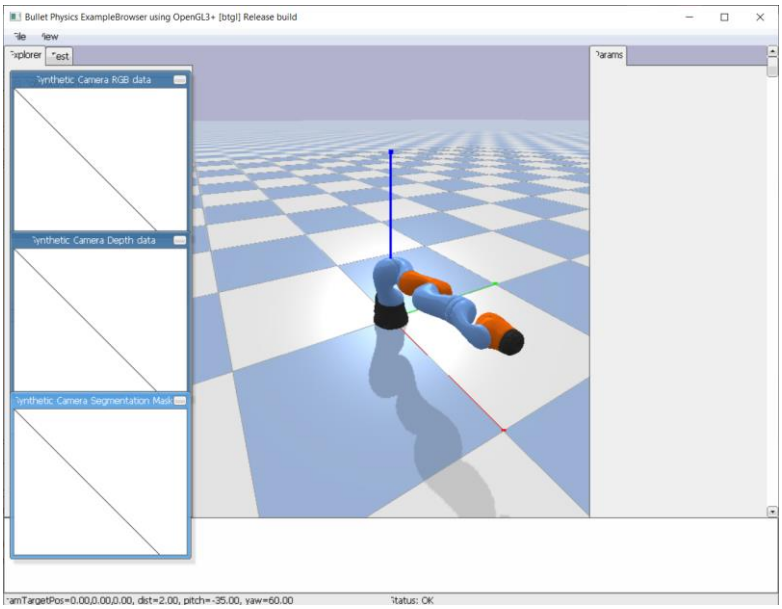

Рисунок 3.4 - Базова симуляція із рухом кінцівки

Запуск p.stepSimulation() у циклі дозволив переконатися, що маніпулятор завантажився без помилок, підлога та інші об’єкти відображаються коректно, фізичні взаємодії працюють. Візуально в GUI-режимі PyBullet спостерігалося,

що маніпулятор стоїть на підлозі, не провалюється крізь неї і фізично реагує на сили, що діють на нього, що є підтвердженням коректності налаштувань середовища.

В результаті етапу отримано повністю робоче середовище з базовою сценою і завантаженим маніпулятором, що дозволяє почати реалізацію керування рухом суглобів, зворотної кінематики та взаємодії з об’єктами. Завдяки використанню GUI PyBullet, стає можливим відразу бачити результати виконання команд у реальному часі та оцінювати поведінку моделі перед наступними кроками.

3.2. Базове керування суглобами

На другому етапі практичної частини проведено реалізацію базового керування рухомими суглобами маніпулятора в середовищі PyBullet. Обрано режим POSITION_CONTROL, який дозволяє задавати цільові кути суглобів і контролювати їх плавне досягнення у межах фізичних обмежень. Для кожного суглоба отримано ідентифікатор та межі рухів із URDF. Після цього проведено первинну перевірку коректності підключення маніпулятора та стабільності симуляції.

Для демонстрації базового керування реалізовано по черзі рух окремих суглобів: другого, четвертого та шостого. Для кожного з них обрано кути 45° та 60° (0.785 та 1.047 рад), щоб уникнути контакту кінцівки з підлогою. Для цього використано функцію p.setJointMotorControl2 із параметром POSITION_CONTROL. Основний цикл симуляції включав виклик p.stepSimulation() та затримку time.sleep(1./240.) для забезпечення реалістичної швидкості руху.

Результат моделювання представлено на рис.3.5. Відповідний код наведений в Додатку Д.

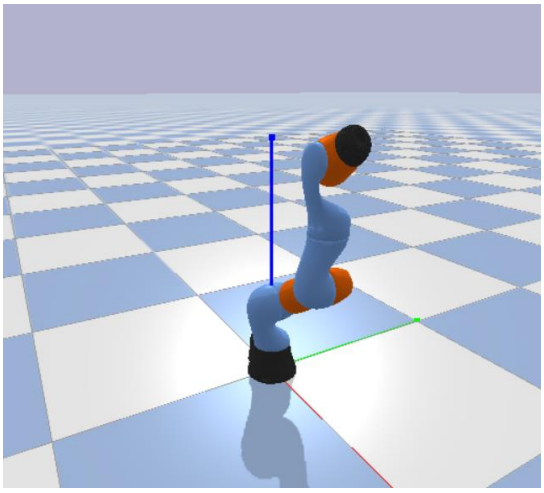

Рисунок 3.5 - Керування окремо кожним суглобом

Спостереження показали, що маніпулятор не виходить за межі фізичних обмежень, рух ланок плавний, без різких ривків або нестабільності сцени. Візуально помітно, що кінцівка завжди повертається до цільового положення в межах заданого кута і не порушує контакт з підлогою або іншими частинами маніпулятора.

Далі було проведено тестування комбінацій рухів кількох суглобів одночасно. Це дозволило оцінити поведінку маніпулятора при складніших сценаріях, коли одночасно змінюються кути кількох ланок. Результати показали, що середовище стабільне, фізичні взаємодії між ланками та підлогою залишаються коректними, а кінцівка досягає заданих положень у допустимому діапазоні. Такі спостереження важливі для подальшої інтеграції алгоритмів комп’ютерного зору, оскільки точне позиціювання кінцівки є критичним для захоплення об’єктів.

Важливо відзначити, що використання циклу симуляції разом із POSITION_CONTROL дозволяє гнучко змінювати швидкість руху і плавність переміщень. Змінюючи параметр force, можна контролювати силу, з якою суглоб намагається досягти цільового кута, що є корисним для моделювання реальних умов експлуатації робота. Крім того, такий підхід забезпечує

можливість додаткової інтеграції алгоритмів планування траєкторій і реакції на зовнішні сили.

Таким чином, було продемонстровано успішне налаштування та тестування базового керування суглобами маніпулятора, стабільність фізичної сцени, дотримання обмежень рухів та готовність для інтеграції наступних етапів практичної частини, зокрема системи зору та складніших сценаріїв переміщень кінцівки.

3.3. Зворотна кінематика

На цьому етапі реалізовано базову систему зворотної кінематики (Inverse Kinematics, IK), яка дає змогу керувати положенням кінцівки маніпулятора не шляхом безпосереднього задання кутів суглобів, а через визначення координат кінцевого ефектора у просторі. Це дозволяє користувачеві формулювати команду на зразок «перемістити руку в точку (x, y, z)», а система самостійно розраховує, які кути мають прийняти суглоби для досягнення заданої позиції.

Принцип зворотної кінематики полягає в тому, що для заданої структури маніпулятора (його URDF-моделі) і відомої позиції кінцевого ефектора у тривимірному просторі розв’язується обернена задача - знаходження вектору кутів, які забезпечують таке положення. PyBullet має вбудований розв’язувач цієї задачі, що дозволяє спростити реалізацію. У бібліотеці використовується функція calculateInverseKinematics(), яка приймає ідентифікатор моделі, індекс ланки (кінцевого ефектора), а також координати цільової точки. Опціонально можна задати орієнтацію (у вигляді кватерніона), але на даному етапі дослідження використовувалася лише позиційна складова, без контролю орієнтації.

Для проведення перевірки було обрано кілька контрольних точок у робочій зоні маніпулятора. Наприклад:

1) точка 1: (0.4, 0.0, 0.3) - перед маніпулятором, трохи вище площини підлоги;

2) точка 2: (0.3, 0.2, 0.2) - зміщення праворуч і нижче;

3) точка 3: (0.35, -0.15, 0.25) - зміщення ліворуч та ближче до основи.

Для кожної з цих точок функція calculateInverseKinematics() обчислювала набір кутів у радіанах для всіх керованих суглобів. У типовому випадку для маніпулятора KUKA iiwa повертається сім значень - по одному для кожного ступеня свободи. Під час тестування спостерігалось, що отримані значення кутів лежали у межах від -2.8 до +2.8 рад, що відповідає фізично допустимим обмеженням згідно з URDF-моделлю.

Розраховані кути передавались до PyBullet через команду setJointMotorControlArray(), у якій використовувався режим POSITION_CONTROL. Це забезпечувало поступовий і плавний рух кожного суглоба до заданого положення. Для уникнення різких коливань і можливих перехресних зіткнень між ланками використовувався короткий часовий інтервал оновлення симуляції (time.sleep(1./240.)), що дозволяло візуально оцінювати процес руху.

Візуальна перевірка показала, що маніпулятор коректно досягає зазначених цільових координат. У всіх тестових випадках кінцівка залишалась стабільною, не провалювалась крізь підлогу, а фізичні реакції симуляції (гравітація, інерція) залишались природними. Зокрема, при спробі задати цільову точку поза робочою зоною (наприклад, надто близько до основи або нижче площини підлоги), розв’язувач повертав крайові значення кутів, що відображалося у вигляді максимального розтягнення або обмеження руху ланок, але без аварійного завершення симуляції.

На рис. 3.6 показано один із результатів роботи системи зворотної кінематики: кінцівка маніпулятора переміщена у позицію (0.4, 0.0, 0.3), яка відповідає центральній ділянці робочого простору. На рисунку видно, що всі ланки зберігають правильну геометрію та не перетинаються, а положення

кінцевого ефектора відповідає очікуваному. Відповідний програмний код наведено в Додатку Е.

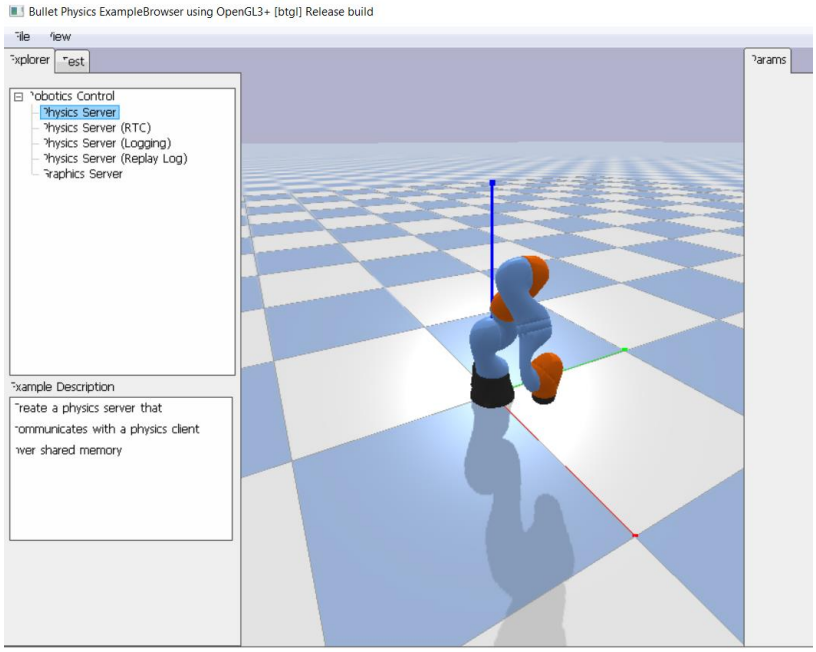

Рисунок 3.6 - Результат роботи зворотної кінематики

Таким чином, реалізація зворотної кінематики у PyBullet підтвердила свою ефективність. Отримані результати дозволяють не лише візуалізувати процес позиціонування, а й забезпечують основу для подальших експериментів - зокрема, переходу до взаємодії маніпулятора з об’єктами сцени, що стане наступним етапом дослідження.

3.4. Взаємодія з об’єктами у середовищі PyBullet

При розробці віртуального маніпулятора основна увага мусить бути приділена моделюванню взаємодії маніпулятора з фізичними об’єктами, що знаходяться у віртуальній сцені. Якщо попередні етапи (підрозділи 3.1-3.3) були спрямовані на створення робочої сцени, візуалізацію моделі робота та організацію керування рухом кінцівки, то на цьому етапі додається повноцінна фізична взаємодія. Саме вона є ключовою складовою моделювання

роботизованих маніпуляторів, оскільки дозволяє досліджувати поведінку системи у середовищі, де присутні зовнішні об’єкти, сили та обмеження.

3.4.1. Додавання фізичного об’єкта в сцену

Першим кроком було включення до середовища PyBullet одного з базових об’єктів (куба), який буде використовуватись як ціль для маніпуляцій. Цей об’єкт можна створити декількома способами (наприклад, за допомогою функцій createCollisionShape, createVisualShape та createMultiBody), що дозволяє повністю визначити їх фізичні характеристики. Для прикладу, куб мав розміри 5×5×5 см, масу 0.2 кг, коефіцієнт тертя 0.8 та коефіцієнт реституції (пружності при відскоку) 0.1.

Об’єкт розміщується на підлозі сцени перед маніпулятором так, щоб кінцевий ефектор міг до нього дістатися без необхідності змінювати положення основи робота. Це дозволило перевірити коректність роботи зворотної кінематики та точність позиціювання кінцівки у просторі. На рис. 3.7 наведено загальний вигляд сцени після додавання об’єкта.

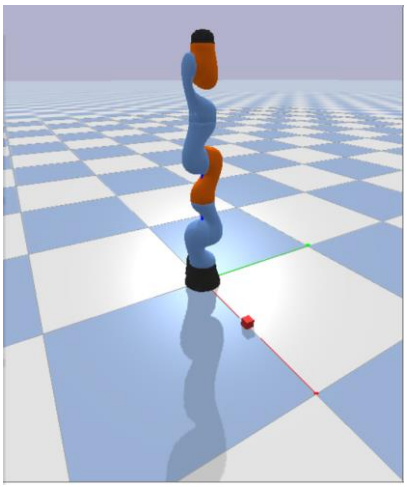

Рисунок 3.7 - Сцена після додавання куба як цілі для маніпуляції

3.4.2. Налаштування взаємодії та моделювання захоплення

Наступним етапом було налаштування можливості маніпулятора «взаємодіяти» з фізичним об’єктом. У робототехнічних симуляціях існує два основні підходи:

1. Взаємодія без захоплення. Це найпростіший підхід, коли кінцівка робота торкається об’єкта, штовхає його або переміщує без створення жорсткого зв’язку. У PyBullet така взаємодія виникає природно завдяки наявності фізичної моделі: колізії, контактні сили та тертя забезпечують рух об’єкта при дотику до нього. Цей варіант підходить для демонстрацій, але не дає можливості переміщувати об’єкт у повітрі.

2. Імітація захоплення (constraint-зачеплення). У PyBullet це реалізується через створення фіксації (p.createConstraint) між кінцевим ефектором маніпулятора та об’єктом. Такий підхід дозволяє створити жорстке або напівжорстке з’єднання, що імітує захоплення пальцями, магнітне зчеплення або вакуумний присос. Створюється прив’язка, яка забезпечує рух об’єкта разом з маніпулятором, а звільнення досягається видаленням constraint.

У нашій моделі застосовано саме другий варіант - фіксацію об’єкта у точці кінцевого ефектора після досягнення ним необхідної позиції. У результаті об’єкт поводиться так, ніби його надійно затиснуто.

3.4.3. Сценарій маніпуляції: захоплення та переміщення

Треба змоделювати реалістичний цикл руху маніпулятора, починаючи від моменту захоплення куба і завершуючи акуратним його встановленням у кінцеву позицію. В симуляції слід досягти ефекту спокійного й стабільного утримання предмета без дрижання, що вимагає переходу від ідеально жорсткого зчеплення до напівжорсткого або пружного з’єднання. Цей тип зв’язку дозволяє інструменту трохи компенсувати мікроколивання, що виникають при роботі фізичного рушія. Реалізується це за рахунок використання пружних параметрів з’єднання, де фіксатори не блокується

миттєво і абсолютно, а мають внутрішнє демпфування - завдяки йому захоплений предмет не тремтить у пальцях маніпулятора навіть при зміні позиції.

Після того як маніпулятор торкається об’єкта, симуляція повинна забезпечити невелику паузу для стабілізації положення. Це не штучне очікування у вигляді тривалого тайм-ауту, а коротке фізичне «осідання» — час, за який напівжорсткий зв’язок гасить перехідні коливання. Після стабілізації відбувається фактичне зчеплення: у моделі це зміна режиму взаємодії з «контактного зіткнення» на «фіксований або пружний зв’язок». Після захоплення маніпулятор плавно піднімає куб, але без різких ривків, завдяки чому блок залишається нерухомим відносно захоплювача. На цьому етапі важливо забезпечити рух за траєкторією, де зміна положення відбувається без миттєвих переходів, а з помірною швидкістю і прискоренням, що імітує реальну кінематику.

Коли об’єкт переноситься до нової точки, знову робиться невелика фізична пауза перед тим, як почати фазу опускання. Опускання куба має бути плавним, щоб рух униз не створив зворотних коливань при контакті з поверхнею. У симуляції це досягається шляхом обмеження максимальної швидкості вертикального переміщення у фінальній ділянці, коли залишається менше двох-трьох сантиметрів до поверхні. У момент дотику куба до площини його нижня грань повинна майже не мати вертикальної швидкості, що створює візуальний ефект акуратного встановлення. Тільки після цього зчеплення інструмента змінюється назад з пружного на звичайний контактний режим, але не миттєво - послідовність має включати короткий проміжок, у якому сила фіксації поступово зменшується. Такий підхід допомагає уникнути раптового стрибка або перекидання куба після відпускання.

Візуально це показано на рис. 3.8.

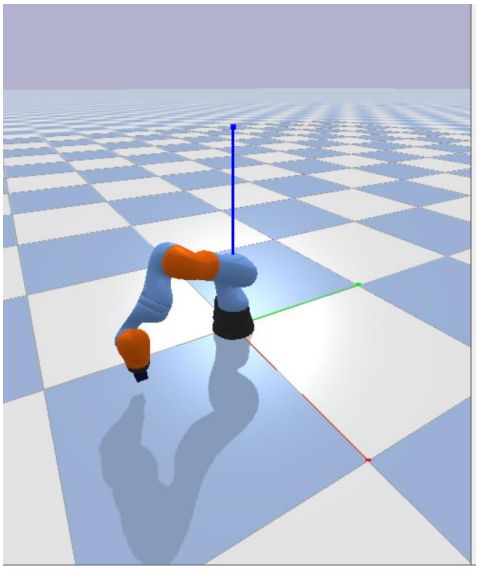

Рисунок 3.8 - Захоплення і перенос об’єкта

Після того як об’єкт акуратно встановлений і зв’язок повністю розімкнутий, маніпулятор повинен піднятися вгору на безпечну висоту. Цей рух виконується вже без об’єкта і не вимагає додаткової стабілізації.

Останньою фазою циклу було повернення куба на попереднє місце (повторюються попередні рухи в зворотній послідовності). Після цього маніпулятор знову мусить піднятись на небезпечну висоту. Результат показаний на рис. 3.9.

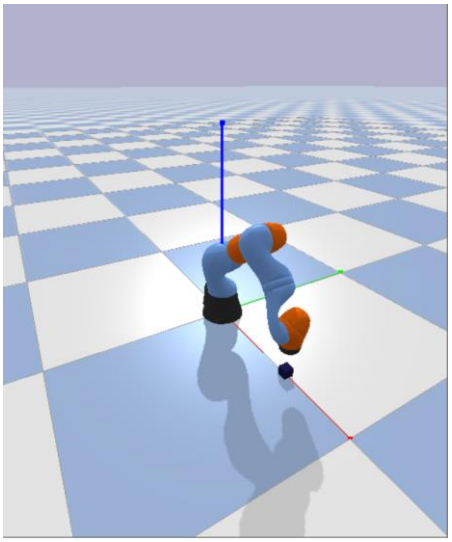

Рисунок 3.9 - Кінцева сцена

3.4.4. Візуальна перевірка результатів

GUI-режим PyBullet дозволив спостерігати всі етапи маніпуляції у реальному часі. Це надало можливість провести візуальний аналіз таких аспектів роботи системи:

- плавність рухів;
- коректність обчислення кутів ІК;
- точність позиціювання кінцівки відносно об’єкта;
- відповідність фізичної моделі очікуваній поведінці під час захоплення та відпускання.

Особлива увага приділялася стабільності під час переміщення: у разі неправильної конфігурації зв’язку могло б спостерігатися провертання або коливання об’єкта. Після серії тестів було підтверджено, що обрана модель constraint забезпечує надійний контроль і дозволяє маніпулятору поводитись у фізичному середовищі передбачувано та стабільно. Відповідний програмний код наведено в Додатку Ж.

Таким чином, реалізація взаємодії з фізичними об’єктами дала можливість продемонструвати здатність роботизованої системи виконувати базові маніпуляційні операції. Маніпулятор не лише досягає заданих координат, а й здатний переміщувати об’єкти та змінювати їх положення у просторі. Успішне виконання захоплення та перенесення підтверджує правильність роботи моделі, налаштувань середовища PyBullet та реалізованих алгоритмів інверсної кінематики.

3.5 Використання комп’ютерного зору для визначення положення об’єктів

Розглянемо процес отримання інформації про положення об’єкта за допомогою віртуальної камери в середовищі PyBullet. Після виконання

маніпуляційних дій у попередньому етапі стає очевидно, що цілеспрямовані рухи маніпулятора повинні ґрунтуватися на реальних вимірюваннях, а не на попередньо заданих координатах. Саме тому тут реалізується базова модель системи комп'ютерного зору, яка дозволяє автоматично визначати положення куба у просторі сцени та передавати ці дані блоку інверсної кінематики. Таким чином створюється замкнений контур сприйняття та керування.

Першим кроком стала організація віртуальної камери. В PyBullet камера визначається набором параметрів: положенням, напрямком погляду, кутами огляду та параметрами проєкції. Камера розміщується так, щоб мати чіткий огляд робочої зони, в якій маніпулятор виконує захоплення та переміщення кубика. В процесі моделювання камера фіксується над сценою під невеликим кутом, забезпечуючи хороший огляд робочої області без значних перспективних спотворень. Після налаштування параметрів камера починає формувати зображення у вигляді RGB-кадру, а також надавати карту глибини та сегментаційний масив, що дозволяє проводити попіксельний аналіз сцени. Зразок отриманого зображення наведений на рис. 3.10

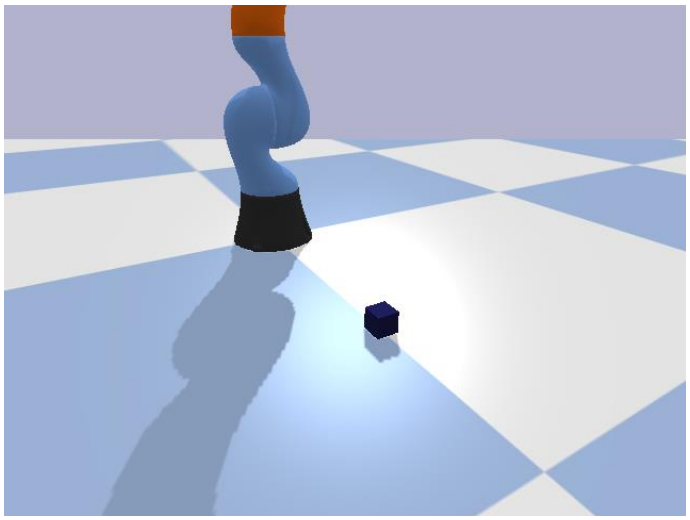

Рисунок 3.10 - Зображення з камери

Для виявлення куба використовується інформація із сегментаційного зображення, яке PyBullet формує автоматично. У цьому зображенні кожному пікселю відповідає ID об’єкта у сцені, тому куб виділяється без необхідності застосовувати алгоритми класичного комп’ютерного зору. Такий підхід виявився значно стабільнішим порівняно зі спробами сегментації за кольором або яскравістю, оскільки в ньому виключений вплив освітлення, бликів, тіней та інших візуальних перешкод. На основі сегментаційної маски формується двовимірна область, що відповідає кубу. Для наочності її можна візуалізувати як окреме зображення, що демонструє виділений об’єкт (на рис. 3.11 бачимо відразу 3 варіанти - зображення з камери, синтетичну depth-картинку та маску).

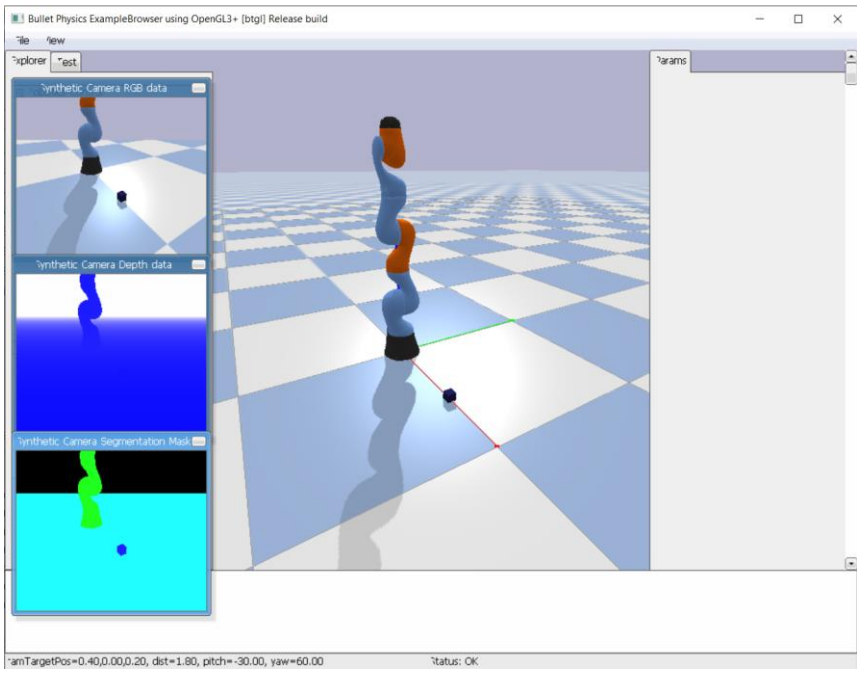

Рисунок 3.11 - Виділення об’єкта через маску

Центр маски визначає положення куба на зображенні в координатах пікселів. Після цього потрібно перейти від координат зображення до просторових координат сцени. Для цього використовується глибинна карта, що постачається разом із RGB-кадром. Кожен піксель глибинної карти містить інформацію про відстань від камери до відповідної точки сцени. Таким чином,

для знайденого центрального пікселя сегментації можна отримати значення глибини. Наступний етап полягає у застосуванні матриць проєкції та огляду, отриманих із PyBullet. Використовуючи ці матриці, координати пікселя перетворюються у промінь у тривимірному просторі, після чого точка перехрестя променя з відповідною глибиною відновлюється у світових координатах. Саме ці координати і визначають положення куба у моделі.

Отримані координати куба використовуються для формування цільової точки, до якої переміщується кінцевий ефектор через алгоритм інверсної кінематики. Відтепер маніпулятор не покладається на заздалегідь відомі позиції, а реагує на фактичне розташування об’єкта у сцені. Це дозволяє компенсувати можливі зміщення куба, неточності початкового розміщення або сторонні впливи, які можуть змінити положення об’єкта. Після визначення координат формується команда руху маніпулятора над куб і його подальше захоплення (рис. 3.12). Таким чином створюється базовий приклад повноцінної системи взаємодії «зір → аналіз → дія».

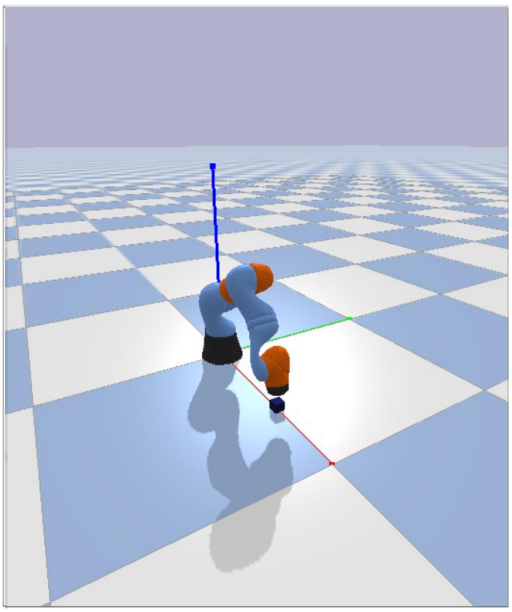

Рисунок 3.12 - Переміщення маніпулятора в точку над об’єктом

Моделювання продемонструвало, що підхід із використанням віртуальної камери PyBullet та сегментаційних масок забезпечує стабільне та точне визначення положення невеликого об’єкта, такого як контрольний куб. Виявлені координати відрізняються мінімальним шумом, а сам процес обчислення координат залишається детермінованим завдяки повному контролю над віртуальним середовищем. У реальних системах комп’ютерного зору ситуація значно складніша, проте описана модель дозволяє протестувати логіку та структуру майбутньої програми до переходу на реальні камери.

Готовий скрипт для використання комп’ютерного зору наведено в Додатку К. Він запускає сцену з KUKA і кубом, рендерить кадр з віртуальної камери PyBullet, отримує RGB, depth і segmentation, знаходить пікселі куба (по segmentation mask), вираховує центр маски, перетворює координату центру-пікселя + значення глибини у світові 3D-координати, виводить знайдені координати і командує IK перемістити кінцевий ефектор над знайденою точкою (невеликий Z-офсет). Код містить усі необхідні перетворення (unproject) і працює в GUI.

Таким чином продемонстрована інтеграція базового комп’ютерного зору з модулем управління маніпулятором. Віртуальна камера забезпечує інформацію про положення об’єкта, розрахунок координат дозволяє передати дані до інверсної кінематики, а робот отримує можливість виконувати маніпуляції на основі зорового сприйняття. Це формує фундамент для подальшого розвитку системи, включаючи впровадження більш складних алгоритмів обробки зображень та розпізнавання, а також використання реальних камер у фізичній робототехнічній системі.

3.6. Можливості подальшої інтеграції комп’ютерного зору

На цьому етапі в роботі не реалізовано повноцінну обробку зображень у реальному часі, проте описуються способи, які можуть бути використані для

майбутнього розвитку системи, а також демонструється принципова взаємодія віртуальної камери та маніпулятора через OpenCV.

Для початку можна додати у функції програми можливість отримувати зображення сцени у форматі RGB з камери PyBullet. PyBullet дозволяє встановлювати віртуальні точки огляду та рендерити сцену, а отримані кадри можна передавати у OpenCV для подальшої обробки. Наприклад, для захоплення кадру використовуються функції getCameraImage(), які повертають RGB-дані у вигляді масиву NumPy. Ці дані можна перетворити на стандартний формат OpenCV та обробляти за допомогою кольорових масок, фільтрів або алгоритмів сегментації. Можна відзначити, що використання OpenCV дає можливість точно визначати позиції об’єктів у піксельних координатах, аналізувати їх колір та форму, а також виділяти конкретні об’єкти у сцені, наприклад куби або циліндри. У рамках системи маніпулятора це дозволяє реалізувати сценарії «захоплення за кольором», коли координати кінцевого ефектора коригуються відповідно до положення об’єкта, виявленого на зображенні.

Другий напрямок удосконалення – це перетворення піксельних координат в координати у віртуальному просторі. Тут можна застосувати принципи перспективної проекції: знання матриці вигляду та проекції дозволяє обчислити напрямок променя, який проходить через конкретний піксель камери. Цей промінь можна використати для визначення координат об’єкта у 3D-просторі, після чого передати ці координати функції зворотної кінематики маніпулятора. Такий підхід дозволяє роботу «бачити» об’єкти без необхідності явно знати їх координати в моделі сцени.

Третій напрямок – обробка кадрів у реальному часі з використанням OpenCV. Після отримання RGB-зображення можна застосовувати порогові маски для виділення певного кольору або прості методи виявлення контурів. Для куба темно-синього кольору можна застосувати поріг за компонентами R, G, B або у HSV-просторі, що дозволяє зменшити вплив освітлення та отримати

більш стабільне виділення. Далі, визначивши центр маси виділеного об’єкта, можна отримати координати його піксельного центру. Ці координати передаються у функції перетворення піксель → світ, після чого маніпулятор рухається до заданої позиції.

Четвертий напрямок - інтеграція з існуючими функціями руху маніпулятора. Вже наявні функції для переміщення кінцівки через зворотну кінематику можна доповнити додатковим параметром - координатою об’єкта, визначеною зображенням камери. Це дозволяє створити сценарій, у якому маніпулятор реагує на зміну положення об’єкта у віртуальному просторі у режимі реального часу. В майбутньому можна розширити алгоритм так, щоб робот рухався до об’єкта, захоплював його і переміщував, при цьому використовуючи інформацію з камери для уточнення позиції під час руху.

Ще один аспект – перевірка коректності виявлення та рухів. Для цього можна додати візуальні підказки, наприклад, накладення координат центру об’єкта на кадр у OpenCV, кольорові прямокутники або точки, що відображають траєкторію руху кінцівки. Такий підхід дозволяє виявляти помилки у визначенні координат та коригувати параметри алгоритму.

Таким чином, існує достаньо багато можливостей майбутнього апгрейду системи з використанням OpenCV. Це включає отримання кадрів сцени, виділення об’єктів за кольором, перетворення координат пікселя у 3D-простір, інтеграцію з функціями руху кінцівки та візуальне контролювання точності рухів. Такий підхід відкриває перспективи для розширення системи до автономного захоплення та переміщення об’єктів, а також створює базу для подальшої розробки повноцінного комп’ютерного зору у робототехнічній моделі. Віртуальна сцена з маніпулятором та об’єктами дозволяють наочно оцінити принципи їх роботи без реалізації складного алгоритму.

Висновки до розділу

Таким чином, в даному розділі було послідовно реалізовано повний цикл розробки програмного рішення для моделювання віртуального маніпулятора з комп’ютерним зором, а також розроблено алгоритмічну основу для визначення координат об'єктів за результатами обробки зображень. Реалізовано та протестовано базові функції програмного модуля, оцінено їх коректність і стабільність роботи, проведено аналіз можливих похибок та обмежень обраного підходу. Окрему увагу приділено перспективам розвитку системи за рахунок інтеграції бібліотек, що дозволяє розширити функціональні можливості, підвищити точність визначення координат, автоматизувати процеси виявлення та відстеження об’єктів, а також забезпечити гнучкість подальшої модернізації програмного комплексу. В результаті сформовано цілісне уявлення про поточний стан системи та окреслено напрямки її подальшого апгрейда.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Охорона праці при розробці та експлуатації віртуального маніпулятора з системою комп’ютерного зору передбачає забезпечення безпечних умов роботи з комп’ютерною технікою, програмними засобами, а також дотримання вимог безпеки при можливому переході до роботи з реальними робототехнічними системами. Незважаючи на те, що основним середовищем дослідження є симуляційна платформа, робочий процес супроводжується використанням електронного обладнання, яке може становити потенційну небезпеку для здоров’я та життя користувача у разі недотримання встановлених правил.

У процесі виконання робіт оператор перебуває за робочим місцем, оснащеним персональним комп’ютером, периферійними пристроями та програмним забезпеченням для моделювання. До основних небезпечних і шкідливих факторів належать тривале статичне навантаження, зорове перенапруження, вплив електромагнітного випромінювання, перевтома, а також можливість ураження електричним струмом у разі порушення правил експлуатації електрообладнання. Неправильна організація робочого місця та недотримання ергономічних вимог можуть призводити до погіршення самопочуття, зниження працездатності та виникнення професійних захворювань.

Робоче місце оператора повинно відповідати санітарно-гігієнічним нормам та правилам охорони праці. Стіл і стілець мають забезпечувати правильне положення тіла, при якому навантаження на хребет мінімізується, а екран монітора знаходиться на відстані 50–70 см від очей на рівні або трохи нижче лінії зору. Освітлення робочого приміщення повинно бути рівномірним і достатнім, без різких контрастів та відблисків на екрані. Рекомендується поєднання природного та штучного освітлення з використанням ламп нейтрального спектра.

Тривала робота за комп’ютером потребує регламентованих перерв для відновлення працездатності. Раціональний режим праці передбачає чергування періодів інтенсивної роботи з короткими паузами, під час яких доцільно виконувати вправи для очей та м’язів, змінювати положення тіла і забезпечувати розслаблення. Це дозволяє знизити ризик розвитку зорового перевантаження та опорно-рухових порушень.

Особливу увагу необхідно приділяти електробезпеці. Усі технічні засоби мають бути справними, підключеними до мережі через заземлені розетки або джерела безперебійного живлення. Забороняється використовувати пошкоджені кабелі, несправні розетки або здійснювати підключення обладнання з оголеними провідниками. У разі виявлення сторонніх запахів, іскріння або перегріву пристроїв робота повинна бути негайно припинена, а обладнання відключене від електромережі.

Програмні засоби, що використовуються для симуляції та розробки, також потребують дотримання інформаційної безпеки. Необхідно використовувати ліцензійне програмне забезпечення, здійснювати регулярне оновлення систем та антивірусний захист, а також уникати запуску неперевірених файлів, які можуть завдати шкоди системі або призвести до втрати даних. Створення резервних копій проєктів є обов’язковою умовою для запобігання втратам результатів роботи.

З точки зору безпеки у надзвичайних ситуаціях необхідно враховувати можливі аварійні обставини, пов’язані з пожежами, відмовами електроживлення, задимленням або технічними збоями обладнання. Приміщення, в якому проводяться роботи, має бути оснащене первинними засобами пожежогасіння, зокрема вогнегасниками, та відповідати вимогам пожежної безпеки. Забороняється перекривати евакуаційні шляхи, захаращувати проходи та розміщувати зайві предмети поблизу електрообладнання.

У разі виникнення пожежі або задимлення необхідно негайно припинити роботу, відключити живлення та повідомити відповідні служби. Якщо ситуація дозволяє, слід використати вогнегасник для локалізації осередку займання, дотримуючись правил безпеки. Евакуація повинна здійснюватися відповідно до плану дій у надзвичайних ситуаціях, без створення паніки та з урахуванням безпеки всіх працівників.

Хоча основний акцент роботи зроблений на симуляційному середовищі, варто враховувати перспективу переходу до взаємодії з реальними маніпуляторами. У такому випадку спектр небезпечних факторів суттєво розширюється за рахунок рухомих механічних елементів, високих навантажень, можливих зіткнень та затискань. При роботі з реальними робототехнічними системами необхідно застосовувати спеціальні захисні засоби, аварійні кнопки зупинки, обмеження зон доступу та чітке регламентування процедур запуску і зупинки обладнання.

Таким чином, дотримання вимог охорони праці та безпеки у надзвичайних ситуаціях є невід’ємною складовою процесу розробки і використання віртуального маніпулятора з системою зору. Забезпечення безпечних умов праці сприяє збереженню здоров’я оператора, підвищенню ефективності роботи та формуванню відповідального підходу до експлуатації сучасних інформаційно-технічних систем, як у віртуальному середовищі, так і при потенційній інтеграції з фізичними робототехнічними комплексами.

РОЗДІЛ 5
ВИЗНАЧЕННЯ ЕФЕКТИВНОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Економічна ефективність розробленої системи віртуального маніпулятора з комп’ютерним зором полягає у зменшенні витрат на проєктування, тестування та відлагодження алгоритмів керування захопленням об’єктів порівняно з традиційним підходом, що базується на використанні реального обладнання на ранніх етапах розробки. Запропонована система дозволяє перенести основний обсяг експериментальних досліджень у симуляційне середовище, що дає змогу суттєво скоротити витрати на обладнання, його обслуговування, а також мінімізувати ризики пошкодження маніпулятора в процесі налагодження.

Для оцінювання економічної ефективності було розглянуто два умовних варіанти реалізації проєкту. Перший передбачає розробку алгоритмів керування та захоплення об’єктів із використанням реального маніпулятора, другий — із застосуванням розробленого віртуального симулятора як основної платформи на етапі проєктування та тестування. Як базову модель порівняння прийнято типовий сценарій дослідницьких робіт у лабораторних умовах протягом одного року.

У випадку використання реального маніпулятора основні витрати включають придбання роботизованої платформи, обладнання для його контролю, налаштування робочого місця та періодичне технічне обслуговування. Орієнтовна вартість навчально-дослідного маніпулятора середнього класу разом із контролером становить близько 150 000 грн. Додаткові витрати на допоміжне обладнання, калібрувальні пристрої, ремонт та замінні елементи протягом року можна оцінити на рівні 20 000 грн. Таким чином, загальні одноразові та експлуатаційні витрати за перший рік становлять близько 170 000 грн.

Розробка алгоритмів у реальному середовищі супроводжується також непродуктивними витратами часу, пов’язаними з паузами на переналаштування, усунення механічних збоїв, повторне калібрування та усунення наслідків помилок керування. У середньому один цикл тестування алгоритмів може займати на 30–40% більше часу, ніж у симуляційному середовищі. Якщо прийняти, що загальний обсяг робіт становить 400 годин розробки та тестування, то додаткові витрати часу при використанні реального маніпулятора становитимуть приблизно 140 годин. За умови середньої погодинної оплати праці спеціаліста на рівні 150 грн, ці витрати у грошовому вираженні складуть 21 000 грн.

У разі використання розробленого віртуального маніпулятора витрати набувають іншої структури. Основними статтями є витрати на розробку програмного забезпечення та організацію обчислювального середовища. Оскільки використовуються безкоштовні програмні засоби з відкритою ліцензією, такі як Python та PyBullet, прямі витрати на програмне забезпечення відсутні. Для виконання симуляції достатньо персонального комп’ютера середнього рівня, який у навчально-дослідних умовах зазвичай уже є в наявності. Таким чином, одноразові матеріальні витрати можна вважати нульовими або незначними.

Основу витрат у цьому випадку становить оплата праці розробника. Якщо прийняти, що на проєктування, реалізацію та тестування віртуального маніпулятора було витрачено 400 годин, то при тій самій погодинній ставці 150 грн загальні витрати становитимуть 60 000 грн. Подальше використання симулятора дозволяє повторно застосовувати створене програмне забезпечення без суттєвих додаткових витрат, що знижує собівартість кожного наступного експерименту.

Таким чином, порівнюючи два підходи, загальні витрати у випадку використання реального маніпулятора за перший рік становлять орієнтовно 191 000 грн (170 000 грн на обладнання та обслуговування і 21 000 грн на додаткові

витрати часу), тоді як при використанні віртуального середовища — близько 60 000 грн. Абсолютна економія коштів становить 131 000 грн протягом першого року.

Річний економічний ефект можна визначити як різницю між витратами традиційного підходу та витратами на використання розробленої системи. Таким чином, економічний ефект Е дорівнює 191 000 – 60 000 = 131 000 грн. Витрати на створення віртуального симулятора становлять 60 000 грн, тому термін окупності проєкту можна визначити як відношення витрат до річного економічного ефекту. У даному випадку термін окупності дорівнює 60 000 / 131 000 ≈ 0,46 року, тобто менше шести місяців. Це свідчить про високу економічну доцільність розробки.

Окрім прямого фінансового ефекту, впровадження симуляційного середовища забезпечує низку додаткових переваг, які опосередковано впливають на економічні показники. До них належить зниження ризику пошкодження обладнання, можливість безпечного тестування нестандартних режимів роботи, прискорення циклу розробки та підвищення якості алгоритмів за рахунок багаторазового відтворення експериментів у контрольованих умовах. Це дозволяє скоротити час виведення готової системи на практичний рівень та зменшити витрати на доопрацювання у майбутньому.

Отже, розробка віртуального маніпулятора з комп’ютерним зором є економічно обґрунтованою та доцільною. Вона забезпечує суттєве зниження витрат на етапах проєктування та дослідження, скорочує термін підготовки системи до практичного застосування та створює умови для масштабування результатів без значного зростання фінансових витрат. Отримані розрахунки підтверджують, що використання симуляційного середовища як основної платформи для розробки та тестування алгоритмів є ефективним рішенням як з технічної, так і з економічної точки зору.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В ході виконання кваліфікаційної роботи була розроблена та досліджена система віртуального маніпулятора з елементами комп’ютерного зору для задач захоплення та переміщення об’єктів у симуляційному середовищі. В процесі виконання роботи вирішені такі задачі:

1. Проведено аналіз предметної області робототехніки, визначено основні напрями застосування, сучасні тенденції розвитку та ключові проблеми, пов’язані з керуванням, захопленням об’єктів і використанням комп’ютерного зору. На основі цього сформульовано мету та завдання дослідження.

2. Розглянуто роль комп’ютерного зору в задачах маніпулювання об’єктами, проаналізовано можливості його використання у віртуальних середовищах моделювання.

3. Виконано обґрунтування вибору інструментарію вирішення задачі. Визначено архітектурні підходи до побудови програмного рішення та методи керування маніпулятором.

4. Реалізовано модель віртуального маніпулятора в симуляційному середовищі, виконано налаштування фізичних параметрів та базових режимів роботи.

5. Розроблено та продемонстровано сценарії взаємодії маніпулятора з об’єктами, включно з підходом до об’єкта, захопленням, переміщенням та акуратною постановкою у задану позицію.

6. Розглянуто можливості використання комп’ютерного зору для визначення положення об’єктів у сцені, а також описано перспективи подальшої інтеграції бібліотек комп’ютерного зору.

7. Проаналізовані питання охорони праці та безпеки у надзвичайних ситуаціях при роботі з комп’ютерними системами та програмними комплексами моделювання.

8. Виконано оцінку економічної ефективності розробки, яка показала доцільність використання віртуального симулятора для зменшення витрат на розробку та тестування алгоритмів керування маніпуляторами.

9. По результатам роботи опубліковані тези доповіді на студентській науковій конференції [29].

Отримані результати свідчать про те, що розроблена система віртуального маніпулятора є придатною для навчальних і дослідницьких цілей, а також може використовуватися як основа для подальшого перенесення алгоритмів керування на реальні робототехнічні платформи.

Подальший розвиток роботи може бути спрямований на розширення функціональності системи комп’ютерного зору, впровадження більш складних методів обробки зображень та машинного навчання, інтеграцію сенсорних даних, а також адаптацію розроблених алгоритмів для використання на реальних маніпуляторах у промислових або сервісних застосуваннях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Морзе Н.В. Основи робототехніки: навчальний посібник / Н.В. Морзе, Л.О. Варченко-Троценко, М.А. Гладун. – Кам’янець-Подільський : ПП Буйницький О.А., 2016. – 184 с.

2. Нові тенденції у розвитку робототехніки у 2025 році. URL: https://robo-education.com.ua/novi-tendenciyi-u-rozvitku-robototexniki-u-2025-roci/

3. Робототехніка: застосування та перспективи в майбутньому. URL: https://duikt.edu.ua/ua/news-1-0-11333-robototehnika-zastosuvannya-ta-perspektivi-v-maybutnomu

4. Промислові роботи - тренди 2024. URL: https://astra-s.com.ua/blog/promyshlennye-roboty-trendy-2024.html

5. У яких сферах застосовують робототехніку? URL: https://robocode.ua/news-embedded-ua

6. Які є різні типи промислових роботів. URL: https://www.evsint.com/uk/industrial-robots-types/

7. Литвин О., Паньков С. Роботизовані маніпулятори особливого призначення. URL: https://www.researchgate.net/publication/343340241_ROBOTIZOVANI_MANIPULATORI_OSOBLIVOGO_PRIZNACENNA

8. Струтинський В.Б., Гуржій А.М. : Монографія. – Житомир: ПП «Рута», 2023. – 524 с.

9. Як роботизовані руки трансформують виробництво у 2025 році. URL: https://www.evsint.com/uk/robotic-arms-transforming-manufacturing-2025/

10. Комп’ютерний зір – технологія, яка допомагає створити безпечне середовище на виробництві. URL: https://proit.ua/kompiutiernii-zir-tiekhnologhiia-iaka-dopomaghaie-stvoriti-biezpiechnie-sieriedovishchie-na-virobnitstvi/

11. Німець К. С. Проблеми та перспективи використання систем комп'ютерного зору у робототехніці / К. С. Німець // Computer-integrated technologies, automation and robotics 2024 : proceedings of the I st All-Ukrainian Conference,

Kharkiv, May 16-17, 2024. – Kharkiv, 2024. – P. 14-16. URL: https://openarchive.nure.ua/handle/document/26192

12. Комп'ютерний зір – це майбутнє автоматизації. URL: https://www.zaptest.com/uk/компютерний-зір-це-майбутнє-автом

13. Прикладні системи штучного інтелекту: комп'ютерний зір. URL: https://nasu-periodicals.org.ua/index.php/visnyk/article/view/12603/11735

14. Робототехнічні системи: проєктування і моделювання [Електронний ресурс]: навч. Посіб. для студ. спеціальності 126 «Інформаційні системи та технології» / М. М. Поліщук, М.М. Ткач; КПІ ім. Ігоря Сікорського, 2021. 112 с. URL: https://ela.kpi.ua/bitstream/123456789/41388/1/RTS.pdf

15. А. В. Чорна. Використання віртуальних середовищ для навчання освітньої робототехніки під час дистанційного навчання. DOI: https://doi.org/10.32782/1992-5786.2023.91.29

16. Як створити робота: огляд сучасних технологій. URL: https://evergreens.com.ua/ua/articles/how-to-create-robots.html

17. Gazebo: Симулятор робототехніки з відкритим кодом, що підтримується Open Robotics. URL: https://blog.desdelinux.net/uk/симулятор-робототехніки-альтанки--відкрита-робототехніка/

18. Simulating your robots with Webots. URL: https://cyberbotics.com/

19. NVIDIA Isaac Sim. URL: https://developer.nvidia.com/isaac/sim

20. PyBullet - EasyBuild - building software with ease. URL: https://docs.easybuild.io/version-specific/supported-software/p/PyBullet/

21. Google Colaboratory та його переваги для розробників Python. URL: https://nofluffjobs.com/uk/log/tehnologiji/google-colaboratory-and-its-benefits-for-python-developers/

22. Віртуалізація робочих місць (VDI). URL: https://netwave.ua/blog/virtualizacziya-robochih-mists-vdi/

23. Робототехніка та автоматизація: всебічний аналіз додатків, тенденцій та соціальних ефектів. URL: https://xpert.digital/uk/всебічний-аналіз-робототехніки-та-автоматизації/

24. Підручник з Python — Python 3.14.2 documentation. URL: https://docs.python.org/uk/3/tutorial/index.html

25. Python. URL: https://uk.wikipedia.org/wiki/Python

26. Технологія створення освітніх симуляторів для розвитку фахової компетентності педагога професійної освіти в умовах новітніх викликів: електронний освітній ресурс / Андрій Геревенко, Біла Церква: БІНПО, 2023. 115 с. URL: https://binpo.com.ua/wp-content/uploads/2024/02/ЕЛЕКТРОННИЙ-ОСВІТНІЙ-РЕСУРС-ГЕРЕВЕНКО-БІНПОpdf.pdf

27. Система керування роботом-маніпулятором. Serhii Kostiuchko. 2020, Computer-integrated technologies: education, science, production. DOI: https://doi.org/10.36910/6775-2524-0560-2020-40-06

28. Реалізація комплексної системи керування промисловим роботом-маніпулятором. URL: https://www.svaltera.ua/solutions/typical/mashinostroenie/8762.php

29. Бекар Д.Р. Середовища віртуального моделювання роботів. Матеріали конференції КІТ-2025, Харків, ХНАДУ, 25.11.2025. С.83-85. URL: https://drive.google.com/file/d/1TP6nULLlvp1JlM6OWajTPOqDCXRek_Zf/view

ДОДАТКИ

Додаток А

Код для створення порожньої сцени

```
import pybullet as p
import pybullet_data
import time

# Підключення до PyBullet у GUI-режимі
physicsClient = p.connect(p.GUI)
p.setAdditionalSearchPath(pybullet_data.getDataPath())

# Основний цикл симуляції
try:
    while True:
        p.stepSimulation()
        time.sleep(1./240.)  # реалістична швидкість симуляції
except KeyboardInterrupt:
    p.disconnect()
```

Додаток Б

Створення підлоги в сцені та симуляція фізичного тяжіння

```
import pybullet as p
import pybullet_data
import time

# Підключення до PyBullet у GUI-режимі
physicsClient = p.connect(p.GUI)
p.setAdditionalSearchPath(pybullet_data.getDataPath())

# Налаштування фізики
p.setGravity(0, 0, -9.81)

# Завантажити підлогу
plane_id = p.loadURDF("plane.urdf")

# Основний цикл симуляції
try:
    while True:
        p.stepSimulation()
        time.sleep(1./240.)  # реалістична швидкість симуляції
except KeyboardInterrupt:
    p.disconnect()
```

Додаток В

Завантаження моделі маніпулятора

```
import pybullet as p
import pybullet_data
import time

# Підключення до PyBullet у GUI-режимі
physicsClient = p.connect(p.GUI)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.resetDebugVisualizerCamera(
    cameraDistance=2.0,   # менше значення – ближче
    cameraYaw=60,
    cameraPitch=-35,
    cameraTargetPosition=[0,0,0]
)

# Налаштування фізики
p.setGravity(0, 0, -9.81)

# Завантажити підлогу
plane_id = p.loadURDF("plane.urdf")

# Завантаження маніпулятора KUKA
robotStartPos = [0, 0, 0]  # стартова позиція
robotStartOrientation = p.getQuaternionFromEuler([0, 0, 0])
robotId = p.loadURDF("kuka_iiwa/model.urdf", robotStartPos, robotStartOrientation)

# Вивід інформації про суглоби
num_joints = p.getNumJoints(robotId)
print(f"Кількість суглобів маніпулятора: {num_joints}")
for i in range(num_joints):
    joint_info = p.getJointInfo(robotId, i)
    print(f"{i}: {joint_info[1].decode('utf-8')} | Type: {joint_info[2]} | Limits:
({joint_info[8]}, {joint_info[9]})")

# Основний цикл симуляції
try:
    while True:
        p.stepSimulation()
        time.sleep(1./240.)  # реалістична швидкість симуляції
except KeyboardInterrupt:
    p.disconnect()
```

Додаток Г

Базова симуляція з рухом кінцівки

```
import pybullet as p
import pybullet_data
import time
import math

# Підключення до PyBullet у GUI-режимі
physicsClient = p.connect(p.GUI)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.resetDebugVisualizerCamera(
    cameraDistance=2.0,   # менше значення – ближче
    cameraYaw=60,
    cameraPitch=-35,
    cameraTargetPosition=[0,0,0]
)
p.setGravity(0, 0, -9.81)

# Підлога
plane_id = p.loadURDF("plane.urdf")

# Завантаження маніпулятора KUKA
robotStartPos = [0, 0, 0]
robotStartOrientation = p.getQuaternionFromEuler([0, 0, 0])
robotId = p.loadURDF("kuka_iiwa/model.urdf", robotStartPos, robotStartOrientation)

# Кількість суглобів
num_joints = p.getNumJoints(robotId)

# Основний цикл симуляції з невеликим рухом кінцівки
try:
    t = 0
    while True:
        # Простий синусоїдальний рух першого суглоба
        target_position = 1.5 * math.sin(t)
        p.setJointMotorControl2(
            bodyIndex=robotId,
            jointIndex=1,  # 2-й суглоб
            controlMode=p.POSITION_CONTROL,
            targetPosition=target_position,
            force=500
        )

        p.stepSimulation()
        time.sleep(1./240.)
        t += 1./240.
except KeyboardInterrupt:
    p.disconnect()
```

Додаток Д

Керування набором суглобів

```
import pybullet as p
import pybullet_data
import time
import math

# Підключення до PyBullet у GUI-режимі
physicsClient = p.connect(p.GUI)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.resetDebugVisualizerCamera(
    cameraDistance=2.0,   # менше значення – ближче
    cameraYaw=60,
    cameraPitch=-35,
    cameraTargetPosition=[0,0,0]
)
p.setGravity(0, 0, -9.81)
plane_id = p.loadURDF("plane.urdf")
robotStartPos = [0, 0, 0]
robotStartOrientation = p.getQuaternionFromEuler([0, 0, 0])
robotId = p.loadURDF("kuka_iiwa/model.urdf", robotStartPos, robotStartOrientation)
num_joints = p.getNumJoints(robotId)

# Основний цикл симуляції з невеликим рухом кінцівки
try:
    t = 0
    while True:
        # Вузли для руху
        joints_to_move = [1, 3, 5]  # у PyBullet нумерація з 0
        target_positions = [0.785, 1.047]  # 45° і 60° в радіанах

        for joint in joints_to_move:
            for pos in target_positions:
                p.setJointMotorControl2(robotId, joint, p.POSITION_CONTROL,
targetPosition=pos)
                for _ in range(240):
                    p.stepSimulation()
                    time.sleep(1. / 240.)

except KeyboardInterrupt:
    p.disconnect()
```

Додаток Е

Код для зворотної кінематики

```
import pybullet as p
import pybullet_data
import time
import math

# Підключення до PyBullet у GUI-режимі
physicsClient = p.connect(p.GUI)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.resetDebugVisualizerCamera(
    cameraDistance=2.0,   # менше значення – ближче
    cameraYaw=60,
    cameraPitch=-35,
    cameraTargetPosition=[0,0,0]
)
p.setGravity(0, 0, -9.81)
plane_id = p.loadURDF("plane.urdf")
robotStartPos = [0, 0, 0]
robotStartOrientation = p.getQuaternionFromEuler([0, 0, 0])
robot_id = p.loadURDF("kuka_iiwa/model.urdf", robotStartPos, robotStartOrientation)
num_joints = p.getNumJoints(robot_id)

# --- Цільові точки для кінцевого ефектора ---
target_positions = [
    (0.4, 0.0, 0.3),   # Точка 1 – перед маніпулятором
    (0.3, 0.2, 0.2),   # Точка 2 – праворуч
    (0.35, -0.15, 0.25)  # Точка 3 – ліворуч
]

# --- Основна функція руху ---
def move_to_position(target_pos, steps=240):
    # Обчислення зворотної кінематики для кінцевого ефектора (7-ма ланка)
    joint_angles = p.calculateInverseKinematics(robot_id, 6, target_pos)

    # Вивід розрахованих кутів
    print(f"Цільова точка: {target_pos}")
    print("Кути суглобів (рад):", [round(a, 3) for a in joint_angles])

    # Плавне переміщення у цільове положення
    for i in range(steps):
        p.setJointMotorControlArray(
            bodyUniqueId=robot_id,
            jointIndices=list(range(7)),
            controlMode=p.POSITION_CONTROL,
            targetPositions=joint_angles
        )
        p.stepSimulation()
        time.sleep(1. / 240.)

# --- Демонстрація руху ---
try:
    for target in target_positions:
```

```
            move_to_position(target)
            time.sleep(2)  # пауза між руками

        print("Рух завершено. Натисніть Ctrl+C для виходу.")
        while True:
            p.stepSimulation()
            time.sleep(1. / 240.)

except KeyboardInterrupt:
    print("Симуляцію завершено користувачем.")
    p.disconnect()
```

Додаток Ж

Взаємодія з об’єктом

```
import pybullet as p
import pybullet_data
import time

# Підключення до PyBullet у GUI-режимі
physicsClient = p.connect(p.GUI)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.setGravity(0, 0, -9.81)
p.resetDebugVisualizerCamera(
    cameraDistance=2.0,   # менше значення – ближче
    cameraYaw=60,
    cameraPitch=-35,
    cameraTargetPosition=[0,0,0]
)

# Завантаження підлоги
plane_id = p.loadURDF("plane.urdf")

# Завантаження маніпулятора KUKA
robotStartPos = [0, 0, 0]
robotStartOrientation = p.getQuaternionFromEuler([0, 0, 0])
robotId = p.loadURDF("kuka_iiwa/model.urdf", robotStartPos, robotStartOrientation,
useFixedBase=True)

# Додавання куба у сцену
cube_start_pos = [0.6, 0, 0.05]
cube_start_orientation = p.getQuaternionFromEuler([0, 0, 0])
cube_id = p.loadURDF("cube_small.urdf", cube_start_pos, cube_start_orientation,
globalScaling=1)
p.changeVisualShape(cube_id, -1, rgbaColor=[0.1, 0.1, 0.3, 1])

# Функція руху кінцівки через IK
def move_ee_to(target_pos, target_orient=None, steps=240):
    if target_orient is None:
        target_orient = p.getQuaternionFromEuler([0, 3.14, 0])
    for _ in range(steps):
        joint_poses = p.calculateInverseKinematics(robotId, 6, target_pos,
target_orient)
        for i in range(7):
            p.setJointMotorControl2(robotId, i, p.POSITION_CONTROL, joint_poses[i],
force=500)
        p.stepSimulation()
        time.sleep(3./240.)

# 1. Підняти руку до куба
cube_hover = [cube_start_pos[0], cube_start_pos[1], cube_start_pos[2] + 0.15]
move_ee_to(cube_hover)

# 2. Опустити кінцівку до куба
cube_pick = [cube_start_pos[0], cube_start_pos[1], cube_start_pos[2] + 0.05]
move_ee_to(cube_pick)
```

```
# 3. Створити напівжорстке зчеплення
constraint_id = p.createConstraint(robotId, 6, cube_id, -1, p.JOINT_FIXED, [0,0,0],
[0,0,0], [0,0,0])
p.changeConstraint(constraint_id, maxForce=50)

# 4. Підняти куб
move_ee_to(cube_hover)

# 5. Перемістити куб у нову позицію
cube_target = [0.2, -0.6, 0.15]
move_ee_to(cube_target)

# 6. Опустити куб на підлогу
cube_place = [cube_target[0], cube_target[1], 0.05]
move_ee_to(cube_place)

# 7. Зняти constraint
p.removeConstraint(constraint_id)

# 8. Підняти руку після постановки
move_ee_to([cube_target[0], cube_target[1], 0.2])

# --- повернення куба на початкову позицію ---

# 9. Підтягнути руку до куба
cube_hover_back = [cube_place[0], cube_place[1], cube_place[2] + 0.15]
move_ee_to(cube_hover_back)

# 10. Опустити кінцівку до куба
cube_pick = [cube_place[0], cube_place[1], cube_place[2] + 0.05]
move_ee_to(cube_pick)

# 11. Створити constraint для повторного захоплення
constraint_id_back = p.createConstraint(robotId, 6, cube_id, -1, p.JOINT_FIXED,
[0,0,0], [0,0,0], [0,0,0])
p.changeConstraint(constraint_id_back, maxForce=50)

# 12. Підняти куб
move_ee_to([cube_place[0], cube_place[1], 0.15])

# 13. Перемістити куб на початкову позицію
move_ee_to(cube_hover)

# 14. Опустити куб на стартову позицію
cube_start_place = [cube_start_pos[0], cube_start_pos[1], cube_start_pos[2] + 0.05]
move_ee_to(cube_start_place)

# 15. Зняти constraint і підняти руку
p.removeConstraint(constraint_id_back)
move_ee_to([cube_start_pos[0], cube_start_pos[1], 0.2])

# Основний цикл для огляду сцени
try:
```

```
    while True:
        p.stepSimulation()
        time.sleep(1./240.)
except KeyboardInterrupt:
    p.disconnect()
```

Додаток К

Код для комп'ютерного зору

```
import pybullet as p
import pybullet_data
import time
import numpy as np
from PIL import Image

# ---------- Налаштування сцени ----------
p.connect(p.GUI)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.setGravity(0, 0, -9.81)
p.resetDebugVisualizerCamera(cameraDistance=1.8, cameraYaw=60, cameraPitch=-30,
cameraTargetPosition=[0.4, 0, 0.2])

plane = p.loadURDF("plane.urdf")
robot = p.loadURDF("kuka_iiwa/model.urdf", [0, 0, 0], useFixedBase=True)

# куб як об'єкт для виявлення
cube_start_pos = [0.6, 0.0, 0.025]
cube_id = p.loadURDF("cube_small.urdf", cube_start_pos)
p.changeVisualShape(cube_id, -1, rgbaColor=[0.1, 0.1, 0.3, 1])

# даємо GUI трохи часу, щоб усе промальовалось
for _ in range(120):
    p.stepSimulation()
    time.sleep(1. / 240.)

# ---------- Параметри камери ----------
width, height = 640, 480
fov = 60
aspect = width / height
near = 0.1
far = 3.0

# позиція камери
cam_target = [0.45, 0.0, 0.1]
cam_distance = 1.0
yaw = 60
pitch = -25
roll = 0
up_axis_index = 2

view_matrix = p.computeViewMatrixFromYawPitchRoll(
    cameraTargetPosition=cam_target,
    distance=cam_distance,
    yaw=yaw,
    pitch=pitch,
    roll=roll,
    upAxisIndex=up_axis_index
)

proj_matrix = p.computeProjectionMatrixFOV(
```

```
    fov=fov,
    aspect=aspect,
    nearVal=near,
    farVal=far
)

# ---------- Допоміжні функції ----------
def depth_buffer_to_z(depth_buffer, near=near, far=far):
    """
    Перетворює буфер глибини з getCameraImage у реальні z (відстань від камери).
    Формула для Bullet depth buffer (не лінійний).
    """
    # depth_buffer: значення в [0,1]
    z = (2.0 * near * far) / (far + near - (2.0 * depth_buffer - 1.0) * (far - near))
    return z

def unproject_pixel(u, v, z, view_mat, proj_mat, width, height):
    """
    Перетворює піксель (u,v) з глибиною z (у світовому вимірі від камери) в світові
Координати. Ми реконструюємо clip space -> view space -> world space.
    """
    # перетворимо матриці у питру та отримаємо повну матрицю трансформації
    view = np.array(view_mat).reshape((4, 4)).T  # pybullet дає матрицю у форматі list
column-major
    proj = np.array(proj_mat).reshape((4, 4)).T

    # інверсія проєкційної * view матриці
    inv = np.linalg.inv(proj @ view)

    # перехід від пікселя до NDC простору
    x_ndc = (2.0 * u) / width - 1.0
    y_ndc = 1.0 - (2.0 * v) / height  # звертаємо вісь Y
    z_ndc = 2.0 * (z - 0.0) / 1.0 - 1.0  # але ми не маємо нормалізованого z; замість
цього використаємо підхід через ray

    # Класичний unproject з використанням depth у clip space є складнішим через
Нелінійність.
    # Тому тут ми скористаємося іншим трюком: збудуємо дві точки на ближній/дальній
площині в NDC
    ndc_near = np.array([x_ndc, y_ndc, -1.0, 1.0])
    ndc_far  = np.array([x_ndc, y_ndc,  1.0, 1.0])

    # перетворюємо їх у світові координати
    world_near = inv @ ndc_near
    world_far  = inv @ ndc_far
    world_near /= world_near[3]
    world_far  /= world_far[3]

    # тепер за допомогою реальної z від камери (depth) інтерполюємо між near та far
    cam_pos = world_near[:3]  # точка на ближній площині в світі
    cam_far  = world_far[:3]

    # розрахунок параметру t по відстані: знаємо бажану відстань від камери = z_world
    # обчислимо відстані від ками до near і far
    dist_near = np.linalg.norm(cam_pos - world_near[:3])  # це 0
    # Для простоти знайдемо напрямок променя та підставимо
    ray_dir = (cam_far - cam_pos)
```

```
    ray_dir = ray_dir / np.linalg.norm(ray_dir)
    # обчислимо точку вздовж променя на відстані z від камери
    # але для цього потрібно знати позицію камери у світі:
    # камера світова позиція можна знайти як inverse(view) * [0,0,0,1]
    inv_view = np.linalg.inv(view)
    cam_world = (inv_view @ np.array([0,0,0,1]))[:3]

    # ми маємо реальну відстань z (від камери до точки) -> world_point = cam_world +
ray_dir * z
    world_point = cam_world + ray_dir * z
    return world_point

def find_object_world_position(segmentation, depth_buffer, target_body_id, view_mat,
proj_mat, w, h):
    """
    Знайти центр маси пікселів з сегментацією == target_body_id, взяти глибину у цьому
пікселі, і конвертувати у світові координати.
    """
    seg = np.array(segmentation).reshape((h, w))
    mask = (seg == target_body_id)
    if not mask.any():
        return None, None, None  # не знайдено

    ys, xs = np.where(mask)
    cx = int(np.mean(xs))
    cy = int(np.mean(ys))

    # отримуємо depth buffer (піксель з индексом (cx, cy) у flatten-формат img[3])
    depth = np.array(depth_buffer).reshape((h, w))[cy, cx]
    # перетворюємо значення depth buffer у відстань від камери
    z_world = depth_buffer_to_z(depth, near=near, far=far)

    world_point = unproject_pixel(cx, cy, z_world, view_mat, proj_mat, w, h)
    return (cx, cy), z_world, world_point

# ---------- Отримуємо зображення ----------
img = p.getCameraImage(width, height, viewMatrix=view_matrix,
projectionMatrix=proj_matrix,
                       renderer=p.ER_BULLET_HARDWARE_OPENGL)

# img structure: (width, height, rgba, depth, seg)
rgb = np.reshape(img[2], (height, width, 4))[:, :, :3].astype(np.uint8)
depth_buffer = np.reshape(img[3], (height, width))
seg = np.reshape(img[4], (height, width))

# Знаходимо пікселі з id == cube_id (в seg масці)
# У PyBullet segmentation mask містить objectUniqueId (як ціле)
(cx_c, cy_c), zdist, world_pt = find_object_world_position(img[4], img[3], cube_id,
view_matrix, proj_matrix, width, height)

if world_pt is None:
    print("Об'єкт не знайдено на зображенні.")
else:
    print(f"Pixel centroid: ({cx_c}, {cy_c})  depth={zdist:.4f} m")
    print("World coordinates of detected object:", world_pt)

    # Збережемо маску
    seg_mask = (seg == cube_id).astype(np.uint8) * 255

    # Збережемо на зображенні позначку центру
```

```
    import cv2
    vis = rgb.copy()
    cv2.circle(vis, (cx_c, cy_c), 6, (0,255,0), -1)

    # ---------- Виклик IK: перемістити EE над об'єктом ----------
    ee_link_idx = 6  # KUKA iiwa end effector index
    pre_grasp = [world_pt[0], world_pt[1], world_pt[2] + 0.12]  # офсет над об'єктом
    print("Moving end-effector to pre-grasp position:", pre_grasp)

    # Плавний рух: кілька кроків
    steps = 240
    for s in range(steps):
        target_angles = p.calculateInverseKinematics(robot, ee_link_idx, pre_grasp)
        for j in range(7):
            p.setJointMotorControl2(robot, j, p.POSITION_CONTROL, target_angles[j],
force=400)
        p.stepSimulation()
        time.sleep(1. / 240.)

    print("End-effector moved to detected object overhead position.")

# Залишаємо вікно відкритим для огляду
print("Готово. Закрийте вікно або натисніть Ctrl+C у терміналі для завершення.")
try:
    while True:
        p.stepSimulation()
        time.sleep(1./240.)
except KeyboardInterrupt:
    p.disconnect()
```