Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки

«Затверджую»
Зав. кафедри теоретичної та
прикладної системотехніки
_____ д.т.н., проф. С. І. Шматков
«___» _____ 2023 р

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «МОДЕЛЮВАННЯ І СТВОРЕННЯ БАЗОВИХ АПАРАТНО – ПРОГРАМНИХ ЕЛЕМЕНТІВ СИСТЕМИ КЕРУВАННЯ РОБОТОМ – МАНІПУЛЯТОРОМ»

Захищено на засіданні
Атестаційної комісії № 40
протокол № __ від 06.12.2023 р.
Оцінка _____ / _____
Голова Атестаційної комісії
_____ Скоб Ю. О.
(підпис) (прізвище та ініціали)

Виконав:
студент 4 курсу, групи КІ– 41
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 «Комп'ютерна інженерія»
Уваров Валентин Юрійович
(прізвище, ім'я та по батькові)

(підпис)

Керівник:
Рало Олександр Миколайович
(прізвище, ім'я та по батькові)

(підпис)
Рецензент:

_____
(підпис)

Харків – 2023

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, трьох розділів, висновку, списку використаних джерел і трьох додатків. Загальний обсяг роботи складає 63 сторінок, із яких 48 сторінки основної частини з 33 рисунками, 10 найменуваннями списку використаних джерел та трьома додатками.

Мета роботи полягає в тому, щоб протестувати можливості ПЗ для симуляції моделювання робототехнічних систем для подальшої роботи з нею.

Об'єкт дослідження: Робот-маніпулятор, як сукупність механізмів, приводів та системи керування.

Предмет дослідження: Моделювання роботи робота-маніпулятора за допомогою програмного забезпечення для симуляції робототехнічних систем.

Проблема, яка вирішується в кваліфікаційній роботі полягає в тому, щоб при розробці робота-маніпулятора заздалегідь, не витрачаючи багато часу, можна було протестувати кожен його елемент разом з системою керування без значних матеріальних витрат.

Область застосування – розробка прототипів і системи керування роботів-маніпуляторів. Розроблений програмний продукт може широко використовуватися в сфері навчання, а також малого та середнього IT-бізнесу.

Ключові слова: CoppeliaSim, PyCharm, Python, робот-маніпулятор, програмне забезпечення, керуючий код, симуляція, імпорт, шарнір.

ABSTRACT

An explanatory note to the master's attestation work is created in the introduction, three sections, conclusions, a list of sources used and three additional substances.

The total volume of work is 63 pages, of which 48 pages of the main part with 33 figures, 10 names of the list of used sources and three additions. The aim of the thesis is testing of the software for simulating the modeling of robotic systems to work with it. The problem that is solved in the diploma collaboration is testing robot-manipulator`s elements and its control system without extra costs and unnecessary time.

Scope – development of prototypes and control system of robot-manipulators. The developed software product can be widely used in the field of study, small and medium IT business.

Keywords: CoppeliaSim, PyCharm, Python, robot-manipulator, software, control code, simulation, import, joint.

ЗМІСТ

ВСТУП

Всі знають, що наше суспільство сильно залежить від цифрової техніки. Про це активно говорять, знімають серіали чи фільми, пишуть книжки і так далі. Але люди не зовсім розуміють, що за цим ховається насправді. У більшості людей в першу чергу програмування асоціюється з хакерами в фільмах. У них декілька моніторів, де з космічною швидкістю в консолі пролітають строчки коду, які навряд чи людини зможе повністю прочитати і проаналізувати. Також пролітаючі строчки може заповнювати байт-код, яким насправді обмінюються складові комп'ютера, з чого можна зробити висновок, що це взагалі має бути приховано від очей користувача. Періодично на цих моніторах з'являється графічний інтерфейс, що показує досить цікаві речі. Можна побачити проекцію будівель разом з локальною мережею, якісь фігури, що являють собою блоки коду і з'єднуються як пазли, і навіть цілий цифровий світ, де ходять завдяки навичкам програмування. В реальному житті це має «трохи» інший вигляд. Динаміка того, що відбувається насправді досить низька. Треба сидіти за столом і годинами читати або писати код. Таке показати глядачу цікаво непосильна задача. Тим більш існує багато різних професій, пов'язаних з програмуванням. Все не робить одна людина. Є багато напрямків розвитку цифрових технологій, і кожен напрям заслуговує великої уваги. Але тема мого диплому натякає на зв'язок з робототехнікою, тому в першу чергу я буду писати про неї.

Робототехніка розвивається дуже впевнено і вносить великий вклад в наше життя. Тому, на мій погляд, якщо ми говоримо про галузі пов'язані з цифровими технологіями, то робототехніка є найбільш важливою для людства, бо ми фізично можемо відчути весь її потенціал. На сьогоднішній день це: безготівкова оплата за допомогою телефона чи годинника, бездротові навушники, розумні пристрої (від чайника до цілого дома), і це ще не говорячи про такі речі, як процесори. Для приклада візьмемо

мега-корпорацію «Intel» та i7 лінійку процесорів для звичайних ПК. Третє покоління таких процессорів має 4 ядра та 8 потоків, а тринадцяте вже 16 ядер і 24 потоки. Можна подумати, що це непогана різниця, але також змінилися: максимальна частота з 3,9 Ггц до 5,4 Ггц, технологічний процес (щільність розміщення транзисторів) з 22 нм до 10 нм, кеш (пам'ять процесора) з 8 Мб до 30 Мб, і в особливості хочу виділити нову архітектуру, яку не тільки дуже добре оптимізували, а ще й ядра розділили на енергоефективні та продуктивні. Також не забуваємо про підтримку процесором нових технологій. Наприклад, як стандарт оперативної пам'яті DDR5, і тд. Це дозволяє опрацьовувати дані, судячи з тестів самої компанії, як мінімум в шість разів швидше. І це різниця всього лише в десять років!

Як не важко зрозуміти, поняття «робототехніка» дуже тісно переплітається з поняттям «робот». Про роботів далі і піде мова. Вони можуть вбирати пил з вашої підлоги, або досліджувати ґрунт на просторах інших планет, але в першу чергу роботи вплинули на промислові виробництва. Такі прилади піднімають великі вантажі, працюють з небезпечними для людини хімікатами і збирають деталі з точністю до мільйонних долей відсотків. І представники такого рода праці - це переважно роботи-маніпулятори.

Об'єкт дослідження: Робот-маніпулятор, як сукупність механізмів, приводів та системи керування.

Предмет дослідження: Моделювання роботи робота-маніпулятора за допомогою програмного забезпечення для симуляції робототехнічних систем.

Актуальність роботи: Плюси купівлі чи сборки робота-маніпулятора очевидні. Найголовніша проблема полягає в тому, що робот-маніпулятор коштує немало, навіть якщо збирати все самому по частинам. Існує програмне забезпечення (ПЗ), яке може допомогти урахувати це питання. Автором буде проаналізовано і виявлено найефективніше ПЗ для вирішення даної проблеми.

Мета роботи: На даний час досить мало матеріалу присвячено ПЗ для симуляції систем керування роботом-маніпулятором, тому дане дослідження повинне допомогти:

- студентам, котрим цікаво поринути в світ робототехніки, не купуючи спеціального обладнання
- викладачам, які думають, що такий вид ПЗ може допомогти їм покращити якість навчального процесу
- підприємцям, що планують вкласти гроші в розробку свого продукту - робота, але прагнуть протестувати свою систему керування до витрат на його складові розробникам, яким треба на початковому етапі проекту продемонструвати в інтерактивній формі, як може виглядати результат їх зусиль.

Ще хотілося б привернути увагу на те, що найбільш акцент буде на освітніх цілях ПЗ, бо шукаючи матеріал про системи керування роботом-маніпулятором, я зрозумів, що ця тема має досить високий поріг входу.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО МАНІПУЛЯТОРИ

1.1. ГНУЧКІСТЬ РОБОТИЗОВАНИХ РОБОЧИХ СТАНЦІЙ

У автоматизації заводів і не тільки було поширеним використання фіксованого місця навколо конвеєрів або інших транспортних систем, де кожен робот виконував певне завдання. Ці лінії збірки мали декілька окремих робочих станцій, кожна з яких виконує певну функцію. Роботи використовувались на робочих станціях для виконання таких операцій, як складання, свердління, обробка поверхні, зварювання, укладання на піддони, тощо. В лініях збірки деталі послідовно направляються на робочі станції транспортною системою. Такі системи дуже дорогі в установці, вимагають інженерів для проектування та програмування, а також надзвичайно важко потім щось змінювати або перепрограмувати відповідно до потреб.

Сучасний виробничий сценарій high-mix low-volume (HMLV) полягає в тому, що виробництво може зробити багато різноманітної продукції у певних кількостях, його ще порівнюють з виготовленням на замовлення. Цей метод виробництва часто використовується для обробки та виробництва унікальних і складних продуктів з певними вимогами якості. Описані вище характеристики сценарію призвели до смерті негнучких застарілих конструкцій (рис. 1.1.). [1]

Рисунок 1.1. Завод виробництва автомобілів Ford

На лініях збірки робот стає обмеженим через жорстку послідовну систему в якій він буде працювати. Роботи — це універсальні машини з багатьма можливостями, і їх потенціал можна значно збільшити, взявши їх за основу для гнучких роботизованих робочих станцій, таких як на підприємстві General Motors (рис. 1.2.)

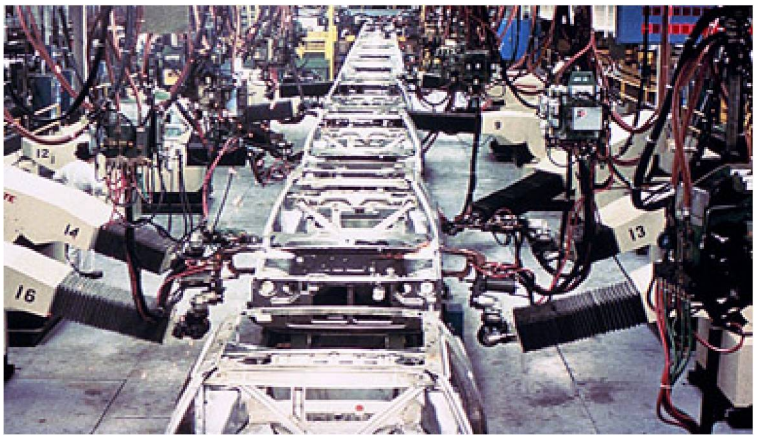

Рисунок 1.2. Перші роботи для точкового зварювання в США

У гнучких роботизованих робочих станціях роботи використовуються для обробки деталей, збірки та інших технологічних операцій. Перепрограмуючи роботів, змінюється повний функціонал робочих станцій. Робоча станція розроблена таким чином, щоб оптимізувати використання робочого простору роботів і таких компонентів, як фрезерні верстати, свердлильні верстати, тощо. Все розміщено так, щоб роботи мали змогу обслуговувати роботизовані робочі станції. Всупереч лініям зборки, фізичне розташування не накладає апріорі фіксовану послідовність операцій або робочих дій. Таким чином, коли вимоги до продукту змінюються, потрібно тільки перепрограмування робочої станції через програмне забезпечення. Робочі станції ідеально підходять для нових умов HMLV у виробництві та інших місцях.

Зростаюча популярність роботизованих робочих станцій зняла акцент з розробки апаратного забезпечення та поставила новий акцент на інноваційних техніках програмного забезпечення та архітектури, що включає планування, координацію та контроль функцій. Щоб надати роботам

гнучкості, точності та функціональності, необхідних для сучасних гнучких робочих станцій, знадобилося багато досліджень контролерів роботів. Далі буде досить детально описано такий розширений контроль техніки. [2]

1.2 КОНФІГУРАЦІЇ ТА ТИПИ КОМЕРЦІЙНИХ РОБОТІВ

Роботи — це високонадійне та технологічно вдосконалене фабричне обладнання. Більшість роботів у світі постачається відомими компаніями, які використовують надійні стандарти для своїх компонентів. Усі комерційні промислові роботи мають два основні елементи (фізично окремі) — маніпулятор і контролер. Базова архітектура більшості комерційних роботів принципово однакова і складається з електричних моторів з цифровими сервоприводами, які керують послідовними кінематичними механізмами, зазвичай не більше ніж шістьма осями (ступенями свободи). Усі вони постачаються з фірмовим контролером. Практично всі програми роботів вимагають значних зусиль щодо проектування та впровадження з боку інженерів і техніків. Що робить кожного робота унікальним, так це те, як компоненти зібрані разом, щоб досягти продуктивності, яка дає конкурентоспроможний продукт. Найважливіші два питання пов’язані з програмою промислового робота - це маніпулювання та інтеграція.[3]

Комбінований ефект кінематичної конструкції, механізмів приводу осей та керування рухом у реальному часі визначають основні характеристики маніпулятора: досяжність і спритність, корисне навантаження, швидкість і точність. Але слід бути обережним, порівнюючи характеристики продуктивності на основі документації опублікованої виробниками, оскільки методи їх вимірювання та звітності не стандартизовані в галузі. Зазвичай тестування руху, симуляція або інші методи аналізу використовуються для перевірки продуктивності для кожної програми.

Досяжність визначається вимірюванням розміру робочого простору, описаного рухом робота, а спритність визначається за допомогою кутового зміщення окремих місць з'єднання. Деякі роботи матимуть спеціальні місця в

межах їхньої досяжності, такі як мертві зони, або окремі пози в місцях з'єднання конструкції (осі).

Вага корисного навантаження вказана виробниками всіх промислових роботів. Також деякі виробники вказують інерційне навантаження для осей обертання. Зазвичай корисне навантаження вказується для екстремально великої швидкості та досяжності. Вага та інерція всіх інструментів, робочих елементів, кабелів і шлангів повинні бути включені як частина корисного навантаження.

Швидкість має вирішальне значення для визначення продуктивності, але її важко визначити з опублікованих специфікацій роботів. Більшість виробників вказують максимальну швидкість для окремих осьових з'єднань або для конкретної кінематичної точки інструменту. Проте *середня швидкість* у робочому циклі є характеристикою *швидкості*, яка нас цікавить.

Точність зазвичай характеризується *повторюваністю* (ці поняття взаємопов'язані). *Повторюваність* — це показник здатності робота постійно досягати заданої точки. *Точність* – це міра похибки між значенням координат точки, на яку робот запрограмований, і значенням точки, до якої робот фактично йде. Практично всі виробники роботів вказують повторюваність статичної позиції. Точність вказується рідко. Також зазвичай не вказується точність, швидкість та прискорення на безперервному шляху та в умовах перевезення.[4]

Усі поширені комерційні промислові роботи є маніпуляторами з послідовними ланками, як правило, з не більше ніж шістьма кінематично пов'язаними осями руху. Часто вони мають структуру руки, описану як «плече», «лікоть» та «зап'ястя». За домовленістю осі руху нумеруються в порядку їх проходження від основи до «зап'ястя». Перші три осі відповідають просторовому позиціонуванню руху робота; їхня конфігурація визначає площину, в якій може бути розташований робот. Будь-які наступні осі в кінематичному ланцюгу зазвичай забезпечують обертальні рухи для орієнтації кінця руки робота та називаються осями «зап'ястя». У

роботизованому зап'ясті три осі зазвичай перетинаються, щоб створити справжній кінематичний аналіз сферичного зап'ясткового механізму робота. В нашому тривимірному просторі потрібні три ступені свободи для повністю незалежного просторового позиціонування та три ступені свободи для повністю незалежного орієнтаційного позиціонування.

Існує два основні типи руху, які вісь робота може здійснити в своїй ланці: обертальний або призматичний. Обертальні з'єднання є антропоморфними (наприклад, як суглоби людини), тоді як призматичні з'єднання здатні витягуватися та стягуватися, як антена в радіо. Часто корисно класифікувати роботів відповідно до орієнтації та типу їхніх перших трьох осей.

Існує декілька поширених комерційних конфігурацій роботів і дві конфігурації (циліндрична та сферична), що тепер зустрічаються достатньо рідко. Ці конфігурації буде наведено нижче.[3]

Шарнірний робот

Роботи з шарнірними "руками" пропонують найбільший ступінь свободи та зазвичай мають від трьох до шести з'єднань (шарнірів). Їх різноманітність дуже велика. Найбільшою перевагою такого робота є його широкий діапазон рухів. Друга і третя осі копланарні та працюють разом для створення руху у вертикальній площині. Перша вісь в основі обертає маніпулятор, щоб ігнорувати великий об'єм робіт. Було розроблено багато різних типів механізмів, щоб дозволити приводним двигунам «зап'ястя» і «передпліччя» встановлюватися поблизу першої та другої осі обертання, таким чином мінімізуючи допустиму масу руки. Добре сконструйовані шарнірні роботи, якщо порівнювати з іншими конфігураціями, коли потрібно п'ять або більше ступенів свободи, мають просто неперевершену ефективність робочого простору (рахується за ступенем спритності та досяжності відносно розміру маніпуляторів). Основним обмежуючим фактором продуктивності шарнірної «руки» є те, що друга вісь повинна працювати, щоб підняти і конструкцію робота, і корисне навантаження. Історично склалося так, що шарнірні

маніпулятори не могли досягти такої високої точності, як інші конфігурації, оскільки всі осі мають похибки положення кута з'єднання, які множаться на радіус ланки та накопичуються для всього плеча.[4]

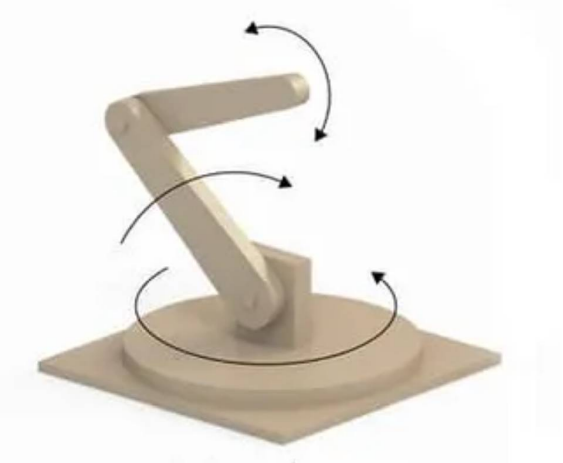

Рисунок 1.3. Шарнірна конфігурація

SCARA робот

Маніпулятори SCARA (Selective Compliance Assembly Robot Arm) часто можуть піднімати більш важкі вантажі, ніж інші типи роботів, тому що перша та друга осі не задіяні в цьому процесі. Третя вісь SCARA забезпечує робочий обсяг шляхом додавання вертикальної осі. Четверта вісь додає обертання для контролю орієнтації в горизонтальній площині. Цей тип роботів рідко зустрічається з більш ніж чотирма осями. Широко використовується для складання електронних компонентів і пристроїв, а також малих і середніх механічних збірок.

Також слід зазначити, що є два типи такої конфігурації. SCARA тип II (також конфігурація з чотирма осями), відрізняється від типу I тим, що перша вісь піднімає дві паралельні поворотні осі та їхні ланки. Для швидкого переміщення важких вантажів (понад 35 кілограмів) на більшу відстань (більше приблизно одного метра) конфігурація SCARA типу II є більш ефективною, ніж тип I.

Рисунок 1.4. Конфігурація SCARA

Декартовий робот

Декартові маніпулятори можуть рухатися вздовж трьох осей за допомогою системи рейок, як друкуюча головка 3D-принтера. Для орієнтації можна додати одну, дві або три обертальні осі “зап’ястя”. Компанії постачають декілька типів роботів з декартовими координатами, де розмір робочого простору може бути від кількох десятків кубічних сантиметрів до тисяч кубічних метрів і корисним навантаженням до кількох сотень кілограмів.

Роботи, які мають конструкцію надземного мосту, є найпоширенішим декартовим типом і добре підходять для вантажно-розвантажувальних робіт, де необхідно обслуговувати великі площі та/або великі вантажі. Вони особливо корисні в таких сферах, як дугове зварювання, гідроабразивне різання та контроль великих і складних прецизійних деталей.

Модульні декартові роботи також дуже поширені. Кожен модуль є самостійним і повністю функціональним одноосьовим приводом. Модулі можуть бути зібрані на замовлення для спеціального призначення.

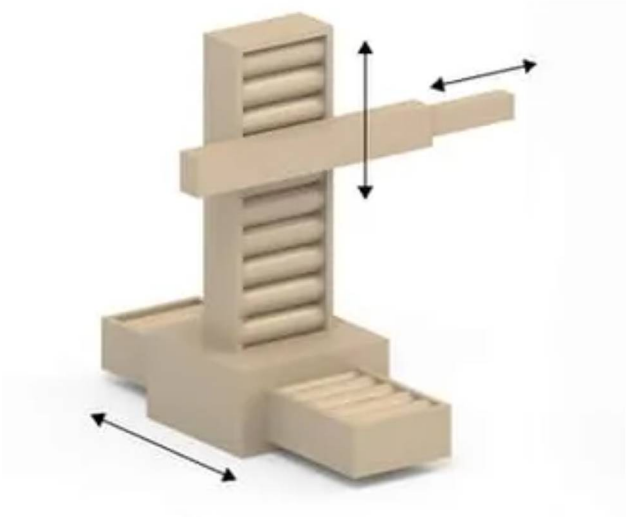

Рисунок 1.5. Декартова конфігурація

Сферичний та циліндричний (координатний) роботи

Сферичні роботи-маніпулятори мають два обертальні шарніри, включаючи в основі, і третій шарнір, який дозволяє згинати «руку». Циліндричні роботи-маніпулятори мають обертальне з'єднання в основі та два призматичних з'єднання: одне для переміщення по осі z і одне для переміщення в горизонтальній площині.

Перші дві осі сферичного координатного робота є обертальними і ортогональними по відношенню одна до одної, а третя вісь забезпечує призматичне радіальне розширення. У результаті виходить сферична координатна система зі сферичним робочим об'ємом. Перша вісь роботів є поворотною базою обертання. Друга і третя є призматичними, що призводить до природного циліндричного руху. Моделі сферичних і циліндричних роботів спочатку були дуже поширеними та популярними по відношенню до догляду за машинами та транспортування матеріалів. Деякі все ще використовуються, але зараз є лише кілька доступних моделей. Зменшення використання цих двох конфігурацій пояснюється проблемами, що виникають через використання призматичної ланки для радіального висування/втягування (міцна стріла вимагає вільного простору для повного втягування)

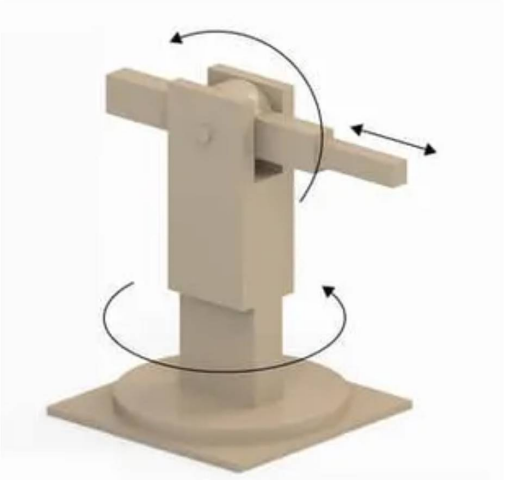

Рисунок 1.6. Сферична конфігурація

Рисунок 1.7. Циліндрична конфігурація

Дельта (паралельний) робот

Для деяких програм спеціального призначення роботи з паралельним зв’язком більш підходять, ніж роботи з послідовним з’єднанням. Ці роботи зазвичай мають три або шість паралельних ланок, кожна ланка прикріплена до нерухомої основи та рухомої робочої платформи. При належному проектуванні маніпулятор з шістьма паралельними ланками може мати шість ступенів свободи руху робочої платформи. На цій основі розроблені тренажери для пілотів, які по суті є великими паралельно зв'язаними роботами, що рухають крісло людини. Ці роботи мають більшу жорсткість і точність, ніж роботи з послідовним зв’язком, де помилки позиціонування кожної ланки ускладнюються, коли ланка рухається назовні від основи.

Таким чином, легкі роботи з паралельними ланками здатні точно переміщувати великі вантажі. Ці роботи використовувалися, наприклад, для обробки поверхонь прецизійних промислових і аерокосмічних компонентів, таких як перегородки та зовнішня обшивка повітряних транспортних засобів. Такі роботи є системою замкнутого кінематичного ланцюга. Її відносно важко проаналізувати. Проблема проектування системи керування є більш складною для цих роботів. [5]

1.3 КОНТРОЛЕРИ КОМЕРЦІЙНИХ РОБОТІВ

Контролери комерційних роботів — це спеціалізовані мультипроцесорні обчислювальні системи, які забезпечують чотири основні процеси, що дозволяють інтегрувати робота в систему автоматизації: генерація траєкторії руху та слідування за нею, інтеграція та послідовність рухів/процесів, інтеграція користувача (людини) та інтеграція інформації.

Генерація траєкторії руху та слідування за нею

Є два важливі аспекти, пов'язані з контролером, у генерації руху промислового робота. Один — це ступінь маніпуляції, який можна запрограмувати, інший — здатність виконувати керований запрограмований рух. Унікальним аспектом кожної роботизованої системи є керування рухом на сервоприводі в реальному часі. Деталі керування в режимі реального часу зазвичай не розкриваються користувачеві з міркувань безпеки та конфіденційності службової інформації. Контролер кожного робота за допомогою програм своєї операційної системи перетворює цифрові дані від координаторів вищого рівня в скоординований рух руки за допомогою точного обчислення та високошвидкісного розподілу та передачі команд руху окремих осей, які виконуються окремими об'єднаними сервоконтролерами. Приблизно 20 років тому дуже активно використовувався вже класичний пропорційно-інтегрально-диференціальний (ПІД) регулятор керування. Для реалізації треба тільки знайти суму трьох доданків: пропорційної, інтегральної та диференціальної складових. Це робило контролери

придатними для руху від точки до точки, але при цьому контроллер мав багато недоліків, які занадто важко контролювати. З тих пір з'явилися більш досконалі контролери, які працюють на різних ОС і з різними мовами програмування.[6]

Інтеграція та послідовність руху/процесу.

Інтеграція руху/процесу передбачає координацію руху маніпулятора з датчиками або іншими пристроями керування процесом. Найпримітивніша інтеграція процесу здійснюється через дискретний цифровий вхід/вихід (input/output). Наприклад, зовнішній контролер машини може надіслати однобайтовий сигнал до контролера робота, вказуючи, що робот готовий до завантаження. Контролер робота повинен мати здатність зчитувати цифровий сигнал і виконувати логічні операції (if then, wait until, do until, тощо). Тобто, деякі контролери роботів мають деякі вбудовані функції програмованого логічного контролера (ПЛК). Також часто забезпечується координація з датчиками (наприклад, зору).

Інтеграція людини.

Людський інтерфейс контролера має вирішальне значення для швидкого налаштування та програмування робототехнічних систем. Більшість контролерів роботів мають два типи інтерфейсу для людини: термінали з клавіатурою для написання та редагування програмного коду в автономному режимі та термінали з графічним інтерфейсом, які використовуються для управління рухом робота за допомогою сенсорних клавіш або джойстиків. Графічний інтерфейс зазвичай є найефективнішим доступним засобом позиціонування робота, а пам’ять контролера дає змогу відтворювати завчені позиції для виконання траєкторій руху. Більшість застосувань роботів наразі залежать від інтеграції досвіду людини на етапі програмування для успішного планування та координації руху робота. Ці механізми інтерфейсу ефективні в безперешкодних робочих просторах.

Новітні вдосконалені методи інтерфейсу роботів базуються на програмуванні поведінки, де різні специфічні поведінки програмуються на

низькому рівні в контролер робота (наприклад, взяти шматок, вставити деталь). Ці дані впорядковуються, а потім на високому рівні людиною вказуються більш конкретні параметри для підвищеної точності.

Інтеграція інформації.

Інтеграція інформації стає все більш важливою, оскільки тенденція до підвищення гнучкості та спритності впливає на робототехніку. Зараз контролери роботів підтримують функції інтеграції інформації, використовуючи інтерфейси ПК через комунікаційні порти, або в деяких випадках через пряме підключення до шини даних контролера робота. Спроби інтеграції зробили можливим доступ до Інтернету, що забезпечило дистанційний моніторинг і контроль для роботів. [6]

1.4 ДАТЧИКИ

Датчики та приводи функціонують як пристрої, за допомогою яких високорівневі системи планування, координації та контролю взаємодіють із апаратними компонентами, що складають робочі станції.

Датчики є життєво важливим елементами, оскільки вони перетворюють стани фізичних пристроїв у сигнали, придатні для введення в систему керування. Невідповідні датчики можуть викликати помилки, які унеможливлюють правильну роботу незалежно від того, наскільки складною чи дорогою є система, тоді як інноваційний вибір датчиків може значно полегшити проблему керування та координації.

Датчики бувають різних типів і мають багато різних застосувань. Якщо проводити аналогію з біологічними системами, пропріорецептори (чутливі нервові закінчення, що дають інформацію про положення різних частин тіла відносно одна одної) — це внутрішні датчики пристрою, які передають інформацію про внутрішній стан цього пристрою (наприклад, датчики кута «суглоба» руки робота). Екстерорецептори (нервові утворення на поверхні тіла людини, що сприймають подразнення та перетворюють їх на нервові збудження) передають інформацію про інше зовнішнє по відношенню до

пристрою обладнання. Цифрові датчики надають інформацію про стан машини або ресурсу (захват відкритий або закритий, завантаженість машини, чи робота завершена). [7] Датчики створюють вихідні дані, необхідні на всіх рівнях, зокрема для:

- управління зворотним зв'язком на серво-рівні
- моніторинг процесів і координація
- моніторинг несправностей і безпека
- перевірка контролю якості.

Тактильні датчики.

Тактильні сенсори залежать від фізичного контакту із зовнішніми об’єктами. Цифрові датчики дають двійкову інформацію про те, чи відбувається контакт чи ні. Аналогові датчики, такі як пружинні стрижні, дають більше інформації. Тактильні датчики на основі гумових вуглецевих або кремнієвих еластомерів із вбудованими електричними чи механічними компонентами можуть надавати дуже детальну інформацію про геометрію деталей, розташування тощо. Еластомери можуть містити резистивні або ємнісні елементи, електричні властивості яких змінюються в міру стискання еластомеру. Також існують конструкції, що можуть виробляти тактильні сітки. Такі датчики створюють «тактильні зображення», які мають властивості, схожі на цифрові зображення з камери, і вимагають схожої обробки даних. Додаткові тактильні датчики підпадають під класифікацію «датчиків сили», яка обговорюється далі.

Датчики відстані.

Безконтактні датчики відстані часто є цифровими, що дають двійкову інформацію про те, чи знаходиться об’єкт поруч. Вони включають пристрої на основі ефекту Холла або індуктивні пристрої на основі електромагнітного ефекту. Ємнісні датчики виявляють будь-яке тверде або рідке тіло поблизу. Оптичні та ультразвукові датчики мають більший радіус дії. Датчики відстані включають далекоміри, такі як ехолоти та лазери. Ехолот, як правило, створює шум із помилковими показаннями, і вимагає фільтрації низьких

частот та іншої обробки даних, спрямованої на зменшення частоти помилкових тривог. Дорогі лазерні далекоміри відрізняються надзвичайною точністю у відстань і мають дуже високу кутову роздільну здатність.

Датчики положення, швидкості та прискорення.

Лінійні прилади для вимірювання положення включають лінійні потенціометри, а також ехолот і лазерний далекомір, про які вже згадувалось вище. Датчики лінійної швидкості можуть бути пристроями на основі ефекту Доплера.

Пропріорецептори положення кута суглоба та швидкості є важливою частиною приводної осі сервоприводу руки робота. Датчики кутового положення включають потенціометри, які використовують постійну напругу, і резольвери, які використовують змінну напругу. Оптичні кодери можуть забезпечити надзвичайну точність за допомогою цифрових методів. Інкрементні оптичні кодери використовують три оптичні датчики та одне кільце чергування областей для надання інформації про кутове положення відносно контрольної точки та кутову швидкість. Дорожчі абсолютні оптичні кодери, мають n концентричних кілець із чергуванням областей і потребують n оптичних датчиків. Вони забезпечують підвищену точність і мінімізують помилки, пов'язані з читанням і передачею даних, особливо якщо вони використовують код Грея, де лише один біт змінюється між двома послідовними секторами. Гіроскопи мають хорошу точність. Доступні різні види акселерометрів на основі тензодатчиків, гіроскопів або властивостей кристала.

Датчики сили і крутного моменту.

Доступні різні датчики крутного моменту. Датчики крутного моменту на свердлильному інструменті, наприклад, можуть вказувати, коли інструменти стають тупими. Лінійну силу можна виміряти за допомогою тензодатчиків. Тензодатчик — це датчик, що перетворює величину деформації на зручний для вимірювання сигнал (зазвичай електричний). П'єзоелектричний ефект, генерація напруги під час застосування сили, також

може бути використаний для визначення сили. Інші методи вимірювання сили базуються на вакуумних діодах, кристалах кварцу (резонансна частота яких змінюється залежно від прикладеної сили) тощо. Датчики сили-крутного моменту на "зап'ясті" робота надзвичайно корисні для виконання завдань маніпулювання з великою спритністю. Комерційно доступні пристрої можуть вимірювати як силу, так і крутний момент уздовж трьох перпендикулярних осей, надаючи повну інформацію про вектор декартової сили F. Стандартні перетворення дозволяють обчислювати сили та крутні моменти в інших координатах. Шестиосьові датчики сили крутного моменту досить дорогі.

Фотоелектричні датчики.

Доступний широкий спектр фотоелектричних датчиків, деякі з них засновані на волоконно-оптичних принципах. Вони корисні для маркування і виявлення частин, сканування оптичних штрих-кодів, підтвердження проходження частин при сортуванні, тощо.

Інші датчики.

Також доступні різні датчики для вимірювання тиску, температури, потоку рідини тощо.

Обробка даних датчиків

Перш ніж будь-який датчик можна буде використовувати, його необхідно відкалібрувати. Залежно від датчика це може потребувати значних зусиль, що виявляються в експериментах, обчисленнях і налаштуваннях після встановлення. Виробники часто надають документацію щодо процедури калібрування, хоча в деяких випадках, такі процедури можуть бути достатньо складними, що потребує додаткового вивчення матеріалу з наукової літератури. Після будь-яких змін у системі може знадобитися тривале повторне калібрування.[8]

Особливо для більш складних датчиків необхідне значне обумовлення та обробка сигналу датчика. Це може включати посилення сигналів, придушення шуму, перетворення даних з аналогового на цифровий або з цифрового на аналоговий, тощо. Апаратне забезпечення зазвичай надається

для таких цілей виробником і повинно розглядатися як частина комплекту датчиків для конструкції роботизованих робочих станцій.

Система технічного зору

Система технічного зору - це система, що забезпечує виявлення, автоматичний контроль та аналіз об'єктів за їхніми зображеннями. Вона може використовуватися як система управління роботом для виконання таких дій, як підйом та переміщення об'єктів та узгоджено виконувати складні та повторювані завдання на високій швидкості,. Усі системи технічного зору мають датчик зображення та програмне забезпечення контролю для обробки зображень та визначення вихідного сигналу системи.[9]

РОЗДІЛ 2

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ СИМУЛЯЦІЇ МОДЕЛЮВАННЯ РОБОТОТЕХНІЧНИХ СИСТЕМ

До ПЗ для симуляції моделювання робототехнічних систем в мене було декілька вимог. Я шукав ПЗ, що як мінімум, повинне:

- бути безкоштовним
- небагато важити і мати низькі системні вимоги
- працювати навіть без виходу в інтернет
- підтримувати різні мови програмування

Маючи досить чітку ціль, таким чином я натрапив на ПЗ під назвою «CoppeliaSim». Прочитавши про нього побільше я зрозумів, що це ідеальне ПЗ для моєї роботи. Воно задовольняє моїм вимогам, має цікаві для дослідження елементи і досить глибокий функціонал. До нього навіть можна підключити код написаний на Java, що для робототехніки, можна сказати, взагалі велика рідкість. Тоді я подумав, що такий потенціал можна направити в розробку цілого конвеєра, де управляючий код можна написати на чому завгодно, налаштувати підключення роботів до мережі, … Але це досить багато роботи, тому далі я опишу більш докладно сутність і основи CoppeliaSim.

Якщо хочеться відчути як керувати промисловими або мобільними роботами, які коштують мільйони, не боячись того, що якась частина обладнання буде зламана, то дане середовище моделювання може допомогти у цьому. CoppeliaSim - це симулятор, створений компанією «Capella Robotics», що має широкі можливості налаштування: майже кожен крок моделювання визначається користувачем. Тут використовуються теорії моделювання динаміки для створення фізичного двигуна, що є досить реалістичним. У цьому середовищі є можливість створювати власні моделі приводів та конструкцій роботів, а також інших динамічних чи статичних

об'єктів. Важлива перевага CoppeliaSim - наявність великої кількості готових моделей роботів з готовими системами керування. У CoppeliaSim є бібліотеки для програмування роботів за допомогою C/C++, Python, Java, Matlab та інших мов. І найголовніше те, що програма дуже зручна і проста у використанні.

2.1. ОСНОВИ РОБОТИ В COPPELIASIM

Навіть якщо керуючі програми будуть написані на Python або C++ та підключені ззовні до симуляції, все одно є сенс розібрати скрипти CoppeliaSim. Нижче за допомогою діаграми можна прослідкувати за різницею написання керуючої частини в самому CoppeliaSim і ззовні. За допомогою таких скриптів можна все полегшити. Частину функціоналу можна інкапсулювати прямо у моделі своїх власних роботів або інших елементів, щоб потім було набагато простіше до них звертатися із зовнішніх програм.

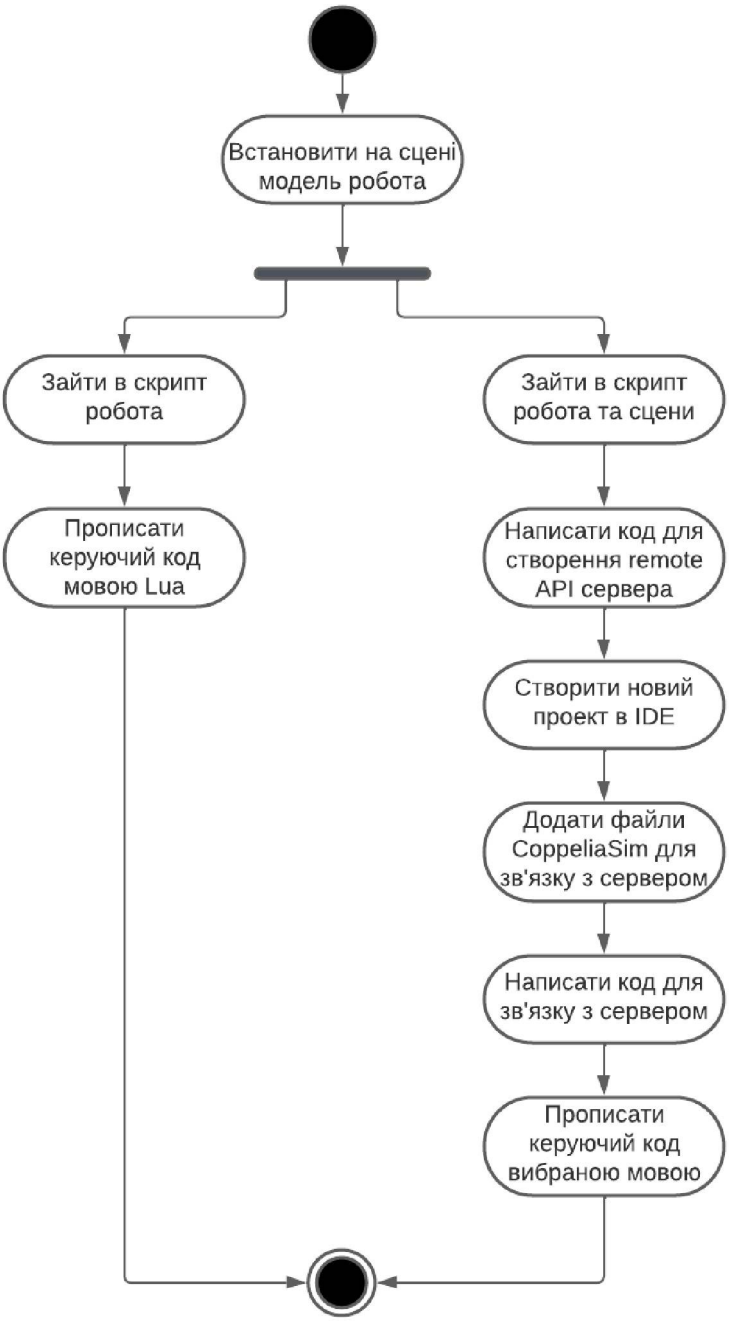

Рисунок 2.1. Написання керуючої програми

Кожному елементу можна призначити на сцені один скрипт. Існують декілька типів скриптів: дочірні скрипти (Child scripts), потокові та непотокові (Threaded and Non-Threaded), а також налаштовуючі (Customizable). Створюються скрипти через контекстне меню на вкладці

«Add». Докладніше про всі види скриптів можна почитати в розділах Writing code\ Embedded scripts мануала до CoppeliaSim (User Manual). Далі буде короткий опис скриптів.

Дочірній потоковий скрипт створює окремий потік у програмі симулятора. У цьому скрипті можна викликати будь-які команди regular API. Створюються дві головні функції «sysCall_cleanup» і «coroutineMain». Також можна створити ініціалізуючі функції, які будуть викликатися заздалегідь при створенні об'єкта або на початку симуляції. «sysCall_cleanup» створюється під час закінчення симуляції або видалення об'єкта, а функція «coroutineMain» викликається на початку симуляції та створює окремий потік, усередині якого ми можемо налаштувати цикл, який вже буде виконувати потрібні дії. Якщо в потоці скрипта відбувається помилка в частині «coroutineMain», воно зупиняє цей скрипт, але при цьому сам симулятор продовжує працювати. Потік симулятора та потік скрипта не синхронізовані, тому можуть виникати різні помилки. Наприклад, потік основного скрипта запитує значення сигналів з датчиків світла, які ще не були встановлені в скриптах відповідних датчиків.

Дочірній непотоковий (non threaded) скрипт працює іншим чином. Він має подібні до потокового скрипту функції ініціалізації та очищення (clean-up) в кінці симуляції. Також він може містити функцію «callback», що викликається на кожному кроці симуляції із самим симулятором. Наприклад, датчик vision-sensor може проводити в цій функції розпізнавання об'єктів за кольорами або формами, переробляти кадр показання освітленості, або щось подібне. Дочірній потоковий скрипт повністю синхронізований з основним потоком виконання. Цей тип скриптів працює лише при запуску симуляції.

Усі функції дочірніх непотокових скриптів можна розділити на чотири фази:

1. ініціалізація - sysCall_init (виконується один раз при створенні об'єкта або запуску симуляції)

2. активація - sysCall_actuation (виконується на кожному кроці симуляції)

3. сенсінг - sysCall_sensing (викликається на кожному кроці симуляції, після активації в цій функції відбувається зчитування та обчислення показань датчиків)

4. очищення - sysCall_cleanup (викликається один раз після видалення об'єкта або після зупинки симуляції)

Головна проблема під час роботи з подібними скриптами (дочірніми непотоковими) - створення функції callback, що викликається на кожному етапі симуляції. Ця функція не створюється автоматично, тому доводиться створювати її «руками». Для цього потрібно розібратися з питанням: «Яка функція у якого об'єкта доступна ?». Наприклад у об'єктів «joint» може бути доступна функція «joint callback», у vision-sensor може бути доступна функція «vision». У всіх об'єктів можуть бути доступні функції dynCallback (dynamic), що викликаються до та після обчислення динамічних переміщень фізичним двигуном симулятора. Також функція «contactCallback» доступна для кожного об'єкта. «contactCallback» викликається, якщо об'єкт до якого приєднаний цей скрипт дотикається до іншого об'єкту. Багато з цих функцій під час створення додаються коментарі, в яких все докладно написано.

Ще варто звернути увагу на те, що функції у дочірніх непотокових скриптах - блокуючі. Управління всієї симуляції переходить до них і повертається до головного потоку тільки після return функції. Якщо всередині цієї функції відбувається помилка, то повністю зупиняється вся симуляція.

Налаштовуючі (або Customizable) скрипти схожі на дочірні непотокові і можуть виконувати їх функції. Але кастомізуючі скрипти додатково мають функції, які виконуються поза симуляцією. Наприклад, за допомогою подібних скриптів можна налаштувати підлогу в усіх сценах чи викликати графічний інтерфейс, в якому також написати налаштування. Ці скрипти можуть виконуватися при додаванні об'єкта з бібліотеки, чи під час його активації, тощо. Далі буде розглянуто тему збереження власних моделей.

Крім роботів від різних компаній, що відразу можна поставити на сцену, також можна створити і зберегти свою модель. Якщо створити на полі дві однакові моделі роботів за замовчуванням, то можна помітити, що вони виконуватимуть однакові дії. Це набагато зручніше, ніж щоразу налаштовувати робота або робити ще щось подібне. Для підготовки моделі робот повинен інкапсулювати у собі всі необхідні скрипти, властивості та параметри. Має сенс переглянути всі компоненти робота та повністю налаштувати всі його параметри, всі його скрипти, щоб у майбутньому отримати доступ до його функціоналу з будь-якої мови програмування. Наприклад, я можу додати скрипт до об'єкта камери (robCam). Щоб потім зчитувати значення (кадри) з цієї камери за допомогою зовнішнього скрипта python. Властивості датчиків, їх область видимості, робочий простір, властивості приводу, такі як: граничні значення, максимально допустима швидкість та зусилля, кути повороту теж повинні бути налаштовані заздалегідь. Все це можна зберегти в моделі, щоб потім застосовувати в кожному новому екземплярі робота при створенні з бібліотеки.

Щоб зберегти модель треба перйти в меню та натиснути “Save model as”. Потім вибрати папку для цього (найкращим рішенням буде - створити свою). Заздалегідь краще знизити область видимості датчиків, щоб потім можна було вдало зберегти зображення моделі робота. Хоча можна і додати будь-яку іншу підготовлену картинку. Також можна зрозуміти, що початкові координати робота не зберігатимуться (що досить логічно). Model browser відображає нову папку з новою моделлю після перезапуску програми. Щоб зберегти поле, потрібно в його властивостях натиснути галочку навпроти пункту «Object is a model base», інакше не буде пункту «Save model as».

Якщо потрібно зберегти відразу кілька геометричних об'єктів, то для цього потрібно налаштувати їхню ієрархію. Якийсь елемент треба позначити як базовий, а інші підпорядкувати йому. У цього базового об'єкта потрібно поставити галочку. Якщо невідомо який об'єкт краще позначити як базовий, можна створити порожній об'єкт (dummy) і підпорядкувати всі об'єкти йому.

Також слід зазначити, що при збереженні своїх моделей в окрему папку, программа гарантує, що вони не пропадуть. В іншому випадку, наприклад, після оновлення програми, в загальній папці всі збережені елементи скоріш за все будуть видалені.

2.2. ІНСТАЛЯЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Інсталяція проводилась на операційній системі «Windows 10».

Встановлення «CoppeliaSim».

Для цього потрібно перейти за наступним посиланням: "https://www.coppeliarobotics.com/downloads", та завантажити версію Edu. Після запустити інсталятор і вибрати потрібні параметри.

Встановлення IDE Python.

Також потрібне IDE (інтегроване середовище розробки) для мови Python. Я використовував «PyCharm». Завантажити його можна за цим посиланням: «https://www.jetbrains.com/pycharm/download/#section=windows»

Вибираємо версію Community. Завантажуємо, встановлюємо.

Встановлення бібліотеки «smach»

Наступним кроком було встановлення бібліотеку «smach». Для цього можна використовувати командний рядок «cmd» (треба викликати команду «виконати» за допомогою клавіш «Win+R» і після цього ввести «cmd.exe»), а також термінал безпосередньо в самій IDE (його іконка розташована знизу зліва). Я використовував термінал.

Перш ніж інсталювати «smach», потрібно встановити необхідну для її роботи бібліотеку «catkin». Для цього в терміналі треба надрукувати або вставити наступну команду: *«pip install catkin_pkg».*

Після інсталяції пакета python може попросити встановити нову версію. Для цього пишемо команду (рис. 2.2.): «*python -m pip install --upgrade pip*»

```
[notice] To update, run: python.exe -m pip install --upgrade pip
(venv) PS D:\projects\CoppeliaSim\pythonPart> python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in d:\projects\coppeliasim\pythonpart\venv\lib\site-packages (22.3.1)
Collecting pip
  Downloading pip-23.0.1-py3-none-any.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 6.9 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.3.1
    Uninstalling pip-22.3.1:
      Successfully uninstalled pip-22.3.1
Successfully installed pip-23.0.1
(venv) PS D:\projects\CoppeliaSim\pythonPart>
```

Рисунок 2.2. Успішна інсталяція нової версії

Далі завантажуємо архів з наступного репозиторія: «https://github.com/ros/executive_smach». Для цього натискаємо кнопку «Code» і вибираємо «Download ZIP» (рис. 2.3.)

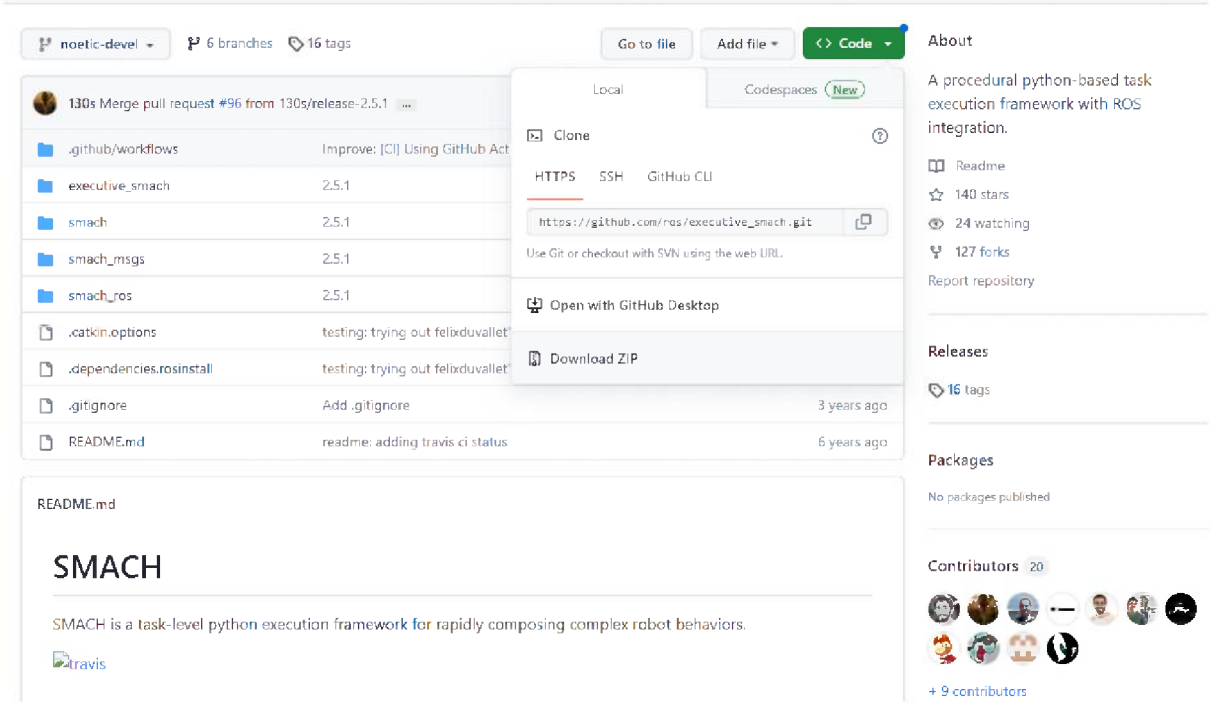

Рисунок 2.3. Завантаження архіву з GitHub

Розархівувати архів можна у будь-яку папку. В мене це папка з такою адресою: «*D:\projects\CoppeliaSim*». Тепер потрібно перейти до каталогу

«smach» цієї папки через термінал (чи командний рядок). Треба написати: «*cd PATH_TO_SMACH\executive_smach-noetic-devel\smach*»

PATH_TO_SMACH - це повний шлях до розархівованих файлів.

Я вводив наступний рядок:

«*cd D:\projects\CoppeliaSim\executive_smach-noetic-devel\smach*»

Перейшовши до нового каталогу, я написав для інсталяції наступну команду (рис. 2.4.): «*python setup.py install*»

Рисунок 2.4. Інсталяція «smach»

Залишилося перевірити, чи встановився модуль «smach». Для цього вбиваємо наступний рядок: «*py -m pydoc modules*». Шукаємо після цього «smach». Якщо він є, то установка пройшла успішно (рис. 2.5.).

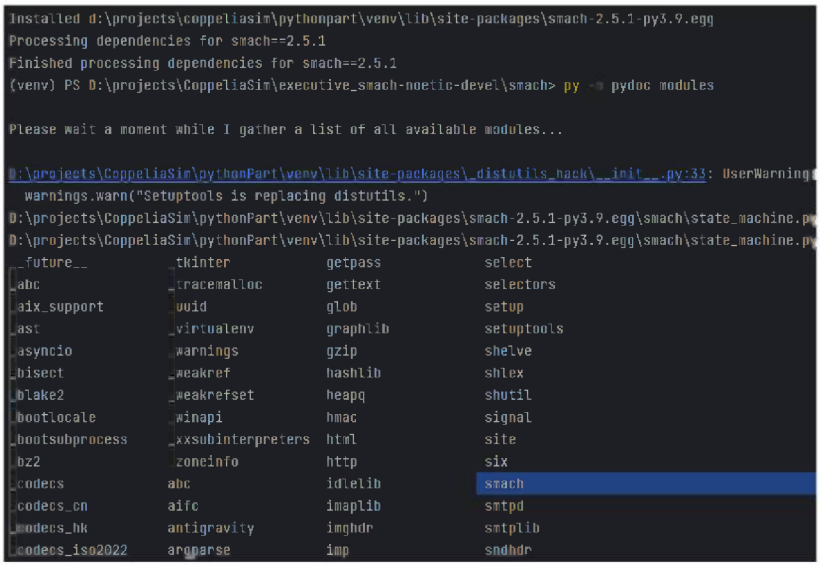

Рисунок 2.5. Перевірка встановленої бібліотеки

2.3. НАЛАШТУВАННЯ COPPELIASIM І PYTHON

Першим кроком необхідно створити нову сцену і розмістити на ній робота (рис. 2.6.).

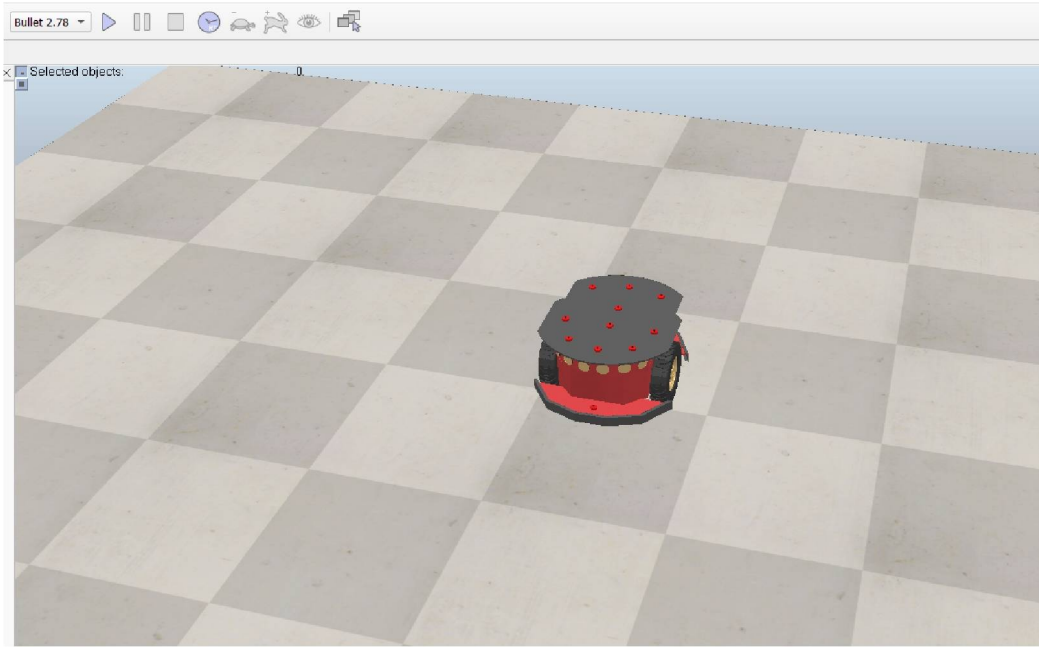

Рисунок 2.6. Розміщення на сцені робота Pioneer

А далі налаштувати сцену, зайшовши до головного скрипта. Потрібно вставити на початку скрипта наступні рядки, що відповідають за створення sim сервера на комп'ютері, а також дозволяють управління з іншого додатка (рис. 2.7.).

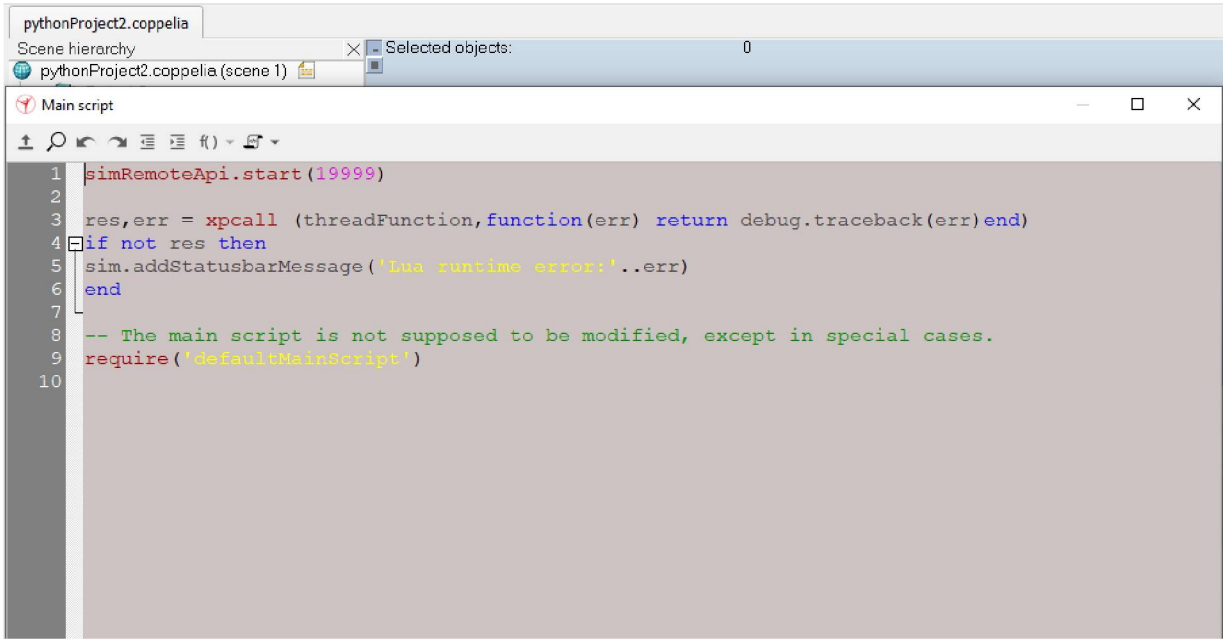

Рисунок 2.7. Написання коду в скрипті сцени

Далі необхідно створити окремий проект в IDE «PyCharm». Для роботи з програмою CoppeliaSim у папку проекту треба помістити файли: sim, simConst, simpleTest. Вони розташовані за адресою: “C:\ProgramFiles\CoppeliaRobotics\CoppeliaSimEdu\programming\legacyRemote Api\remoteApiBindings\python\python” (рис. 2.8.)

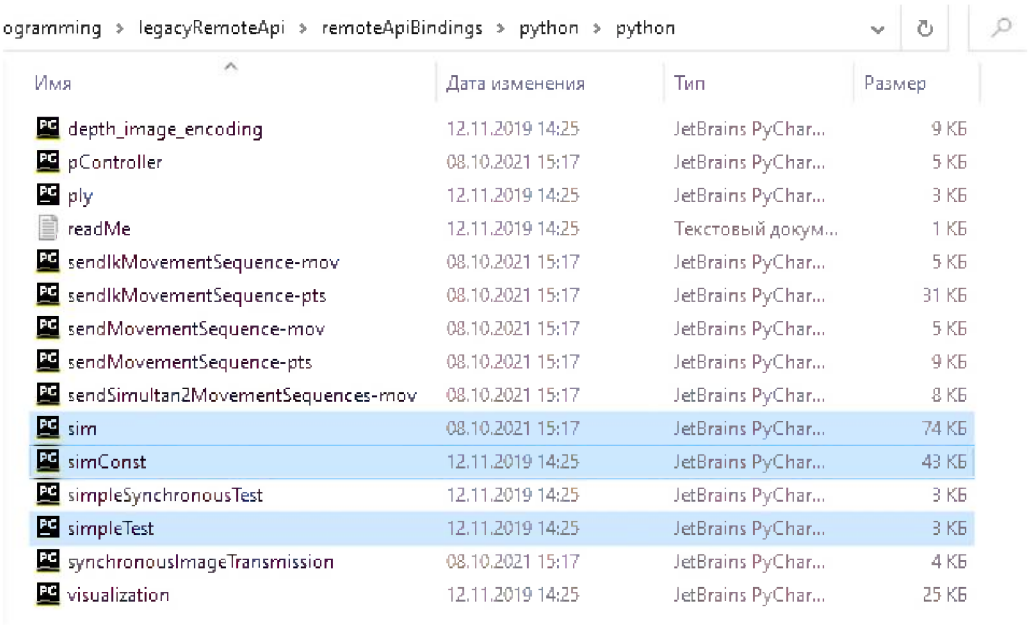

Рисунок 2.8. Розміщення необхідних файлів

Також потрібен файл з назвою «remoteApi.dll», який знаходиться за схожою адресою, де відрізнюються назви тільки двох останніх папок: «C:\ProgramFiles\CoppeliaRobotics\CoppeliaSimEdu\programming\legacyRemote Api\remoteApiBindings\lib\lib\Windows». Додатково, про всякий випадок, можно перевірити імпорт файлів за допомогою простого коду. Далі я зв'язався з сервером CoppeliaSim. Для цього використовується конструкція команд, що:

- закриває всі поточні з'єднання;
- зв'язується з sim-сервером на даному комп'ютері;
- перевіряє з'єднання і виводить результат цього.

В output цього блоку з'являється змінна clientID, яка відповідає за зв'язок Python із сервером «CapeliaSim». Щоб правильно перевірити з’єднання, потрібно запустити симуляцію, а слідом написаний код. Якщо з'єднання встановлено, то все було зроблено правильно.

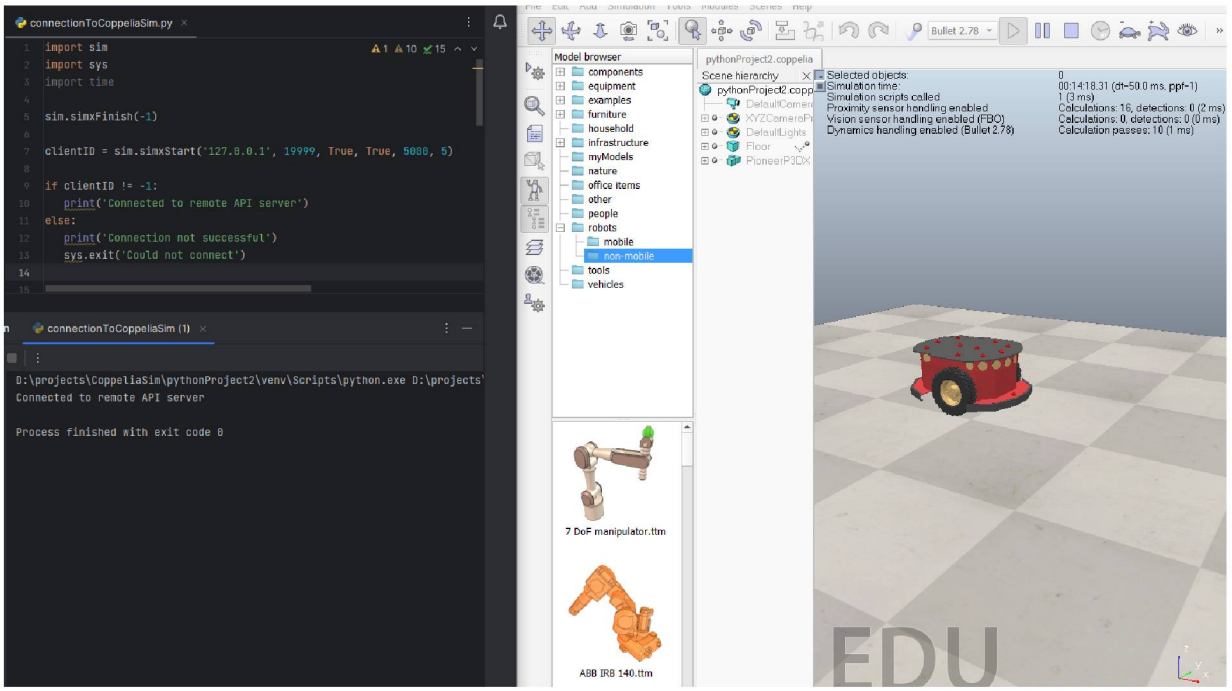

Рисунок 2.9. Вдала спроба з’єднання

Далі залишається тільки написати керуючий код, тому буде наведено приклади основних функцій, що можуть допомогти у цьому. Спочатку в любому випадку треба буде створити посилання в Python на об'єкти CoppeliaSim. Для цього використовується команда:

«errorCode, HandleName = vrep.simxGetObjectHandle(clientID, 'NameObject', vrep.simx_opmode_oneshot_wait)».

В цій функції наведено такі параметри і значення, що повертаються:

- «clientID» - посилання на зв'язок з CoppeliaSim
- «'NameObject'» - ім'я об'єкта на який дається вказівник, де ім'я береться з браузера моделей
- «vrep.simx_opmode_oneshot_wait» - тип зв'язку (в даному випадку для цієї функції використовується лише разовий зв'язок)
- «errorCode» - код помилки (якщо її нема, повертається 0)
- «HandleName» - отримане посилання на об'єкт

Далі для реалізації програми знадобиться положення різних елементів:

«errorCode, Position = vrep.simxGetObjectPosition(clientID, HandleName, -1, vrep.simx_opmode_streaming)»

- «ClientID» - посилання на зв'язок з CoppeliaSim

- «'HandleName'» - посилання на об'єкт, що цікавить
- «-1» визначає абсолютні координати в системі координат сцени
- «vrep.simx_opmode_streaming» - тип зв'язку (в даному випадку це постійна прив'язка параметра CoppeliaSim до параметра коду)
- «errorCode» - код помилки (якщо її нема, повертається 0)
- «Position» - отримані координати об'єкта

Також може знадобитися орієнтація елементів. Для цього використовується функція: «errorCode, Orientation = vrep.simxGetObjectOrientation(clientID, HandleName, -1, vrep.simx_opmode_oneshot_wait)»

- ті ж самі параметри, що і для минулої команди
- «Orientation» -отримані три кути положення об'єкта

І також функція присвоєння швидкості обертання мотора: «errorCode = vrep.simxSetJointTargetVelocity(clientID, HandleName, value, vrep.simx_opmode_oneshot_wait)»

- ті ж самі параметри, що і для минулої команди
- «value» - значення швидкості, яке ми хочемо привласнити

Цих функцій буде достатньо, щоб змусити робота Pioneer розпочати рух. Можна, наприклад, написати програму, що дозволить роботу їхати до певного об’єкту чи просто в заданому напрямку, тощо. Тепер знаючи основи CoppeliaSim (і навіть трішки більше) можна перейти до наступного розділу.

РОЗДІЛ 3
СИМУЛЯЦІЯ РОБОТІВ-МАНІПУЛЯТОРІВ

3.1. МОДЕЛЮВАННЯ ВЛАСНОГО МАНІПУЛЯТОРА

Можна достатньо добре зекономити, якщо купити всі комплектуючі маніпулятора самому, тому було вирішено протестувати можливості ПЗ (програмного забезпечення) і спробувати підготувати тривимірну модель маніпулятора та імпортувати її.

Як надалі виявилось CoppeliaSim дозволяє імпорт тривимірних моделей, які можна зробити за допомогою спеціалізованого ПЗ, як SolidWorks, наприклад. Для цього необхідно знайти (або зберегти) збірку маніпулятора з розширенням STL. Файли у форматі STL зберігають свої координати, тому надалі буде зручно працювати з ними в симуляції. Далі треба запустити CoppeliaSim і в головному меню перейти на вкладку «File\Import\Mesh», вибрати потрібні файли та натиснути «Відкрити». Далі в з'явившомуся вікні задати масштаб вже в новій системі координат. Коли всі налаштування коректно задані, залишається тільки натиснути «ОК».

Після імпорту модель повинна з’явитись на сцені, де буде видно ієрархію компонентів, що відповідають кожній частині промислового маніпулятора (рис 3.1.).

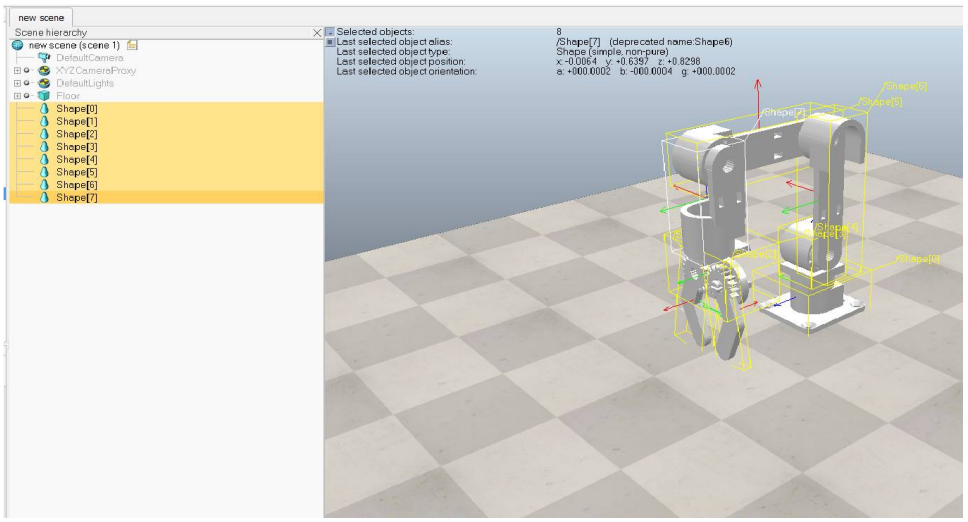

Рисунок 3.1. Вдалий імпорт моделі робота-маніпулятора

Але далі треба ще провести налаштування, щоб можна було керувати створеною моделлю. Це буде описано нижче, але також про це можна більш докладніше прочитати в мануалі до CoppeliaSim (User Manual). [10]

Взагалі, елементи після імпорту можуть бути використані для симуляції, проте робити це не рекомендується, так як елементи цього типу не оптимальні для розрахунку фізичних властивостей. Тому треба створити модифіковані копії цих елементів, які вже будуть використовуватись для моделювання фізики. Виділяємо один елемент моделі, заходимо в контекстне меню за допомогою правої кнопки мишки і натискаємо “Add/Convex decomposition of selection”. Після введення значень, що дозволяють створити оптимізовану сітку, закриваємо меню і отримуємо модифіковані копії комплектуючих. Рекомендується перейменувати нові компоненти так, щоб у них були такі ж назви, як у вихідних, але з додаванням закінчення «_dyn». Далі необхідно зв'язати початкові компоненти з створеними копіями (елементи з закінченням «_dyn») так, щоб динамічні компоненти були «батьками» для початкових компонентів.

Після проведення таких маніпуляцій в сценарії повинні з'явитися об'єкти, що перекривають одне одного. Необхідно ці об'єкти рознести на різні пласти відображення. Знову виділяємо елементи з оптимізованою сіткою і в властивостях знаходимо пункт “Camera visibility layers” і ставимо там галочку. У результаті отримуємо початковий маніпулятор на сцені.

Також можна перемикатися на відображення компонентів іншого типу (рис 3.2.) (рис 3.3.).

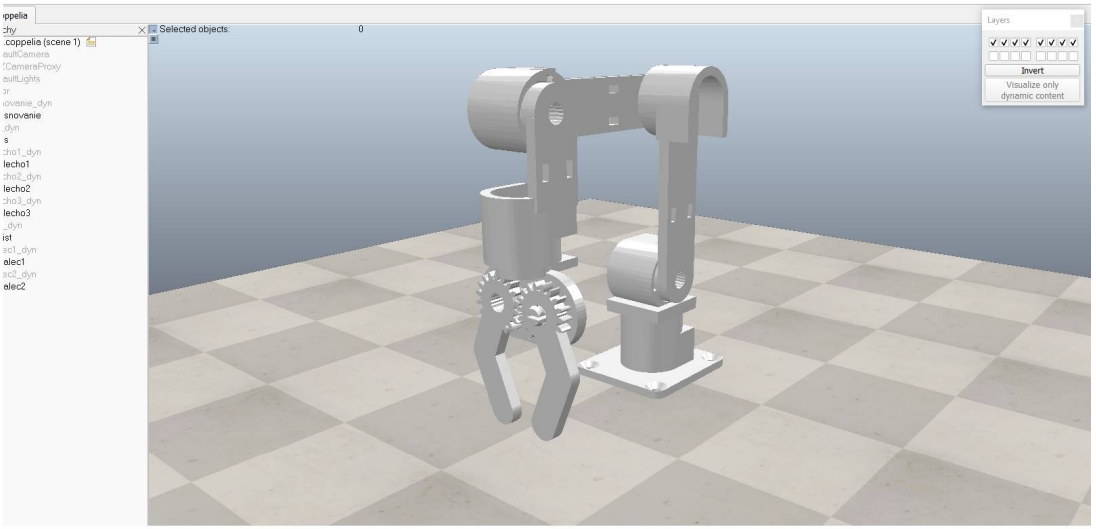

Рисунок 3.2.Перший пласт відображення

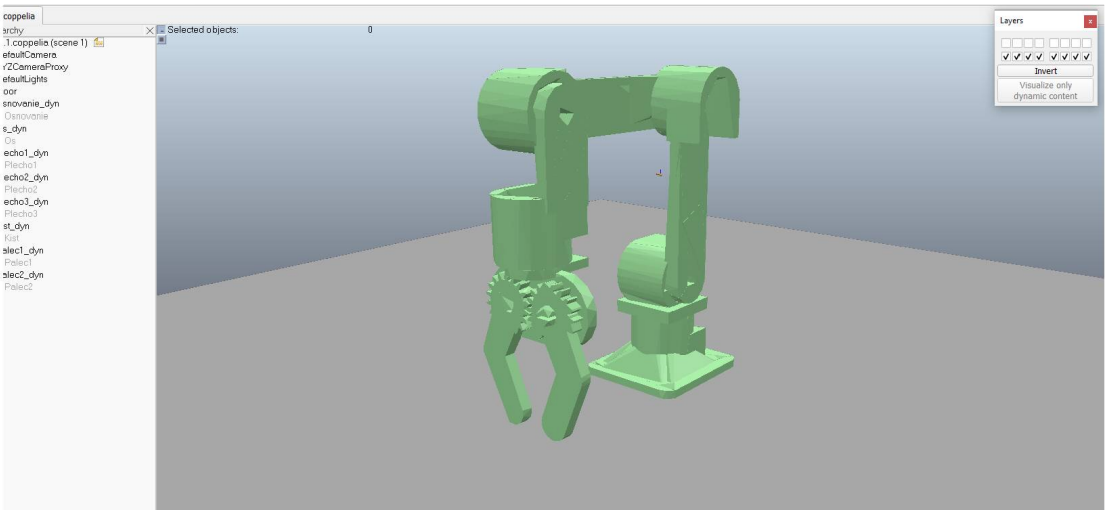

Рисунок 3.3. Другий пласт відображення

Тепер залишилось налаштувати все так, щоб оптимізовані під динаміку елементи моделювали динаміку (що досить логічно), а початкові компоненти не брали участі у розрахунках, а лише повторювали поведінку динамічних елементів. Для цього достатньо поставити декілька галочок у властивостях всіх динамічних елементів: “Body is Respondable” (Розрахунок реакцій, пов'язаних з тілом), “Body is dynamic” (Розрахунок динамічних

характеристик). А також для автоматичних геометричних розрахунків треба натиснути “Compute mass and inertia properties” і зберегти все як є (рис 3.4.).

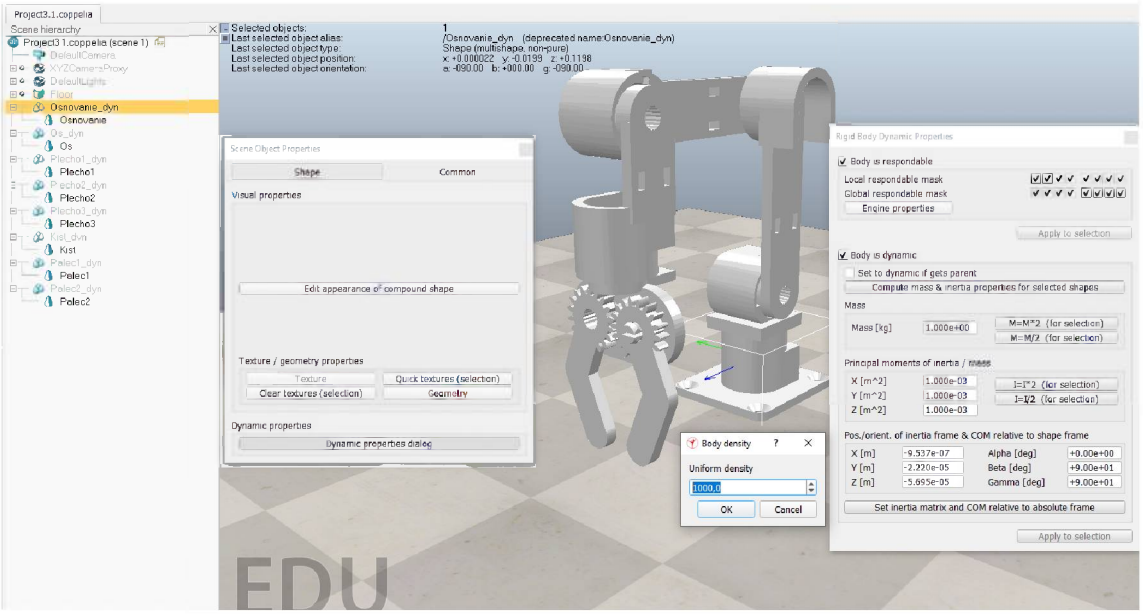

Рисунок 3.4. Налаштування динамічних елементів

Також в CoppeliaSim є функція встановлення маски взаємодії. З її допомогою можна вказати ігнорування взаємодії між елементами. Припустимо, що два тіла з'єднуються за допомогою шарніра. Дуже просто уявити як вони будуть рухатись відносно один одного. І найголовніше, не будуть діяти один на одного. Але в даному ПЗ при моделюванні ми можемо отримати таку ситуацію, що сітки двох поруч розташованих елементів стикаються, і для вирішення цієї проблеми потрібно було б заново створювати більш дрібну сітку, що призвело б до збільшення виділених ресурсів комп'ютера. Використовуючи маски, можна легко вирішити такі проблеми. Достатньо перейти до властивості «Local respondable mask» і поставити галочки в кожному елементі різним чином. В кожного буде унікальна локальна маска.

Тільки треба пам’ятати, що при простому накладанні маски, всі деталі маніпулятора розсиплються, так як відсутні будь-які з'єднання-шарніри (рис 3.5.).

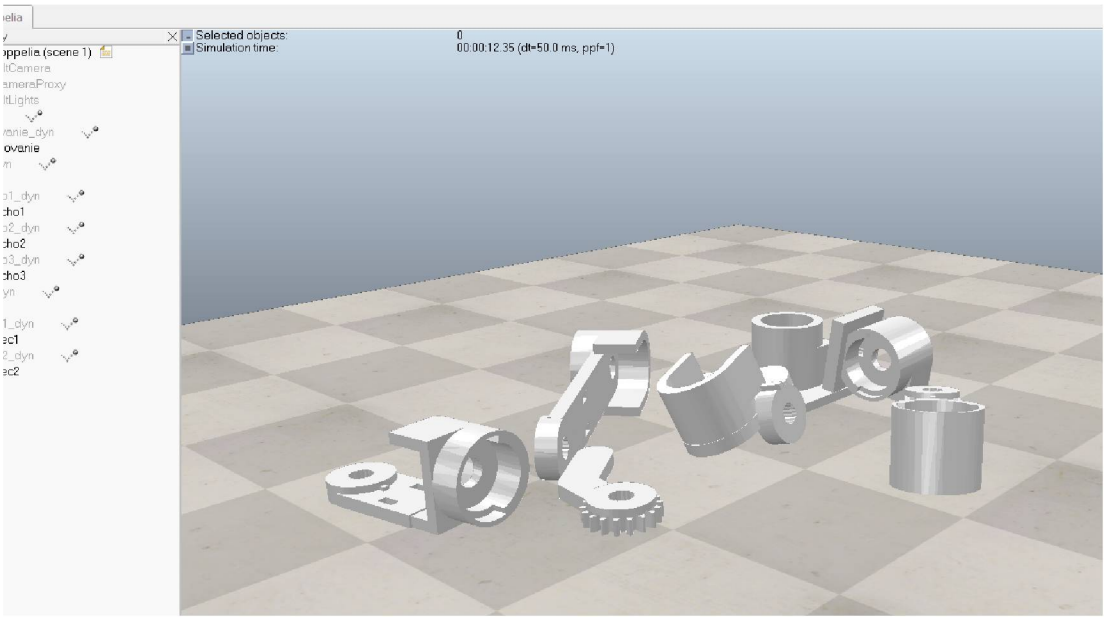

Рисунок 3.5. Запуск симуляції робота-маніпулятора без шарнірів

Для з'єднання частин компонентів маніпулятора потрібно додати шарніри («Joint») обертального типу («Revolute»). А для їх позиціонування є відразу декілька способів (і не просто так). Це найскладніша частина процесу. Описувати цей процесс досить довго, тому якщо підсумувати, то можна виділити наступне. Для початку шарнір можна відкалібрувати відносно частини в яку він буде встановлений. Тоді він буде рівно посередині деталі.

Далі за допомогою інструментів позиціювання і орієнтації регулюємо їх доки не будуть встановлені в потрібне місце - рівно посередині отвору для шарніру (рис 3.6.).

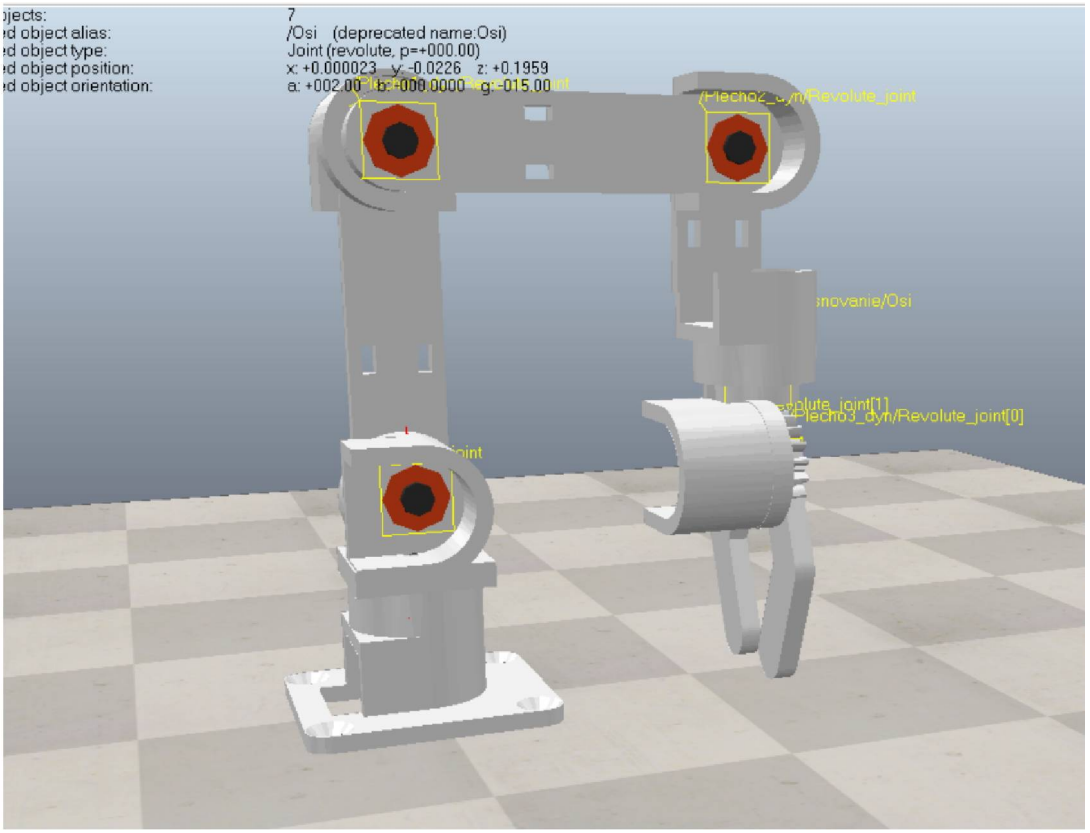

Рисунок 3.6. Відображення моделі разом з шарнірами

І нарешті, можна задавати зв'язки між елементами. Для цього створюємо деревоподібну структуру в ієрархії сцени. Виставлення ієрархії досить просте. Все починається з динамічного об'єкта основи маніпулятора, (він є першим «предком»), далі йдуть два «нащадки» - шарнір і наступний динамічний об'єкт (той що найближчий до основи), і з кожним динамічним об’єктом теж саме.

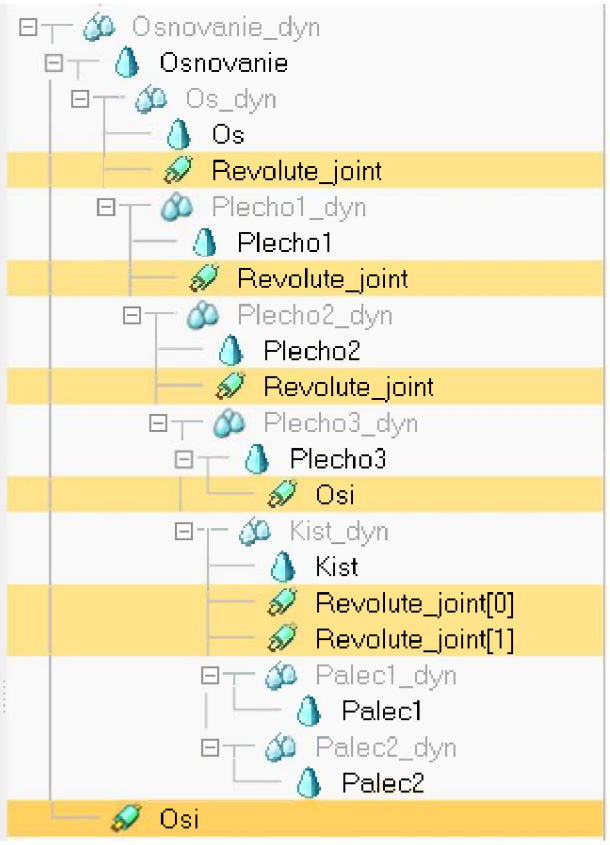

Рисунок 3.7. Ієрархія елементів маніпулятора

Перейдемо до налаштування шарнірів. У контекстному вікні «Joint Dynamic Properties» ставимо галочки навпроти “Motor enabled” (включити двигун), «Control loop enabled» (включити циклічне управління) і «Lock motor when target velocity is zero» (фіксація мотора при швидкості, що дорівнює нулю) у всіх шарнірах маніпулятора.

Тепер цю модель можна зберегти, як було вказано в підрозділі 2.1 і користуватися рівно як тими, що надає ПЗ.

Наприклад, далі можна встановити ще два сенсори за допомогою команди «Add». Перший сенсор буде визначати відстань до об'єкта, а другий колір об'єкту, щоб брати об’єкти потрібного кольора, сортувати їх, тощо. Треба буде задати необхідне положення та орієнтацію сенсорам, потім встановити прив'язку до кінцевої ланки маніпулятора і написати керуючий код.

Таким чином “CoppeliaSim” надає можливість створення своєї моделі, щоб її надалі можна було продемонструвати, протестувати, тощо.

3.2. ЗАДАЧА З КІНЕМАТИКИ ДЛЯ ПРОМИСЛОВОГО РОБОТА

Тепер нарешті перейдемо до програмування існуючого промислового робота-маніпулятора, тільки попередньо ще трішечки торкнемося теорії.

У робототехніці існує дві основні задачі кінематики: пряма та зворотна [3]. Пряма задача – це обчислення положення (по координатам) робочого органу маніпулятора за заданою орієнтацією його ланок та за його кінематичною схемою. Звідси можно зробити висновок, що рішення такої задачі вкаже положення робочого органу маніпулятора відносно заданих кутів. А назва “зворотна” каже сама за себе (хід рішення тепер буде зворотнім). Отримані теоретичні рішення задачі треба підтвердити на практиці, але застосування для таких цілей реального промислового робота не завжди є можливим. Тому оптимальним варіантом, на мій погляд, є застосування програм для симуляції моделювання робототехнічних систем. Треба тільки написати програмну реалізацію. Нижче для данного завдання наведена UML-діаграма.

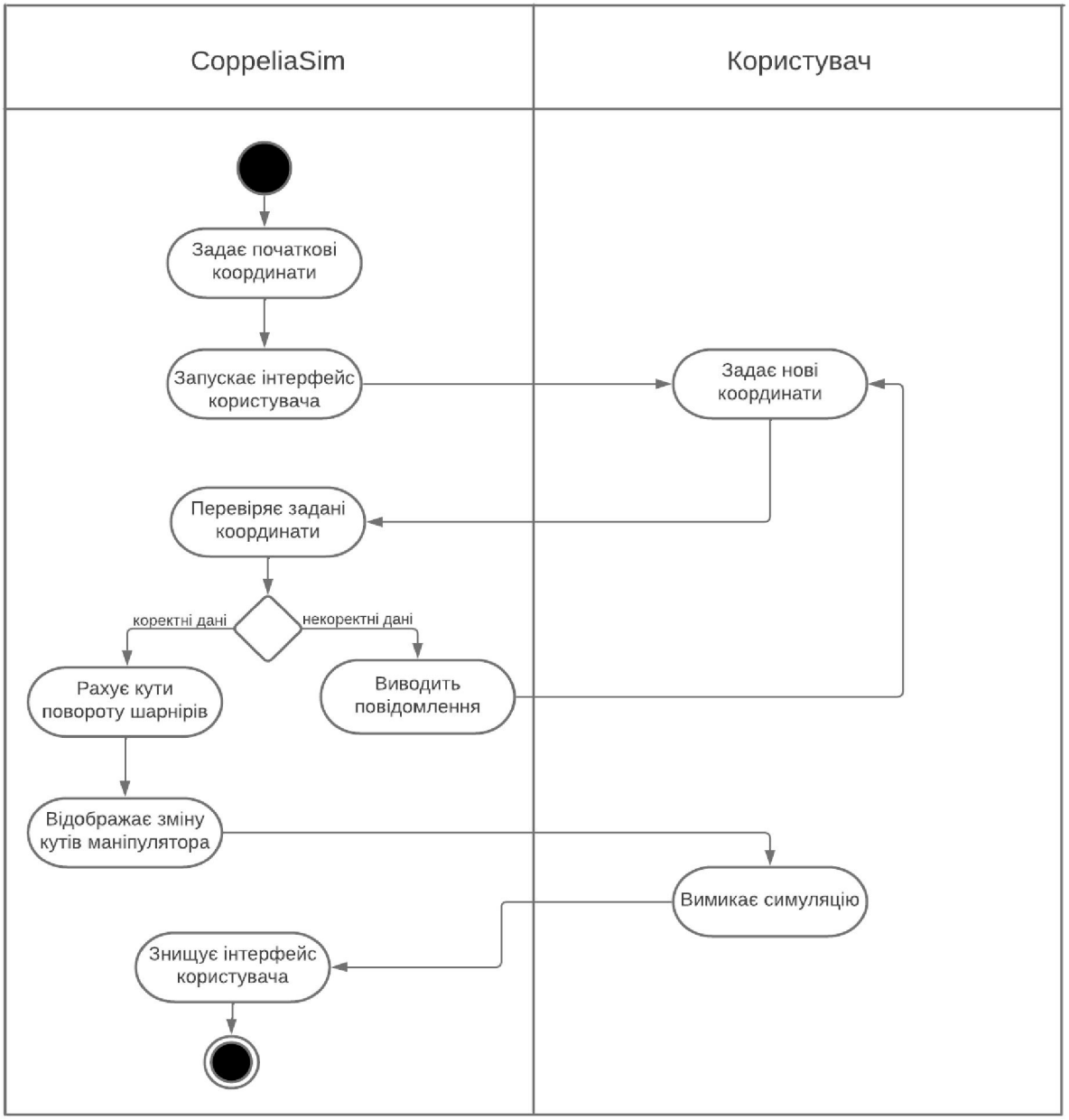

Рисунок 3.8. UML-діаграма для реалізації поставленого завдання

Модель робота ABB IRB 140 - це компактний та ефективний промисловий робот з шістьма ступенями свободи, здатний працювати із завантаженням до 6 кг.

Додамо модель робота ABB IRB 140 на сцену (рис. 3.9.).

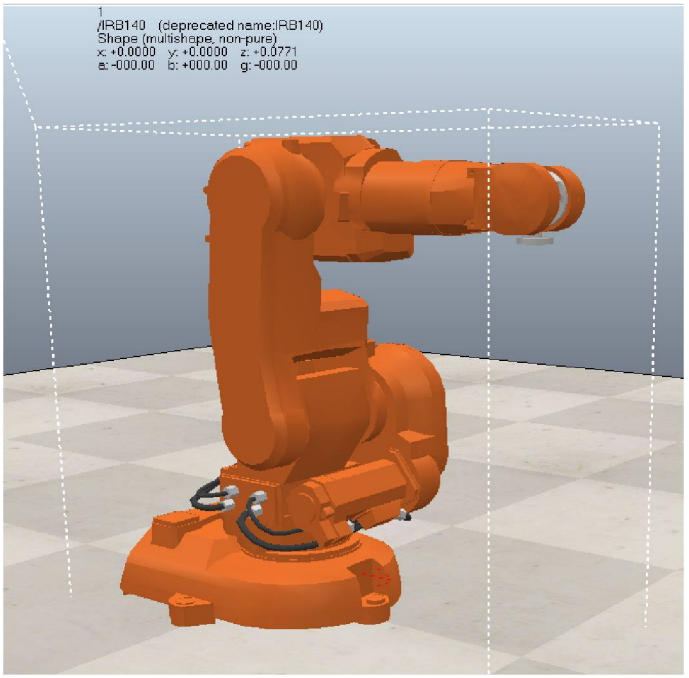

Рисунок 3.9. Розміщення на сцені ABB IRB 140

Для роботи з ним необхідно видалити непотрібні для прямої задачі об'єкти в ієрархії сцени та бажано перейменувати решту для того щоб, не плутати елементи робота (рис. 3. 10.).

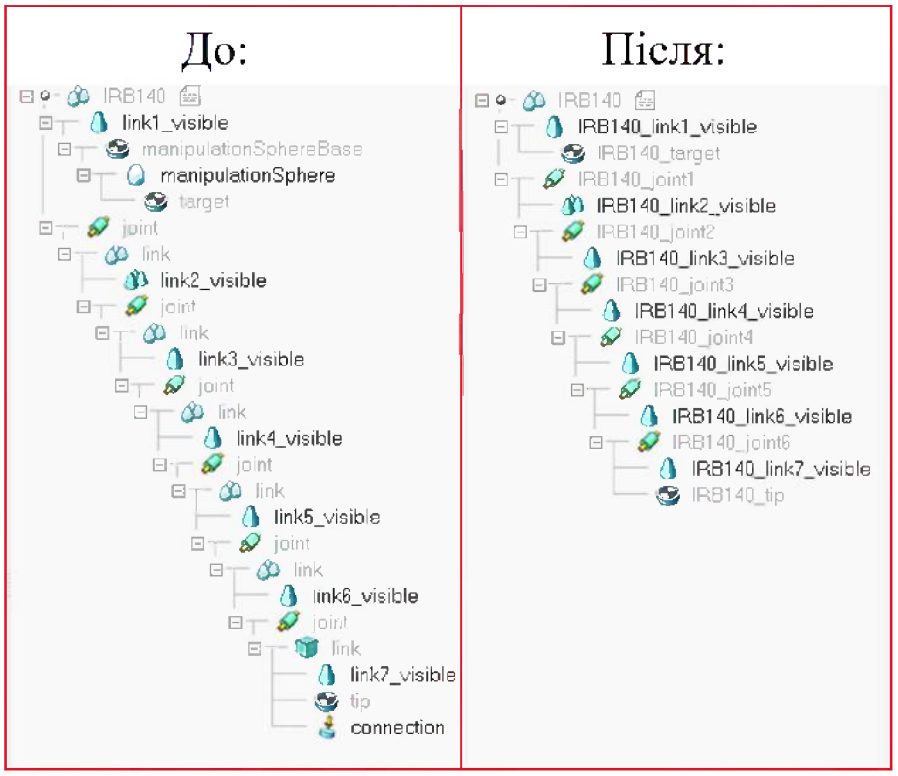

Рисунок 3.10. Вигляд ієрархія сцени до та після маніпуляцій

Для всіх шарнірів (IRB140_joint1 і тд) у вікні «Scene Object Properties» у вкладці «Joint» у полі «Mode» значення «Torque/force mode» необхідно змінити на «Passive mode».

Далі розглянемо структуру непотокового дочірнього скрипта управління роботом (рис. 3.11.). Тут буде доповнюватись інформація до підрозділу 2.1. Скрипти пишуться на мові Lua.

```
1  function sysCall_init()
2      -- do some initialization here
3  end
4
5  function sysCall_actuation()
6      -- put your actuation code here
7  end
8
9  function sysCall_sensing()
10     -- put your sensing code here
11 end
12
13 function sysCall_cleanup()
14     -- do some clean-up here
15 end
16
```

Рисунок 3.11. Загальна структура скрипту управління

Скрипт у цьому прикладі виконує чотири основні функції, а саме:

sysCall_init() – функція ініціалізації. Викликається під час запуску симуляції. Вона викликається лише один раз і зазвичай використовується для ініціалізації змінних, інтерфейсу користувача і т.п. Функція sysCall_init() завжди викликається до sysCall_actuation().

sysCall_actuation() – функція активації. Виконується циклічно у процесі симуляції. Є основною функцією управління роботом.

sysCall_sensing() - Функція датчиків. Ця функція призначена для опитування стану датчиків під час симуляції, і виконується на кожному етапі симуляції.

sysCall_cleanup() – функція очищення. Ця функція викликається після зупинки симуляції. Функція відповідає за відновлення початкової конфігурації робота, очищення станів датчиків, тощо.

CoppeliaSim може створити інтерфейс, що настроюється, за допомогою вбудованого плагіна Qt. При створенні інтерфейсу користувача використовується спеціальний XML синтаксис. Приклад синтаксису добре показано в навчальній сцені «customUI.ttt» в папці «scenes», що знаходиться в місці встановлення самої програми. Далі розглянемо розробку керуючого скрипта.

Для функції sysCall_init() було написано код з початковими налаштуваннями та XML частиною. (рис. 3.12.)

```
function sysCall_init()
    simJoints={0,0,0,0,0,0}
    for i=1,6,1 do
        simJoints[i]=sim.getObjectHandle('IRB140_joint'..i)
    end
    xml = '<ui title="Forward Kinematics" closeable="true" on-close="closeEventHandler" resizable="false">'..[[
    <label text="Forward Kinematics for ABB IRB 140 using CoppeliaSim and Qt"/>
    <group layout="hbox">
    <label text="J1" id="11" />
    <hslider id="21" minimum="-500" maximum="500" on-change="sliderChange" />
    <edit value="0" id="31" on-editing-finished="jointEntry" />
    </group>
    <group layout="hbox">
    <label text="J2" id="12"/>
    <hslider id="22" minimum="-500" maximum="500" on-change="sliderChange" />
    <edit value="0" id="32" on-editing-finished="jointEntry" />
    </group>
    <group layout="hbox">
    <label text="J3" id="13"/>
    <hslider id="23" minimum="-500" maximum="500" on-change="sliderChange" />
    <edit value="0" id="33" on-editing-finished="jointEntry" />
    </group>
    <group layout="hbox">
    <label text="J4" id="14"/>
    <hslider id="24" minimum="-500" maximum="500" on-change="sliderChange" />
    <edit value="0" id="34" on-editing-finished="jointEntry" />
    </group>
    <group layout="hbox">
    <label text="J5" id="15"/>
    <hslider id="25" minimum="-500" maximum="500" on-change="sliderChange" />
    <edit value="0" id="35" on-editing-finished="jointEntry" />
    </group>
    <group layout="hbox">
    <label text="J6" id="16"/>
    <hslider id="26" minimum="-500" maximum="500" on-change="sliderChange" />
    <edit value="0" id="36" on-editing-finished="jointEntry" />
    </group>
    </ui>
    ]]
    ui=simUI.create(xml)
end
```

Рисунок 3.12. Блок коду в функції sysCall_init()

В функції sysCall_cleanup() була написана команда simUI.destroy(ui), що знищує інтерфейс користувача після зупинки симуляції. (рис. 3.13.)

```
69 function sysCall_cleanup()
70 simUI.destroy(ui)
71 end
```

Рисунок 3.13. Блок коду в функції sysCall_cleanup()

Далі я додав ще одну функцію closeEventHandler(), що відповідає за обробку подій кнопки “закрити” інтерфейсу користувача (рис. 3.14.).

```
73 function closeEventHandler()
74 simUI.hide(ui)
75 end
```

Рисунок 3.14. Блок коду в функції closeEventHandler()

Також я додав функцію обробки подій слайдерів (рис. 3.15.).

```
43 function sliderChange(ui,id,newVal)
44     for i=1,6,1 do
45         if (id==20+i) then
46             simUI.setLabelText(ui,10+i,string.format('J' ..i))
47             simUI.setEditValue(ui,30+i,string.format(newVal))
48             sim.setJointPosition(simJoints[i],newVal*math.pi/180)
49             break
50         end
51     end
52 end
53
```

Рисунок 3.15. Блок коду в функції sliderChange()

І остання - це функція обробки подій редагованих полів (рис. 3.16.).

```
function jointEntry(ui,id,newVal)
    if (tonumber(newVal)==nil) then
        print("Error: could not convert number")
        return
    end
    for i=1,6,1 do
        if (id==30+i) then
            simUI.setLabelText(ui,10+i,string.format('J'..i))
            simUI.setEditValue(ui,30+i,string.format(newVal))
            sim.setJointPosition(simJoints[i],newVal*math.pi/180)
            break
        end
    end
end

```

Рисунок 3.16. Блок коду в функції jointEntry()

Вмикаємо симуляцію. І після цього можна керувати роботом за допомогою інтерфейсу користувача (рис. 3.17.).

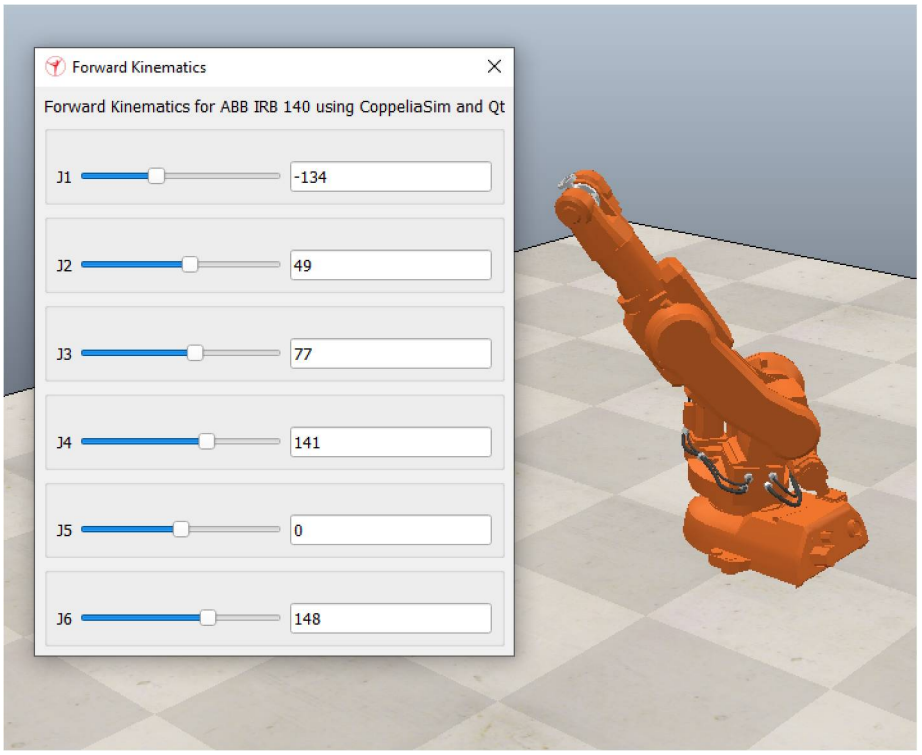

Рисунок 3.17. Вікно інтерфейсу користувача

Тепер для кожного шарніра можна задавати кути повороту в градусах і спостерігати за зміною положення його ланок, вирішивши пряме завдання за відомою кінематичною схемою – перевірка результатів практичним шляхом.

ВИСНОВОК

По-перше, треба зазначити, що застосування симулятора робототехнічних систем в освіті, дослідженнях та промисловості має низку переваг. Віртуальне моделювання доступне для всіх і не потребує значних фінансових витрат. Також ПЗ для симуляції ліквідує ризики. Немає змоги задати жодної матеріальної шкоди деталям та вузлам робота, внаслідок можливої людської помилки, а також відсутня небезпека для життя або здоров'я людини при роботі з роботом. В даній роботі було проаналізовано ПЗ для моделювання робототехнічних систем “CoppeliaSim”. Воно має декілька різних видів скриптів, що надають можливість повного налаштування симуляції. Пишуться скрипти на мові Lua, але також мною було протестовано підключення керуючої програми (що була написано автором) на мові Python зовні до симуляції, де була розміщена мобільна модель робота. Це показує велику свободу дій при реалізації проектів. Для порівняння підходів була створена UML-діаграма. До ПЗ можна додати свою модель робота створену з нуля. Воно має широкий спектр налаштувань, що частково було вказано в вигляді інструкції при додаванні автором власної моделі. Також слід зазначити, що наявність готових до роботи реально існуючих комерційних моделей роботів не розчаровує, особливо це торкається роботів-маніпуляторів. Були успішно протестовані можливості однієї з таких моделей, де в першу чергу були розглянуті такі основні характеристики як точність і досяжність робота. Пряма кінематична задача для цього підійшла дуже добре. Для відображення роботи програми спочатку була створена UML-діаграма, а вже після була написана програмна реалізація на мові Lua.

1. Робототехніка. Підручник / [В. І. Костюк, Г. О. Спину, Л. С. Ямпольський, М. М. Ткач.] - К.: Вища школа. - 1994. - 447 с.

2. Бучинський М. Я., Горик О. В., Чернявський А. М., Яхін С. В. Основи творення машин / [За редакцією О. В. Горика, доктора технічних наук, професора, заслуженого працівника народної освіти України]. — Харків: Вид-во «НТМТ», 2017. — 448 с.

3. Робототехніка та мехатроніка: навч. посіб. / Л.І. Цвіркун, Г. Грулер ; під заг. ред. Л.І. Цвіркуна ; М-во освіти і науки України, Нац. гірн. ун-т. – 3- тє вид., переробл. і доповн. – Дніпро: НГУ, 2017. – 224 с.

4. Михайлов Є.П. Маніпулятори та промислові роботи [Текст]: підручник / Михайлов Є.П., Лінгур В.М. — Одеса: ОНПУ, 2019, -233 с.

5. Дудюк Д. Л. Гнучке автоматизоване виробництво і роботизовані комплекси: Навч. посібник / Д. Л. Дудюк, С. С. Мазепа, М. М. Мисик. - Львів: "Магнолія плюс" СПД ФО В. М. Піча, 2005. - 278 с.

6. Кошель С. О. Проектування промислових роботів та маніпуляторів: посібник / С. О. Кошель, Ю. Ковалёв, О. П. Манойленко — К. :Центр навчальної літератури, 2019. — 256 с.

7. Greer, R. Advances in Control Systems for Construction Manipulators / Greer, R., Haas, C., Gibson, G.. – Austin, 2014. – 615 с. – (ISARC).

8. Лисенко, С. М. Напрямки досліджень та розвитку комп'ютерної інженерії/ Лисенко, С. М. — К. : ХНУ, 2015. — 142 с.

9. Румбешта В. О. Технологія складання, регулювання та випробування приладів / В. О. Румбешта. – Київ, 2013. – 360 с.

10. Мануал до CoppeliaSim [Електронний ресурс] – Режим доступу: https://www.coppeliarobotics.com/helpFiles/

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) бакалавр
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри теоретичної
та прикладної системотехніки
д.т.н., проф. Шматков С. І.

«19» жовтня 2020 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ (ПРОЕКТ)

Уварова Валентина Юрійовича
(прізвище, ім'я, по батькові студента)

1. Тема роботи: "Моделювання і створення базових апаратно – програмних елементів системи керування роботом – маніпулятором"

керівник роботи Рало Олександр Миколайович, старший викладач
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "17"березня 2021 року №0210-05/477

2. Строк подання студентом роботи *29 травня 2023р.*

3. Перелік питань, які потрібно розробити
   1. Аналіз існуючих моделей роботів-маніпуляторів.
   2. Аналіз принципів роботи роботів-маніпуляторів.
   3. Аналіз комплектуючих роботів-маніпуляторів.
   4. Вибір програмного забезпечення для моделювання роботи роботів.
   5. Вивчення можливостей програмного забезпечення.
   6. Побудова алгоритмів системи керування.
   7. Створення моделі робота-маніпулятора.
   8. Розробка системи керування.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Аналіз та пошук методичної літератури	Листопад 2022-Грудень 2022
2	Огляд і аналіз існуючих рішень для системи управління роботом – маніпулятором	Грудень 2022
3	Підбір і налаштування програмно-апаратного забезпечення	Січень 2023
4	Аналіз документації до вибраного програмного забезпечення	Січень 2023
5	Розробка алгоритму управління	Лютий 2023-Березень 2023
6	Імплементація системи управління роботом – маніпулятором	Березень 202 – Квітень 2023
7	Оформлення пояснювальної записки	Травень 2023

5. Дата видачі завдання «19» жовтня 2020 року

Студент В. Ю. Уваров

Керівник роботи О. М. Рало

Додаток Б

Затверджую
___№0210-05/477___
«_____»_____2023 р.

Технічне завдання
на розробку програмного виробу «Базові апаратно – програмні елементи системи керування роботом – маніпулятором»

1.	Введення	1.1. Назва: Базові апаратно – програмні елементи системи керування роботом – маніпулятором. 1.2. Галузь застосування: комп'ютерні технології, робототехніка.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 123 – Комп'ютерна інженерія 2.2. Завдання на кваліфікаційну роботу бакалавра № 0210-05/477 від «___» _____2022 (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3.	Призначення розробки	3.1. Мета розробки: дослідження програмного забезпечення для симуляції систем керування роботом-маніпулятором. 3.2. Призначення розробки надає можливість розробки системи керування та налаштування кожного елемента робота-маніпулятора на базі вже існуючих моделей або створених самим користувачем. 3.3. Вихідні дані розробки: візуальне відображення роботи робота-маніпулятора на сцені ПЗ для симуляції, а також дані, що надає графічний інтерфейс ПЗ.
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик:немає 4.2. Вимоги до надійності: немає 4.3.Вимоги до умов експлуатації: немає 4.4. Вимоги до складу і параметрів технічних засобів: ПК з досить низькими технічними характеристиками. 4.5. Вимоги до інформаційної та програмної сумісності: Рекомендовані системні вимоги ПЗ для ПК: ● 32-розрядний (x86) або 64-розрядний (x64) процесор із тактовою частотою 1 ГГц або швидший; ● 1 гігабайт (ГБ) RAM (для 32-розрядної версії) або 2 ГБ (для 64-розрядної версії);

		• 16 ГБ (для 32-розрядної версії) або 20 ГБ (для 64-розрядної версії) вільного місця на жорсткому диску; • графічний пристрій із підтримкою DirectX 9 і драйвером WDDM 1.0 або новішим. З більшими характеристиками ПЗ буде працювати тільки краще. 4.6. Вимоги до маркування та упаковки: немає 4.7. Вимоги до транспортування і зберігання: на цифрових носіях інформації. 4.8. Спеціальні вимоги: немає.	
5.	Вимоги до програмної документації	Програмною документацією до виробу «Базові апаратно – програмні елементи системи керування роботом – маніпулятором» вважати: 1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Опис налаштування та основ роботи ПЗ (у вигляді всього другого розділа пояснювальної записки до кваліфікаційної роботи). 3) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи)	
6.	Вимоги до техніко-економічних показників	Програмною документацією до виробу «Базові апаратно – програмні елементи системи керування роботом – маніпулятором» вважати: 1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Опис налаштування та основ роботи ПЗ(у вигляді всього другого розділа пояснювальної записки до кваліфікаційної роботи). 3) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи)	
7.	Стадії і етапи розробки	Дата	Назва етапу
		від 10 листопада 2022 до 10 грудня 2022	Аналіз та пошук методичної літератури
		від 10 грудня 2022 до 1 січня 2023	Огляд і аналіз існуючих рішень для системи управління роботом – маніпулятором

		від 1 січня 2023 до 15 січня 2023	Підбір і налаштування програмно-апаратного забезпечення
		від 15 січня 2023 до 1 лютого 2023	Аналіз документації до вибраного програмного забезпечення
		від 1 лютого 2023 до 10 березня 2023	Розробка алгоритму управління
		від 10 березня 2023 до 15 квітня 2023	Імплементація системи управління роботом – маніпулятором
		від 15 квітня 2023 до 31 квітня 2023	Оформлення пояснювальної записки
8.	Порядок контролю і приймання програмного продукту (моделі)	1. Перевірку ходу розробки програми виконувати раз в 3 тижні. 2. Захист розробленої моделі провести на засіданні Атестаційної комісії. 3. Пояснювальну записку подати на паперових носіях в 1 примірнику і в електронному вигляді в 1 примірнику на CD-R компакт-диску.	

Виконавець
студент групи КІ-41
Уваров В. Ю.

Замовник
ст. викладач
Рало О. М.

Додаток В

Програма і методика випробувань програмного виробу

«Базові апаратно – програмні елементи системи керування роботом – маніпулятором»

1 Об'єкт випробувань

1. Назва програмного виробу : «Базові апаратно – програмні елементи системи керування роботом – маніпулятором»
2. Галузь застосування : Комп'ютерні технології та робототехніка
3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі мого персонального комп'ютера, що відповідає рекомендованим системним вимогам (Додаток Б).

3. Обсяг випробувань

Приймальні випробування програмного вироба проводяться в обсязі відповідному цієї програми і методики випробувань.

4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Програмний виріб повинен задовольняти наступним вимогам:

1. працювати на операційній системі Windows
2. передбачити захист від некоректних дій користувача;
3. сумісність з іншими програмними продуктами;
4. вимоги до складу і параметрів технічних засобів;
5. вимоги до маркування та упаковки (не висуваються);
6. вимоги до транспортування і зберігання (не висуваються).

Спеціальні вимоги (не пред'являються).

5. Вимоги до програмної документації

Програмною документацією до виробу «Базові апаратно – програмні елементи системи керування роботом – маніпулятором» вважати:

1. Справжнє технічне завдання на розробку програми (представити як Додаток Б до пояснювальної записки до кваліфікаційної роботи);
2. Програму і методику випробувань розробленої програми (представити як Додаток В до пояснювальної записки до кваліфікаційної роботи);
3. Рекомендацій щодо застосування створеної програмної стандартизації у проектах (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи).
4. Текст програми(представити як Додаток Г до пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань треба інтегроване середовище розробки для Python з використанням бібліотек catkin і smach, а також програмне забезпечення для симуляції моделювання робототехнічних систем "CoppeliaSim" з попередніми налаштуваннями.

6.2 Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

-ознайомчий (1-й етап);

-випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

1. Перевірку комплектності програмної документації.
2. Перевірка комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.
3. Перевірку комплектності складу технічних і програмних засобів.
4. Методику проведення перевірок на 1 етапі випробувань.
5. Якість програмної документації перевіряється на відповідність вимогам стандартів ЕСПД.

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

1. перевірку відповідності технічних характеристик програми вимогам технічного завдання;
2. перевірку ступеня виконання функціональних вимог до програми;
3. методику проведення перевірок, що входять до переліку по 2 етапу випробувань.

1. Програма працює відповідно до умов експлуатації операційної системи Windows.
2. Для роботи необхідне інтегроване середовище розробки для Python та програмне забезпечення для симуляції моделювання робототехнічних систем “CoppeliaSim”
3. Порядок проведення випробувань:

3.1. Запуск програми здійснюється за допомогою запуску симуляції в “CoppeliaSim”;

3.2. Після запуску треба запустити відповідний код на мові Python;

3.3. Ввести дані в графічний інтерфейс “CoppeliaSim”

3.4. Далі на сцені “CoppeliaSim” з’явиться відповідний результат.

Для проведення випробувань пропонується тест 1, тест 2, тест 3, тест 4 та тест 5

Тест 1. Перевірка невдалого підключення до sim-сервера

1. Запуск програми на Python без попереднього запуску симуляції
2. Отримання повідомлення про невдале підключення

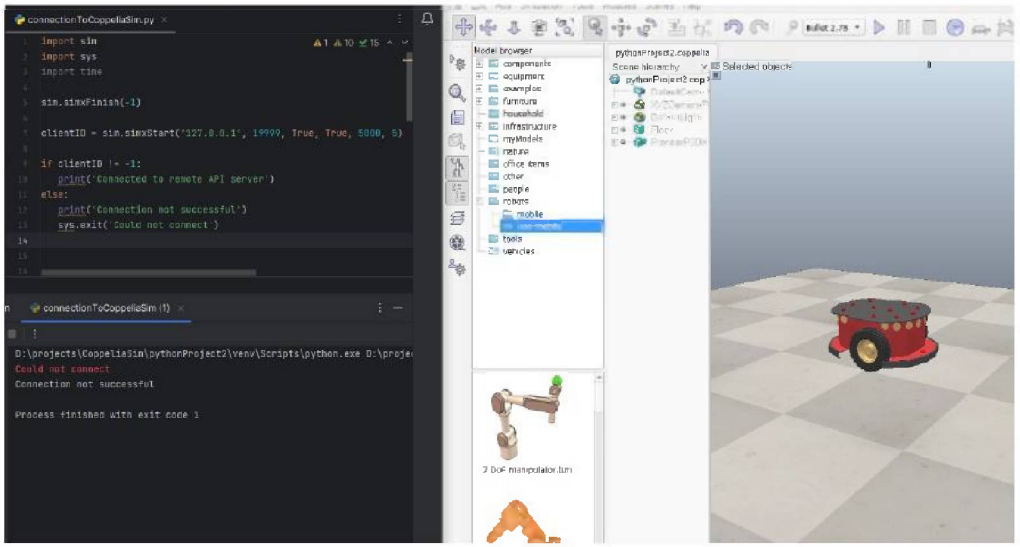

Рисунок В.1. Тест 1

Тест 2. Перевірка вдалого підключення до sim-сервера

1. Запуск симуляції в ПЗ “CoppeliaSim”
2. Запуск програми на Python
3. Отримання повідомлення про вдале підключення

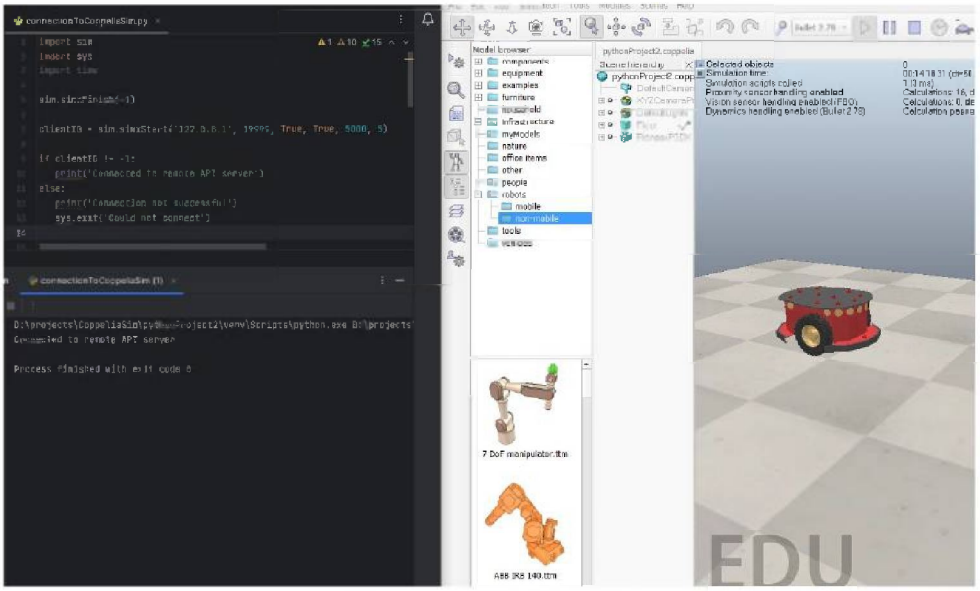

Рисунок В.2. Тест 2

Тест 3. Перевірка виконання керуючого коду з “PyCharm”

1. Запуск симуляції в ПЗ “CoppeliaSim”
2. Запуск керуючої програми на Python
3. Відображення руху робота на сцені відповідно до керуючого коду

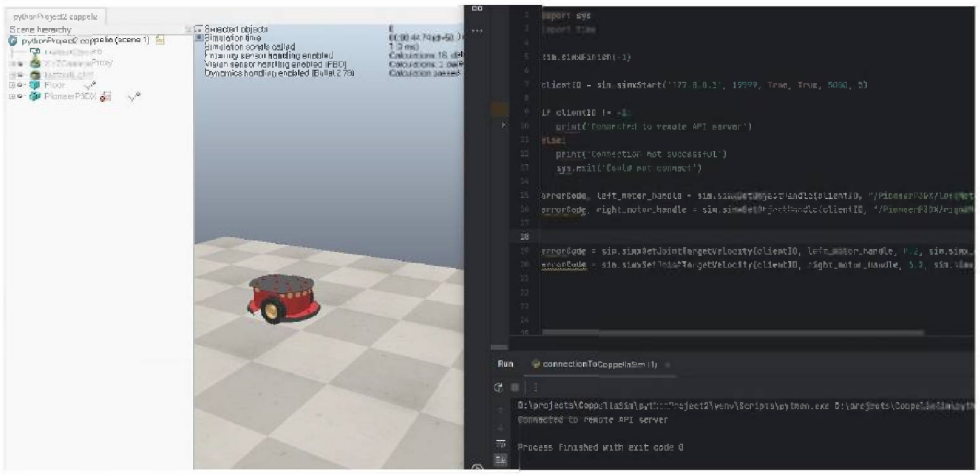

Рисунок В.3. Тест 3

Тест 4. Перевірка роботи інтерфейсу користувача

1. Запуск симуляції в ПЗ “CoppeliaSim”
2. Відображення графічного інтерфейсу
3. Закриття графічного інтерфейсу без помилок (автоматично або вручну)

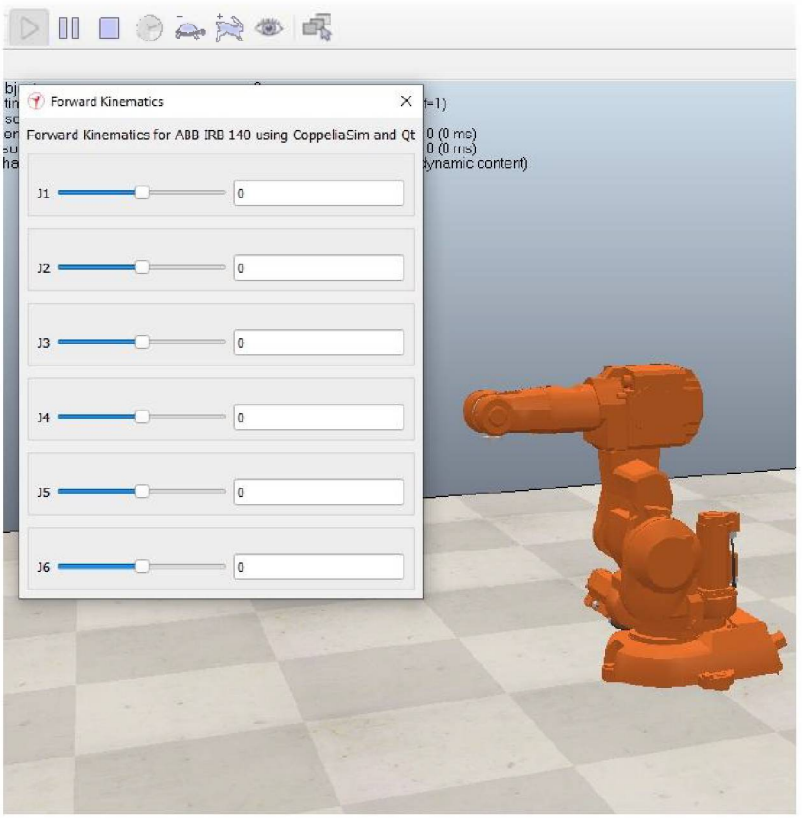

Рисунок В.4. Тест 4

Тест 5. Перевірка роботи керуючого коду на мові Lua

1. Запуск симуляції в ПЗ “CoppeliaSim”
2. Ввод даних положення шарнірів робота-маніпулятора за допомогою інтерфейсу користувача
3. Відображення положення шарнірів робота-маніпулятора на сцені

6

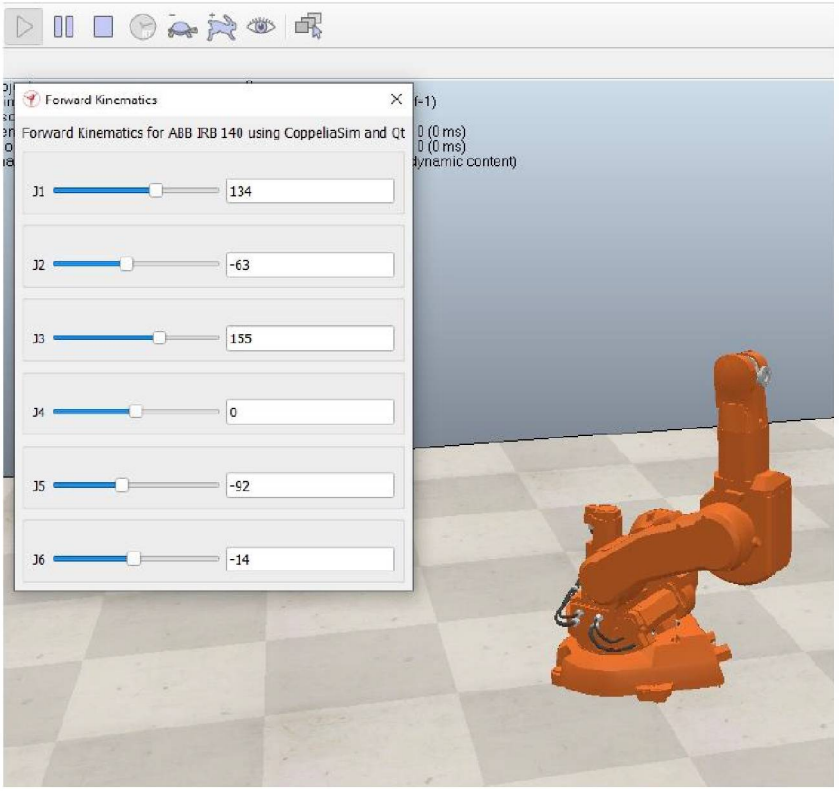

Рисунок В.5. Тест 5

Тест вважається пройденим, якщо відбуваються вказані операції і їх відображення у програмному продукті.

Висновки: тест 1 успішно пройшов випробування, тест 2 успішно пройшов випробування, тест 3 успішно пройшов випробування, тест 4 успішно пройшов випробування і тест 5 успішно пройшов випробування. Випробування пройшло успішно.

Виконавець: студент групи КІ-41, Уваров В. Ю.