Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

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

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

перший (бакалаврський)
(рівень вищої освіти)

Розроблення віртуального макету для моделювання автоматизованої лінії розподілу товарів в внутрішньоскладських логістичних системах
(тема)

Виконав:
здобувач 4 року навчання,
групи АКТАКІТ 21-3
Михайло КРАСНОПЬОРОВ
(власне ім'я, прізвище)

Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Автоматизація та комп'ютерно-інтегровані технології
(повна назва освітньої програми)

Керівник доцент Олена ЧАЛА
(посада, власне ім'я, прізвище)

Допускається до захисту
Зав. кафедри КІТАР _____ Ігор НЕВЛЮДОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Я, Краснопьоров Михайло Романович, як здобувач вищої освіти ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Я не використовував штучний інтелект для підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на віповідне джерело.

"9" червня 2025 р.  Михайло КРАСНОПЬОРОВ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет Автоматики і комп'ютеризованих технологій
Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
Рівень вищої осві перший (бакалаврський)
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва)
Тип програми Освітньо-професійна
Освітня програма Автоматизація та комп'ютерно-інтегровані технології
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри КІТАР _____
(підпис)
« 30» квітня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Краснопьорову Михайлу Романовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення віртуального макету для моделювання автоматизованої лінії розподілу товарів в внутрішньоскладських логістичних системах
Затверджена наказом по університету від від 19 травня 2025 р. № 390 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 11.06.2025р.
3. Вихідні дані до роботи: Середовище розробки Visual Studio 2022; 3D моделювання в Blender; Quixel Megascans, Sketchfab; середовище розробки Pycharm; використання бібліотек комп'ютерного зору; максимальна похибка сортування до 3%.

4. Перелік питань, що потрібно опрацювати в роботі
Аналіз модулів розподілу товарів в автоматизованих лініях, вибір середовища розробки, побудова структурної схеми, побудова алгоритму роботи, вибір бібліотек для розробки, програмування логіки обробки відеопотоку, розробка віртуального макету, питання теорії автоматичного керування та охорони праці.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)

Презентація PowerPoint на 10 слайдів.

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Приміт-ка
1	Аналіз технічного завдання	30.04.2025	Виконано
2	Аналіз літератури за темою	06.05.2025	Виконано
3	Вибір та обґрунтування програмних засобів	14.05.2025	Виконано
4	Розробка віртуального макету	16.05.2025	Виконано
5	Оформлення пояснювальної записки	01.06.2025	Виконано
6	Подання роботи на перевірку Інтернет-сервісом StrikePlagiarism	11.06.2025	Виконано
7	Подання роботи на рецензію	12.06.2025	Виконано
8	Подання роботи на підпис зав. кафедри	20.06.2025	Виконано
9	Подання кваліфікаційної роботи в ЕК	23.06.2025	Виконано

Дата видачі завдання 2025 р.

Здобувач _____ Михайло КРАСНОПЬОРОВ
(підпис) (посада, власне ім'я, прізвище)

Керівник роботи _____ Олена ЧАЛА
(підпис) (власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 73 с., 1 табл., 24 рис., 3 дод., 42 джерела.

АВТОМАТИЗАЦІЯ, СОРТУВАННЯ, МОДУЛЬ, КОНВЕЄР, ТОВАРИ, ЛОГІСТИЧНІ СИСТЕМИ, ВНУТРІШНЬОСКЛАДСЬКІ ПРИМІЩЕННЯ.

Об’єкт розробки – автоматизований розподіл товарів у внутрішньоскладських логістичних системах.

Предмет розробки – віртуальний макет для моделювання автоматизованої ліній розподілу товарів.

Мета роботи – підвищення якості автоматизованого сортування в внутрішньоскладських логістичних системах шляхом проведення симуляції виробничих параметрів розподілу товарів при сортуванні.

Методи дослідження – моделювання процесу автоматизованого сортування у внутрішньоскладських логістичних системах; метод цифрових двійників, метод аналізу зображення машинним зором.

Було проаналізовано предметну область, досліджено основні модулі автоматизованої лінії сортування товарів у внутрішньоскладських логістичних системах, розглянуто методи сортування.

Упровадження розробленого макету буде корисним для тестування та налагодження автоматизованих та автоматичних сортувальних ліній в логістичних системах, та при проведенні лабораторних та практичних занять для здобувачів першого (бакалаврського) рівня вищої освіти за освітньо-професійною програмою: «Автоматизація та комп’ютерно-інтегровані технології» (АКТАКІТ).

ABSTRACT

Explanatory note: 73 p., 1 table, 24 fig., 3 appendix, 42 sources.

AUTOMATION, SORTING, MODULE, CONVEYOR, GOODS, LOGISTICS SYSTEMS, INTERNAL WAREHOUSE PREMISES

The object of development is automated distribution of goods in intra-warehouse logistics systems.

The subject of development is a virtual model for modeling automated distribution lines of goods.

The purpose of the work is to improve the quality of automated sorting in intra-warehouse logistics systems by simulating the production parameters of goods distribution during sorting.

Research methods are modeling of the automated sorting process in intra-warehouse logistics systems; digital twin method, method of image analysis by machine vision.

The subject area was analyzed, the main modules of the automated goods sorting line in intra-warehouse logistics systems were investigated, and sorting methods were considered. The implementation of the developed model will be useful for testing and debugging automated and automatic sorting lines in logistics systems, and when conducting laboratory and practical classes for applicants of the first (bachelor's) level of higher education in the educational and professional program: "Automation and Computer-Integrated Technologies" (AKTAKIT).

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

КІТАР – Комп’ютерно-інтегровані технології, автоматизація та робототехніка;

ОЕЗ – обчислювально-електронний засіб;

ОПБ – орієнтовний прямокутний бокс;

ОПП – освітньо-професійна програма;

РЕЗ – радіоелектронний засіб;

САПР – система автоматизованого проєктування;

ЦСР – цілі сталого розвитку;

LIDAR – Light Detection and Ranging;

ROI – Region of interest;

ToF – Time of Flight;

UE – Unreal Engine.

ВСТУП

Впродовж останніх десятиліть стрімко попит на електронні «електронну торгівлю» (e-commerce), що призвело до підвищення навантаження на компанії з доставки [1].

Разом із тим навантаженість на внутрішньоскладські логістичні процесси є порівняно вищою, ніж раніше [2-4].

Постає питання автоматизації цих процесів, збільшення швидкості обробки посилок, зменшення людського фактору для підвищення якості сортування товарів за критерієм точності сортування на складах. Тому, можна вважати , що питання ро зорзвязано в роботі є актуальними та своєчасними.

Одним з таких процесів є вимірювання та сортування за габаритами. Зазвичай у реальних промислових умовах це вирішується системами на базі LIDAR датчиків, Time of Flight камерами, або лазерними завісами. Все це вимагає значних витрат на інсталяцію, налаштування. Для зменшення витрат при випробовуванні різних методів вимірювання та сортування за габаритами доцільним є побудова цифрового двійника та тестування його у віртуальних системах. У рамках дипломної роботи створюється цифровий двійник сортувальної лінії в середовищі Unreal Engine 5.5 із конвеєрним модулем та віртуальной ToF камерой.

Однак, не зважаючи на велику кількість досліджень в галузі автоматизації сортування товарів в логістичних хабах виникає протиріччя між підвищенням якості за сортування товарів і обмеженістю існуючих підходів, модернізації обладнання та швидкості сортування.

Вище сказаного, можна зробити висновок, що виникає нагальна потреба в розробці модернізованих системи з можливістю перевірки за рахунок налаштувань різних видів налаштування камер, їх позиціонування,

тестування різних варіантів алгоритмів, що дадуть можливість виправлення спотворень при використанні симуляторів для калібрування та попереднього тестування перед використанням в умовах реального виробництва.

Об’єкт розробки – автоматизований розподіл товарів в внутрішньоскладських логістичних системах.

Предмет розробки – віртуальний макет для моделювання автоматизованої ліній розподілу товарів.

Мета роботи – підвищення якості автоматизованого сортування в внутрішньоскладських логістичних системах шляхом проведення симуляції виробничих параметрів розподілу товарів при сортуванні.

Кваліфікаційна робота оформлена згідно з ДСТУ 3008:2015 [5] та Положення про протидію академічному плагіату в ХНУРЕ [6], методичних вказівок [7], навчального посібника з кваліфікаційної роботи [8].

Матеріали за тематикою роботи було опубліковано у вигляді статті [4].

Проведені дослідження відповідають цілям сталого розвитку (ЦСР), а саме: ЦСР 4, ЦСР 9 [9].

1. АНАЛІЗ МОДУЛІВ РОЗПОДІЛУ ТОВАРІВ В АВТОМАТИЗОВАНИХ ЛІНІЯХ

1.1 Визначення основних модулів розподілу товарів в автоматизованих лініях

Автоматизація розподілу(сортування) товарів – це комплекс заходів, технічних і програмних рішень спрямованих на оптимізацію процесу сортування.

Автоматизовані сортувальні лінії ідентифікують окремі предмети на конвеєрі та перенаправляють їх у призначені місця за допомогою різних пристроїв, керованих спеціальним програмним забезпеченням. Сортувальні пристрої застосовуються для різних застосувань залежно від типу продукту та швидкості конвеєра.

Ці системи підвищать швидкість сортування предметів, одночасно знизивши рівень помилок. Автоматизовані сортувальні станції можна використовувати в розподільчому центрі різними способами. Деякі приклади: отримання, комплектування, пакування та відправлення [10].

До складу автоматизованої лінії входять наступні модулі:

– система подачі для подачі товарів у належній орієнтації і відстані між один одним;

– конвеєри для безперервного транспорту товарів по складу;

– сортувальний модуль [10].

1.2 Системи подачі (сингулятори)

Для досягнення необхіної пропускної здатності лінії та для подальшого розподілення перед входженням в сортувальну лінію масовий

потік посилок має бути розділений, тобто пакунки повинні бути рівномірно розділені та вирівняні. Для цього використовуються так звані сингулятори (рис. 1.1).

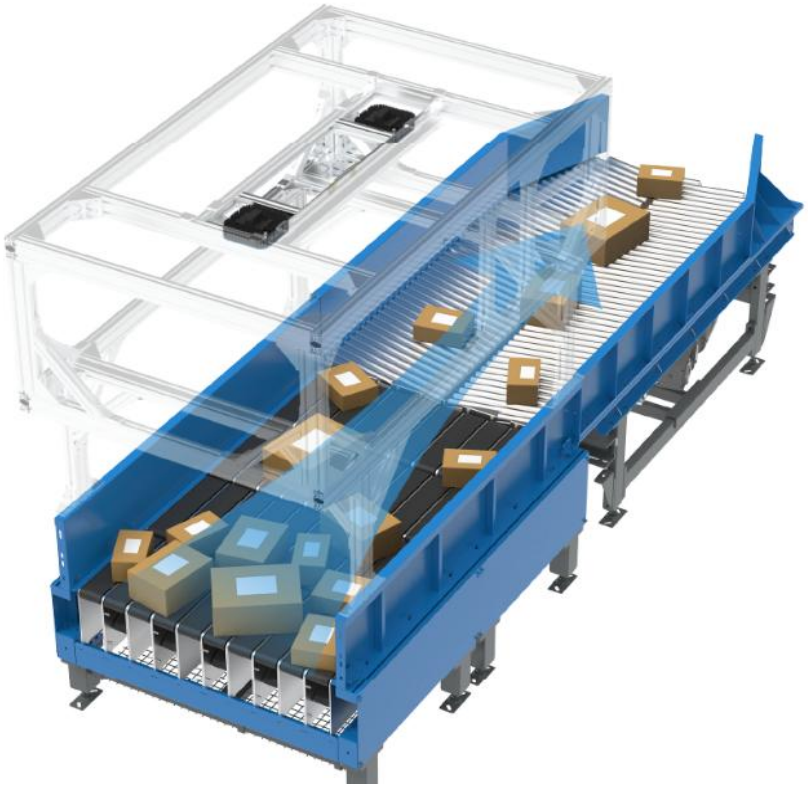

Рисунок 1.1 – Сингулятор масового потоку товарів [10]

Завдяки високоякісним рішенням для автоматичного розділення посилки транспортуються із зони масового розвантаження до автоматизованої машини для розділення за допомогою ліній подачі, які можуть бути налаштовані за допомогою повністю автоматичних самоскидів контейнерів, висувних конвеєрів і завантажувальних конвеєрів.

У деяких додатках конвеєри пропонують зсувні ролики, через які проходять невеликі партії посилок, коли вони потрапляють у роздільник. Вони вирівнюють пакети таким чином, щоб вони випускалися одним потоком з рівномірним проміжком між кожними пакетами [11].

Безпроблемна інтеграція автоматизованого розділювача в систему циклічного сортування дозволяє використовувати систему витягування, яка регулює потік [12].

1.3 Транспортні модулі

Конвеєр (траснпортер) – це скалдське устаткування призначене для автоматизації переміщення матеріалів та виробів на виробничих підприємствах та в складах [13].

Існує велика кількість видів цих конвеєрів але два самих розповсюдженних це стрічкові (рис. 1.2) [12] та роликові (рольганги) (рис 1.3) [13].

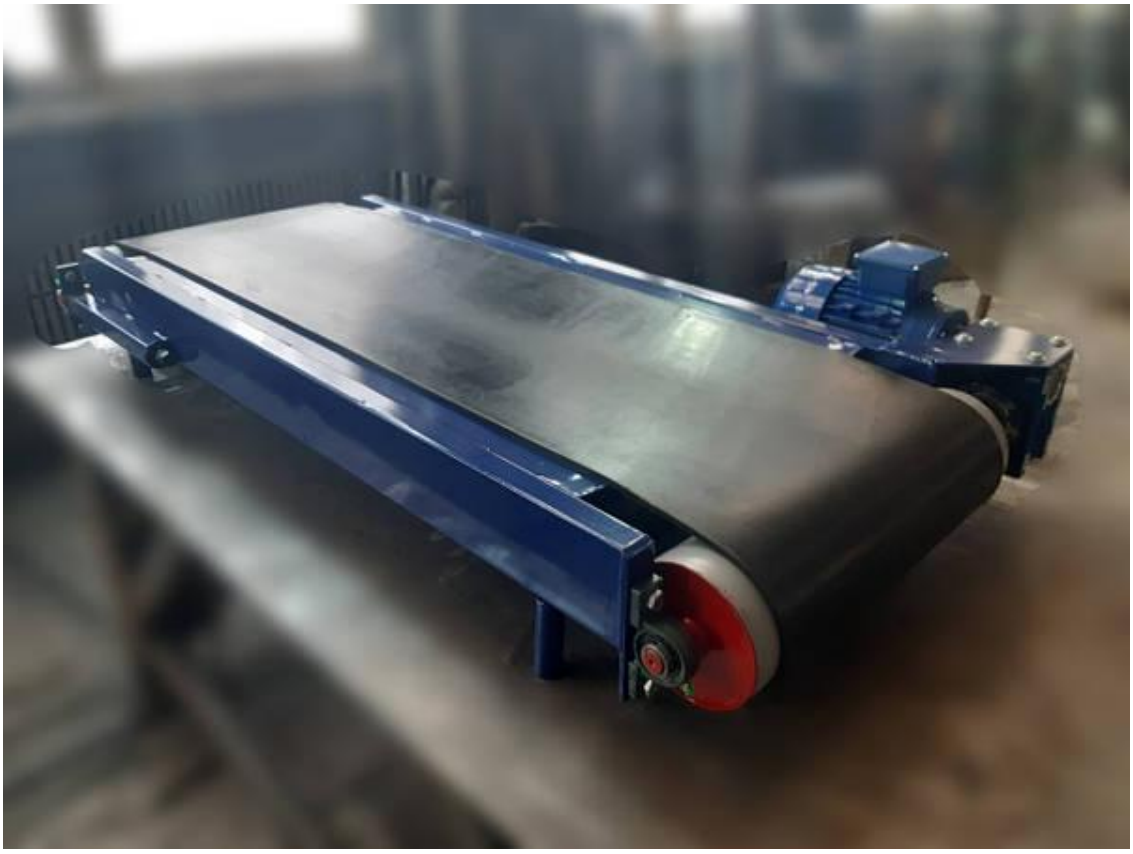

Рисунок 1.2 – Стрічковий конвеєр [12]

Стрічкові конвеєри забезпечують транспортування на довгі дистанції штучного і сипучого вантажу. Стрічкове пристрій не є універсальним, але він доступний за ціною та користуються популярністю. Особливістю стрічкових конвеєрів є робоча зона – стрічка, по якій транспортується товар. Стрічкові конвеєри широко застосовуються на виробництвах в сортувальних цехах, що збільшує швидкість операцій зі збирання чи відвантаження продукції/товару [14].

Рисунок 1.3 – Роликовий конвеєр [13]

Роликові конвеєри поділяються на гравітаційні і приводні.

Ролики, встановлені на бічній рамі, забезпечують поверхню кочення. При установці під кутом нахилу деталі переміщаються власними силами. Однак при використанні гравітаційних конвеєрів для деталей різного розміру та ваги це може бути складнішим.

Потрібно буде контролювати швидкість та кут. Приводні роликові конвеєри використовують один або кілька роликів з приводом

електродвигуна для кожної зони, щоб створити серію зон вздовж конвеєра [15].

1.4 Сортувальні модулі

Сортувальний модуль приймає постійний потік продукту та направляє його до належного місця призначення. Сортувальники є унікальним типом конвеєра, хоча багато з них базуються на інших типах конвеєрів, багато з них унікальні. Незалежно від типу, усі вони застосовуються однаково. Різниця полягає в їх номінальній потужності та продуктах, які можна обробляти.

Усі сортери мають дві спільні риси. По-перше, це спосіб подачі продукту в сортувальник у правильному положенні та на відстані. По-друге, це ємності для продукту, коли він виходить із сортувальника. Це можуть бути гравітаційні або силові конвеєри або жолоби [16].

Роздивимося декілька видів сортувальних модулів, що використовуються в логістичних системах.

1.4.1 Поперечний стрічковий сортувальник (Cross-belt sorters)

Поперечний стрічковий сортувальник (рис. 1.4) [17] складається з десятків або сотень транспортних засобів стрічкового конвеєра, які рухаються перпендикулярно до основного напрямку, тоді як система сортування складається з частини подачі (автоматичної чи ручної), дистанції, позиціонування посилок, ідентифікації, хоста сортувальника, частини вивантаження, системи керування тощо.

Вона може ефективно сортувати всі види коробок, коробок, посилок з одягом, друкованих виробів та Інші посилки, і в даний час є основним високошвидкісним і ефективним обладнанням для сортування в сферах електронної комерції, експрес-доставки тощо. Похибки таких сортувальників доходять до 1% - 2% [16].

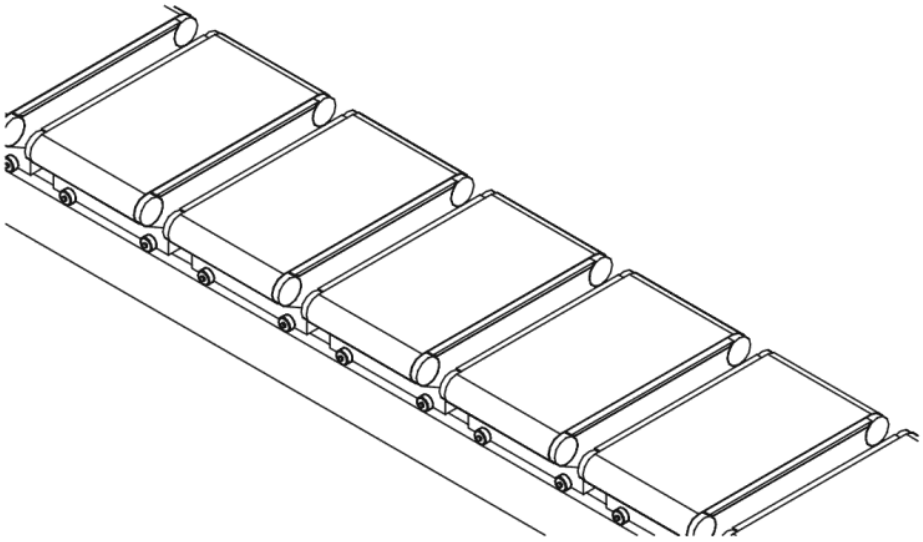

Рисунок 1.4 – Схематичне зображення поперечного стрічкового сортувальника [17]

1.4.2 Сортувальники з перекидними лотками

Сортувальники з перекидними лотками (рис. 1.5) [18] зазвичай приводяться в рух індивідуально лінійними двигунами, встановленими на кожному носієві, а наступні носії з’єднуються разом, щоб утворити безперервну петлю. Існує безліч інших доступних концепцій приводів, але жодна з них не використовується настільки широко.

Варто зазначити, що кілька невеликих виробників створили сортувальник лотків із нахилом, у якому лотки та механізм нахилу кріпляться до стрічки. У цьому випадку сортувальник не є безперервним циклом, а просто прямим сортувальником.

Будь-який продукт, який не перенаправляється, має бути повторно індукований, а не допускатися до рециркуляції. У кінці сортувальника лотки слідують за стрічкою внизу через повернення до вхідного кінця сортувальника. Похибки таких сортувальників доходять до 3% - 5% [17].

Рисунок 1.5 – Сортувальники з перекидними лотками [18]

1.4.3 Сортувальники штовхачів або тягачів

Сортувальники штовхачів або тягачів (рис. 1.6) використовують пристрої, встановлені збоку від конвеєра, для відхилення коробок під кутом 90 градусів за допомогою повітряного циліндра.

Для цього типу сортувальників використовуються як стрічкові, так і роликові транспортери.

Стрічкові конвеєри забезпечують краще відстеження місця розташування продукту, але бічна сила відхилення продукту може призвести до того, що стрічка зійде з траєкторії.

Поточні роликові конвеєри забезпечують кращий бічний рух роликів, але деяка точність відстеження продукту втрачається через прослизання роликів під продуктом.

Цей тип сортувальника хороший для великих, важких, стійких агрегатів. Штовхач або знімач не є гарним вибором для крихких елементів через дещо грубу природу відводу [17].

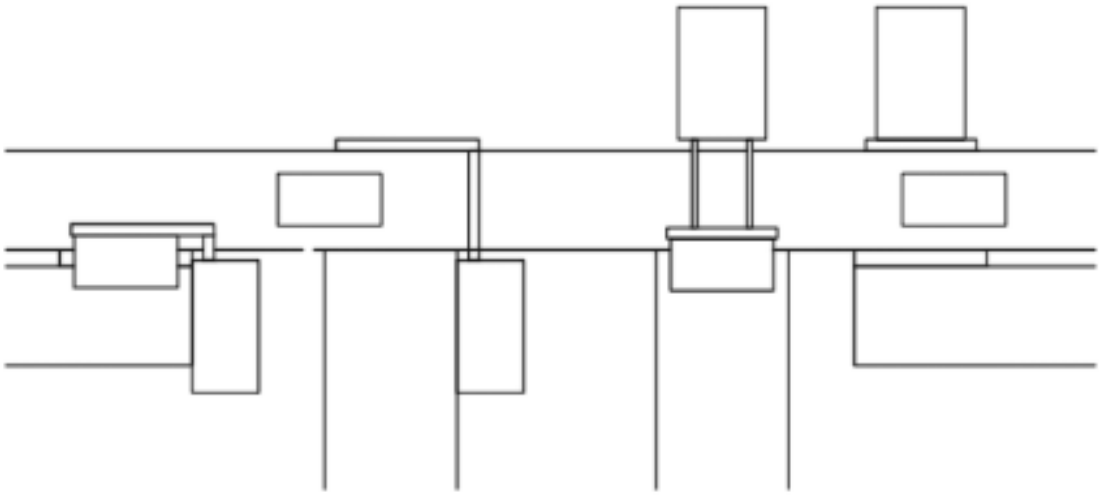

Рисунок 1.6 – Схематичне зображення сортувальника штовхачів/тягачів [17]

1.4.3 Лопаткові сортувальники

Лопаткові сортувальники, які часто називають сортувальниками бейсбольних биток, використовують поворотний відводний важіль, який встановлено поруч із конвеєром і відкидає продукт убік (рис 1.7).

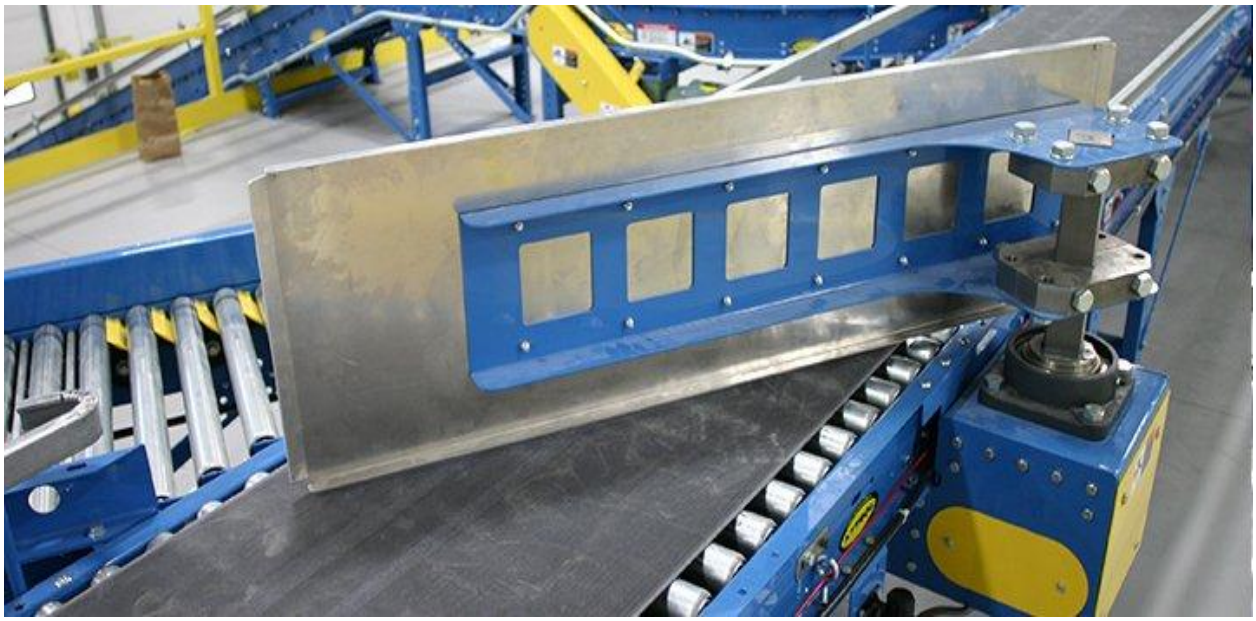

Рисунок 1.7 – Зображення лопаткового сортувальника [19]

Вони підходять для низькошвидкісних програм і зазвичай використовуються для одиниць неправильної форми, таких як пачки або поштові мішки. Самі весла зазвичай приводяться в дію за допомогою повітряного циліндра або електродвигуна.

Лопатка активується, коли виріб знаходиться перед лопаткою. Час циклу відносно повільний; тому цей тип сортувальника зазвичай використовується лише в програмах із низькою швидкістю [17].

1.4.4 Сортувальники з шарнірними колесами

Сортувальники з шарнірними колесами (рис. 1.8) є гнучкими та простими у застосуванні.

Цей варіант сортування не потребує багато місця, що робить його чудовим вибором для сортування на невеликій ділянці конвеєра або якщо на відносно невеликій ділянці конвеєра потрібно декілька відводів [20].

Сортувальники з поворотними колесами використовують два комплекти коліс, які обертаються незалежно та послідовно.

Сортувальник із шарнірним колесом, розміщений біля кожного відхилення, розподіляє продукти та дозволяє залишати невеликі проміжки між кожним продуктом.

Потрібна відповідна відстань між продуктами, щоб гарантувати, що станція перенаправлення мала час для циклу, і запобігає накопиченню продуктів для створення невпорядкованої резервної копії.

Сортувальники з обертовими колесами опрацьовують приблизно до 90 коробок на хвилину.

Похибки таких сортувальників доходять до 5% [20].

Рисунок 1.8 – Зображення сортувальники з шарнірними колесами [20]

З проведеного аналізу модулів сортувальних ліній можна зробити висновки, що основним критерієм якості таких пристроїв є точність та швидкість сортування.

Похибки, що виникають при сортуванні, відстежуються вже в умовах виробництва при використанні устаткування протягом певного часу.

Для зменшення похибок, якими є неправильна ідентифікація товару на лінії, було прийнято рішення розробки віртуального макета-симулятора для налагодження та тестування ліній сортування в логістичних процесах.

2 ПРОЄКТУВАННЯ ВІРТУАЛЬНОГО МАКЕТУ ДЛЯ МОДЕЛЮВАННЯ АВТОМАТИЗОВАНОЇ ЛІНІЇ РОЗПОДІЛУ ТОВАРІВ

2.1 Розробка структурної схеми

Розроблення структурної схеми допоможе визначити елементи, що входять в систему, що розробляється. На цій схемі виріб поділений на частини, що виконують певну дію.

Зазвичай автоматизована лінія розподілу товарів складається з вивантажувачів, сигнуляторів, конвеєрних ліній, та сортувальних модулів. Оскільки макет є віртуальним деякі з цих елементів можуть бути вилучені або представлені в іншому вигляді для імітування їх роботи.

Оскільки задачею проекту є створення віртуального макету для моделювання автоматизованої лінії розподілу товарів необхідно визначити за якими саме параметрами буде проводитись розподіл товарів.

Зазвичай розподіл відбувається за місцем доставки або за габаритними розмірами коробки в якій знаходиться товар. В нашому випадку буде використовуватись габаритні розміри, а саме об’ємна вага.

Об’ємна вага визначається за формулою [21]:

$$\frac{L*W*H}{5000} = Vw, \tag{2.1}$$

де L – довжина коробки (см),

W – ширина (см),

H – висота (см),

V_w – об’ємна вага (см^3/ кг).

Також необхідно визначити метод сортування. У випадку де в конвеєрну лінію вбудований модуль визначення габаритів надходжень доцільно використовувати методи сортування з розгалуженням конвеєрної лінії. Це можуть бути лопаткові сортувальники, сортувальники штовхачів або лопаткові сортувальники. Для простоти реалізації було обрано лопатковий сортувальник.

Для визначення габаритів коробок необхідно мати модуль визначення габаритів. Серед багатьох методів визначення розмірів на конвеєрних лініях, серед яких LIDAR датчики, фотоелектричні датчики, методи комп'ютерного зору, було обрано метод з використанням ToF (Time of Flight) камери (наприклад Intel RealSense, ORBBEC Femto Bolt).

ToF камера – це нове покоління продуктів для визначення відстані та технологій 3D-зображень. Він безперервно надсилає світлові імпульси до цілі, а потім використовує датчик, щоб отримати світло, що повертається від об'єкта, і отримує відстань до цільового об'єкта, визначаючи час польоту (туди й назад) світлового імпульсу. Коли використовується ця камера, розмір можна виміряти за допомогою зображення, що дуже зручно. І цей спосіб використання полягає в тому, що відбиття світла дозволяє визначити відстань, обчисливши час повернення, і більш адекватне сприйняття можна отримати за допомогою датчика. Перевага використання такого типу камери дуже очевидна. Не тільки пікселі вищі, але й додавання цього датчика може зробити отримання даних на карті розмірів більш реалістичним, і немає потреби в рухомих частинах, а кращі результати можна отримати лише шляхом вимірювання [22].

Далі необхідно обрати програмний застосунок для розробки віртуального макету. Це повинна бути віртуальна середа в якій можна створити необхідні елементи для нашої системи. Оскільки було обрано сортування за габаритами, а саме за об'ємною вагою, яка визначається за формулою (2.1) то це повинна бути тривимірна середа з можливістю роботи з

тривимірними об’єктами і можливістю симуляції деяких процесів. З такими задачами може працювати Unreal Engine.

Unreal Engine – інструмент для створення ігор і сцен із використанням 3D-моделей, розроблений компанією Epic Games. Ця програма призначена для створення ігор і симуляцій. Unreal Engine надає засоби для створення високоякісного освітлення, рендеринг і підтримку віртуальної реальності. Також він дає змогу моделювати об'єкти, відтворювати їхній рух і взаємодію з навколишнім середовищем реалістично. Unreal Engine використовує мову програмування C++, а також має систему розширень та графічний інтерфейс для того, щоб зробити гру, не знаючи кодингу. В наш час така практика широко використовується для симуляції робототехніки, тестуванні алгоритмів комп’ютерного зору, тестування приладів розроблених в програмах Simulink/Matlab тощо. В цьому рушії є можливість симулювати роботу конвеєра, камери для визначення габаритів, робити захват відео для подальшої обробки, і сортувати лопатковим механізмом. Для комортної роботи в даному рушії необхідно мати наступні характеристики комп’ютера:

– Операційна система: Windows 10 64-бітна версія;

– процесор: Чотирьох ядерний Intel або AMD, 2.5 Ггц або швидше;

– озп: 16 ГБ або більше;

– відеокарта: DirectX 11 або 12 сумісна графічна карта;

– відеопам’ять: 8Гб або більше [23].

Для обробки даних, які надходять з симульованої ToF камери, необхідно використовувати бібліотеки комп’ютерного зору, а оскільки маємо ще глибинне зображення то виникає потреба його опрацьовувати і створювати по ньому хмару точок для визначення розмірів. В Unreal Engine 5, який працює з C++ кодом досить важко інтегрувати сторонні бібліотеки, які відсутні в офіційній підтримці вбудованими плагінами. Отже треба відокремлювати алгоритми обробки відеопотоку від Unreal Engine. Для цього будемо використовувати мову програмування Python, оскільки на ній

реалізовані зручні біндінги для роботи з багатьма бібліотеками, наприклад OpenCV і Open3D, які можна підключити в проект однією командою в терміналі і вже мати їхній повний функціонал для подальшого використання. Офіційний Python-плагін для Unreal Engine 5 працює лише в режимі редактора і не може працювати в кінцевій збірці під час симуляції або гри, тому доцільно використати окремий сервер з Python який зможе приймати відеопотік з Unreal Engine, що буде виступати в ролі клієнта, і там же його обробляти і повертати данні назад. Для зв'язку між Unreal Engine та Python сервером використаємо мережевий протокол TCP/IP, оскільки мережеві бібліотеки для Python і Unreal Engine мають підтримку цього протоколу, також це є своєрідною імітацією промислового протоколу передачі даних.

На основі проведеного аналізу данні занесемо до таблиці 2.1 куди заносимо основні елементи і будуємо структурну схему (рис. 2.1).

Таблиця 2.1 – Елементи структурної схеми

Елемент	Опис
Unreal Engine	Рушій в якому буде симулюватись робота конвеєра і захват камери
Конвеєр	Конвеєрна стрічка по якій переміщаються коробки
Коробка	Об'єкт який ми сортуємо
ToF камера	Камера яка робить захват відео і глибини сцени
Лопатковий сортувальник	Сортувальний модуль з лопатковим сортувальником і розгалуженням конвеєра
Обробка	Обробка отриманого відеопотоку алгоритмами комп'ютерного зору
Визначення розмірів	Визначення розмір по побудованій хмарі точок
Розрахунок об'ємної ваги	Розрахунок об'ємної ваги коробки по формулі (2.1)
Обробка інформації про вагу	Отримання розрахованої об'ємної ваги і прийняття рішення щодо сортування

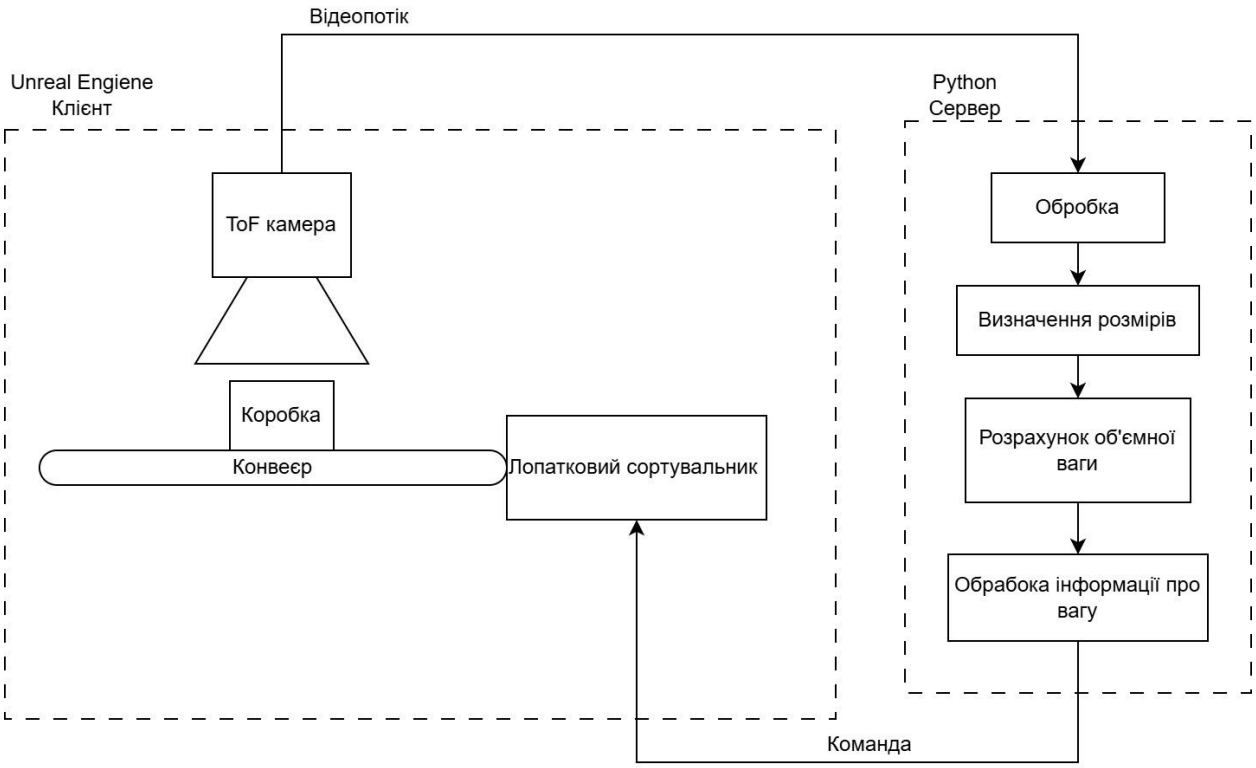

Рисунок 2.1 – Структурна схема

2.2 Розробка алгоритму роботи комп’ютерного зору

Оскільки було визначено, що для обробки відеопотоку який надходить з симульованої ToF камери, що робить захват сцени, необхідно створити сервер з Python де буде прийматися цей відео поток і оброблятися. Для спрощення процесу розробки та подальшої коректної роботи програми розробимо алгоритм обробки відеопотоку та визначення об’ємної ваги товарів на сортувальній лінії.

Для визначення габаритної ваги, по якій буде проходити сортування товарів на автоматизованій сортувальні лінії, за формулою (2.1) необхідно знати розміри довжини, ширини і висоти коробки товару. Для цього в проекті використовується метод визначення габаритів на основі Time-of-Flight камери, яка має можливість робити захоплення так званого «дальнісного зображення» або «мапу глибини». Карта глибини зазвичай є зображенням у градаціях сірого, де кожне значення представляє відстань між

світловідбиваючою поверхнею та камерою [22]. Фактично ця мапа э двовимірним масивом з розміром кадру, де кожен елемент масива, тобто піксель, має інформацію про відстань від камери до цього пікселя(вісь Z). На основі неї можна побудувати тривимірний об’єкт який буде складатися з точок в тривимірному просторі і називатися «хмарою точок», де кожна точка це відповідний піксель зображення з наданою йому третьою координатою від карти глибини. Перетворення двовимірної карти глибини в тривімирну модель відбувається наступним чином.

Для кожної точки тривимірного простору існує її проектне двовимірне зображення. В контексті цифрового зображення для визначення положення двовимірної точки в площині пікселів відповідно до її проектного тривимірного положення використовуються однорідні координати. Однорідні координати це формально проекція точок з вищого виміру до меншого.

В системі однорідних координат для точки y$[u,v,1]^T$ еквівалентною є точка x$[X,Y,Z,1]^T$ помножена на матрицю камери (2.2) [24]:

$$y \sim K \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} x, \tag{2.2}$$

де K – це матриця внутрішніх параметрів (2.3),

R, T – зовнішні параметри (R – матриця повороту, T – це положення початку світової системи координат, виражене в координатах камероцентрованої системи координат) [23].

$$\begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.3}$$

де f_x та f_y – це фокусні відстані в пікселях, u_0 та v_0 координати центру зображення в пікселях,

γ – кут нахилу між осями x та y, зазвичай приймається як 0 [25].

Фокусні відстані f_x та f_y в пікселях можна знайти за формулою, знаючи кут огляду і розміри кадру (рисунок 2.2).

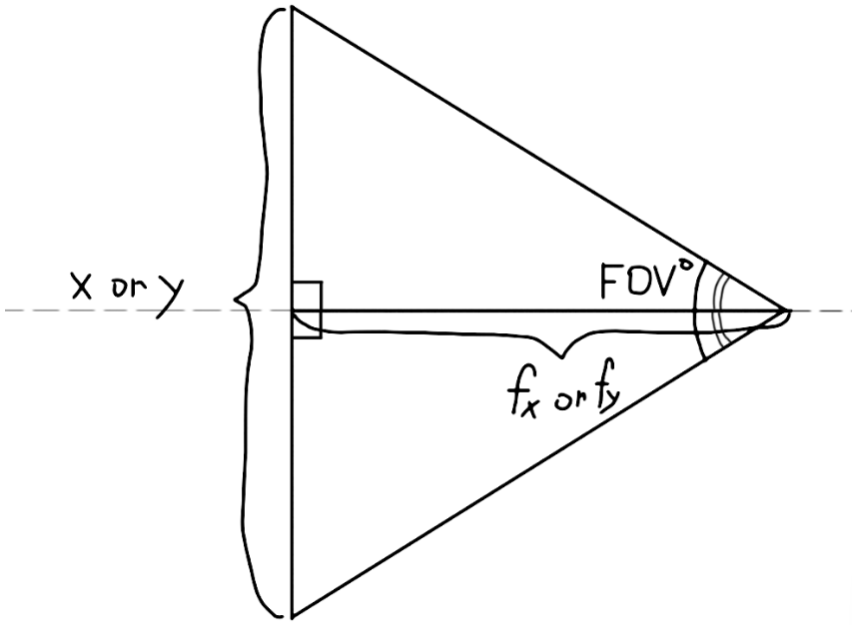

Рисунок 2.2 – Схема фокусної відстані

Зі схеми отримуємо формулу для визначення фокусної відстані f_x та f_y:

$$f_{xy} = \frac{\left(\frac{x,y}{2}\right)}{tg\left(\frac{FOV}{2}\right)}, \tag{2.4}$$

де x,y – ширина і висота кадру, FOV це кут огляду у градусах.

З цього в декартовій системи виходять наступні формули для переведення з тривимірного до двовимірного зображення [26]:

$$u = \frac{Xf_x}{Z} + u_0, \tag{2.5}$$

$$v = \frac{Xf_y}{Z} + v_0. \tag{2.6}$$

Для переведення з двовимірного в тривимірне формули будуть виглядати наступним чином [24]:

$$X = (u - u_0)\frac{z}{f_x}, \tag{2.7}$$

$$Y = (v - v_0)\frac{z}{f_y}. \tag{2.8}$$

На основі цієї хмари точок і буде визначатися розміри коробки товару. Отже, алгоритм представлено на рисунку 2.3.

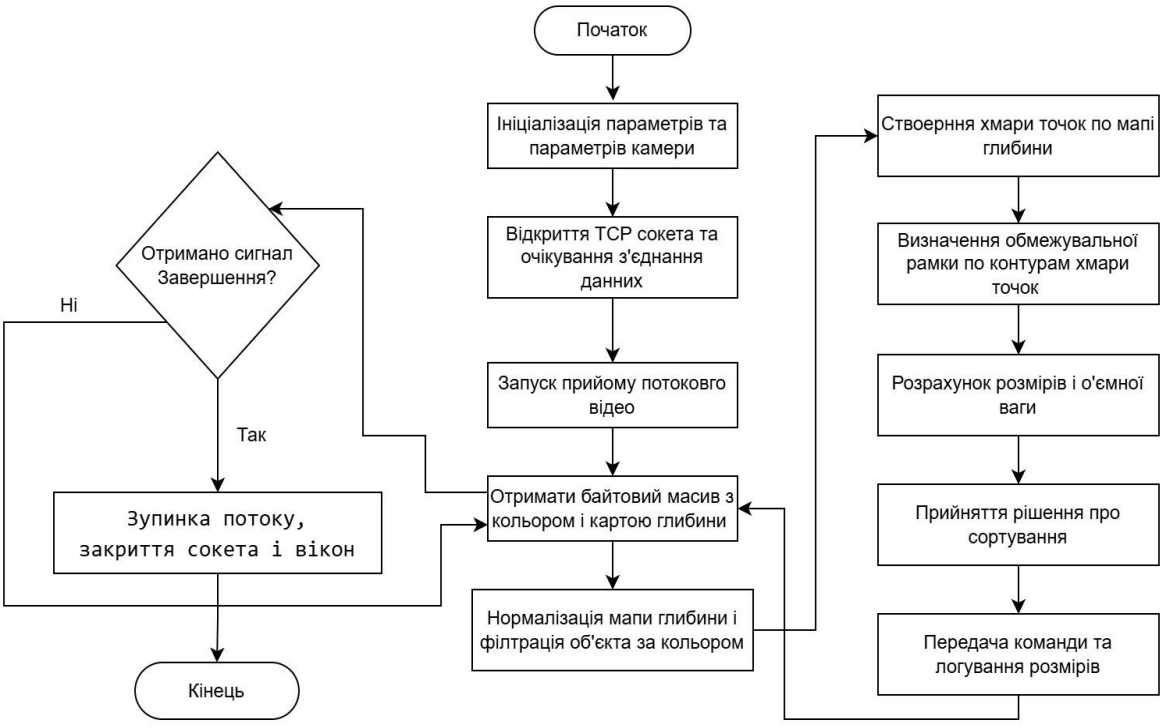

Рисунок 2.3 – Алгоритм роботи прорами

Сегментація коробки по кольору буде зроблена за допомогою бібліотеки OpenCV. OpenCV – це бібліотека комп'ютерного зору з відкритим кодом, яка призначена для аналізу, класифікації та обробки зображень.

Ця бібліотека є кросплатформовою та доступна для Windows, Linux, Android, MacOS, FreeBSD, OpenBSD, ARM, Android, Maemo, iOS та інших платформ. Також вона наявна для мов програмування C, C++, Python та Java [27, 28].

Для роботи з хмарою точок буде використовуватись бібліотека Open3D. Open3D – це бібліотека з відкритим кодом, яка підтримує швидку розробку програмного забезпечення для роботи з 3D-даними. Фронтенд Open3D надає набір ретельно відібраних структур даних та алгоритмів як на C++, так і на Python. Бекенд високо оптимізований та налаштований на паралелізацію.

Open3D було розроблено з чистого аркуша з невеликим та ретельно продуманим набором залежностей.

Його можна налаштувати на різних платформах та скомпілювати з вихідного коду з мінімальними зусиллями. Код чистий, має узгоджений стиль та підтримується за допомогою чіткого механізму перевірки коду. Open3D використовувався в низці опублікованих дослідницьких проектів та активно розгортається в хмарі [29].

3 РОЗРОБКА ВІРТУАЛЬНОГО МАКЕТУ ДЛЯ МОДЕЛЮВАННЯ АВТОМАТИЗОВАНОЇ ЛІНІЇ РОЗПОДІЛУ ТОВАРІВ

3.1 Розробка сцени в Unreal Engine 5

Для розробки віртуального макету для моделювання автоматизованої лінії розподілу товарів було визначено, що необхідно використовувати рушій Unreal Engine 5 для симуляції проходження товарів по конвеєрній лінії, захоплення відеопотоку та симуляції сортувального механізму лопатковими сортувальниками. А також відокремлений від рушія сервер з мовою програмування Python для обробки відеопотку з рушія, опрацювання, визначення розмірів, габаритної ваги та прийняття рішення щодо сортування.

Для того щоб почати розробляти віртуальний макет необхідно завантажити рушій. Для цього на офіційному сайті Unreal Engine знаходимо сторінку для завантаження, де вказано, що для того щоб почати працювати з UE5 потрібно встановити Epic Games Launcher. Це додаток звідки можна завантажити рушій. Після встановлення та запуску він буде виглядати настяк показано на рисунку 3.1. Далі нам потрібна вкладка Unreal Engine та в правому верхньому куту буде кнопка «Install Engine», натискаємо на неї, вибираємо директорію встановлення(або за замовчуванням) і чекаємо кінця завантаження. Після встановлення на робочому столі повинен з’явитися ярлик програми Unreal Engine, при запуску якого відкривається програма Unreal Editor (рис. 3.2), що є своєрідним менеджером проектів. Там створюємо бланк-проект в Games або Simulation, обов’язково з підтримкою Blueprints, даємо йому назву і він автоматично відривається вже в редакторі.

Проект завантажується і ми бачимо інтерфейс користувача і пустий рівень де присутні стартовий набор елементів, серед яких елементи пейзажу, направлене світло, атмосфера неба, небесне світло та ін. Усі елементи в

цьому рушії представлені у вигляді об’єктів які мають клас, що визначає їхні властивості. Серед них основними є:

– Актор – це будь-який об'єкт, який можна розмістити на рівні, наприклад, камера, статична сітка або початкова локація гравця. Актори підтримують 3D-трансформації, такі як переміщення, обертання та масштабування. Їх можна створюват та знищувати за допомогою ігрового коду (C++ або Blueprints) [23];

– пішак – це базовий клас усіх Акторів, якими можуть керувати гравці або ШІ. Пішак (Pawn) – це фізичне представлення гравця або сутності ШІ у світі. Пішак визначає візуальний вигляд гравця або сутності ШІ, те, як він взаємодіє зі світом з точки зору зіткнень та інших фізичних взаємодій [23];

– світ, рівень – об'єкт, що характеризує загальні властивості «простору», наприклад, силу тяжіння й туман, у якому розташовуються всі актори.

Далі створюємо новий рівень куди розміщуємо елементі що показані на рисунку 3.3.

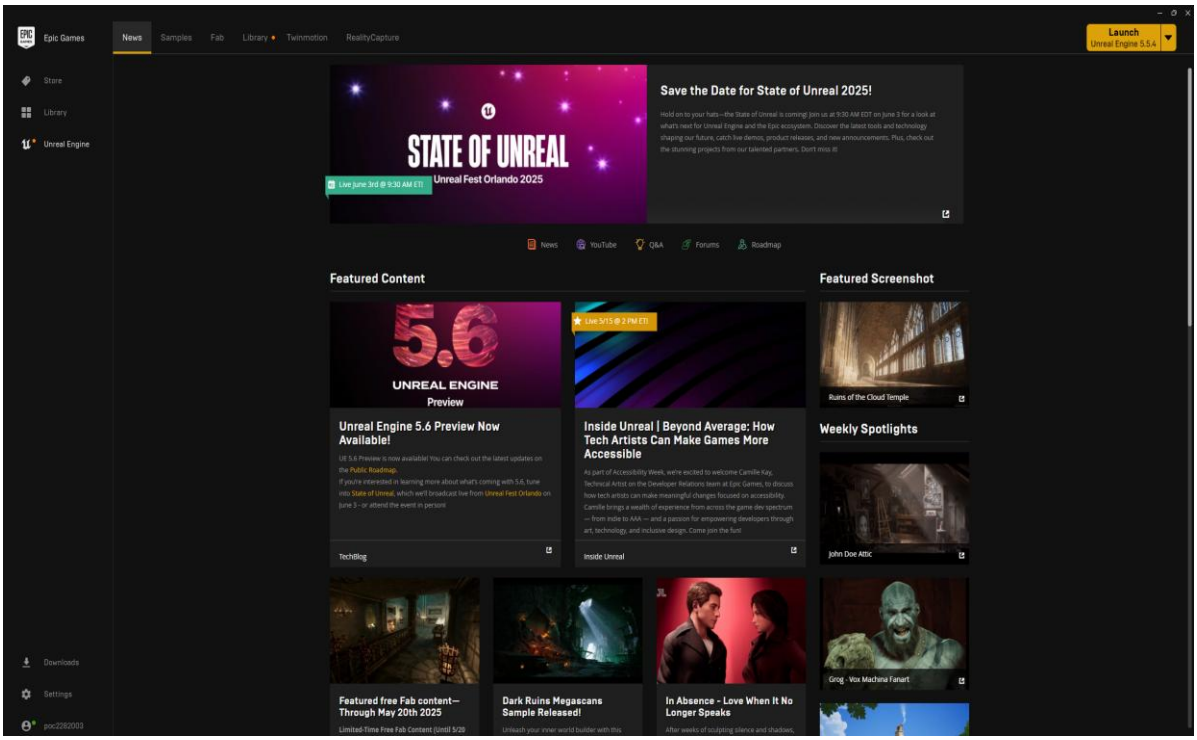

Рисунок 3.1 – Робоче вікно Epic Games Launcher

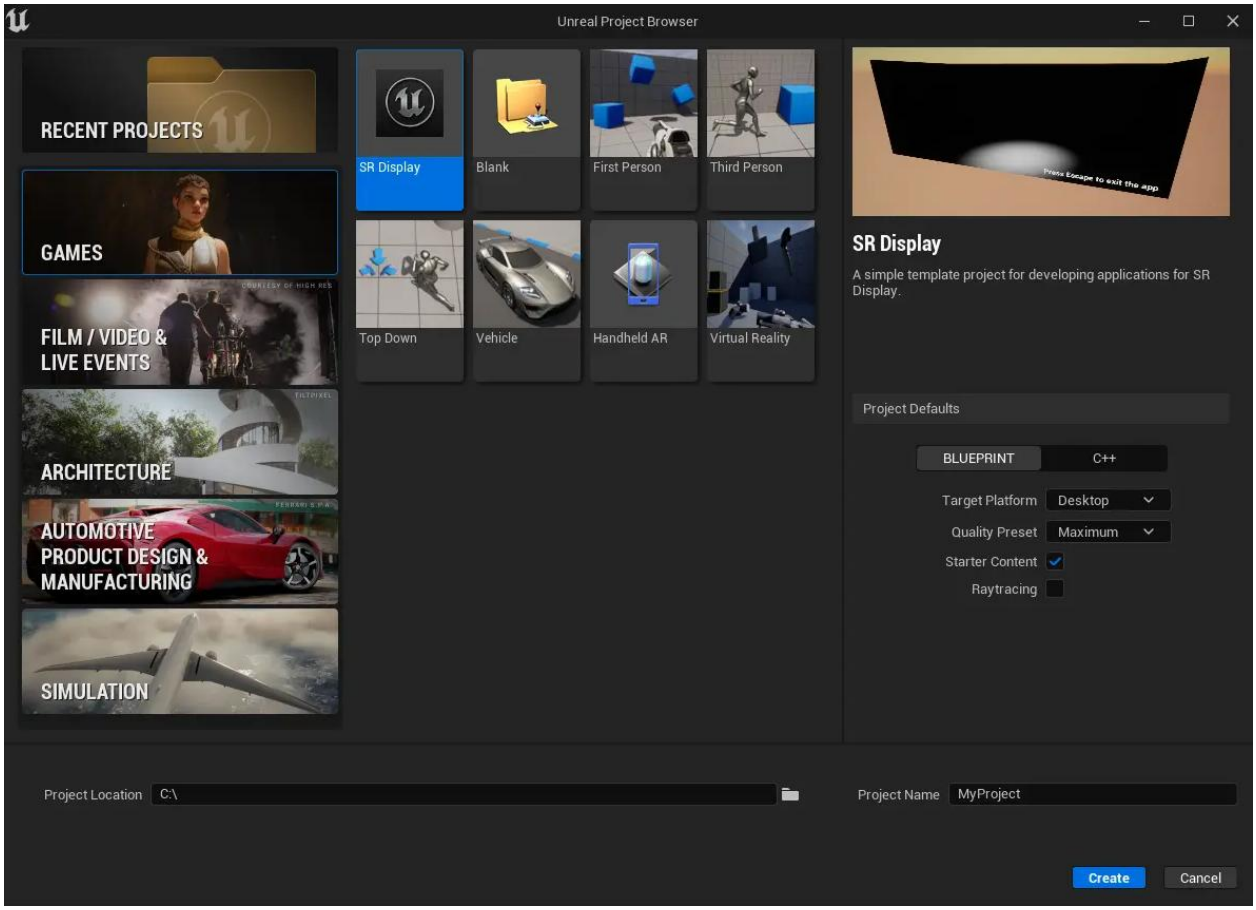

Рисунок 3.2 – Вікно створення проекту в Unreal Editor

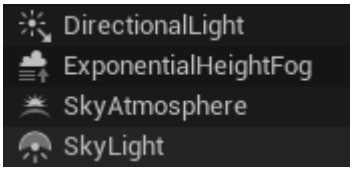

Рисунок 3.3 – Елементи для розміщення

Далі підготуємо необхідні 3D моделі. Для цього можна скористатися інтернет-ресурсами де можна безкоштовно завантажити необхідні моделі і використовувати їх в роботі. Серед таких ресурсів є два сайти, це Sketchfab [30] та Quixel Megascans [31], знаходимо необхідні нам моделі та скачуємо їх. Моделі мають бути в розширенні FBX. За допомогою програм для роботи з 3D моделями, наприклад Blender, доопрацьовуємо моделі за необхідності. В Content Browser, що є файловим менеджером по проекту в UE, додаємо папку Models і в ній підпапку для кожної моделі, і простим Drag'n'Drop переносимо

наші моделі в відповідні папки. В нашому випадку це моделі конвеєра, моделі коробок(декілька різних штук) та інші моделі для створення сцени-чернетки. Приклад використаних моделей показано на рисунках 3.4 – 3.5 .

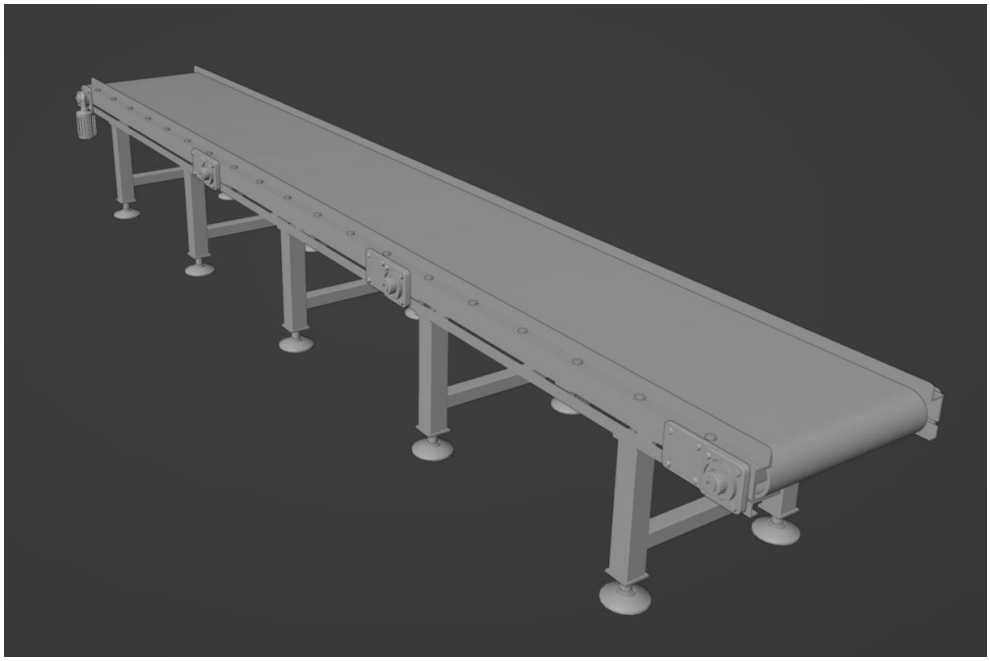

Рисунок 3.4 – 3D модель конвеєра

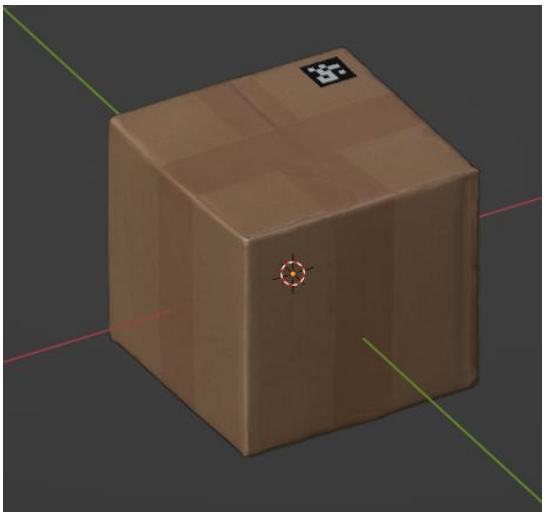

Рисунок 3.5 – 3D модель коробки

В редакторі Unreal Engine підключаємо додаток Bridge для того щоб можна було завантажувати необхідні нам матеріали з ресурсу Quixel

Megascans і однією кнопкою додавати їх в сцену. Розставивши моделі наступний чернетковий варіант сцени (рис. 3.6).

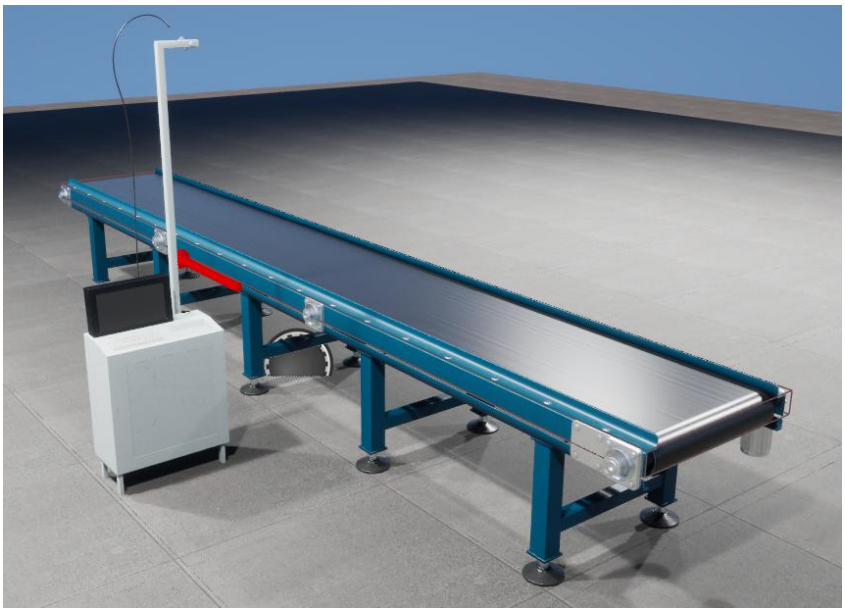

Рисунок 3.6 – Сцена «чернетка»

Далі створюємо Blueprint камери для захоплення потоку. Система візуальних скриптів Blueprint в Unreal Engine – це повноцінна система скриптів ігрового процесу, заснована на концепції використання інтерфейсу на основі вузлів для створення елементів ігрового процесу в Unreal Editor. Як і багато поширених мов скриптів, вона використовується для визначення об'єктно-орієнтованих (ОО) класів або об'єктів у рушії [23].

Для цього створюємо модель ToF камери (рис. 3.7), в окремій папці в файловому менеджері створюємо додатково для елементи з назвою Render Target, та створюємо в UE Blueprint, клас Actor. Заходимо в його налаштування і бачимо інший інтерфейс користувача і пусту сцену (рис. 3.8).

Додаємо в цю сцену створену модель камери, та в лівому верхньому кутку через кнопку Add Component додаємо два компоненти з назвою Scene Capture 2D. Scene Capture 2D захоплює сцену з її усіченого кута зору, зберігає цей вигляд як зображення, яке потім використовується в матеріалі. Фактично, представлення Scene Capture 2D в Unreal Editor – це камера [23]. Render

Target являє собою буфер для зберігання захопленої сцени. В налаштуваннях для однієї з камер в полі Texture Target обираємо один з наших створених Render Target-ів(нехай його назва буде RT_RGB) і в полі Capture source обираємо Scene Color (HDR) in RGB, 0 in A.

Ця «камера» буде відповідати за захоплення кольорового зображення з камери. Для іншого Scene Capture 2D в полі Texture Target обираємо інший Render Target(нехай його назва буде RT_Depth), і в полі Capture Source обираємо SceneDepth in R.

Друга «камера» буде відповідати за захоплення глибини сцени. Створений Blueprint камери буде виглядати як показано на рисунку 3.9. Розміщуємо в сцену Blueprint ToF камери над конвеєром під кутом (рис. 3.10). Також необхідно налаштувати Render Target-и.

Для цього заходимо в налаштування обох і для того який є буфером для кольорового зображення в полі Render Target Format обираємо RTF RGBA 8, для іншого – RTF R16f.

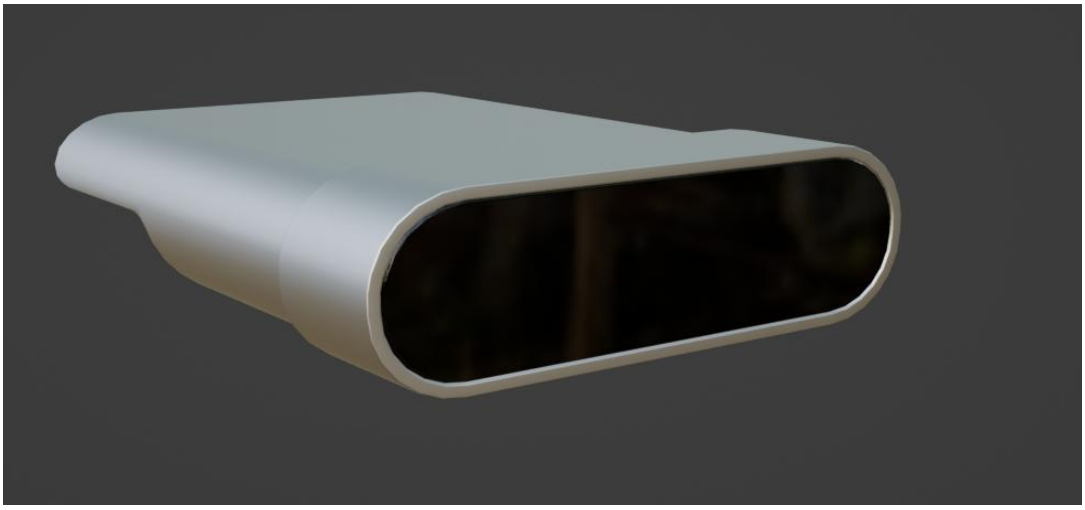

Рисунок 3.7 – Модель камери

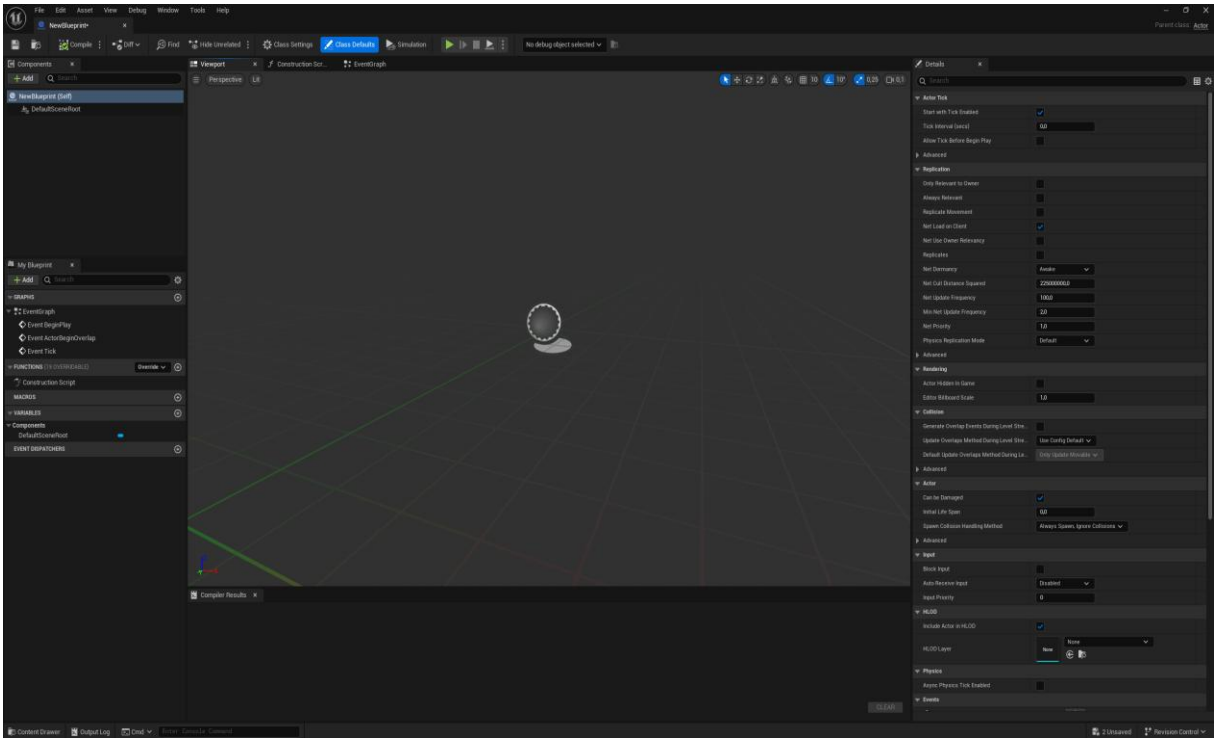

Рисунок 3.8 – Редактор Blueprint

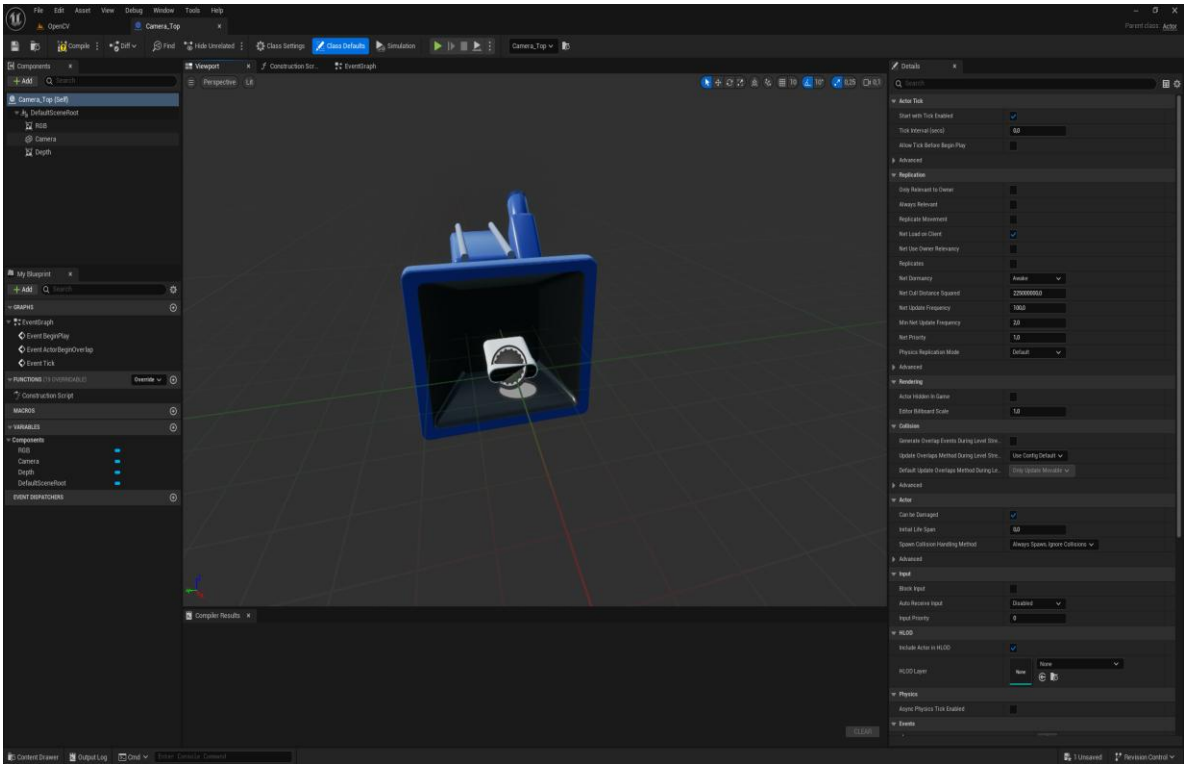

Рисунок 3.9 – Blueprint камери

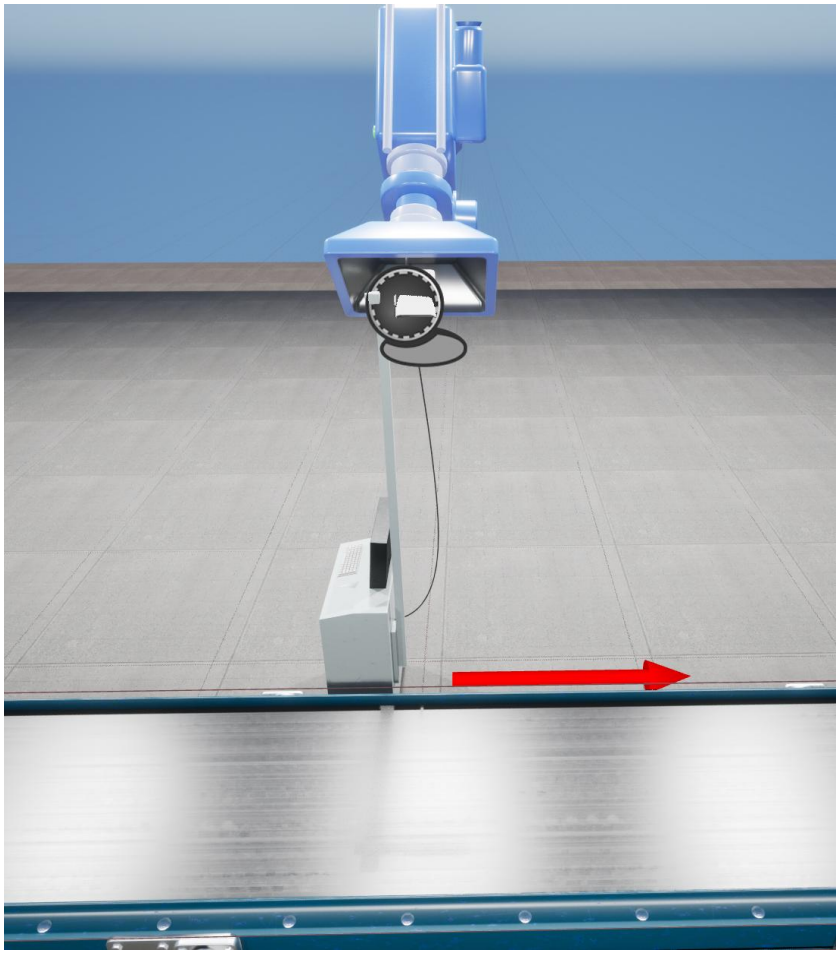

Рисунок 3.10 – Камера в сцені

3.2 Розроблення класу для відправки потокового відео

Тепер треба створити C++ клас який буде відповідати за надсилання відеопотоку з імітованої ToF камери на сервер по протоколу передачі даних TCP/ IP. TCP/IP в нашому випадку буде гарним рішенням, бо він передає данні по заданому порядку, з гарантією цілісності, і не має обмеження на кількість передаваємих байтів інформації.

Так як весь проект знаходиться на одному пристрої, то відправляти дані будемо на локальну IP адресу 127.0.0.1 і використовувати порт 5005. Працювати з C++ кодом будемо в середові розробки Visual Studio 2022.

Створюємо клас Актора з назвою PictureSender. Відривається Visual Studio з файлами PictureSender.h та PictureSender.cpp де вже є шаблонний код,

оскільки клас створювався не пустим, а наслідувався від головного класу Актора. Почнемо розробку класу із заголовного файлу PictureSender.h. Підключаємо наступні бібліотеки:

```
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Engine/TextureRenderTarget2D.h"
#include "Sockets.h"
#include "SocketSubsystem.h"
#include "Containers/Queue.h"
#include "Async/Async.h"
#include "PictureSender.generated.h"
```

де CoreMinimal.h базова бібліотека для роботи з класами в UE, GameFramework/Actor.h батьківський клас з якого наслідувались, Engine/TextureRenderTarget2D.h клас для роботи з Render Target-ами, Sockets.h та SocketSubsystem.h для роботи з TCP підключенням, Queue.h неблокуюча багатопотокова черга щоб не блокувати основий потік рушія довгими операціями, Async.h для запуску завдань у фоновому потоці, PictureSender.generated.h це автоматично згенерований файл програмою UnrealHeaderTool для подальшої сумісності з Blueprints.

Далі описуємо інтерфейс класу.

```
// Максимальний розмір буфера для сокета (8 мегабайт)
#define BUFFER_SIZE (8 * 1024 * 1024)

// Позначаємо клас для Unreal Engine — включається в систему рефлексії, дозволяє
// налаштовувати властивості в Blueprint та редакторі
UCLASS()
class OPENCVTEST_API APictureSender : public AActor
{
    GENERATED_BODY()  // Вставляє автоматично згенерований код для рефлексії та
Blueprint
```

Оголошуємо клас APictureSender. UClass — це клас, який використовується для мета/рефлексії [32]. Рефлексія в програмуванні – це здатність програми аналізувати та змінювати свою структуру та поведінку під час виконання. За допомогою рефлексії програмісти можуть отримувати

доступ до інформації про класи, методи та поля об’єктів, а також змінювати їх [33]. Мета/рефлексія є поширеною частиною великих ігрових двигунів, але не вбудовано в C++ нативно. Оскільки це не вбудовано в C++, рушіям потрібно знайти спосіб відображення цих даних та забезпечити, щоб користувачі знали, які дані відображаються, простим способом, розробники Unreal вирішили, що позначка вашого класу синтаксисом, подібним до UCLASS(), ваших методів за допомогою UFUNCTION(), а властивостей за допомогою

UPROPERTY() – це чудовий спосіб явно визначити, які дані/методи будуть відображатися [32]. Тобто UClass позначає оголошений клас APictureSender як компонент системи рефлексії в рушії Unreal Engine.

OPENCVTEST_API – це макрос що відповідає за позначення функцій, класів або даних як публічних у DLL-файлі модуля. Під модулем мається на увазі основний будівельний блок архітектури програмного забезпечення Unreal Engine, що інкапсулюює певні інструменти редактора, функції середовища виконання, бібліотеки або інші функції в окремих одиницях коду [34].

Public AActor – це наслідування від основного класу Актора. Мається на увазі що він буде мати властивості актора і його можна буде розмістити на рівні.

GENERATED_BODY() це автоматично згенерований код вбудованим застосунком UnrealHeaderTool.

```
public:
    // Конструктор: викликається при створенні актора (навіть до запуску гри)
    APictureSender();

    // BeginPlay — викликається один раз на старті гри/рівня або при спавні актора
    virtual void BeginPlay() override;

    // Tick — викликається що кадр, DeltaTime = час (в секундах) з останнього виклику
    virtual void Tick(float DeltaTime) override;

    // EndPlay — викликається перед знищенням або зупинкою рівня,
    // тут завершуємо фонові процеси
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override;
```

APictureSender() – конструктор класу що викликається при створенні рушієм екземпляру цього класу.

Virtual void BeginPlay() override – віртуальна функція, що викликається при старті гри. Ключове слово virtual вказує на функції, які можна перевизначити в дочірніх класах. Функцію BeginPlay() було оголошено в класі Unreal Engine C++ AActor, тому будь-який клас, що належить до ієрархії AActor, може перевизначити BeginPlay() [35].

Override використовується для того, щоб переконатися, що програміст дійсно перевизначає функцію батьківського класу. Програміст може переплутати кількість або тип параметрів під час перевизначення функції. При використанні override компілятор генеруватиме помилку, якщо еквівалентної функції не знайдено в батьківському класі.

Virtual void Tick(float DeltaTime) override – метод що викликається кожного кадру якщо в конструкторі прописано PrimaryActorTick.bCanEverTick = true.

virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override –Метод що викликається перед повною зупинкою гри.

```
protected:
    // Метод для налаштування та підключення TCP-сокета
    void InitializeSocket();

    /**
     * Рендер-таргет для зйомки глибини (формат RTF_R16f)
     * Редагується в Blueprint/EditAnywhere
     */
    UPROPERTY(EditAnywhere, Category = "Capture")
    UTextureRenderTarget2D* DepthTarget;

    /**
     * Рендер-таргет для зйомки кольору (формат RTF_RGBA8)
     * Редагується в Blueprint/EditAnywhere
     */
    UPROPERTY(EditAnywhere, Category = "Capture")
    UTextureRenderTarget2D* ColorTarget;
```

void InitializeSocket() – метод що буде використовуватись для налаштування підключення TCP сокета.

UPROPERTY(EditAnywhere, Category = "Capture") UTextureRenderTarget2D* DepthTarget; UPROPERTY(EditAnywhere, Category = "Capture") UTextureRenderTarget2D* ColorTarget; UPROPERTY() – макрос використовується для відображення змінних у редакторі Unreal Engine та для включення властивості до системи керування пам'яттю Unreal Engine. Параметри UPROPERTY() називаються специфікаторами властивостей. У наведеному вище прикладі було використано EditAnywhere та Category, що використовуються для групування змінних у вікні властивостей [36]. В нашому випадку цей макрос створює поле в інспекторі(деталях) класа, де можна вибрати з якого рендер таргета буде надсилатися відеопотік.

```
private:
    // Вказівник на TCP-сокет клієнта, ініціалізується в InitializeSocket()
    FSocket* ClientSocket = nullptr;

    // IP-адреса та порт сервера, вказуються в InitializeSocket()
    TSharedPtr<FInternetAddr> ServerAddress;

    // Багатопотокова черга пар <TagBytes, Payload>:
    // – TagBytes (4 байти) вказує тип пакета ('COLR' або 'LDEP'),
    // – Payload містить власне дані пікселів
    TQueue<TPair<TArray<uint8>, TArray<uint8>>, EQueueMode::Mpsc> FrameQueue;

    // Атомарний флаг для коректного завершення фонового потоку
    TAtomic<bool> bRunSender{ true };
};
```

FSocket* ClientSocket = nullptr – вказівник на об’єкт сокета для встановлення TCP-з’єднання та відправки/прийому даних.

TSharedPtr<FInternetAddr> ServerAddress – покажчик Unreal Engine на об’єкт FInternetAddr, який містить IP адресу і порт сервера, на який буде відправлятись відеопотік.

TQueue<TPair<TArray<uint8>, TArray<uint8>>, EQueueMode::Mpsc> FrameQueue – черга пакетів що будуть надсилатися по TCP/IP на сервер. Перший <TArray<uint8> – це тег пакету, відповідає за розрізнення пакету

відеопотоку кольорового зображення від пакету з картою глибини. Другий <TArray<uint8> – це власне байти даних відеопотоку. EQueueMode::Mpsc режим відправки Multiple producer, Single consumer, що означає що відправників буде декілька а фоновий потік один.

TAtomic<bool> bRunSender{ true } – Атомарний флаг для коректного завершення фонового потоку. Атомний означає, що лише один потік має доступ до змінної (статичний тип). Atomic є потокобезпечним, але він повільний. Неатомний означає, що кілька потоків отримують доступ до змінної (динамічний тип). Nonatomic небезпечний для потоків, але він швидкий [37].

Після цього в файлі PictureSender.cpp, описуємо методи, які прописали в інтерфейсі класу.

```
#include "PictureSender.h"
#include "Engine/TextureRenderTarget2D.h"    // Для UTextureRenderTarget2D і ресурсів рендер-таргета
#include "RenderUtils.h"                        // Для роботи з ReadPixels / ReadLinearColorPixels
#include "SocketSubsystem.h"                       // Для ISocketSubsystem і FInternetAddr
#include "HAL/PlatformProcess.h"            // Для FPlatformProcess::Sleep
```

Підключення всіх необхідних бібліотек.

```
/**
 * Блочне відправлення всіх Size байт з буфера Data через сокет Socket.
 * Повертає true, якщо всі байти успішно відправлені, і false при помилці.
 */
static bool BlockingSendAll(FSocket* Socket, const uint8* Data, int32 Size)
{
    int32 Total = 0;  // скільки байт вже відправлено
    while (Total < Size)
    {
        int32 Sent = 0;
        // Отримуємо, скільки байт реально відправлено при цьому виклику Send
        if (!Socket->Send(Data + Total, Size - Total, Sent) || Sent <= 0)
        {
            // При помилці читаємо код останньої помилки й логуємо
            int32 Err = ISocketSubsystem::Get(PLATFORM_SOCKETSUBSYSTEM)
                ->GetLastErrorCode();
            UE_LOG(LogTemp, Error, TEXT("SendAll failed: Sent=%d Err=%d"), Sent, Err);
            return false;
        }
```

```
            Total += Sent;  // накопичуємо лічильник надісланих байт
        }
        return true;
    }
```

Булева функція BlockingSendAll блочно відправляє всі байти розміру Size з буферу Data через TCP/IP сокет на який вказує покажчик FSocket* Socket. Повертає True якщо всі байти розміру Size були відправлені успішно, False якщо сталась помилка і логує її з кодом помилки.

```
    /**
     * Формує заголовок із 4-байтного Tag і 4-байтного розміру payload,
     * а потім блочним чином відправляє Header і Payload через BlockingSendAll.
     */
    static void BlockingSendPacket(FSocket* Socket, const char Tag[4],
TArray<uint8>&& Payload )
    {
        // 1) Формуємо 8-байтовий заголовок: [ Tag[0..3] | size (little-endian) ]
        uint8 Header[8];
        FMemory::Memcpy(Header, Tag, 4);       // копіюємо 4 байти тегу
        uint32 PS = Payload.Num();             // довжина даних
        Header[4] = PS & 0xFF;                 // молодший байт розміру
        Header[5] = (PS >> 8) & 0xFF;
        Header[6] = (PS >> 16) & 0xFF;
        Header[7] = (PS >> 24) & 0xFF;         // старший байт розміру

        // 2) Відправляємо спочатку заголовок, потім власне payload
        BlockingSendAll(Socket, Header, sizeof(Header));
        BlockingSendAll(Socket, Payload.GetData(), PS);

    }
```

Функція BlockingSendPacket формує 8 байтовий заголовок, який складається з масиву 4 байтів тегу(Tag), що вказує який відеопотік кольоровий чи карти глабини, і масиву 4 байтів розміру Payload в якому знаходятся масив данних про пікселі. І через два виклики BlockingSendAll надсилаємо через TCP/IP наш пакет данних.

```
    /**
     * Конструктор актора: дозволяє виклик Tick() що кадр
     */
    APictureSender::APictureSender()
    {
        PrimaryActorTick.bCanEverTick = true;
    }
```

APictureSender::APictureSender() конструктор актора з викликом функції PrimaryActorTick.bCanEverTick = true щоб функція Tick() викликалась що кадру.

```
        /**
         * BeginPlay: викликається на старті рівня або при спавні актора.
         * Логує стан таргетів, ініціалізує TCP-сокет і запускає фоновий потік відправки.
         */
        void APictureSender::BeginPlay()
        {
            Super::BeginPlay();

            // Вивід у лог інформації про наявність і розміри DepthTarget та ColorTarget
            UE_LOG(LogTemp, Warning,
                TEXT("BeginPlay: Depth=%s %dx%d, Color=%s %dx%d"),
                DepthTarget ? TEXT("OK") : TEXT("NULL"),
                DepthTarget ? DepthTarget->SizeX : 0,
                DepthTarget ? DepthTarget->SizeY : 0,
                ColorTarget ? TEXT("OK") : TEXT("NULL"),
                ColorTarget ? ColorTarget->SizeX : 0,
                ColorTarget ? ColorTarget->SizeY : 0
            );

            // Налаштовуємо й підключаємо сокет
            InitializeSocket();

            // Запускаємо єдиний фоновий потік: він зчитує з FrameQueue і відправляє пакети
            TWeakObjectPtr<APictureSender> WeakThis(this);
            Async(EAsyncExecution::Thread, [WeakThis]()
                {
                    // Цикл працюватиме доти, поки актор валідний, bRunSender=true і сокет з’єднано
                    while (WeakThis.IsValid() && WeakThis->bRunSender.Load() &&
                        WeakThis->ClientSocket &&
                        WeakThis->ClientSocket->GetConnectionState()               == SCS_Connected)
                    {
                        // Спроба діставання одного елемента з черги
                        TPair<TArray<uint8>, TArray<uint8>> Item;
                        if (WeakThis->FrameQueue.Dequeue(Item))
                        {
                            // Розпаковуємо тег і payload, відправляємо через BlockingSendPacket
                            char Tag[4];
                            for (int32 i = 0; i < 4; ++i)
                                Tag[i] = Item.Key[i];
                            BlockingSendPacket(WeakThis->ClientSocket,           Tag, MoveTemp(Item.Value));
                        }
                        else
                        {
                            // Якщо черга порожня, невеликий sleep, щоб не завантажувати CPU
```

```
                        FPlatformProcess::Sleep(0.001f);
                    }
                }
            });
    }
```

Метод BeginPlay(). Викликається при старті сцени, логує готовність рендер таргетів, які були обрані в налаштуваннях класа Актора розміщеному на рівні, ініціалізує та підключає сокет, та запускає фоновий потік Async(EAsyncExecution::Thread, [WeakThis]() в якому буде діставатися пакети з черги і передає їх в метод BlockingSendPacket.

```
    /**
     * Tick: викликається кожен кадр.
     * Зчитує пікселі з DepthTarget і ColorTarget, пакує їх у FrameQueue.
     */
    void APictureSender::Tick(float DeltaTime)
    {
        Super::Tick(DeltaTime);

        // Перевіряємо, чи сокет ініціалізований і з'єднаний
        if (!ClientSocket || ClientSocket->GetConnectionState() != SCS_Connected)
            return;

        // --- Обробка глибинного кадру ---
        if (DepthTarget)
        {
            // Отримуємо ресурс рендер-таргета (GameThread)
            if (auto* Res = DepthTarget->GameThread_GetRenderTargetResource())
            {
                int32 NumPx = DepthTarget->SizeX * DepthTarget->SizeY;
                // Масив для зчитування float RGBA
                TArray<FLinearColor> Lin;
                Lin.SetNumUninitialized(NumPx);

                // Зчитуємо всі пікселі як FLinearColor
                if (Res->ReadLinearColorPixels(Lin))
                {
                    // Готуємо тег і буфер під float-дані (тільки компонент R)
                    TArray<uint8> Tag = { 'L','D','E','P' };
                    TArray<uint8> Payload;
                    Payload.SetNumUninitialized(NumPx * sizeof(float));
                    float* D = reinterpret_cast<float*>(Payload.GetData());
                    for (int32 i = 0; i < NumPx; ++i)
                        D[i] = Lin[i].R;  // копіюємо лише R → значення глибини

                    // Додаємо в чергу на відправку
                    FrameQueue.Enqueue({ MoveTemp(Tag), MoveTemp(Payload) });
                }
                else
                {
                    UE_LOG(LogTemp,   Error,   TEXT("Depth   ReadLinearColorPixels
failed"));
```

```
            }
        }
    }

    // --- Обробка кольорового кадру ---
    if (ColorTarget)
    {
        if (auto* Res = ColorTarget->GameThread_GetRenderTargetResource())
        {
            int32 NumPx = ColorTarget->SizeX * ColorTarget->SizeY;
            // Масив для зчитування FColor (RGBA8)
            TArray<FColor> Col;
            Col.SetNumUninitialized(NumPx);

            // Зчитуємо всі пікселі
            Res->ReadPixels(Col);

            // Готуємо тег і буфер під RGBA8
            TArray<uint8> Tag = { 'C','O','L','R' };
            TArray<uint8> Payload;
            Payload.SetNumUninitialized(NumPx * sizeof(FColor));
            FMemory::Memcpy(Payload.GetData(), Col.GetData(), Payload.Num());

            // Додаємо в чергу
            FrameQueue.Enqueue({ MoveTemp(Tag), MoveTemp(Payload) });
        }
    }
}
```

Метод Tick(float DeltaTime). Викликається кожен кадр, перевіряє підключення до серверу, отримує данні з рендер таргетів, зчитує з них пікселі як FColor для кольоровго або як FLinearColor, формує тег і створює масив Payload з розміром NumPx * sizeof(FColor) для кольорового або NumPx * sizeof(float) для глибини , в який копіюється весь масив утворений зі считування рендер таргетів. І в кінці записує тег і Payload в чергу FrameQueue.Enqueue.

```
/**
 * EndPlay: викликається при завершенні гри/рівня або видаленні актора.
 * Змінює флаг bRunSender, щоб фоновий потік завершився.
 */
void APictureSender::EndPlay(const EEndPlayReason::Type EndPlayReason)
{
    bRunSender.Store(false);  // сигналізує фон-потоку про зупинку
    Super::EndPlay(EndPlayReason);
}
```

Метод EndPlay викликається для сігналізації фон потку про завершення роботи при завершенні гри чи знищенні актора з рівня.

```
        /**
         * InitializeSocket: створює TCP-сокет, налаштовує його і підключається
         * до сервера за IP=127.0.0.1, порт=5005.
         */
        void APictureSender::InitializeSocket()
        {
            // 1) Створення сокета через SocketSubsystem
            ClientSocket = ISocketSubsystem::Get(PLATFORM_SOCKETSUBSYSTEM)
                ->CreateSocket(NAME_Stream, TEXT("TCP_Client"), false);
            if (!ClientSocket)
            {
                UE_LOG(LogTemp, Error, TEXT("InitSocket: cannot create"));
                return;
            }

            // 2) Налаштування сокета
            ClientSocket->SetNoDelay(true);                // вимикає Nagle, зменшує
затримку
            ClientSocket->SetNonBlocking(false);      // робимо блокуючим (у фоні)

            int32 Buf;
            ClientSocket->SetReceiveBufferSize(BUFFER_SIZE, Buf);
            ClientSocket->SetSendBufferSize(BUFFER_SIZE, Buf);

            // 3) Підготовка адреси сервера
            auto Addr = ISocketSubsystem::Get(PLATFORM_SOCKETSUBSYSTEM)
                ->CreateInternetAddr();
            bool bValid = false;
            Addr->SetIp(TEXT("127.0.0.1"), bValid);
            Addr->SetPort(5005);
            if (!bValid)
            {
                UE_LOG(LogTemp, Error, TEXT("Invalid IP"));
                return;
            }
            ServerAddress = Addr;

            // 4) Підключення до сервера
            if (ClientSocket->Connect(*ServerAddress))
            {
                UE_LOG(LogTemp, Warning,
                    TEXT("Connected to %s:%d"),
                    *ServerAddress->ToString(false), ServerAddress->GetPort());
            }
            else
            {
                int32 Err = ISocketSubsystem::Get(PLATFORM_SOCKETSUBSYSTEM)
                    ->GetLastErrorCode();
                UE_LOG(LogTemp, Error,
                    TEXT("Connect failed, Err=%d"), Err);
            }
        }
```

InitializeSocket створює TCP сокет для роботи з сервером а адресою 127.0.0.1 та портом 5005, встановлює блокуючий режим відправлення щоб кожен наступний пакет відправлявся лише тоді, коли відправляться попередні, логує успішність підключення до серверу та помилки якщо підключення не вдалось.

В результаті маємо клас який відправляє на IP адресу 127.0.0.1:5005 кожен раз коли спрацьовує Tick(), що дорівнює кількості кадрів за секунду, два пакети байтів вигляду <4-byte Tag><4-byte Size><Payload>, де перші 4 байти це тег COLR для кольору, LDEP для глибини, другі 4 байти це вказувач на довжину payload, і payload це масив даних пікселів.

При розборі пакету на серверній стороні читають спочатку 4 байти тегу, потім 4 байти розміру, і вже потім зчитують з пакету саме розмір байт даних із payload.

4 ПРОГРАМУВАННЯ ЛОГІКИ СОРТУВАННЯ ДЛЯ ВІРТУАЛЬНОГО МАКЕТУ ДЛЯ МОДЕЛЮВАННЯ АВТОМАТИЗОВАНОЇ ЛІНІЇ РОЗПОДІЛУ ТОВАРІВ

4.1 Розроблення зовнішньої логіки для обробки відеопотоку

Вже з’ясовано, що для сортування товарів необхідно визначити об’ємну вагу коробки товару, та на основі отриманих даних робити розподіл за вагою. Для визначення об’ємної ваги, а отже і розмірів, буде використовуватись метод з використанням ToF камери. Для обробки відеопотоку з симульованої камери буде використовуватись мова програмування Python версією 3.9.13 [38] з використанням бібліотек для комп’ютерного зору, створення та обробки хмари точок по карті глибини, а також для роботи з TCP/IP підключенням та математичних розрахунків. Для роботи з комп’ютерним зором буде використовуватись бібліотека OpenCV, для створення та обробки хмари точок – Open3D.

Для написання Python скрипта буде використовуватись середовище розробки PyCharm зі студентською ліцензією.

```
import socket
import struct
import time
import threading
import math
from itertools import product
import numpy as np
import cv2       #OpenCV
import open3d as o3d
```

Імпорт необхідних бібліотек для роботи, серед них socket, struct, threading для роботи з протоколами підключення та роботою в окремому потоці, math для математичних розрахунків, numpy для роботи з масивами.

```
T # ---- TCP SETTINGS ----
TCP_IP      = '127.0.0.1'   # Локальна IP-адреса, на якій слухаємо відеопотік
TCP_PORT    = 5005           # Порт для підключення відеопотоку
TCP_PORT2   = 5006           # Порт для підключення команд сортування
BUFFER_SIZE = 8_000_000          # Розмір буфера прийому даних (8 МБ)

# ---- CAMERA INTRINSICS (для 2D-проекції) ----
W, H    = 1024, 820         # Розміри кадру (ширина W, висота H)
FOV_DEG = 45.0           # Поле огляду камери в градусах
FX      = W / (2 * math.tan(math.radians(FOV_DEG/2)))  # Фокусна відстань по
X, обчислена з FOV
FY      = FX          # Фокусна відстань по Y (однакова)
CX, CY  = W/2.0, H/2.0        # Координати головної точки (центр кадру)

# ---- ROI: full height, horizontal margins 15% of frame width ----
margin = int(W * 0.15)         # Відступ зліва/справа 15% від ширини кадру
ROI_X1, ROI_X2 = margin, W - margin  # Межі ROI по горизонталі: ліво =
margin, право = W - margin
ROI_Y1, ROI_Y2 = 0, H      # Межі ROI по вертикалі: зверху = 0, знизу = H
(повна висота)

# Shared buffers and control flag
latest_color = None      # Змінна для зберігання останнього кольорового кадру
latest_depth = None       # Змінна для зберігання останнього кадру глибини
lock = threading.Lock()        # M’ютекс для синхронізації доступу до буферів
running = True            # Флаг, який показує, чи працює головний цикл

# Box identification and measurement storage
next_box_id = 1                      # Лічильник для присвоєння унікальних ID
коробкам
current_id = None        # Змінна для поточного ID коробки, що обробляється
dims_list = []       # Список для збору всіх виміряних розмірів пакунка в ROI

LOG_FILENAME = "boxes_log.txt"  # Ім’я файлу для логів результатів вимірювань
```

Оголошення змінних в яких визначається параметри для запуску серверу, для внутрішніх параметрів матриці камери (2.3), та для майбутньої зони інтересу(ROI). А також оголошення інших глобальних змінних для подальшої роботи

```
def recv_all(sock, n):  # Функція, яка читає точно n байт з сокета або
повертає None
    buf = b''       # Порожній буфер для накопичення байт
    while len(buf) < n:      # Поки не накопичимо n байт
        chunk = sock.recv(min(n - len(buf), BUFFER_SIZE))  # Читаємо чергову
порцію байт
        if not chunk     # Якщо повернули порожні байти, значить з'єднання
закрите
            return None  # Повертаємо None
        buf += chunk     # Додаємо отриману порцію байт до буфера
    return buf           # Повертаємо буфер довжини n
```

Функція recv_all отримує байти з TCP сокета в кількості n та записує в байтовий масив. Якщо данних замало або з'єднання розірване повертає None.

```
        def io_thread(conn):
            global latest_color, latest_depth, running
            try:
                while running:       # Поки флаг running True, продовжуємо прийом
                    hdr = recv_all(conn, 8)   # Читаємо заголовок: перші 8 байт
                    if hdr is None: continue  # Якщо заголовок не отримано,
переходимо до наступної ітерації
                    tag, size = struct.unpack('<4sI', hdr)  # Розпаковуємо 4 байти
тегу + 4 байти розміру (uint32)
                    data = recv_all(conn, size)  # Читаємо payload заданого розміру
                    if data is None: continue  # Якщо payload не отримано, переходимо
далі
                    with lock:           # Синхронізуємо запис у спільні змінні
                        if tag == b'COLR':    # Якщо тег 'COLR' (кольоровий кадр)
                            latest_color = data  # Зберігаємо дані в latest_color
                        elif tag == b'LDEP':  # Якщо тег 'LDEP' (кадр глибини)
                            latest_depth = data  # Зберігаємо дані в latest_depth
            except:
                pass                         # Ігноруємо будь-які виключні ситуації
```

Функція io_thread запускає в фоновому потоці і читає отримані пакети з сокета. Читає спочатку заголовок де зазначено тег і розмір Payload, а потім читає сам Payload розміру що вказаних в заголовку. Також у випадку розриву з'єднання код буде працювати далі.

```
        def normalize_depth(Z):
            p1, p99 = np.percentile(Z, (1, 99))  # Обчислюємо 1-й та 99-й перцентили
масиву глибин
            Zn = np.clip((Z - p1) / (p99 - p1), 0, 1)  # Масштабуємо значення Z у
діапазон [0,1]
            return (Zn * 255).astype(np.uint8)  # Перетворюємо в діапазон [0,255] і
повертаємо uint8-масив
```

```
        Функція normalize_depth потрібна для приведення карти глибини до 8бітного
зображення та виводу його в вікно відображення.
```

```
        def log_box(id_, dims_cm, vol_w, decision, note=""):
            with open(LOG_FILENAME, 'a') as log:  # Відкриваємо файл для дозапису
                log.write(
                    f"ID {id_} "                     # Пишемо ID пакунка
                    f"Length {dims_cm[0]:.2f} "      # Пишемо довжину в см з двома
знаками після коми
                    f"Width {dims_cm[1]:.2f} "       # Пишемо ширину в см
                    f"Height {dims_cm[2]:.2f} "      # Пишемо висоту в см
                    f"VolW {vol_w:.2f} "             # Пишемо об'ємну вагу (кг)
```

```
                    f"Dec {decision} "              # Пишемо рішення сортування
(FlapIndex)
                    f"{note}\n"                     # Пишемо додаткову примітку
(якщо є) і завершальний символ рядка
                )
```

Функція log_box призначена для запису в текстовий файл отриманих результатів вимірювань об’єкта.

Далі йде функція main.

```
    global latest_color, latest_depth, running # використання зовнішніх змінних
      global next_box_id, current_id, dims_list

      # Створюємо або очищуємо файл логів і записуємо заголовок
      with open(LOG_FILENAME, 'w') as f:
          f.write("ID Length(cm) Width(cm) Height(cm) VolW(kg) Dec Note\n")

      # Налаштування TCP-сервера для відеопотоку
      srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # Створюємо TCP-
сокет
      srv.bind((TCP_IP, TCP_PORT))    # Прив’язуємо сокет до IP і порту
      srv.listen(1)           # Починаємо прослуховування одного підключення
      print(f"[+] Listening on {TCP_IP}:{TCP_PORT}")  # Виводимо повідомлення
про старт прослуховування
      conn, addr = srv.accept()       # Чекаємо і приймаємо підключення клієнта

    # Налаштування TCP-сервера для команд сортування
      srv_cmd = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # Створюємо
другий сокет
      srv_cmd.bind((TCP_IP, TCP_PORT + 1))  # Прив’язуємо до TCP_PORT2 (5006)
      srv_cmd.listen(1)                      # Прослуховуємо підключення команд
      print(f"[+] Listening for sort commands on {TCP_IP}:{TCP_PORT + 1}")  #
Повідомлення
      conn_cmd, addr_cmd = srv_cmd.accept()  # Приймаємо підключення для команд
      print(f"[+] Command connection by {addr_cmd}")  # Виводимо адресу
підключення команд

    conn.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)  # Вимикаємо
алгоритм Nagle
      print(f"[+] Connected by {addr}")  # Повідомлення про підключення відео
```

На початку створюємо або чистимо файл для логів і записуємо заголовок майбутнього запису. Створюємо два TCP сокета, один для прийому відеопотоку, інший для відправки команд сортувальнику, прив’язуємо до зазначених раніше адреси і порту, і починаємо прослуховувати підключення від Unreal Engine та приймаємо підключення.

```
    # Запускаємо фоновий потік для прийому відеоданих
      threading.Thread(target=io_thread, args=(conn,), daemon=True).start()
```

```
            # Ініціалізуємо вікно OpenCV для відображення кадру та проекції
            cv2.namedWindow("RGB+3DBox", cv2.WINDOW_NORMAL)

        vis = o3d.visualization.Visualizer()  # Створюємо візуалізатор
            vis.create_window("Live Cloud + Box", width=800, height=600)  # Створюємо
вікно Open3D
            pcd    = o3d.geometry.PointCloud()     # Створюємо об’єкт порожньої хмари
точок
            box_ls  =  o3d.geometry.LineSet()      #  Створюємо  об’єкт  для  лінійних
сегментів куба
            vis.add_geometry(pcd)          # Додаємо пусту хмару до візуалізатора
            vis.add_geometry(box_ls)               # Додаємо лінійний об’єкт
            opts = vis.get_render_option()         # Отримуємо параметри відображення
            opts.point_size = 5.0         # Встановлюємо розмір точок у візуалізаторі
            opts.line_width = 3.0                  # Встановлюємо товщину ліній

            proc_ctr  =  0       #  Лічильник  кадрів  (для  зменшення  частоти  обробки
відображення)
            first_view = True # Прапорець для первинного налаштування камери

            # Параметри RANSAC для сегментації площин
            dist_thr = 0.005  # Поріг відстані для RANSAC
            ransac_n = 3      # Кількість точок для побудови однієї площини
            n_iters  = 1000   # Кількість ітерацій RANSAC
```

Далі запускаємо в фоновому потоці функцію io_thread. Створюємо вікно OpenCV для виводу відеопотоку і вікно Open3D для відобрвження хвари точок. Додаємо пусту хмару точок, лінійний об’єкт в вікні Open3D та встановлюємо параметри для візуалізації. Також додаємо нові змінні для майбутнього методу визначення площини граней коробки.

```
        try:
                while True:  # Головний цикл обробки кадрів
                    with lock:
                        cbuf = latest_color  # Копіюємо останній кольоровий буфер
                        dbuf = latest_depth  # Копіюємо останній буфер глибини
                        latest_color = None  # Очищаємо буфер кольору
                        latest_depth = None  # Очищаємо буфер глибини

                    if  cbuf  is  None  or  dbuf  is  None:   #  Якщо  якого-небудь  буфера
немає
                        time.sleep(0.005)             # Чекаємо 5 мс і повторюємо
                        continue

                    proc_ctr += 1  # Збільшуємо лічильник кадрів
                    if  proc_ctr  %  3  !=  0:   #  Обробляємо  кожен  третій  кадр,  щоб
зменшити навантаження
                        continue

                    # Декодування кольорового кадру (RGBA) → BGR для OpenCV
                    rgba = np.frombuffer(cbuf, np.uint8).reshape((H, W, 4))
                    bgr  = cv2.cvtColor(rgba, cv2.COLOR_BGRA2BGR)
```

```
                    # Побудова маски пікселів коробки у HSV-просторі
                    Z    = np.frombuffer(dbuf, np.float32).reshape((H, W)) * 0.01  # 
Конвертуємо depth у метри
                    hsv  = cv2.cvtColor(bgr, cv2.COLOR_BGR2HSV)  # Переводимо BGR → 
HSV
                    mask = cv2.inRange(hsv, (5,35,35), (25,255,255))   # Маска для 
коричнево-помаранчевого кольору
                    ys, xs = np.nonzero(mask)  # Індекси пікселів, що потрапили в 
маску
                    if ys.size < 100:  # Якщо дуже мало точок (коробка не знайдена)
                        cv2.imshow("RGB+3DBox", bgr)  # Відображаємо просто BGR
                        if cv2.waitKey(1) == 27: break   # Якщо натиснуто Esc, 
виходимо з циклу
                        continue
```

Далі копіюємо буфери з latest_color, latest_depth та очищаємо їх. Якщо буфери порожні то робимо паузу на 5мс і повторюємо. Оскільки Unreal Engine надсилає приблизно 30 кадрів в секунду, а Python не встигає їх обробляти, робимо вибірку 1 кадр через 3 щоб кадри не накопичувались і система не висла. Робимо декодування кольорового зображення з формату RGBA (Red, Green, Blue, Alpha) в формат який використовує OpenCV BGR (Blue Green Red) а з нього в HSV(Hue, Saturation, Value) для сегментації по кольору картона. Перевіряємо кількість пікселів в масці для розуміння чи є коробка в кадрі чи ні. Також переводимо отриману карту глибини в двовимірний масив де кожне значення в буде в метрах.

```
        # Створюємо 3D-точки з маски та глибини
                    zs  = Z[ys, xs]  # Z-координати (глибина в метрах)
                    x3d = (xs - CX) * zs / FX  # Обрахунок X у світових координатах
                    y3d = (CY - ys) * zs / FY  # Обрахунок Y (з урахуванням інверсії 
осі у зображенні)
                    pts = np.stack((x3d, y3d, zs), axis=1)[::5]  # Збираємо масив [X, 
Y, Z] і робимо підвибірку через 5

                    # Оновлюємо хмару точок у Open3D
                    pcd.points = o3d.utility.Vector3dVector(pts)
                    vis.update_geometry(pcd)

                    # Спроба побудувати орієнтований Bounding Box за допомогою двох 
RANSAC-площин
                    try:
                        m1, in1 = pcd.segment_plane(dist_thr, ransac_n, n_iters)  # 
Знаходимо першу площину RANSAC
                        rest1   = pcd.select_by_index(in1, invert=True)        # 
Залишаємо всі точки, що не належать першій площині
```

```
                        if len(rest1.points) < ransac_n: raise RuntimeError       #
Якщо залишилось замало точок, кидаємо помилку
                        m2, _ = rest1.segment_plane(dist_thr, ransac_n, n_iters)  #
Будуємо другу площину RANSAC
                        n1 = np.array(m1[:3]); n1 /= np.linalg.norm(n1)           #
Нормалізуємо нормаль m1
                        n2 = np.array(m2[:3]); n2 /= np.linalg.norm(n2)           #
Нормалізуємо нормаль m2
                        ax = np.cross(n1, n2); ax /= np.linalg.norm(ax)           #
Визначаємо третій напрям (вісь X)
                        ay = np.cross(n1, ax); ay /= np.linalg.norm(ay)           #
Вісь Y як перпендикулярна до n1 і ax
                        az = n1.copy()                                            #
Вісь Z = n1 (перший нормаль)
                        if np.dot(np.cross(ax, ay), az) < 0: az = -az             #
Перевіряємо правий/лівий набір векторів
                        R      = np.vstack((ax, ay, az)).T                        #
Формуємо матрицю обертання 3×3
                        coords = pts @ R       # Переносимо точки в локальні
координати пакунка
                        mn, mx = coords.min(axis=0), coords.max(axis=0)           #
Мінімум і максимум по кожній осі
                        dims   = mx - mn        # Розміри пакунка в локальних
координатах
                        center_l  = (mn + mx) / 2    # Центр пакунка в локальних
координатах
                        corners_l = np.array([     # Генеруємо 8 локальних вершин OBB
                            center_l + np.array([sx, sy, sz]) * (dims / 2)
                            for sx, sy, sz in product([-1, 1], repeat=3)
                        ])
                        corners = corners_l @ R.T                                  #
Переводимо локальні вершини назад у світові координати
                    except:
                        aabb    = pcd.get_axis_aligned_bounding_box()  # Якщо RANSAC
не спрацював, беремо Axis-Aligned BB
                        dims    = np.asarray(aabb.get_extent())        # Розміри AABB
                        corners = np.asarray(aabb.get_box_points())    # Вершини AABB
```

По переведеній в масив карті глибини створюємо масив точок з урахуванням внутрішніх параметрів матриці камери по формулам 2.7 та 2.8 для x та y відповідно. А після обрахунку всіх координат формуємо хмару точок та оновлюємо вікно Open3D для її відображення. В циклі try йде спроба сформувати орієнтовний прямокутний бокс (ОПБ) по створеній хмарі точок. Для цього використовується метод RANSAC. Це ітеративний метод оцінки математичної моделі з набору даних, що містить викиди. Алгоритм RANSAC працює шляхом ідентифікації викидів у наборі даних та оцінки бажаної моделі, використовуючи дані, що не містять викидів. RANSAC виконується за допомогою таких кроків: випадковий вибір підмножини

набору даних, підбір моделі до вибраної підмножини, визначення кількості викидів повторення попередніх кроків для заданої кількості ітерацій [39]. Якщо RANSAC метод не спрацьовує то використовується метод Axis-Aligned Bounding Box (AABB). Обмежувальна рамка, вирівняна по осях (AABB), — це просто прямокутний паралелепіпед, кожна грань якого перпендикулярна до одного з базисних векторів [40]. По отриманому ОПБ отримуємо розміри в метрах.

```
            # Малюємо 3D-коробку (LineSet) у Open3D
                    lines = [
                        (0,1),(1,3),(3,2),(2,0),
                        (4,5),(5,7),(7,6),(6,4),
                        (0,4),(1,5),(2,6),(3,7)
                    ]
                    ls = o3d.geometry.LineSet(
                        points=o3d.utility.Vector3dVector(corners),    # Встановлюємо
вершини
                        lines=o3d.utility.Vector2iVector(lines)         #
Встановлюємо сполучення вершин у лінії
                    )
                    ls.colors = o3d.utility.Vector3dVector([[1,0,0] for _ in lines])
# Задаємо колір ліній (червоний)
                    box_ls.points  = ls.points   # Оновлюємо колекцію точок ліній
                    box_ls.lines   = ls.lines    # Оновлюємо вектори ліній
                    box_ls.colors  = ls.colors   # Оновлюємо масив кольорів
                    vis.update_geometry(box_ls)  # Оновлюємо об’єкт LineSet у
візуалізаторі

                    if first_view:    # Якщо це перший кадр, налаштувати вигляд
камери
                        vis.reset_view_point(True)      # Скинути положення камери на
дефолтне
                        first_view = False     # Вимикаємо прапорець першого вигляду

                    # Проекція 3D-вершин у 2D-координати для відображення на
зображенні
                    img2d = bgr.copy()  # Копіюємо оригінальне зображення
                    proj2d = []         # Список для збереження 2D-координат проекцій
вершин
                    for i, j in lines:  # Проходимо по всіх ребрах куба
                        X1, Y1, Z1 = corners[i]  # Координати першої вершини ребра
                        X2, Y2, Z2 = corners[j]  # Координати другої вершини ребра
                        u1 = int((X1 * FX) / Z1 + CX); v1 = int(CY - (Y1 * FY) / Z1)
# Переклад у 2D-пікселі першої вершини
                        u2 = int((X2 * FX) / Z2 + CX); v2 = int(CY - (Y2 * FY) / Z2)
# І другої вершини
                        proj2d += [(u1, v1), (u2, v2)]  # Додаємо пари координат у
список
                        cv2.line(img2d, (u1, v1), (u2, v2), (255, 0, 255), 2)  #
Малюємо фіолетову лінію між двома точками

        # Малюємо ROI (прямокутник) на 2D-зображенні
```

```
                    cv2.rectangle(img2d, (ROI_X1, ROI_Y1), (ROI_X2, ROI_Y2), (255, 0,
0), 2)
                    cv2.imshow("RGB+3DBox", img2d)  # Показуємо зображення з 3D Box
та ROI
```

Далі створюється сет ліній які будуть з’єднувати вершини створеного раніше ОПБ у вікні Open3D щоб візуально можна було побачити наш паралелепіпед. Для цього створюється список ребер lines, де значення указують на пари індексів вершин прямокутного паралелепіпеда. Виглядає це наступним чином (рис. 4.1).

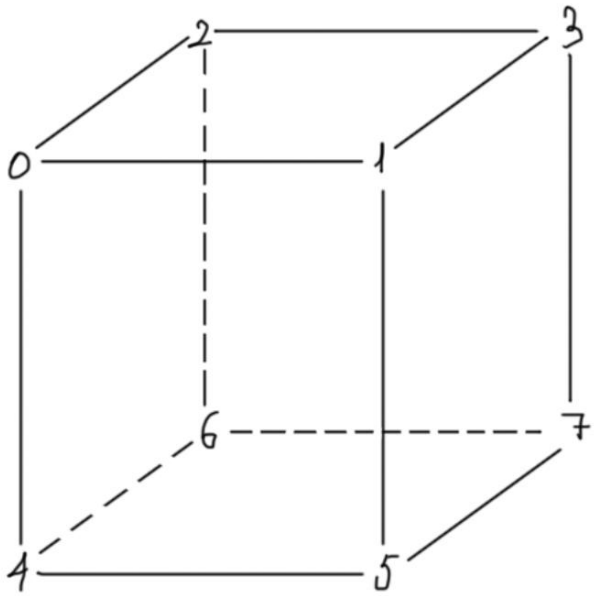

Рисунок 4.1 – Індекси вершин

Наступним кроком створюється об'єкт LineSet у вікні Open3D в якому ми беремо раніше отримані вершини (corners) і наш список ребер lines. Встановлюється червоний колір для ліній. Далі отриманий паралелепіпед з ліній проектуємо у вікно кольорового зображення. Для цього працюємо з копією зображення, оголошуємо масив, що буде списком для координат вершин у двовимірному просторі. Беручи всі координати вершин з масиву corners переводимо їх у 2D простір використовуючи вже відомі формули для проекції тривімірного зображення у двовимірне (2.5) та (2.6).

```
        in_roi = all(
                        ROI_X1 <= u <= ROI_X2 and ROI_Y1 <= v <= ROI_Y2
                        for (u, v) in proj2d
                    )

                    if in_roi:  # Якщо всі проекції у межах ROI
                        if current_id is None:
                            # Якщо ще не присвоєно ID цій коробці
                            current_id = next_box_id  # Присвоюємо новий унікальний
ID
                            next_box_id += 1          # Інкрементуємо лічильник
                            dims_list = []    # Очищаємо список розмірів для нової
коробки
                        dims_list.append(dims)  # Додаємо поточні виміряні розміри у
список
                    else:
                        if current_id is not None:  # Якщо коробка вже мала ID і
щойно вийшла з ROI
                            if dims_list:  # Якщо вдалось зібрати хоча б один набір
розмірів
                                mean_dims = np.mean(dims_list, axis=0)  # Обчислюємо
середні розміри вимірів
                                cm = mean_dims * 100       # Конвертуємо у сантиметри
                                vol_w = (cm[0] * cm[1] * cm[2]) / 5000.0 # Обчислюємо
об’ємну вагу (тариф 5000)
                                if 1 <= vol_w < 5:    decision = 1      # Рішення 1
для ваги [1,5)
                                elif 5 <= vol_w < 10: decision = 2      # Рішення 2
для ваги [5,10)
                                elif 10 <= vol_w < 50: decision = 3     # Рішення 3
для ваги [10,50)
                                else:                   decision = 4     # Рішення 4
для усього іншого
                                note = ""               # Пустий рядок нотатки
                            else:
                                # Якщо не вдалося виміряти жодного разу
                                cm = [0.0, 0.0, 0.0]   # Розміри 0
                                vol_w = 0.0            # Об’ємна вага 0
                                decision = 4           # Рішення 4 (невдалося
виміряти)
                                note = "MEASURE_FAIL"  # Нотатка про невдалий вимір
```

Перевіряємо присутність всіх вершин двовимірної проекції в середині зони інтересну. Якщо коробка повністю присутня, присвоюємо їй ідентифікаційний номер (ID), збільшуємо лічильник на 1 та присвоюємо їй її розміри. Коли коробка вже отримала свій номер, і вона вийшла з зони інтересу хоча б однією вершиною, йде обрахунок середньоарифметичного всіх отриманих розмірів поки коробка була в зоні інтересу, конвертується в сантиметри і обчислюється об’ємна вага по формулі (2.1). Отримана вага

порівнюється з еталонами і формується рішення щодо «напряму сортування». Перший напрям якщо об’ємна вага від 1 до 5 кг, другий – вага віж 5 до 10, третій – від 10 до 50.

```
        # Формуємо текстовий payload для надсилання в Unreal
                            payload = f"{current_id},{vol_w:.2f},{decision}"
                            hdr  =  struct.pack('<4sI',  b'RESL',  len(payload))    #
Пакуємо тег 'RESL' + довжину payload (uint32)
                            conn_cmd.sendall(hdr  +  payload.encode('utf-8'))      #
Відправляємо заголовок і payload у Unreal

                            import sys;
                            sys.stdout.flush()       #  Очищаємо  буфер  виводу  для
негайного друку
                            print("→ SENT TO UNREAL:", hdr, payload)  # Виводимо в
консоль для дебагу
                            sys.stdout.flush()                                # Очищаємо
буфер ще раз

                            # Логування результатів у файл
                            log_box(current_id, cm, vol_w, decision, note)

                            print(f"Logged     ID={current_id},     dims={cm}     cm,
vol_w={vol_w:.2f} kg, decision={decision}, {note}")
                            # Скидаємо змінні для наступної коробки
                            current_id = None
                            dims_list = []

                    vis.poll_events()    # Обробка подій у Open3D
                    vis.update_renderer()# Оновлення рендерингу у Open3D
                    if cv2.waitKey(1) == 27:  # Якщо натиснуто Esc у вікні OpenCV
                        break                # Виходимо з циклу

            finally:
                running = False               # Змінюємо флаг, щоб зупинити фоновий
потік
                conn.close()               # Закриваємо сокет з відео
                vis.destroy_window()       # Закриваємо вікно Open3D
                cv2.destroyAllWindows()    # Закриваємо вікно OpenCV
```

Фінальними діями є формування пакету даних з ID коробки, її об’ємною вагою та рішенням про сортування, та відправлення його назад у Unreal Engine. Також логуємо отримані результати у файл, і в консоль. Оновлюємо рендер у вікні Open3D і фіксуємо натиснення ESC у вікні OpenCV. При закінченні роботи закриваємо сокет і вікна. В результаті маємо скрипт який визначає розміри коробок, їх об’ємну вагу, формує пакет з

параметрами кожної коробки і відправляє їх назад в Unreal Engine. Приклад роботи скрипта показано на рисунку 4.2.

Похибка програмно-виміряних значень від реальних розмірів об’єкта виміряних в 3д редакторі складає ±1-3%, що є прийнятним результатом. Максимальний розмір який можна визначати це 60см×60см×60см.

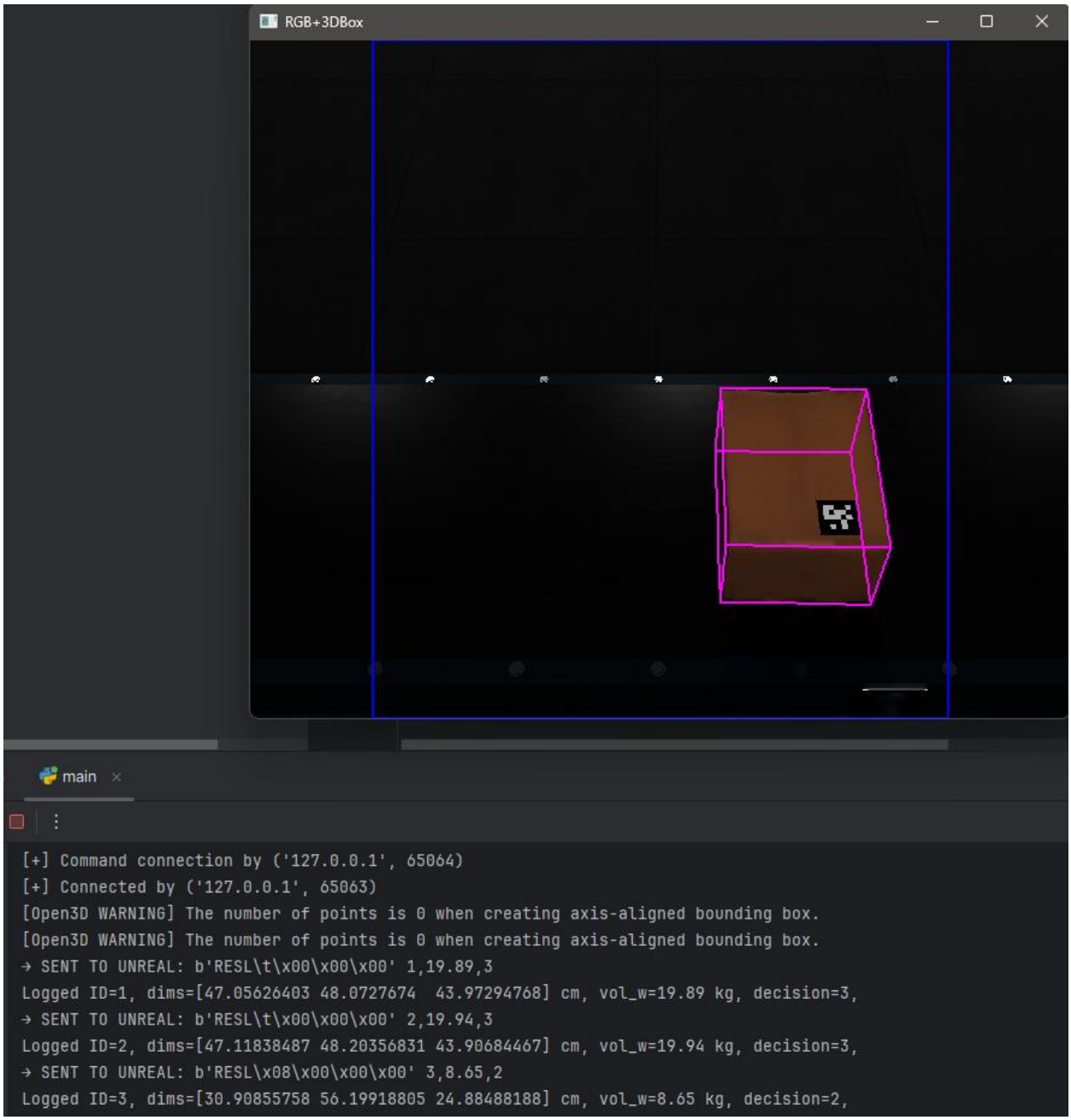

Рисунок 4.2 – Результат роботи

4.2 Розробленя логіки прийому команд та передача їх в сортувальний модуль

На даному етапі ми маємо команду яка приходить від сервера з Python скриптом але ніяк не оброблюється рушієм. Для цього створюємо два класи. Перший CommandRecieverRunnable (Додаток Б) який в фоновому потоці буде безперервно чекати повідомлення від сервера, декодувати повідомлення за тією ж логікою що і на стороні сервера, тобто читати заголовок який складається з тегу і розміру Payload і читати сам Payload. Розбиває отриманий рядок на ID коробки (BoxID), об’ємну вагу (VolW) і рішення про сортування (FlapIndex). Отримані данні передаються в переданий callback об’єкт у конструкторі і викликає його. Другий класс CommandRecieverComponent (Додаток Б) при старті викликає клас CommandRecieverRunnable і передає йому дані про сервер і callback об’єкт, який викликається не в робочу потоці класа Runnable, а за допомогою AsyncTask в головний потік рушія. Там викликається Blueprint-делегат OnCommandRecieved який надалі обробляється в Blueprint-і BP_ComandListener. Ці два класи отримують від Python команди щодо сортування і дають змогу далі працювати з Blueprints.

Далі логіка роботи буде наступною. Для кожної коробки створюються три параметри які відповідають ти що приходять від сервера, тобто BoxID, VolW (об'ємна вага) і Destination (надалі буде вважатись, що це пункт призначення виражений в номері конвеєра на який треба переправити коробку лопатковим сортувальником). В ці 3 параметри будуть записуватись значення що були отримані і декодовані в написаних раніше классах. Для цього створюється додатковий Blueprint, який називається BP_SortManager. В ньому буде 3 масиви які відповідають за 3 параметри для коробки і зона спрацьовування. В ці масиви будуть записуватись отриманні данні з BP_CommandListener. З цих масивів буде діставатися перший об’єкт і

назначатися на коробку яка попала в зону спрацьовування. Після цього цей же перший об’єкт буде видалятися з масивів, щоб не назначатися на наступну коробку в черзі.

Наступним кроком допрацьовуємо сцену. Для цього в конвєр додаємо логіку переміщення акторів по ньому. Створюємо Blueprint який буде відповідати за появу коробок на рівні. Робимо розгалуження в конвеєрній лінії які будуть рахуватись за пункти призначення (рішення про сортування) в параметрах коробок. Додаємо Blueprint для лопаткового сортувальника, в якому буде сам механізм сортувальника і зона спрацьовування яка буде сканувати у коробки її параметр призначення, і порівнювати його з тим який це конвеєр за рахунком і спрацьовувати, тим самим пересовувати коробку на потрібний конвеєр. Приклад роботи лопаткових сортувальників показано на рисунку 4.3.

Рисунок 4.3 – Робота лопаткових сортувальників

Для того щоб сортувальна лінія була спроможною сортувати велику кількість товарів та мати похибку сортування якомога меншу необхідно визначити оптимальні параметри неї, а саме: швидкість конвеєра, відстань між товарами, пропускну здатність в годину, та похибку спрацьовувань. Експериментальним методом було визначено, що оптимальними є наступні показники:

– швидкість конвеєра v: 3,5 м/с;

– відстань між товарами L: 2,8 м.

Знаючи ці показники можна розрахувати пропускну здатність лінії за годину:

$$T = \frac{L}{v}, \tag{4.1}$$

де T – це період одного товару.

Тобто T буде дорівнювати:

$$T = \frac{2,8}{3,5} = 0,8 \text{ с.}$$

Далі можемо розрахувати пропускну здатність за годину (3600 секунд):

$$T = \frac{t}{N}; \tag{4.2}$$

$$N = \frac{t}{T}; \tag{4.3}$$

$$N = \frac{3600}{0,8} = 4500 \text{ товарів.}$$

Отже пропускна здатність сортувальної лінії складає 4500 товарів за годину. За отриманих оптимальних показників середня похибка сортування не перевищила 1%.

4.3 Розрахунок показників стійкості системи

Розглянемо лінійну САУ яка буде складатися з конвеєра з постійною швидкість 3,5 м/с і лопаткового сортувальник який обертається з положення 0 у положення 1 за 0,2 секунди, і з положення 1 у положення 0 за 0,1 секунду. Передаточна функція для конвеєра буде:

$$W(s) = \frac{3,5}{s}. \tag{4.4}$$

А передаточна функція сортувальника буде мати наступних вигляд:

$$W(s) = \frac{1}{(0,2s+1)(0,1s+1)}. \tag{4.5}$$

Передаточна функція для розімкнутого контуру буде наступною:

$$W_{\text{роз}}(s) = \frac{3,5}{s} + \frac{1}{(0,2s+1)(0,1s+1)} = \frac{3.5}{s(0,2s+1)(0,1s+1)}. \tag{4.6}$$

Стійкість системи будемо перевіряти за критерієм Гурвіца. Для цього складаємо характеристичне рівняння системи:

$$0,02\lambda^3 + 0,3\lambda^2 + \lambda + 3,5 = 0. \tag{4.7}$$

За Гурвіцом для того, щоб система була стійкою, необхідно та достатньо, щоб усі визначники Гурвіца, складені з характеристичного рівняння виду $a_0\lambda^n + a_1\lambda^{n-1} + ... + a_n$, були додатніми $\Delta_i > 0$ [41].Тобто:

$$a_0 > 0, \Delta_1 = a_1 > 0, \Delta_2 = a_1a_2 - a_0a_3 > 0, \Delta_3 = a_3\Delta_2 > 0.$$

Розраховуємо визначники Гурвіца:

$$a_0 = 0{,}02 > 0;$$
$$\Delta_2 = a_1a_2 - a_0a_3 = 0{,}23 > 0;$$
$$\Delta_3 = a_3\Delta_2 = 0{,}23 > 0.$$

Отже за критерієм Гурвіца система є стійкою.

4.4 Охорона праці

Приміщення, в якому провоиться розробка запропонованого віртуального макету оснащенно комьтерною технікою.

Роботодавець повинен враховувати санітарні нормативи освітлення, вимоги до параметрів мікроклімату (температура, відносна вологість), ступеня і сили вібрації, звукового шуму і вогнестійкості приміщення, а також характеристики електромагнітного, ультрафіолетового та інфрачервоного полів. Так, наприклад, роботодавцю заборонено установлювати комп’ютери в приміщеннях, розташованих у підвалах будинків [42].

Для уникнення можливих аварій та замикань, поряд з приміщеннями, де вестиметься робота з комп’ютером (над чи під ними), також не дозволяється проведення робіт, що потребують здійснення надмірно вологих технологічних процесів. Відповідне приміщення повинно бути укомплектоване системами центрального або індивідуального опалення,

кондиціонування чи вентиляції повітря. Але при установці зазначених систем, необхідно переконатись, що батареї опалення, водопровідні труби, вентиляційні кабелі тощо, надійно сховані під захисними щитками, які перешкоджатимуть можливому потраплянню робітника під напругу [42].

Роботодавець, який використовує найману працю робітників, повинен забезпечити відповідність їхніх робочих місць комфортним та безпечним умовам. Розмір одного робочого місця має становити не менше 6 квадратних метрів. При необхідності, суміжні робочі місця співробітників, що працюють з комп'ютером, слід розділити перегородками висотою до 2 метрів. При визначенні достатнього розміру приміщення і робочого місця на одну особу необхідно додатково враховувати шафи, сейфи, тумби або інші предмети меблів чи обладнання, які знаходяться в кімнаті [42].

На столі працівника можливо розмістити допоміжні для роботи пристрої (принтери, колонки, сканери), а також місця для зберігання документів, за умови, що це не обмежуватиме видимість екрану і не заважатиме працівнику. У разі надмірного шуму чи вібрації технічного обладнання, роботодавець повинен забезпечити працівників антивібраційними килимками [42].

Робочий стілець співробітника має бути підйомно-поворотним, легко регульованим за висотою та забезпечувати належну підтримку та зручне положення спини і хребта особи Щодня необхідно проводити вологе прибирання приміщення, та очищати робоче місце та безпосередньо монітор комп'ютера від запиленості. На підприємстві забороняється: проводити ремонт та технічне обслуговування комп'ютера за робочим місцем працівника; самочинно ремонтувати або намагатись здійснити технічне налагодження комп'ютера без залучення компетентних спеціалістів; складувати на робочому місці зайві документи, деталі та предмети, що не потрібні для роботи; використовувати монітори з нечітким зображенням та монітори, у яких наявні поламки екрану [42].

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було розроблено віртуальний макет для симуляції та налаштування лінії розподілу товарів в внутрішньоскладських логістичних системах, шо дало можливість тестування та виключення помилок в робті ліннії та підвищити якість сортування. Проведена симуляція цифрового двійника лінії сортування показала, що при отриманих оптимальних показниках середня похибка сортування не перевищує 1%.

Під час виконання роботи було визначено, що перед впровадженням автоматизованої лінії розподілу товарів у внутрішньоскладські приміщення, для підвищення якості автоматизованого сортування необхідно проводити симуляцію виробничих параметрів розподілу товарів при сортуванні. Для розробки макету симуляції було проведено аналіз модулів сортувальної лінії, аналіз методів сортування, після чого було вирішено використовувати рушій Unreal Engine 5 та зовнішню логіку (сервер) на мові Python.

У роботі проведено розробка макету який симулює роботу конвеєра з товарами на ньому і лопатковим модулем сортування. Сортування проводиться за розрахованою по розмірам коробки товару об’ємною вагою. Для визначення розмірів використовується симульована ToF камера і зовнішня логіка обробки відепотоку з камери на мові Python.

У практичній частині наведено приклад створення сцени чернетки, наведені фрагменти коду з поясненями для відправки данних з рушія до серверу (зовнішньої логіки), а також логіку обробки відеопотку для визначення розмірів і подальшого розрахунку об’ємної ваги.

Завершальним етапом стало визначення та розрахунок параметрів для сортувальної лінії, а саме швидкість конвеєра, відстань між товарами, пропускну здатність та похибку сортування

Упровадження розробленого макету буде корисним для тестування та налагодження автоматизованих та автоматичних сортувальних ліній в логістичних системах, та при проведенні лабораторних та практичних занять для здобувачів першого (бакалаврського) рівня вищої освіти за освітньо-професійною програмою: «Автоматизація та комп'ютерно-інтегровані технології».

Матеріали за тематикою роботи було опубліковано у статті.

Проведені дослідження відповідають цілям сталого розвитку (ЦСР), а саме: ЦСР 4, ЦСР 9.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Vzhesnievskyi, M., and O. Chala. "Автоматизація внутрішньо-складських виробничих логістичних процесів для впровадження концепції INDUSTRY 4.0: енергоощадливість, продуктивність, мобільність, модульність, автономність." Системи управління, навігації та зв'язку. Збірник наукових праць 2.76 (2024): 34-38.

2. Vzhesnievskyi M. Automated Warehouse Logistics Systems Energy Efficiency Improvment / M. Vzhesnievskyi, O. Chala // Electromechanical and energy saving systems. - 2024. - № 4/2024 (67). - P. 18-26. - DOI : https://doi.org/10.32782/2072-2052.2024.4.67.2.

3. Nevliudov I., Maksymova S., Chala O., Bronnikov A. Vzhesnievskyi Maksym. Automated Logistics Processes Improvement in Logistics Facilities. Multidisciplinary Journal of Science and Technology. 2023. Vol. 3 (3). P. 157–170. URL: https://mjstjournal.com/index.php/mjst/ article/view/202.

4. М. Краснопьоров Автомазоматизація логістичних систем з використанням кіберфізичних підходів // Збірник студентських наукових статей «Автоматизація та приладобудування» «Automation and Development of Electronic Devices» ADED-2024 (Випуск 2) , Харків. 2024, С.: 186-190.

5. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. структура та правила оформлення. Введ. 2015-06-22. К. Держстандарт України, 2017. 29 с.

6. Положення про протидію академічному плагіату в ХНУРЕ [Електронний ресурс]: Наказ ХНУРЕ від 31.12.2024 р. No 386. Режим доступу:
https://nure.ua/wpcontent/uploads/Main_Docs_NURE/386_31.12.2024.pdf

7. Методичні вказівки з підготовки кваліфікаційної роботи для здобувачів першого (бакалаврського) рівня вищої освіти денної і заочної

форми навчання спеціальності 151 – Автоматизація та комп’ютерно-інтегровані технології освітньої [Електронний ресурс] / упоряд.: І. Ш. Невлюдов, О. І. Филипенко, О. В. Токарєва, С. П. Новоселов, О.В Сичова ; М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – електрон. вид. – Харків : ХНУРЕ, 2023. – 760 Kb.

8. Навчальний посібник з підготовки кваліфікаційної роботи бакалавра для здобувачів вищої освіти денної і заочної форм навчання спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» освітньої програми «Автоматизація та комп’ютерно-інтегровані технології» : Навчальний посібник / І. Невлюдов, О. Филипенко, О. Токарєва, С. Новоселов, О. Сичова. Харків : Видавництво Іванченка І. С., 2023. – 151 с.

9. United Nations. The 17 Sustainable Development Goals. United Nations, 2025, Режим доступу: sdgs.un.org/goals (дата звернення: 12.04.2025).

10. Increased efficiency in parcel sorting. Materialfluss. URL: http://www.materialfluss.de/conveyor-and-lifting-technology/increased-efficiency-in-parcel-sorting.htm (date of access: 21.04.2025).

11. Brown T. Choosing a singulation method. Parcel. URL: https://parcelindustry.com/article-4901-Choosing-a-Singulation-Method.html (date of access: 21.04.2025).

12. Стрічковий конвеєр 1,5 метра. СПЕЦМАШ - завод специализированного машиностроения в Украине. URL: https://zavodspetsmash.com/ua/p1129552710-lentochnyj-konvejer-metra.html (дата звернення: 23.04.2025).

13. Конвеєри: види, класифікація та сфера застосування, в блозі SSK.UA. Sklad Service | Комплексные решения для склада. URL: https://ssk.ua/ua/blog/konvejery-vidy-klassifikaciya-i-sfera-primeneniya-482 (дата звернення: 21.04.2025).

14. Standard Gravity Roller Conveyor at 50000.00 INR in New Delhi | V S R Systems. Tradeindia. URL: https://www.tradeindia.com/products/standard-gravity-roller-conveyor-c10623018.html (date of access: 21.04.2025).

15. Типи, категорії та галузі застосування конвеєрів. TEN 24. URL: https://ten24.com.ua/ua/blog/tipy-kategorii-i-oblasti-primeneniya-konveyerov/ (дата звернення: 22.04.2025).

16. Cross-belt Sorter-The Trustworthy Provider of intelligent logistics equipment and solution. The Trustworthy Provider of intelligent logistics equipment and solution. URL: http://www.damon-group.com/intelligent-sorting-system-15 (date of access: 23.04.2025).

17. McGuire P. M. Conveyors: application, selection, and integration. Taylor & Francis Group, 2009. 210 p.

18. BG SORTER ET HOCHLEISTUNGSSORTIERUNG AUF KIPPSCHALEN. BEUMER GROUP. URL: https://www.beumergroup.com/app/uploads/2019/03/BEUMER-BG-Sorter-ET-DE.pdf (date of access: 22.04.2025).

19. Paddle & Pusher Sorters. Bastian Solutions a Toyoyta automated logistics company. URL: https://www.bastiansolutions.com/solutions/technology/sortation/conveyor/paddle-sorters-pusher-sorters/ (date of access: 23.04.2025).

20. Pivot Wheel Sorter. Bastian Solutions a Toyoyta automated logistics company. URL: https://www.bastiansolutions.com/solutions/technology/sortation/conveyor/pivot-wheel-sorter/ (date of access: 22.04.2025).

21. What is Volumetric Weight? | ParcelBroker. Cheap Parcel Delivery: UK & International | Compare Courier Prices. URL: https://parcelbroker.co.uk/help/what-is-volumetric-weight-volume-calculator/#:~:text=Volumetric%20weight%20refers%20to%20the,use%20a%20divisor%20of%204000). (date of access: 26.04.2025).

22. News - what is TOF camera? And how it works?. https://www.hampotech.com/. URL: https://www.hampotech.com/news/what-is-tof-camera-and-how-it-works/ (date of access: 14.05.2025).

23. Unreal Engine 5.5 Documentation. Epic Games Unreal Engine 5.5. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation (date of access: 14.05.2025).

24. Zisserman A., Hartley R. Multiple View Geometry in Computer Vision. 2nd ed. Cambridge University Press, 2004. 672 p.

25. What is camera calibration? - MATLAB & simulink. MathWorks - Maker of MATLAB and Simulink - MATLAB & Simulink. URL: https://www.mathworks.com/help/vision/ug/camera-calibration.html (date of access: 14.05.2025).

26. RK M. CV-17 (Depth Image to Point Cloud). Medium. URL: https://medium.com/@monishatemp20/cv-17-depth-image-to-pointcloud-d83247a5d8e8 (date of access: 14.05.2025).

27. Вступ до OpenCV. Комп'ютерний зір – IT Master - електроніка та програмування. Головна – IT Master - електроніка та програмування. URL: https://itmaster.biz.ua/programming/vision/opencv.html (дата звернення: 14.05.2025).

28. OpenCV: OpenCV-Python Tutorials. OpenCV documentation index. URL: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html (date of access: 14.05.2025).

29. Open3D – A Modern Library for 3D Data Processing. Open3D – A Modern Library for 3D Data Processing. URL: https://www.open3d.org/ (date of access: 14.05.2025).

30. Sketchfab - The best 3D viewer on the web. Sketchfab. URL: https://sketchfab.com/ (date of access: 16.05.2025).

31. Quixel Megascans. Quixel Megascans. URL: https://quixel.com/megascans/ (date of access: 16.05.2025).

32. What's the difference between UClass and UObject?. GameDev StackExchange. URL: https://gamedev.stackexchange.com/questions/161783/whats-the-difference-between-uclass-and-uobject/ (date of access: 01.06.2025).

33. Рефлексія в програмуванні: що це таке? FoxmindED. FoxmindEd. URL: https://foxminded.ua/refleksiia-v-prohramuvanni/ (дата звернення: 01.06.2025).

34. Topics tagged unreal-engine & question. Epic Developer Community Forums. URL: https://forums.unrealengine.com/tags/intersection/unreal-engine/question (date of access: 02.06.2025).

35. Creating the first function in C++. Romero Blueprints. URL: https://romeroblueprints.blogspot.com/2020/08/creating-first-function-in-c.html (date of access: 02.06.2025).

36. The UPROPERTY() macro. Romero Blueprints. URL: https://romeroblueprints.blogspot.com/2020/10/the-uproperty-macro.html (date of access: 02.06.2025).

37. Що таке атомарна та неатомна змінна?. Каталог спеціаліста. URL: https://brend.liberty.cx.ua/maysternist/shho-take-atomarna-ta-neatomna-zminna.html (дата звернення: 02.06.2025).

38. Welcome to Python.org. Python.org. URL: https://www.python.org/ (date of access: 11.06.2025).

39. What Is RANSAC?. MathWorks. URL: https://www.mathworks.com/discovery/ransac.html (date of access: 03.06.2025).

40. Aligned bounding box. ScienceDirect. URL: https://www.sciencedirect.com/topics/computer-science/aligned-bounding-box (date of access: 03.06.2025).

41. Теорія автоматичного управління (збірник задач): навчальний посібник / І. Ш. Невлюдов, О. В. Токарєва. Харків: ХНУРЕ, 2020. 240 с. 35. Perišić, N., & Jovanović, R. (2023). Control of direct current motor by using

artificial neural networks in Internal model control scheme. FME Transactions, 51(1), 109-116.

42. Охорона праці при роботі з комп'ютером. Компанія Вікторія. URL: https://www.victorija.ua/dovidnik/osnovni-pravyla-dotrymannya-ohorony-pratsi-pry-roboti-na-personalnyh-eom.html (дата звернення: 08.06.2025).