Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеню вищої освіти – магістр
за освітньо-професійною програмою
«Кіберфізичні системи в промисловості, бізнесі та транспорті»

зі спеціальності
174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка

тема роботи:

«*Розробка цифрового двійника комплексу моделювання транспортно-розвантажувальних операцій з використанням рушіїв візуалізації*»

Виконав ст. гр. АКІТР-23-1м. _____ Куксенко Р.Є.

Керівник _____ Рубан С.А.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2024

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Магістр*

Спеціальність: *174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка*

ЗАТВЕРДЖУЮ

Зав. кафедри: *к.т.н. Рубан С.А.*

_____

« __»_____2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи АКІТР-23-1м. *Куксенку Романові Євгоновичу*

1. Тема кваліфікаційної роботи: *«Розробка цифрового двійника комплексу моделювання транспортно-розвантажувальних операцій з використанням рушіїв візуалізації»*

затверджено наказом по університету № *595с* від *04.07.2024 р.*

2. Термін здачі кваліфікаційної роботи: *01.12.2024 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 101с., додатки, презентація у Microsoft PowerPoint (12 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-3 зав. Кафедри Рубан С.А.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*10.07.24*
2	*Розділ 1*	*15.07.24*
3	*Розділ 2*	*18.08.24*
4	*Розділ 3*	*19.09.24*
5	*Висновки*	*15.10.24*
6	*Оформлення кваліфікаційної роботи*	*20.11.24*
7	*Підготовка презентації та графічного матеріалу*	*28.11.24*
8	*Підготовка доповіді до захисту*	*01.12.24*

6. Дата видачі завдання: *28.06.2024р.*

Керівник _____ /Рубан С.А./

7. Запевнення: *Я, Куксенко Роман Євгенович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Здобувач _____ / Куксенко Р.Є./

АНОТАЦІЯ

Куксенко Р.Є. «Розробка цифрового двійника комплексу моделювання транспортно-розвантажувальних операцій з використанням рушіїв візуалізації».

Кваліфікаційна робота на здобуття ступеню вищої освіти магістр за освітньо-професійною програмою «Кіберфізичні системи в промисловості, бізнесі та транспорті» зі спеціальності 174 – Автоматизація, комп'ютерно – інтегровані технології та робототехніка– Криворізький національний університет, Кривий Ріг, 2024.

Об'єктом дослідження є комплекс моделювання транспортно-розвантажувальних операцій та створення його цифрового двійника.

У першому розділі було описано технологічний процес, проаналізовано принципи створення цифрових двійників та їх класифікацію, переваги та недоліки, можливі технології та патерни реалізації.

У другому розділі було розроблено структуру та модель системи, налаштовано інтеграцію компонентів, створено та описано схеми інформаційних потоків.

У третьому розділі було програмно реалізовано цифрового двійника системи за допомогою рушія візуалізації Unreal Engine 5, WebSocket, OPC UA, Node Red та TIA Portal.

Ключові слова:

ЦИФРОВИЙ ДВІЙНИК, SCARA, ТРАНСПОРТНО-РОЗВАНТАЖУВАЛЬНІ ОПЕРАЦІЇ, РУШІЙ ВІЗУАЛІЗАЦІЇ, ІГРОВИЙ РУШІЙ, UNREAL ENGINE, NODE RED, OPC UA, WEBSOCKET.

ANNOTATION

Kuksenko R.E. «Development of a digital twin of a complex for modelling transport and handling operations using visualisation engines».

Graduation master`s work for obtaining an educational degree «Master» for the educational and professional program « Cyber-physical systems in industry, business and transport » in specialty 174 – «Automation, computer-integrated technologies, and robotics». – Kryvyi Rih National University, Kryvyi Rih, 2024.

The object of research is a complex for modelling transport and handling operations and creating its digital twin.

In the first section, the technological process was described, the principles of creating digital twins were analyzed and their classification, advantages and disadvantages, possible technologies and implementation patterns.

In the second section, the structure and model of the system were developed, the integration of components was set up, and information flow diagrams were created and described.

In the third section, a digital twin of the system was programmatically implemented using the Unreal Engine 5 visualisation engine, WebSocket, OPC UA, Node Red, and TIA Portal.

Keywords:

DIGITAL TWIN, SCARA, HANDLING OPERATIONS, VISUALISATION ENGINE, GAME ENGINE, UNREAL ENGINE, NODE RED, OPC UA, WEBSOCKET.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ЦД — цифровий двійник.

Unreal Engine 5 – UE5.

Reference Architectural Model Industry 4.0 – RAMI 4.0.

Six-Layer Architecture for Digital Twins with Aggregation – SLADTA.

ВСТУП

Важливим етапом четвертої промислової революції стало впровадження цифрових двійників у виробничі процеси. Дане нововведення змінило те, як промисловість керує, оптимізує та модифікує їх процеси. Цифрові двійники є частиною галузі Інтернета Речей та автоматизації, та виконують дуже важливу роль, оскільки вони пов'язують цифрову та фізичну версію реальних промислових об'єктів.

Цифровий двійник – це віртуальна версія фізичного об'єкту або системи. Вони у реальному часі відображають, аналізують та оптимізують процес, та, на відміну від симуляцій, отримують дані від реальної моделі, та візуалізують його за допомогою сучасних двигунів візуалізації.

Метою даної кваліфікаційної роботи та майбутньої магістерської роботи є реалізація цифрового двійника реальної моделі транспортно-розвантажувальних операцій на базі SCARA-робота, за допомогою двигуна візуалізації та хмарних технологій. Це дозволить відстежувати стан моделі у реальному часі, аналізувати швидкість її роботи, віддавати команди керування одразу з двигуна візуалізації, оптимізувати процес.

Також впровадження цифрового близнюка дозволить у майбутньому проєктувати модифікації до моделі віртуально та переглядати можливі недоліки майбутньої системи на основі отриманих даних у реальному часі.

На даній кваліфікаційній роботі будуть розглянуті призначення та цілі створення системи, а після цього реалізовані її окремі компоненти, розроблені прототипи візуального інтерфейсу керування цифровим двійником у двигуні візуалізації, а також розроблені прототипи відображення даних у реальному часі, написана інструкція користувача системи.

РОЗДІЛ 1
ХАРАКТЕРИСТИКА ТЕХНОЛОГІЧНОГО ПРОЦЕСУ, ЙОГО АНАЛІЗ ТА ОГЛЯД ПОВ'ЯЗАНИХ З ТЕМОЮ РОЗРОБОК

1.1 Загальні відомості про технологічний процес, опис моделі транспортно-розвантажувальних операцій та цифрового двійника системи

Технічний процес загалом представляє собою реалізовану модель транспортно-розвантажувальних операцій на базі SCARA-робота (рис 1.1). Вона реалізована на базі мікроконтролера S7-1200 (рис 1.2), який живить модуль PM1207 та мікропроцесора CPU 1215c.

У якості блока керування, силового модуля та фізичної панелі управління використовується SINAMICS G120.

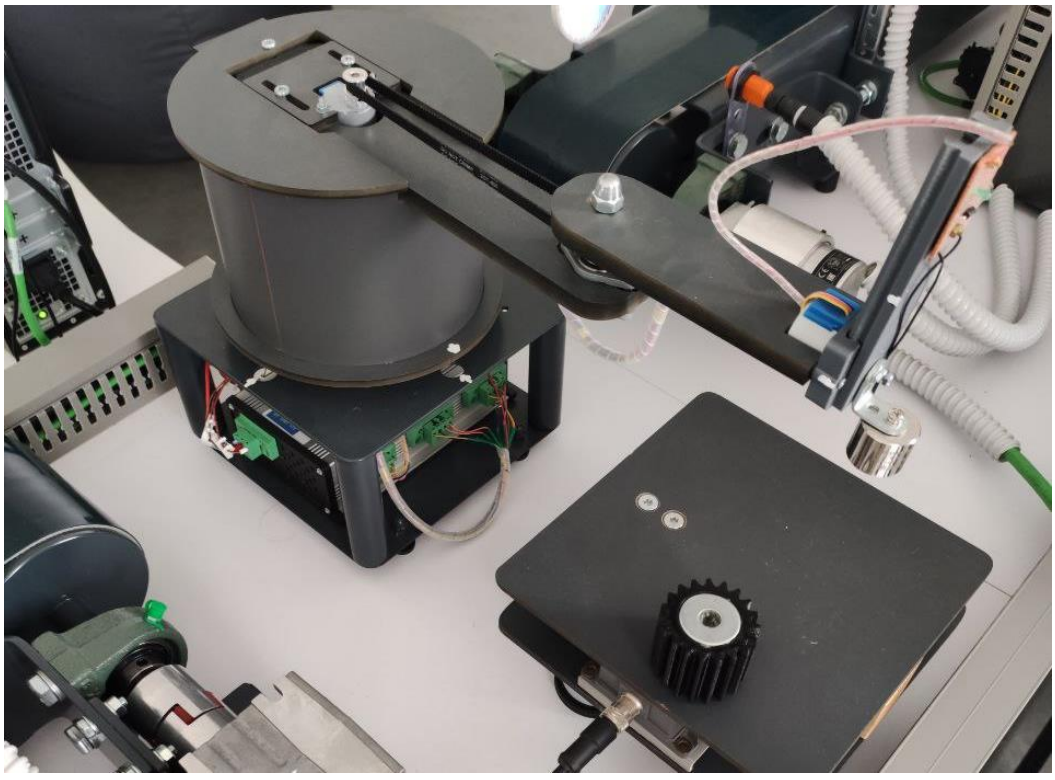


Рисунок 1.1 – Модель транспортно-розвантажувальних операцій на базі SCARA-робота

У складі моделі є ваговий датчик на основі тензометра (яким керує ваговий модуль SIWAREX WP231) , який зважує розміщений на нього вантаж та передає необхідні дані на ПК оператора стенду.

Після цього він надає команду активації SCARA-руки (на базі крокового двигуна 28BJ48-12-300-01 та драйвера TB6560 V2), яка розміщується у стандартне положення, підбирає об’єкт, та розміщує його на стрічку конвеєра.

Об’єкт, розміщений на конвеєрі, перетинає перший лазерний датчик початку шляху, переміщується на максимальну відстань та перетинає другий, останній лазерний датчик, який сигналізує про завершення транспортування.

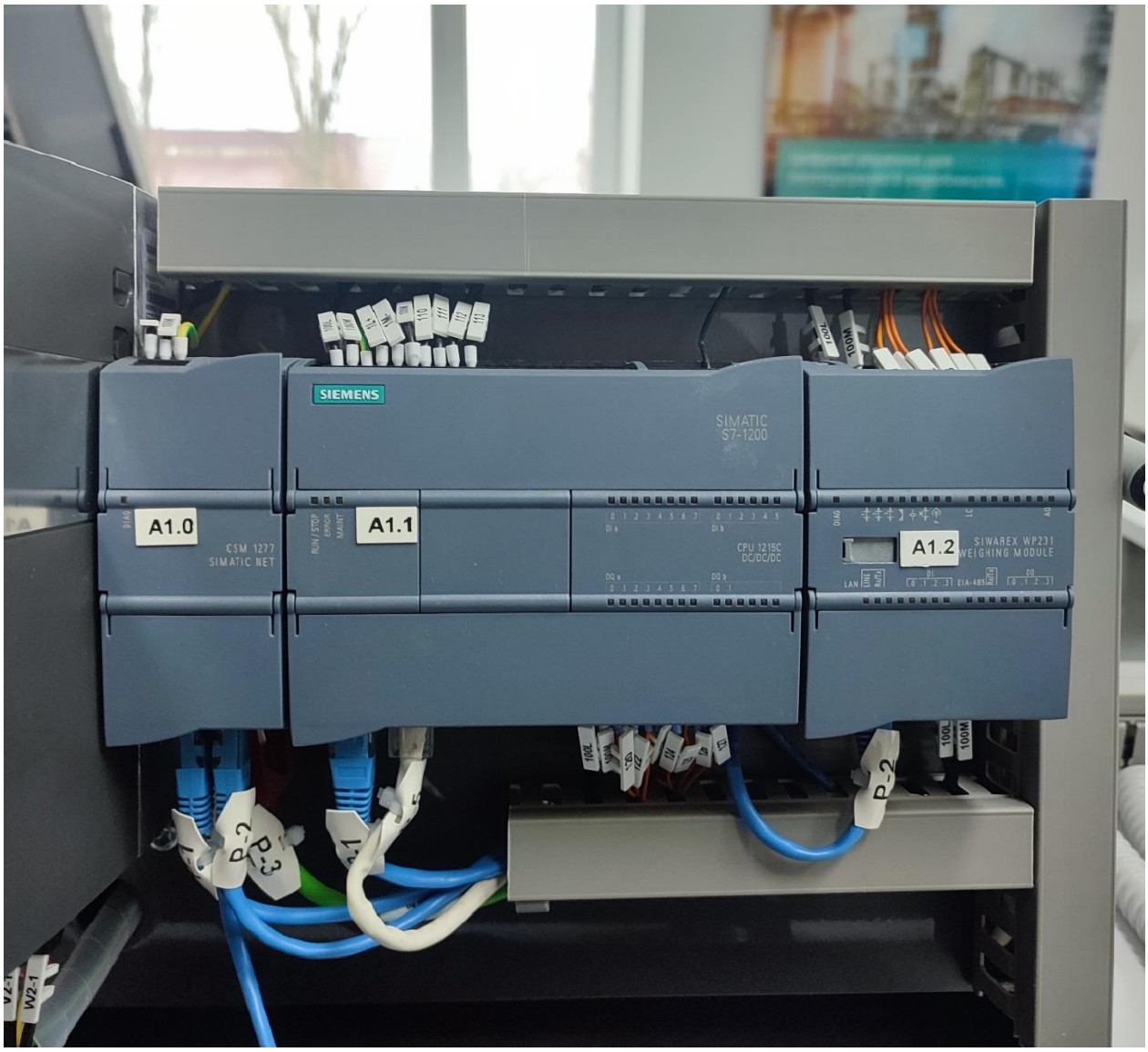


Рисунок 1.2 – Мікроконтролер S7-1200 та модулі Simatic CSM1277 для налаштування зв’язку та WP231 вагового модуля

Рух по конвеєру забезпечується за допомогою трифазного двигуна Siemens 1AV1062B (рис. 1.3), який має максимальний запас навантаження у 5 кг (відповідно, на одній конвеєрній стрічці можуть бути деталі загальною масою у п’ять кілограм. Враховуючи швидкість та максимальну масу підйому SCARA-руки, фактично ліміт маси конвеєра нівелюється).

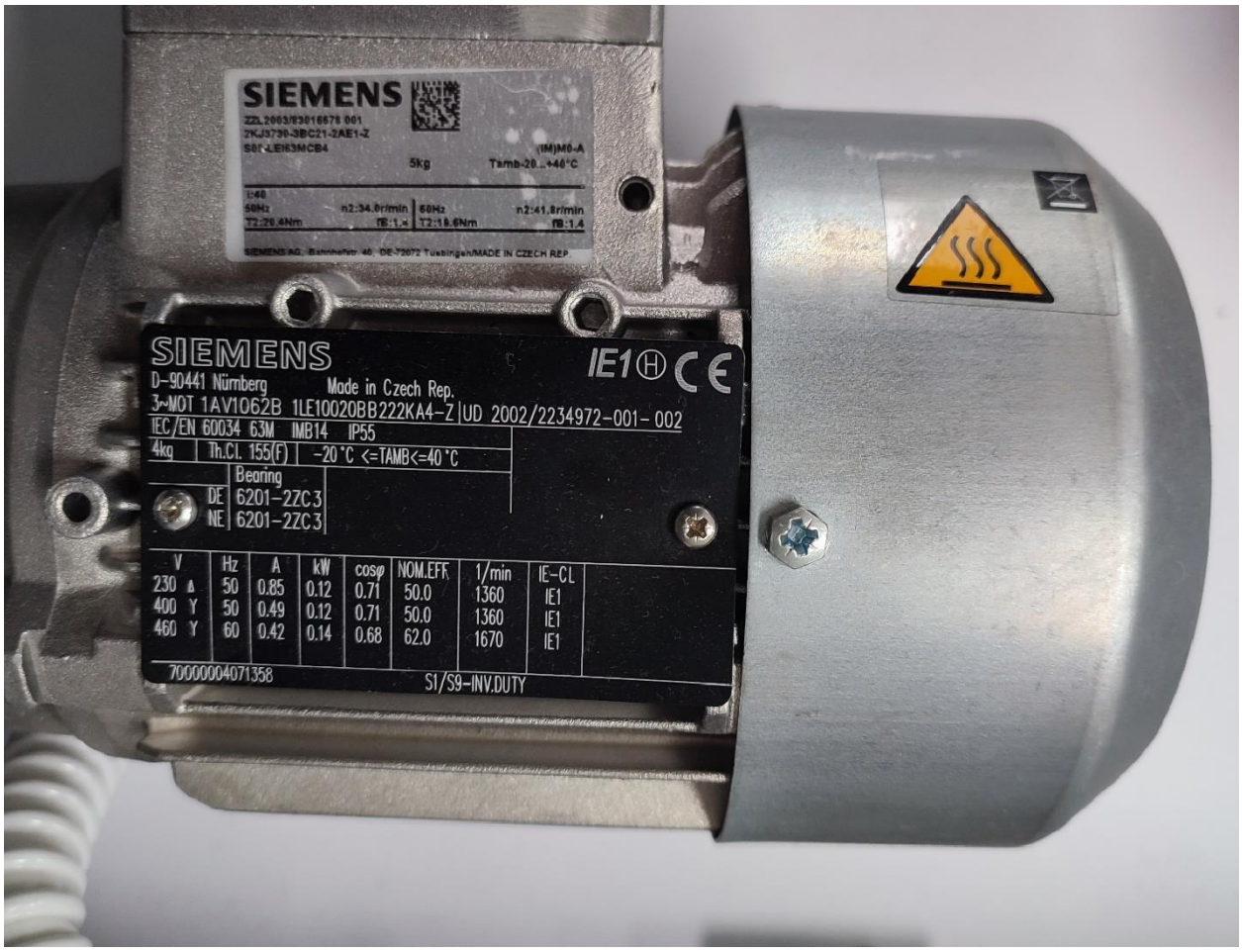


Рисунок 1.3 – Трифазний двигун Siemens 1AV1062B,
який забезпечує рух конвеєра

Окрім цього, можливе додання функціоналу камери, яка візуально ідентифікує декілька об’єктів на початковій платформі. Програма обирає найбільший об’єкт та піднімає саме його. Альтернативна модель роботи — розміщення камери у кінці конвеєра, та передача найбільшого об’єкту вже на наступний конвеєр.

Основною задачею при створенні цифрового двійника системи є синхронізація реального об’єкту та віртуальної моделі. Для цього хмарні сервіси передаються основні змінні об’єкту, а саме: положення SCARA-руки, дані з вагового датчику, лазерні датчики на конвеєрній стрічці, дані з камери що ідентифікує розмір об’єктів.

Для створення повноцінного цифрового двійника можна за допомогою сканувальних AR-технологій створити 3D-моделі частин реальної системи. Як альтернатива, подібні моделі реальної системи можна створити, використовуючи інструменти 3Д-моделювання, такі як Maya 3D або, більш вірогідно, Blender. Оскільки модель SCARA-руки має фіксовану кількість осей обернення та ширину і довжину деталей, використання 3д-моделей для неї не є можливим. Втім, заздалегідь завантажені 3Д-моделі можна використовувати для об’єктів конвеєра (за умови підгону їх параметрів розміру), конвеєра та інших деталей.

Отримані моделі необхідно імпортувати у ігровий рушій, у якому буде реалізована візуальна частина двійника, а також інтерфейс керування. Поєднання двигуна з TIA Portal v18, на базі якого створено програмне забезпечення для роботи моделі, відбувається за рахунок використання хмарних сервісів (на кшталт Microsoft Azure) та мережевих протоколів передачі даних (наприклад MQTT).

Кодування функціоналу ігрового рушія буде відбуватися за рахунок комбінації мов програмування Blueprints та C++. Кодування функціоналу у TIA Portal відбувається за рахунок поєднання FBD, LAD та SCL. Не виключається необхідність програмування на мові блоків Node Red для забезпечення зв’язку стенду та ігрового двигуна візуалізації.

Отже, є реальна модель транспортно-розвантажувальних операцій на базі SCARA-робота. Для неї необхідно створити цифрового двійника за допомогою поєднання технологій ігрових рушіїв, різних мов програмування, протоколів передачі даних та програмного забезпечення контролю над моделлю.

1.2 Загальні відомості про напрямки пов’язані з технологічним процессом та цифровими двійниками

Що ж по суті є цифровим двійником (ЦД)? Вони є віртуальною версією реального об’єкта, яка сама по собі слугує мостом між фізичним та цифровим світом. Завдяки цьому здійснюється моніторинг, а також аналіз з моделюванням реальної системи.

Об’єктом ЦД може бути що завгодно: автомобіль, міська інфраструктура, корабель, розвідний міст, реактивний двигун.

Але, на відміну від симуляції, UI-візуалізації чи простої 3D-моделі, цифрові двійники містять у собі динамічні дані, що відображають поточний стан та поведінку фізичного об'єкта. Ці дані потім використовуються для прогнозування та аналізу ситуацій, де фізичний аналог може некоректно зреагувати на різні чинники та фактори [1].

ЦД відображають корисні дані (як візуальні, так і параметричні), які можуть бути корисними у будь-яких галузях: дизайні об'єктів, архітектурі, інженерінгу, медицині [2]. Завдяки цьому можна виявити певні закономірності, які за допомогою звичайного аналізу можуть бути не помітними, і завдяки цьому підвищити ефективність реального об’єкту [3].

Основними типами цифрових двійників є:

– Продукт. Тобто віртуальна модель фізичного продукту з деталями дизайну, характеристиками, матеріалами та компонентами. Може використовуватися для валідації, тестування та моделювання поведінки реального продукту за різних умов;

– Процес. Цифрова модель даного типу моделює та оптимізує виробничі або операційні процеси. Також допомагає визначати вузькі місця та прогнозувати результати;

– Ефективність. Цей тип займається моніторингом та аналізує дані роботи систем або обладнання в реальному часі. Такі дані допомагають

оцінити стан, продуктивність та ефективність цих систем, забезпечуючи прогнозоване технічне обслуговування та мінімізуючи простої;

– Система. Моделі великих та складних систем (система кондиціонування будівлі, завод тощо). Інтегрує дані з різних джерел для моделювання та оптимізації поведінки, продуктивності та ефективності системи;

– Місто. Модель всієї міської інфраструктури (будівлі, транспорт, комунікації тощо). Може використовуватися міськими планувальниками для моделювання та аналізу різних сценаріїв розвитку міста, сталого розвитку та розподілу ресурсів [4, 5].

Іноді ЦД дозволяють симулювати умови реальної роботи для перевірки системи на наявність проблем (але для цього тоді необхідна наявність заздалегідь прорахованої наближеної моделі пристрою) [6].

Але вони призначені не лише для використання людьми. Часто технологія ЦД використовується як пісочниці для імітації різних взаємодій з об’єктами, або навіть здатні відтворювати реальні процеси взаємодії користувачів та роботів перед тим, як ця система буде фактично інтегрована у реальне середовище підприємства.

Можна сказати і про те, що цифрові двійники — це дуже новітня технологія, яка почала впроваджуватися у компаніях виробництва, охорони здоров’я, архітектури, інженерії, будівництва, менеджменту міст, логістики лише у кінці 2010-х років. У США, наприклад, ринок цифрових двійників зараз оцінюється у 3 мільярди доларів, але до 2030 очікується зростання ринку більш ніж у сорок разів (до приблизно 140 мільярдів доларів). Отже, можна зробити висновок, що галузь дуже пріоритетна [7].

На прикладі вище зазначених галузей можна зазначити, що напрям майбутнього цифрового двійника транспортно-розвантажувальних операцій – це поєднання інженерії та логістики з можливими модифікаціями логістики (наприклад, віртуальне додавання додаткових конвеєрів).

Можна побачити тенденцію останніх років щодо зростання акценту міжнародних компаній на IoT та, передусім, цифрових двійниках. Серед таких компаній: SAP SE, Siemens, ANSYS, Hitachi, Microsoft, Autodesk, IBM, General Electric, Schneider Electric, Airbus.

Останні роки показують все більше і більше нових розробок і дій у даній сфері. У 2022 році NVIDIA запустила платформу для розробки цифрових двійників за допомогою методів машинного навчання. У тому ж році L&T разом з Microsoft та Bentley розгорнула нові практичні матеріали наступного покоління по новому поколінню Інтернету Речей, основою якого стануть саме цифрові двійники промислового та виробничого секторів.

Загалом, протягом останніх трьох років вищезазначені компанії або розробляють, або вже випустили інструменти для створення цифрових двійників.

Важливо зазначити переваги створення цифрових двійників, тобто причини, чому вони отримують все більше уваги від світових корпорацій, а також основні виклики та складні моменти при їх розробці.

Переваги щодо створення цифрового двійника:

– Покращення процесу прийняття рішень;
– Ефективність та можливість оптимізації існуючих процесів;
– Спрощування розробки та модифікації продуктів;
– Синхронізація різних команд продукту;
– Уникнення вірогідності поломок завдяки заздалегідь зпрогнозованому обслуговуванню;
– Зниження ризиків незапланованих сценаріїв;
– Економія витрат на продукт, ефективність використання ресурсів команди розробки;
– Навчання майбутніх робітників на існуючій віртуальній моделі;
– Віддалений моніторинг та візуальне керування системою.

Але для досягнення переваг вище необхідно подолати певну кількість викликів, а саме:

– Необхідно забезпечити безперебійний та надійний потік даних з та в систему;

– Необхідно коректно змоделювати модель, щоб вона могла працювати навіть з мінімальним потоком даних (або взагалі без нього);

– Іноді впровадження цифрових двійників може коштувати великих сум (наприклад, впровадження цифрових двійників літаків), втім внесок даної системи у оптимізацію процесу значно вищий;

– Впровадження цифрових двійників безповоротно змінює менеджмент процесів, і на це теж треба звернути увагу;

– Створення даної моделі потребує гарних знань у сферах візуалізації даних, аналізу даних, симуляції, моделюванні, програмуванні.

Система ЦД у будь-якій галузі має у собі три основних компоненти.

Перший – це фізичний пристрій, об’єкт, система, яку відображує двійник. Це може бути невеликий стенд в університеті, робот, автомобіль, або ціле місто.

Другий – це віртуальний двійник фізичного пристрою, створений за допомогою 3д-моделювання, двигунів візуалізації, програмування та даних.

Третій, хіба що не головний компонент для коректної роботи, міст між фізичним та цифровим пристроєм. Це відбувається за рахунок обміну даних за різними протоколами, датчиками, IoT-девайсами та потоками даних, які передають інформацію у дві сторони, надаючи дані цифровому двійнику або команди реальному об’єкту.

Який базовий принцип роботи цифрових двійників?

1. За допомогою датчиків та IoT-девайсів отримаються дані про об’єкт, у нашому випадку SCARA-роботу та конвеєра це, наприклад, положення руки-контролера, показники ваги об’єкта, його габаритів, інформація що об’єкт розташовано на стрічку, або що він повністю транспортований.

2. Передача даних у реальному часі, з постійним оновленням. Відбувається за рахунок різних протоколів передачі, наприклад MQTT.

3. Аналіз, обробка отриманої інформації, відображення трендів і графіків.

4. Симуляція та візуалізація отриманих даних.

5. Предиктивна поведінка, тобто окремий інтерфейсний елемент, який на основі деяких даних може попереджати про можливу несправність у системі.

6. Віддалений моніторинг системи оператором та надання команд управлінні. Якщо це певний об’єкт управління, то оператору не треба навіть знаходитися у приміщенні, завдяки хмарним технологіям керування може відбуватись будь-де.

Отже, цифрові двійники є віртуальною репрезентацією реального технологічного об’єкту. Вони діляться на декілька категорій, усі з яких дають суттєві переваги підприємствам, а саме зниження ризиків, витрат, покращення та автоматизація процесів, дозволяють віддалено здійснювати моніторинг виробничих процесів. Напрямок ЦД є дуже перспективним та визнається провідними компаніями світу як майбутнє IoT.

1.3 Аналіз інформації щодо конкретної реалізації цифрових двійників та наявних технологій IoT

Створення цифрового двійника – це дуже складна задача, яка поєднує у собі програмування, моделювання, симулювання, візуалізацію, створення інтерфейсу, налаштування потоку даних, налаштування фізичного об’єкту та оптимізацію.

Для того, щоб зрозуміти, які технології необхідні для створення подібної системи, необхідно знову поглянути на базовий принцип роботи цифрових двійників та проаналізувати можливі рішення на ринку технологій.

Для того, щоб створити потік даних, необхідно обрати протоколи передачі даних у хмару. Для цього є декілька варіантів: MQTT, OPC UA, AMQP, HTTP, CoAP [8].

Перевагами MQTT є те, що це дуже ефективний та простий протокол передачі даних, який є ідеальним для IoT та M2M комунікацій. Він підтримується більшістю програм та дуже рідко потребує встановлення сторонніх плагінів для інтеграції. Єдиний мінус – це відсутність вбудованих механізмів захисту даних, але це питання вирішується поєднанням з іншими протоколами.

OPC UA це потужний стандарт взаємодії різних систем, підтримує різні протоколи безпеки та шифрування, але може бути складним у реалізації.

Використання HTTP для IoT є трохи застарілим рішенням. Хоча це і універсальний протокол для комунікацій, для обміну даними у реальному часі він може бути занадто громіздким та застарілим [9].

CoAP хоча і оптимізований для обміну даними у IoT мережах, він дуже рідко вживається та підтримується розробниками ПО та технологій [10].

AMQP це ефективний протокол передачі даних, але зважаючи на те, що він з початку не задумувався як протокол IoT, він може бути значно складніше у реалізації ніж MQTT та може створювати занадто сильне навантаження на систему потоку даних [11].

Загалом через використання S7-1200 буде гарною практикою налаштування OPC UA Server на контролері на базі програмування у клієнті TIA Portal, і щоб дані через цей протокол потрапляли на клієнт Node-Red. Далі дані потрапляють у MQTT брокер.

У якості MQTT брокера у деяких проєктах використовують сервіс HiveMQ. Він має багато переваг щодо передачі даних, надійності, безпеки, але занадто комплексний для заданої задачі. Те саме можна сказати про EMQX, VerneMQ, NanoMQ, CloudMQTT, які хоча і забезбечують високу продуктивність та функціональність, але вони є занадто масштабними.

Оптимальним варіантом MQTT брокера є Mosquitto завдяки його лімітованій масштабності, легкості та простоті у використанні, дуже широкій документації та спільноті. Він підтримує MQTT v5, WebSocket, ACL та веб-моніторінг.

Дуже схожий принцип роботи цифрових двійників описується у роботі Карло Хьюмана та Антона Германа Бейсона щодо вивчення передачі даних з використанням MQTT. Даний проєкт описує можливий механізм, структуру роботи інтеграції IoT.

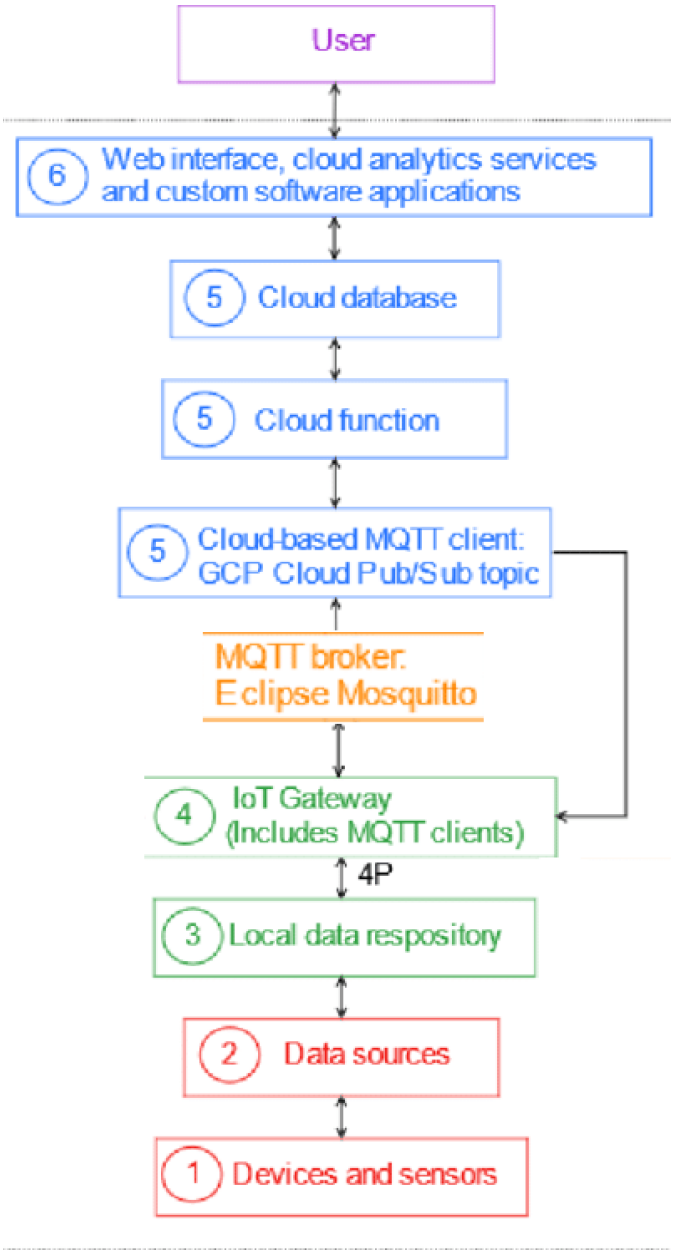


Рисунок 1.4 – Модель SLADTA

Дослідники пропонують модель SLADTA (Six-Layer Architecture for Digital Twins with Aggregation) [12], що складається з шости основних шарів:

1. Девайси та сенсори;
2. Джерела даних;

3. Локальні репозиторії даних;
4. IoT-мости;
5. Хмарні сервіси, що містять дані про об’єкт, або сервери;
6. Емуляція, моделювання, симулювання, візуалізація.

Якщо проаналізувати схему, на якій зображено принцип роботи даного фреймворку та архітектури, то бачимо, що фактично саме так ми і планували реалізувати цифрового двійника транспортної системи та SCARA-робота (рис. 1.4).

Дослідники вивчають модель, яка передає дані з локального серверу у складі девайсу або ПК, на MQTT-клієнт, потім на брокера Mosquitto (у роботі вибір брокера аргументовано простотою та ефективністю), потім у хмарні сервіси та бази даних, потім до юзера. Оскільки у майбутній роботі буде реалізація саме хмарного двійника, тобто такого, який можна запустити з будь-якого ПК, навіть який знаходиться за межами міста, в якому знаходиться об’єкт, у нас додається ще етап з оберненим ланцюгом MQTT брокера та двигуна візуалізації, який буде приймати дані, а замість початкового MQTT-брокера використовується OPC UA Server.

У роботі Агуса Хасана розглянуто модель предиктивного цифрового двійника (ЦД), який постійно оновлюється з даними з реального об’єкту (а саме корабля) для визначення можливих майбутніх помилок та потенційних невдач. Завдяки отриманню даних з датчиків, діагностичних алгоритмів та систем часового прогнозування, цифровий двійник може зпрогнозувати можливі помилки у компонентах корабля, дозволяючи здійснювати ранню діагностику подібних невдач.

На рисунку 1.5 зображено схему запропонованої моделі. Фізична та віртуальна версії корабля постійно здійснює двосторонній обмін даними. Фізична надає параметри з датчиків та стан компонентів, який проходить через декілька шарів систем прогнозування у ЦД, візуалізується у оператора, а потім надає керівні сигнали для реальної системи [13].

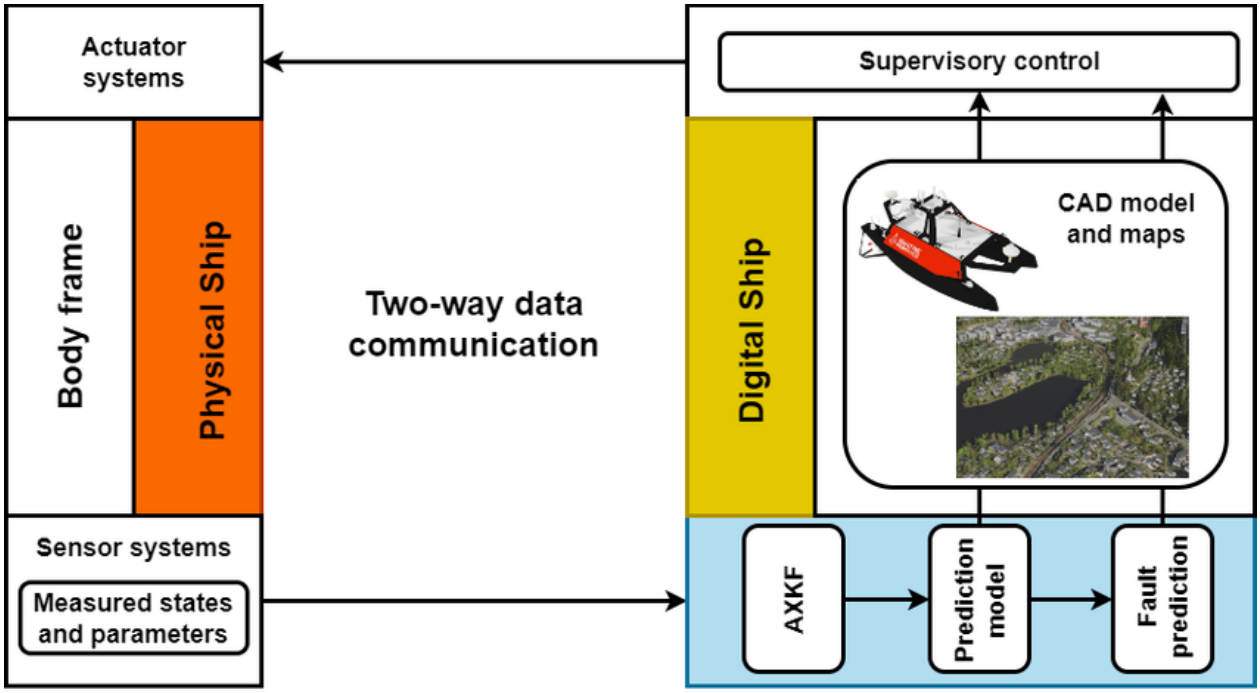


Рисунок 1.5 – Схема двостороннього обміну інформацією між фізичною та віртуальною версією корабля

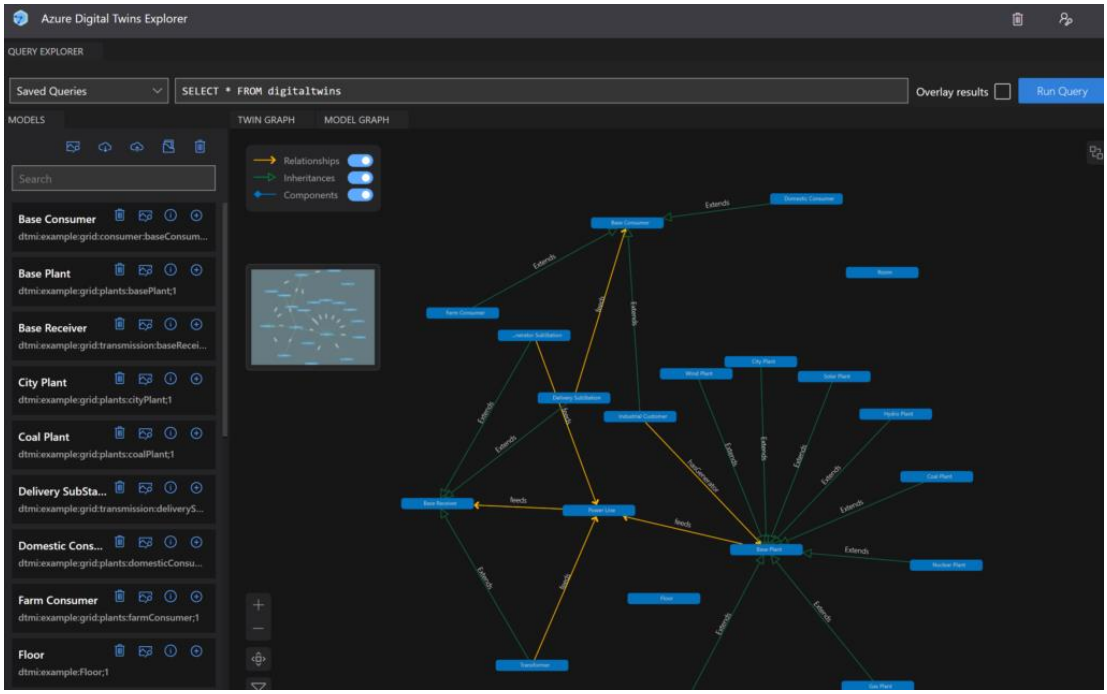


Рисунок 1.6 – Візуалізація потоків даних у Microsoft Azure Digital Twin

У Microsoft Azure є власна система Digital Twin, яка дозволяє у реальному часі формувати зв’язки між різними запросами даних, а після цього

дивитися на надходження сигналів у реальному часі і слідкувати за надсилання даних з серверу у базу даних або іншим клієнтам.

Дана система може бути корисною для візуалізації та оптимізації процесу. Як запевняють дослідники порталу XMPRO, дуже важливим плюсом є також інтеграція з іншими сервісами Azure та Microsoft.

Окрім цього, Azure Digital Twin ефективно інтегрується з MQTT-брокерами та клієнтами за рахунок спрощеної системи передачі даних через різні протоколи.

Приклад інтерфейсу передачі даних цифрового двійника у Mizrosoft Azure Digital Twin зображено на рис. 1.6.

Необхідно також обрати двигун візуалізації. Для ймовірних розширень як фізичних, так і технологічних, необхідно обирати якомога широкий та кастомний двигун, який має найбільше функціоналу та підтримки спільноти та розробника.

Користувачі програм для створення цифрового контенту (3DS Max або Maya) існують досить давно, та мають вбудовані системи написання скриптів для кастомізації. І такі інструменти створені ще десятки років назад. Але насправді до 2020-х років великим бар'єром для створення ЦД була складність навіть початкових етапів налаштування. Через це найм кваліфікованого спеціаліста програмування було не тільки дуже коштовитратним, а і часовитратним процесом. А окрім програмування необхідні і UX-спеціалісти, які б зробили керування ЦД зручним для звичайних операторів об'єктів. Саме тому на ринку ЦД компанії почали звертати увагу на інші варіанти реалізації даних систем.

У контексті IoT та Digital Twin останнім часом все більшої популярності набувають ігрові двигуни розробки. На відміну, від класичних систем цифрових двійників від Siemens, Nvidia, Autodesk, дані двигуни мають більш широку кастомізацію, функціонал для візуалізації, інтеграцію зі сторонніми сервісами та є більш простими у використанні (звісно, при наявності досвіду розробки продуктів у керівників проєктів).

Дослідники Крістіан Клаузен, Ченг Ма та Бо Йоргенсен вивчили дану тему використання ігрових двигунів та їх користі на прикладі системи сонячних панелей реального будинку (рис 1.7) [14].

Повідомляється, що ігрові двигуни забезпечують реалістичне відображення об’єкту, при цьому поєднуючи у собі простоту зміни у процесі розробки. Їх прототип цифрового двійника забезпечує вивід фотоелектричної моделі, яка містить та враховує позицію сонця, позицію індивідуальних модулей і комірок панелей, і навіть об’єктів, що блокують світло. Результати повідомляють, що двигун правильно прораховує вплив блокуючих об’єктів на світло.

Загалом дослідники кажуть, що ігрові двигуни візуалізації дозволяють дуже реалістично реалізувати різні об’єкти автоматизації, завдяки чому налаштування і робота з цими об’єктами відбувається набагато швидше та простіше. І незважаючи на те, що створення цифрового двійника створює обов’язок щодо перевірки коректності відображення процесу реального світу, цей напрям є дуже новітнім і перспективним.

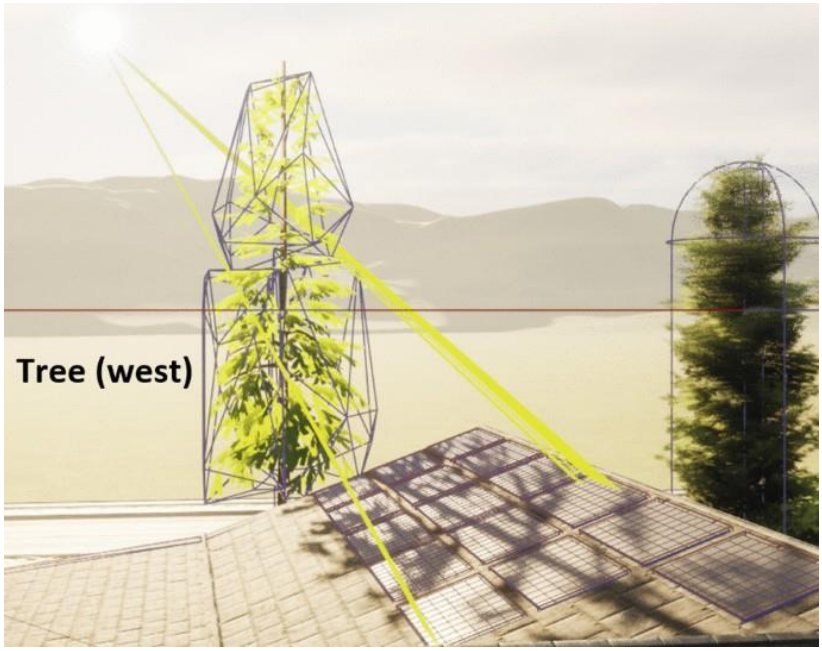


Рисунок 1.7 – Візуалізація промінів світла, які потрапляють на сонячні панелі даха будинку у Unreal Engine 5

Інший дослідник, Томас Стрігль, розглядав обрання двигунів Unity, Unreal Engine 5 та Nvidia Omniverse у контексті створення цифрових двійників.

Автор сприймає Unreal та Unity у якості рівноцінних конкурентів, у той час зазначаючи, що використання Omniverse може призвести до «загрузнення» команди розробки у одному технологічному інструменті. Тобто, перші два інструменти мають значно ширші можливості для інтеграції та комунікації з іншими технологіями [15].

Порівнюючи Unreal та Unity, дослідник зазначає, що вони обидва мають дуже зручні інструменти імпорта MQTT даних, цифрових 3Д-моделей. Більш того, дані інструменти надають можливість дуже зручно при необхідності передавати цифрових двійників на платформи окрім PC — а саме Android та iOS, а це дуже вагомий плюс, враховуючи останні тренди галузі автоматизації щодо використання VR/AR технологій.

У висновку автор каже, що Unity та Unreal є найкращими виборами на ринку, і обрання інструменту є суб'єктивним і засновується на тому, яка мова програмування зручніша — C# (Unity) або C++ (Unreal). Ще у даних двигунів різні ціни монетизації ліцензії для великих компаній [16].

Як Unity, так і Unreal роблять велику ставку на розробку цифрових двійників.

Денні Ланге, старший віце-президент ШІ Unity та Тімоті Вест, віце-президент з інструментів для Unity навели дуже прості, але ефектні приклади цифрових двійників – це, наприклад, Pokemon Go від Niantic, або Google Maps (що фактично абсолютною правдою). Ще одним прикладом цифрового двійника є застосунок EasyWay, який показує рух транспорту у місті у реальному часі.

Денні Ланге каже, що Unity робить значні інвестиції у цю концепцію, і вона дійсно може стати основним кроком у майбутній концепції Метавсесвіту. По суті, це перший крок для створення цифрових світів. Завдяки ньому ми можему тренуватися та модифікувати, перш ніж робити це у реальному [17].

Unreal теж робить велику ставку на цифрових двійників. Компанія постійно проводить вебінари та повноцінні конференції у павільйонах, присвячені цій технології, діляться інсайтами партнерів, публікують навчальні матеріали та навіть пропонують гранти на реалізацію даних проєктів.

Бізнес-менеджер з розробки Unreal Engine, Девід Вейр-МакКол, каже, що цифрові двійники – це фундамент на якому буде побудований майбутній метавсесвіт, провідний тренд IoT. «Завдяки метавсесвіту, цифровому світові, який живе поряд з нашим реальним фізичним світом, який дозволяє нам жити, працювати, навчатися і грати разом, будуть досягнуті нові рубіжі еволюції промислових, ігрових, наукових технологій. Одного дня цифрові репліки будинків та міст будуть об’єднані з реальними для формування паралельного світу, де люди разом соціалізуються.

На рисунку 1.8 можна побачити цифрового двійника метро Чанша від 51World на двигуні Unreal Engine 5.

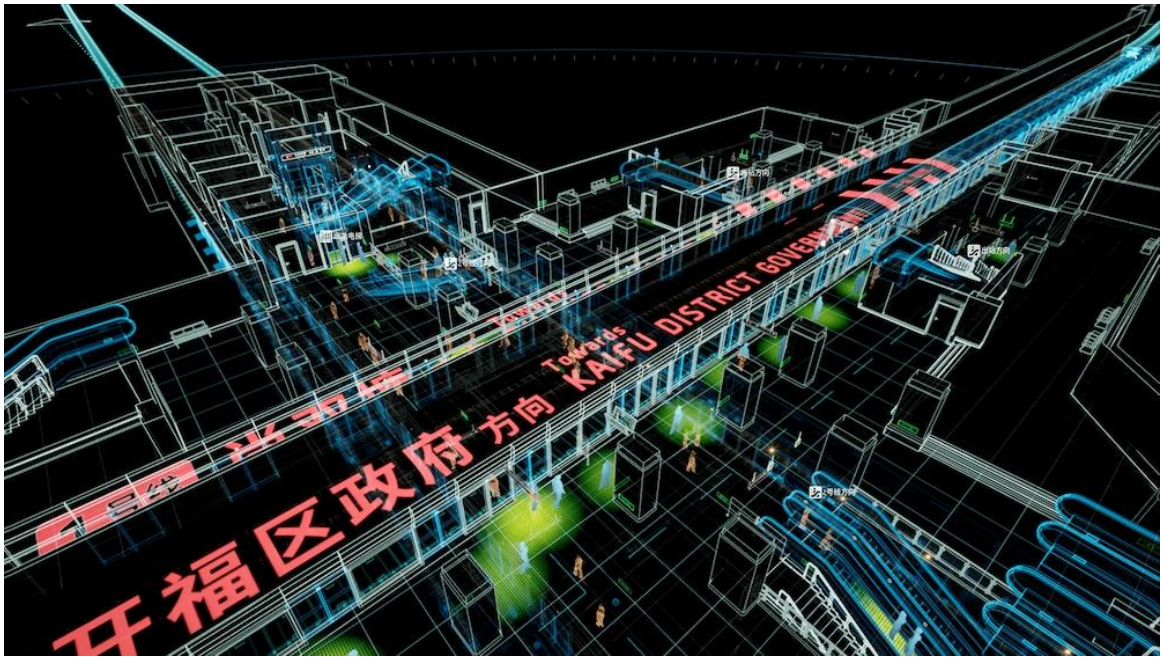


Рисунок 1.8 – Цифровий двійник метро Чанша від 51World

Отже, на основі всієї зазначеної інформації, для майбутньої роботи оберемо саме Unreal Engine 5 та подальший аналіз літературних джерел будемо вести саме у цьому напряму. Але важливо зберігати можливі реалізації і з інших двигунів візуалізації та технічного забезпечення.

1.4 Аналіз інформації щодо створення цифрових двійників на базі двигуна візуалізації

Для з'єднання Azure з Unreal Engine існує плагін-інструмент ADTLink, який, як запевняє розробник, простий і ефективний у використанні. Однак, у зв'язку з виходом Unreal Engine 5 та припиненням підтримки четвертої версії, у магазині Unreal плагін пропав та більше недоступний для завантаження.

Також, користувачі, які завантажили плагін зі сторонніх ресурсів, повідомляють, що програма відображає повідомлення «unreal-plugin-config.json».

Однак, навіть незважаючи на це, якщо подивитися на посилання у коментаріях до плагіна, можна знайти посібник по інтеграції Azure у UE5, який може дати багато важливої інформації. Передусім, проєкт надає схему високорівневої архітектури (рис 1.9), де можна побачити приблизно аналогічну SLADTA структуру, хоча і спрощену. Тут можна побачити розгорнуту структуру проєкта на боці Unreal та хмарного сервісу Azure [18].

Проєкт пропонує широку інтеграцію сервісів Azure, а саме Digital Twin, як основу, IoT Hub для обробки IoT інформації, SignalIR Service для передачі інформації додатку, Time Series Insights для створення візуалізації чартів та графіків [19].

Втім, у даній системі через те, що це навчальний проєкт, не написано про використання вбудованого у Azure інструментарію SQL Server, який також можна використовувати для збереження інформації про об'єкт (щоб в подальшому можна було спрогнозувати певні сценарії роботи або поломок) [20].

Важливим моментом є і використання симуляції IoT девайсів під назвою Mock-Devices, яке може дуже сильно знадобитися при розробці перших версій проєкту без наявного фізичного об'єкту [21].

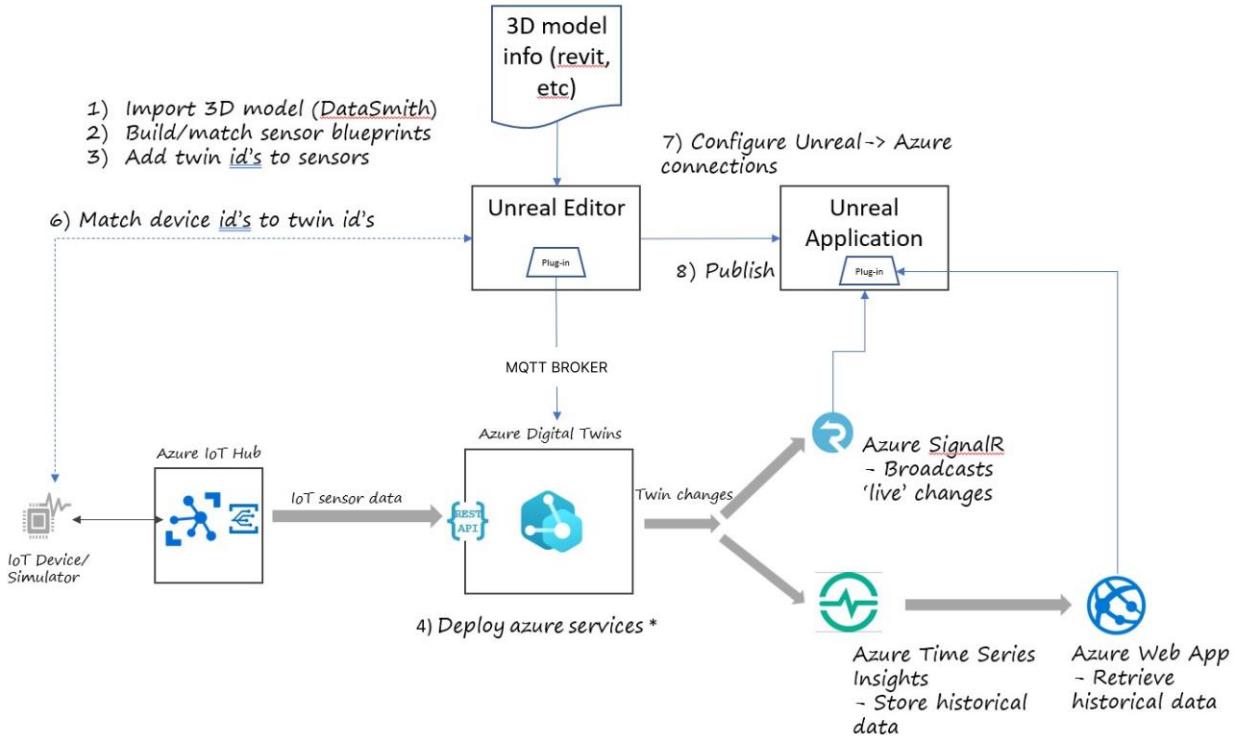


Рисунок 1.9 – Модель цифрового двійника на основі технологій Azure та Unreal Engine

При дослідженні транспортних цифрових двійників на базі Unreal Engine, компанія Epic Games зробила важливий висновок, що цей двигун є гарним вибором у якості основного інструменту, оскільки його дуже легко масштабувати на різні платформи (рис 1.10), а потім інтегрувати у процес модифікації [22].

Наприклад, Unreal Engine може використовуватися для створення віртуального прототипу нового автомобіля, або робота який створює цей автомобіль. Прототип надалі може використовуватися для тестування різних аспектів дизайну автомобіля чи робота, таких як його аеродинаміка, керованість та безпека.

Unreal Engine також може використовуватися для створення віртуального середовища, в якому можна тестувати різні засоби автоматизації. Це середовище може бути наповнене різними небезпеками, щоб допомогти розробникам тестувати, як реальний фізичний об'єкт реагує на різні ситуації [23].

Портал Twinspiration, присвячений розробці цифрових двійників, провів власний аналіз користі Unreal Engine щодо розробки даної технології [24-26].

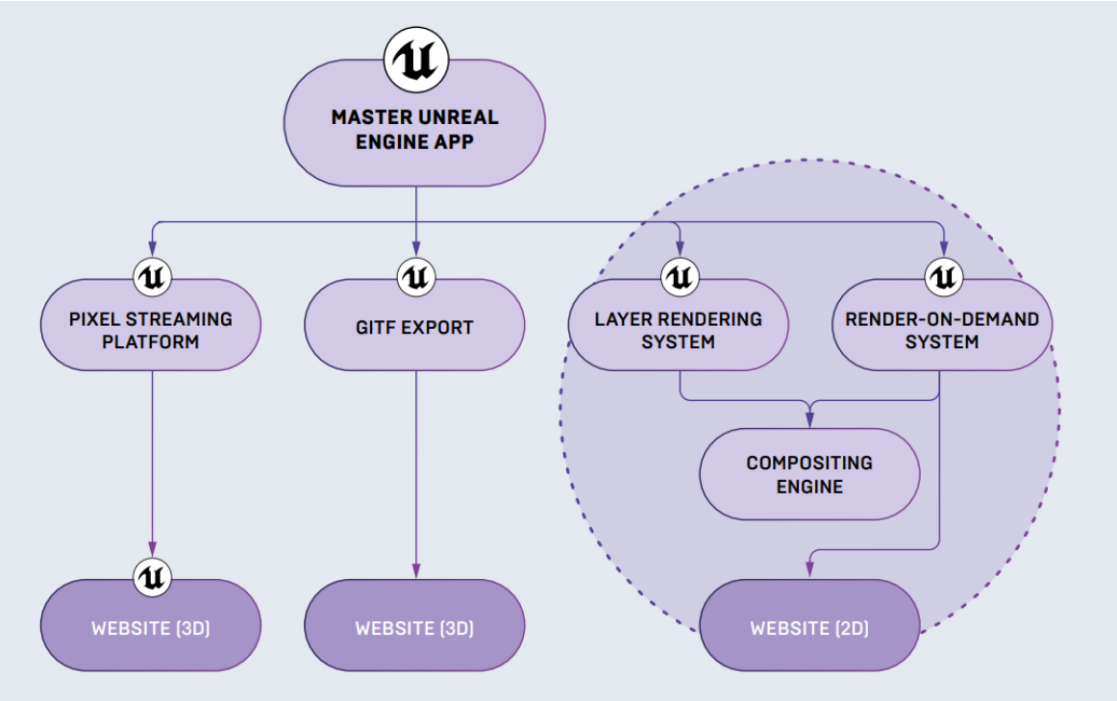


Рисунок 1.10 – Схема масштабування одного проєкту цифрового двійника на різні платформи з різними задачами

Він зазначив, що для майбутніх розробників дуже знадобиться вбудований інструментарій Digital Content Creation, який дозволяє дуже зручно з заздалегідь налаштованими параметрами не тільки інтегрувати моделі та скелет-компоненти об'єктів цифрових двійників, а і завдяки нодовій системі налаштувати вигляд об'єкта та навіть його фізичні властивості (завдяки вбудованій у двигун системі Unreal Physics Handle, яка фактично займається реалістичною симуляцією фізики).

Ініціатива HMI Dojo та порталу Spyro Soft теж варта уваги, оскільки має у собі декілька важливих пунктів для подальшої реалізації проєкту цифрового двійника. У цій ініціативі створювався віртуальний екскаватор з покроковим описом процесу розробки (схема процесу на рис. 1.11). Використання Unreal Engine надає унікальну можливість керування цифровим двійником у абсолютно різних видах. Як повідомляють автори, окрім PC, Android та iOS систем, двигун дуже вдало здатний масштабувати технологію на VR-пристрої,

а також адаптувати її під пристрої, які підтримують технології доповненої реальності [27].

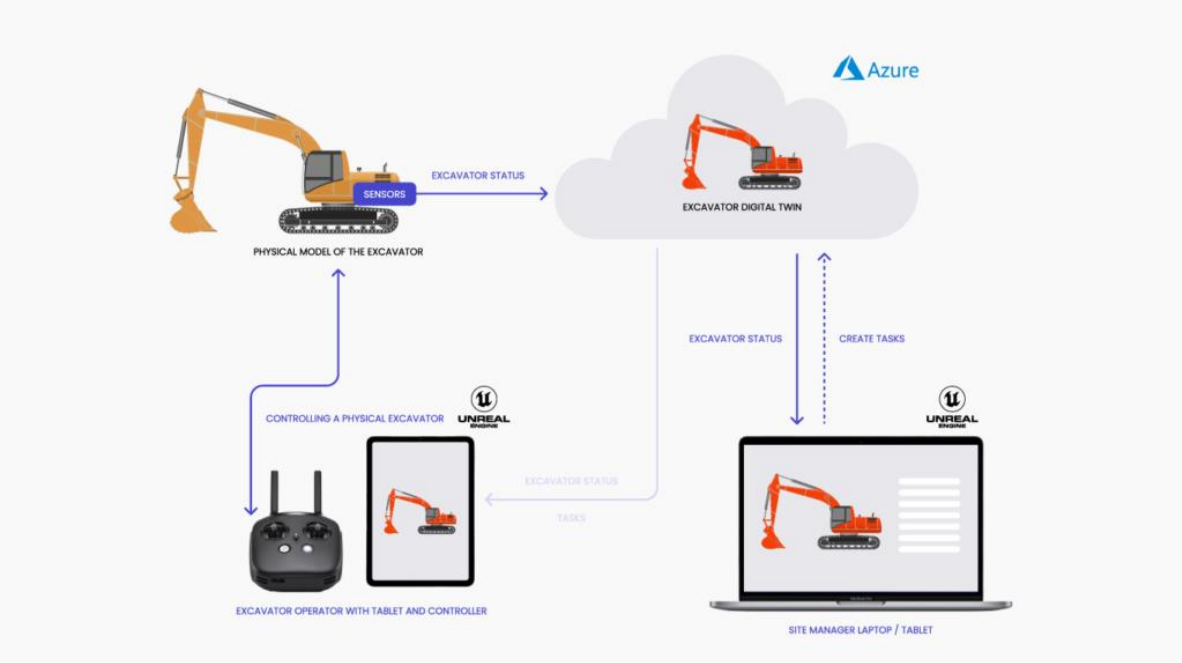


Рисунок 1.11 – Схема цифрового двійника керування моделлю екскаватора

Система, запропонована авторами, також використовує комбінацію різних датчиків, пов’язаних з Azure та пристроями керування за рахунок інтернет протоколів.

Twinmotion каже, що технологія цифрових двійників розвивається дуже швидкими темпами, і що до 2050 року стане пануючою у сфері IoT. Вони приводять власну модель цифрового двійника міської інфраструктури. У якості основної ідеї вони приводять фотограмметрію [28].

Це дуже вдалий прийом, оскільки для точного відображення реального об’єкта необхідно створити точну 3D-копію, а це може бути ресурсозатратно. Як альтернатива, існує фотограмметрія, яка ніби «сканує» об’єкт з різних сторін, та за допомогою ШІ створює його 3D-копію. Існують як і комплексні, так і прості методи реалізації даного процесу, але для прикладу можна використовувати більш прості, наприклад, Android застосунки. Вони пропонують приблизно ідентичний процес сканування 3D-об’єкту: Обернути смартфон навколо необхідної моделі на 360 градусів, а також зверху і, при

можливості, знизу. Після цього модель експортується і модифікується у, відповідно, 3D-редакторах (Maya, Blender, Autocad тощо).

Для реалізації цифрового двійника знадобляться будь яка інформація щодо розробки подібних проєктів. Unreal Engine 5 підтримує дві мови розробки, і якщо подивитися на методи реалізації подібних проєктів ЦД навіть на інших мовах програмування чи двигунах, це буде корисним матеріалом для аналізу.

Чанг Чіхан та Ніван Холоділі створили власного цифрового двійника SCARA-робота (рис 1.12) на базі Unity та Matlab (для комплексного керування рухом руки). Сам робот створений на базі Arduino [29].

У проєкті розробники додали руці візуальний сенсор (який є і в актуальній версії моделі транспортно-розвантажувальних операцій). Але на відміну від нашого пристрою, там рука обирає об’єкт не на основі його габаритів, а на основі кольору.

Корисними є також дані щодо відображення положень руки та наявних у ній об’єктів у інтерфейсі користувача.

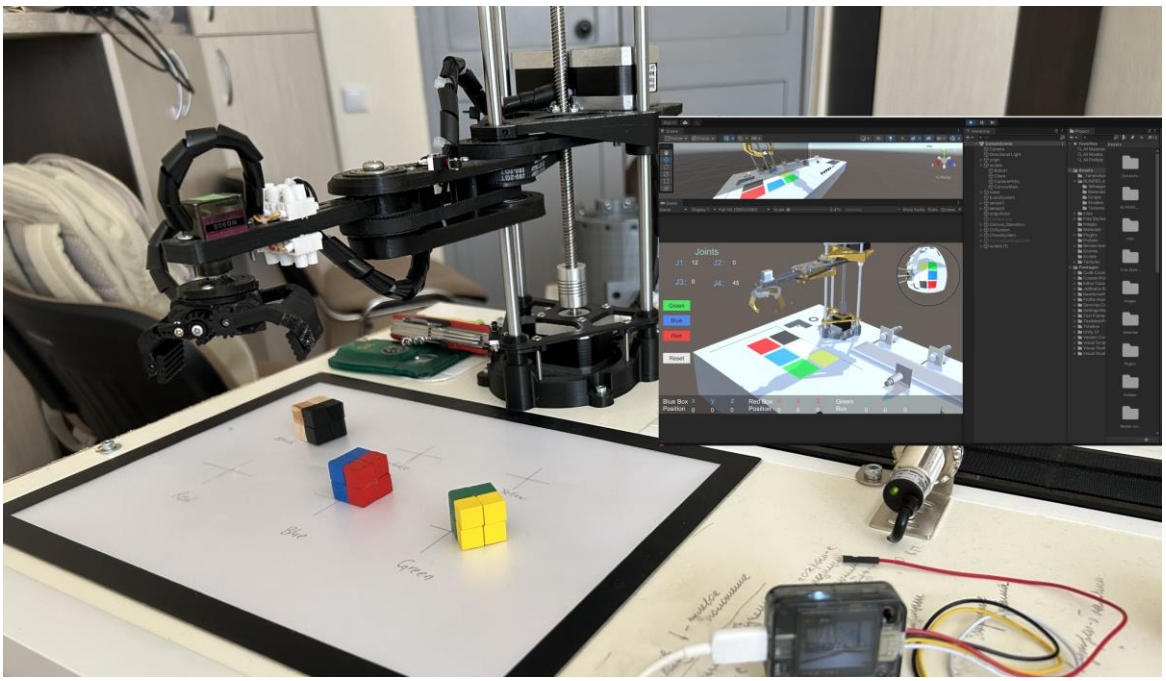


Рисунок 1.12 – SCARA-робот та його цифровий двійник у середовищі Unity

У вебінарі Stable Automation розповідалося про зручність способу підключення промислового контролера S7-1200 до OPC UA. Для цього треба

у налаштуваннях контролера TIA Portal обрати наявну ліцензію на використання OPC UA, вписати адрес серверу.

Після цього створюється сервер-інтерфейс з необхідними змінними для передачі на OPC UA. Запускається клієнт, і на цьому все, налаштування завершено. Достатньо лише запустити симуляцію ПЛК та перевірити наявність необхідних змінних на сервері.

У статті Чінгуй Хе та Ксянміна Янга розглядається схожий проєкт системи цифрового двійника системи конвеєрних стрічок та SCARA-роботів. Вони створили модель, яка керує двонаправленим синхронізованим рухом, саме на основі протоколів OPC UA для керуванням декількома роботами на конвеєрі. Дослідники перевірили здійсненність двохпоточної системи передачі даних між віртуальною та фізичною версіями об’єкту і зауважили, що це найбільш оптимальний метод реалізації цифрових двійників [30].

IEEE теж створила подібний проєкт цифрового двійника SCARA-робота, але, на жаль, схоже, не дуже розібралися з питанням правильного визначення. Цифровий двійник – це відображення реального об’єкта з можливістю керування та візуалізації даних, їх аналізу.

Замість цього IEEE створили віртуального робота SCARA та лімітовану симуляцію поведінки та кінематики реальної моделі за рахунок чисельного моделювання, яке поєднує у собі динамічні рівняння Ейлера з урахуванням рівняння тертя компонентів між собою. Розроблений робот, як кажуть дослідники, може бути впроваджений у реальну схему моделювання та підключений до реальної моделі, використаний у навчальних цілях. Однак, фактично, назва проєкта невірна, бо IEEE створили віртуальну симуляцію SCARA-робота, а не цифрового двійника [31].

Технологічний центр MTAB у 2021 році також створив цифровий двійник SCARA робота (рис 1.13) на власному двигуні візуалізації. Вони завантажили 3D-модель, додала кінетичні та моторні характеристики, підключивши до модулів автоматизації процесів. Після декількох етапів симуляції та автоматизації, модель була під'єднана до реального маніпулятора

та уведена у експлуатацію. Інженери повідомляють, що постійно користуються даною симуляцією для виявлення помилок, мінусів безпеки та оптимізації [32].

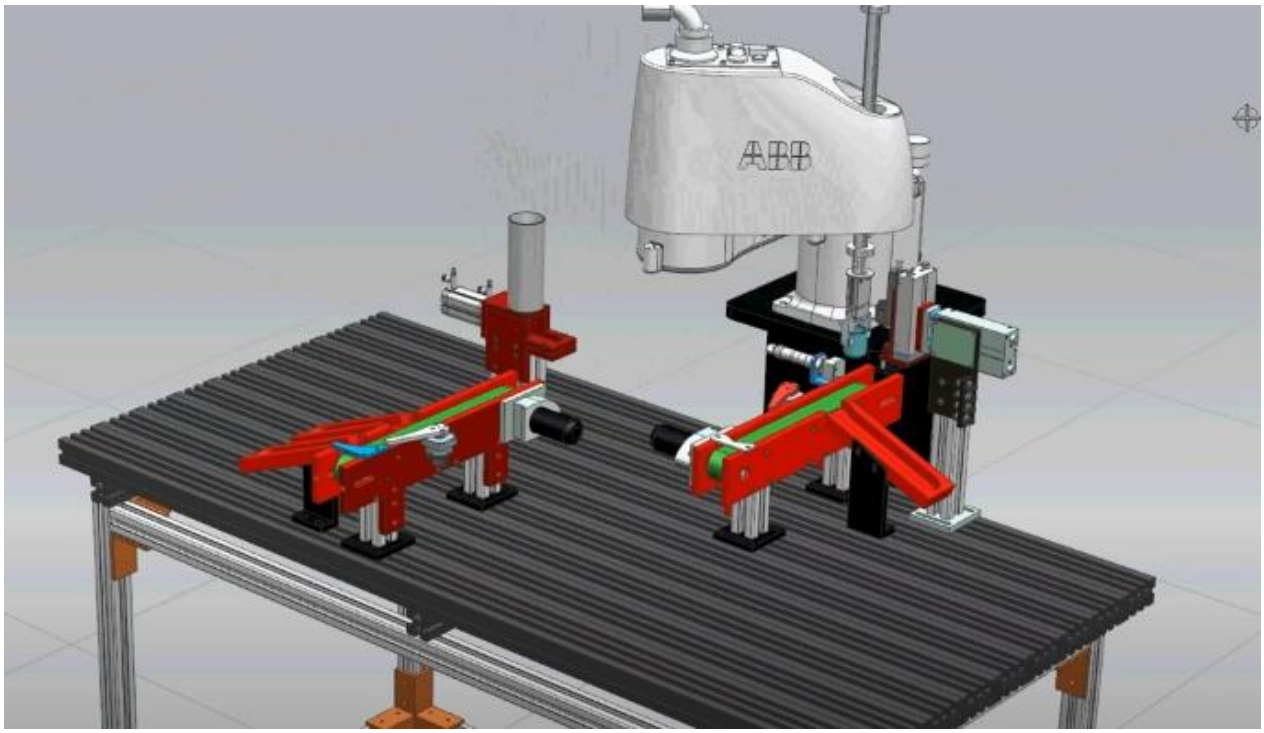


Рисунок 1.13 – ЦД SCARA-робота від MTAB Automation

Маріус Матуліс дослідив схожий проєкт реалізації SCARA-робота на Unity. У його роботі досліджено використання ШІ та ЦД для підвищення ефективності процесів виробництва.

У проєкті запропоновано використовувати віртуальне середовище, яке для прикладу було створене на двигуні Unity, для навчання робота-маніпулятора виконувати завдання у мінливому середовищі. Завдяки цьому підходу, можна одночасно навчати декілька копій маніпулятора, і вони будуть паралельно обмінюватися своїм досвідом, немовби нейрону в мозку людини. Такий метод потенціально може змінити розвиток автономних виробництв, де роботи швидше за людей зможуть знаходити ефективний та стійкий спосіб виконання поставлених задач. Дана робота показує можливу імплементацію ШІ у процес розробки цифрових двійників та плюси цього методу.

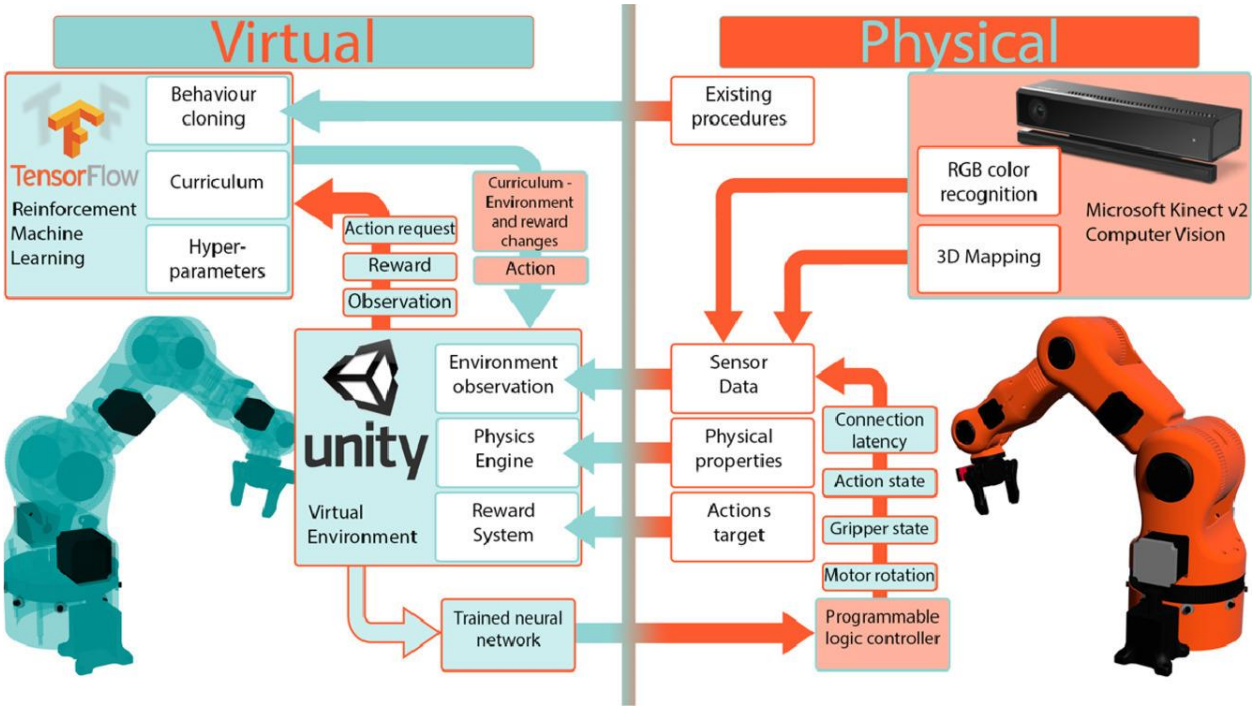


Рисунок 1.14 – Взаємозв’язок віртуальних та фізичних компонентів ЦД

Як основні результати роботи, було розроблено системи для навчання цифрого двійника у Unity за допомогою методу машинного навчання з підкріпленням інформації. Після цього досліджено взаємозв’язок між віртуальними та фізичними компонентами цифрового двійника та запропоновано методику перенесення знань, які отримані у цифровому середовищі, на фізичний рівень. Загалом дослідження відкриває нові можливості для функціоналу ЦД та ШІ.

На рисунку 1.14 можна побачити взаємодію фізичної та віртуальної копій об’єкту. Отримання реальних даних відбувається за рахунок камери Microsoft Kinect та значень з логічного контролеру. Потім вони за допомогою каналів передачі даних потрапляють у середовище Unity, з якого потрапляють на платформу навчання Tensorflow. Зворотній зв’язок відправляють дані з навчання знову у віртуальне середовище, звідки вони йдуть на контролер [33].

Університет Дебрецема створив власний проєкт SCARA-робота у середовищі Unreal Engine (рис 1.15). Основою було відтворення моделі KUKA KR5, надаючи можливість керувати та програмувати його в віртуальному середовищі.

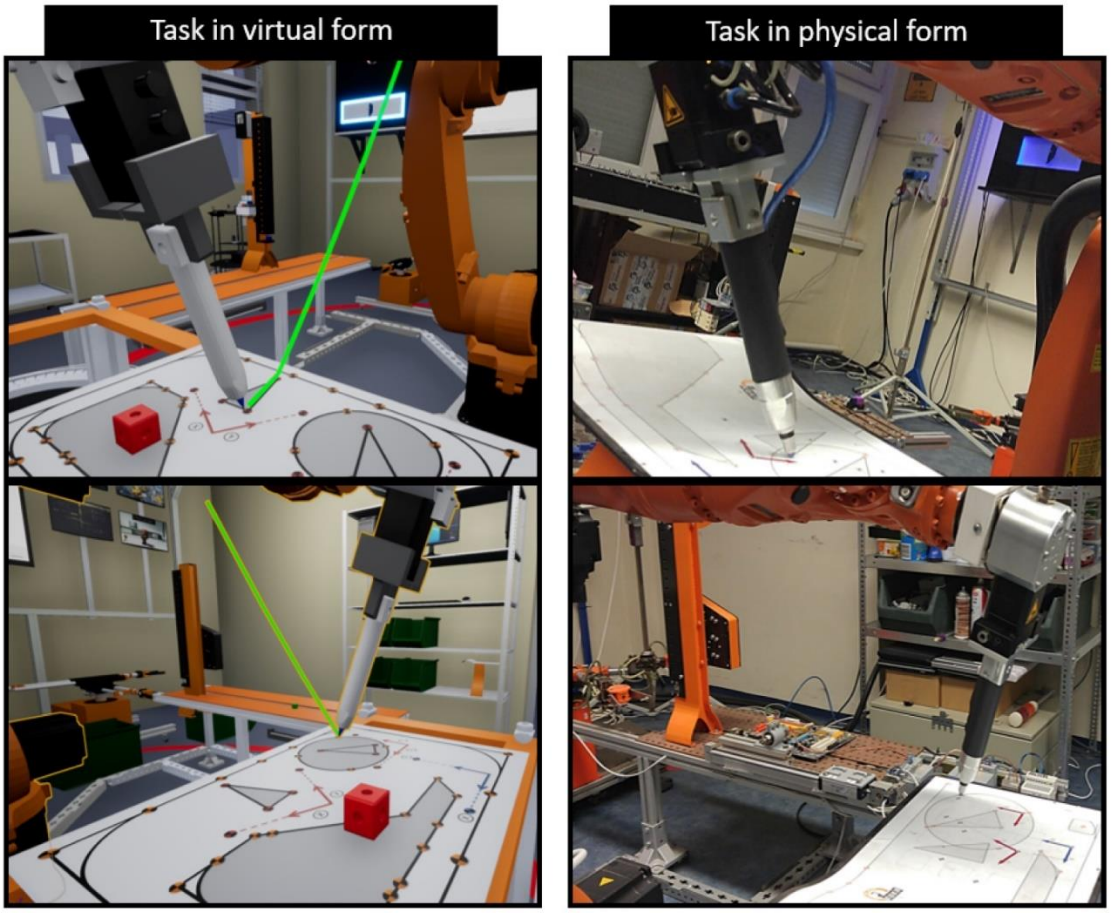


Рисунок 1.15 – Фізична та віртуальна версії KUKA KR5

Для віртуальної моделі та майбутнього навчання спеціалістів було прийнято рішення зробити інтерфейс керування максимально наближеною моделлю до реальних пультів керування. Найбільшими викликами при створенні моделі стали складність реалізації кінематики робота, бо для точного та правильного руху необхідні були точні розрахунки керуючих контурів. Також проблемною стороною було оптимізація виявлення колізій та формування правильної структури коду для плавної роботи програми та подальшої її оптимізації. Забезпечено тестування юзабіліті інтерфейсу та інтераційне реалізування, завдяки цьому вдалося зробити зручний ого варіант.

У результаті було створено різні методи керування – вільна та зворотна кінематика, та заздалегідь запрограмовані послідовності рухів. Створено систему створення та зберігання програм робота за рахунок віртуальних інтерфейсів [34].

Як можливі розширення проєкту, автори запропонували створення симуляції датчиків та зовнішньої комунікації, створення складніших схем руху, планування маршрутів та прогнозування помилок. Створення VR-версії для навчання операторів, симуляційних програм для тренування з робототехніки. Пропонувалося також дослідити потенційні випадки віртуального налагодження фізичної моделі.

Даний проєкт вдало демонструє можливості створення реалістичної та функціональної моделі реального SCARA-робота у середовищі Unreal Engine.

Йонас Соренсен та Ченг Грейс Ма при створенні цифрової моделі повітряного генератора запропонували свою модель цифрового двійника, який співвідноситься зі створюваною моделлю цифрового двійника транспортно-розвантажувальної моделі SCARA-робота, а саме частини передачі даних з Azure до Unreal Engine 5 (рис 1.16).

Дані у цій моделі потрапляють у UE через MQTT-брокера Mosquitto за допомогою плагіна, який дозволяє отримувати дані через MQTT. Але на наявній версії UE5 цей етап ще спрощено, бо функціонал даного плагіна додано власне у стандартну версію двигуна візуалізації [35].

Потім дані обмінюються з плагіном візуалізації повітряної турбіни, відбувається прогнозування руху цифрової турбіни, візуалізуються дані та графіки параметрів об’єкту та здійснюється керування цифрового двійника [36].

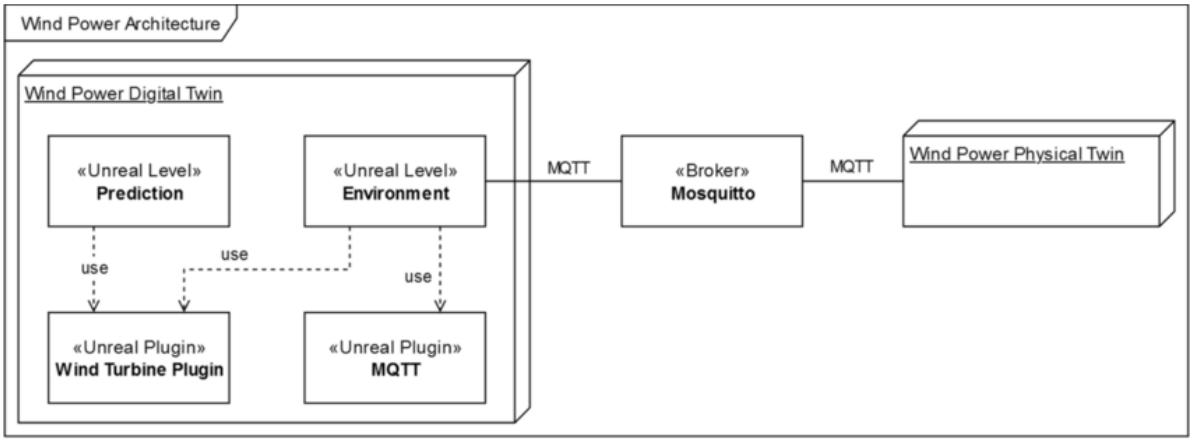


Рисунок 1.16 – Архітектура цифрового двійника повітряної турбіни

Робота Льюїса Вісенте описує нову систему цифрового двійника для робота з миття автоцистерн в контексті Industry 4.0, створену на Unreal Engine 5. Вона інтегрує різні технології ШІ, машинного навчання та IoT. Автори розробили архітектуру системи SCADA на основі програмного забезпечення Ignition, яке дозволяє підключатися до будь-яких ПЛК та пропонує широку масштабованість та кібербезпеку.

Система пропонує кілька режимів управління:

– Керування виключно цифровою симуляцією;

– Керування реальною системою;

– Моніторінг реальної системи за допомогою управління цифровим двійником.

Дана система дозволяє тестувати та модифікувати систему за допомогою ЦД перед реалізацією на реальному об'єкті. Дана система надає нагляд за системою через різні HMI: ПК, планшети, смартфони. Зв’язок з іншими системами реалізований через протоколи передачі даних та базу даних. Підтримується також генерація та управління тривогами у системі на основі заздалегідь визначених рівнів критичності.

Система автоматично генерує звіти, що зберігаються у базі даних. Вона забезпечує ролі та дозволи користувачів у системі.

Можна побачити на рисунку 1.17, що модель системи досить схожа на прогнозовану модель даної роботи, якщо замінити Ignition Server на сервіси Azure. Дані з реальних датчиків та сенсорів потрапляють на логічний контролер, звідки дані потрапляють на сервер через протоколи передачі даних (не зазначено конкретно у цьому випадку, але можливо, що також MQTT/OPC UA). З серверу на базу даних дані потрапляють через протокол MQTT, а на HMI — через HTTP (насправді не дуже оптимізований підхід, який може бути «заважким» для об’єму даних, що передаються).

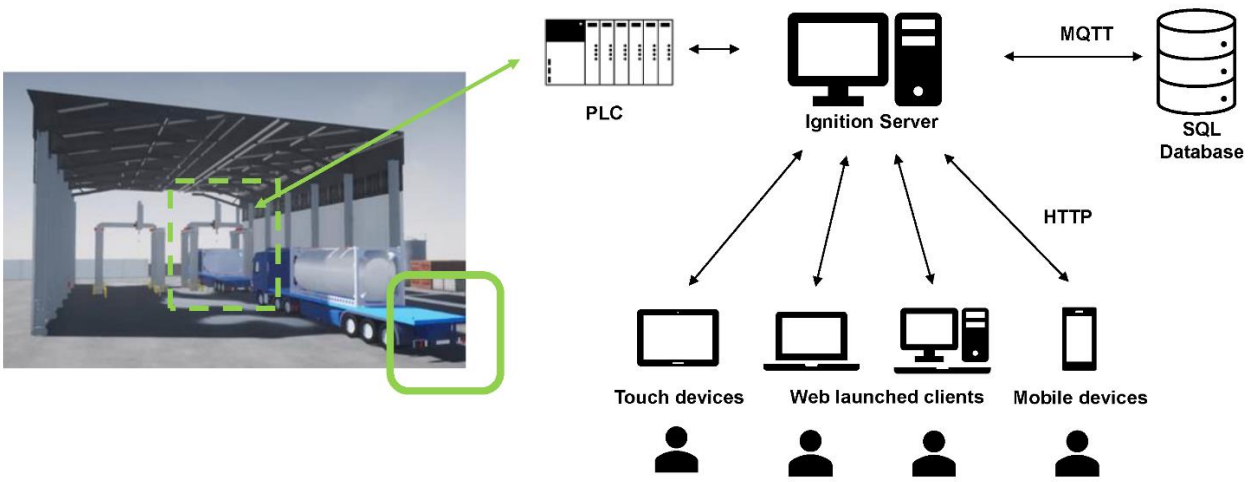


Рисунок 1.17 – Модель цифрового двійника Льюїса Вісенте

TalTech під час створення власного ЦД на Unreal Engine 5 зіштовхнулися з наступними викликами під час виконання реалізації моделі. Вони виконали аналіз існуючих рішень на ринку, дослідивши літературні джерела. Потім створили 3D-модель реального об’єкту, та інтегрували до неї систему управління ПІ-контроллером, та можливість зміни параметрів об’єкта у реальному часі.

Після цього дослідники імплементували двосторонню коммунікацію між віртуальною та фізичною версією та візуалізацію даних.

У результаті (рис 1.18) було створене імерсивне середовище для наглядового контролю та збору даних на основі концепції ЦД. Дослідники наголошують на необхідності звертати увагу на UX та UI складові цифрових двійників, бо, судячи з їх аналізу теперішніх моделей систем, даними моментами часто нехтують та не приділяють необхідної уваги.

Можна побачити як у доповненій реальності цифровий двійник розташовано поряд з реальним та відображається візуалізація даних та їх аналіз над ним [37].

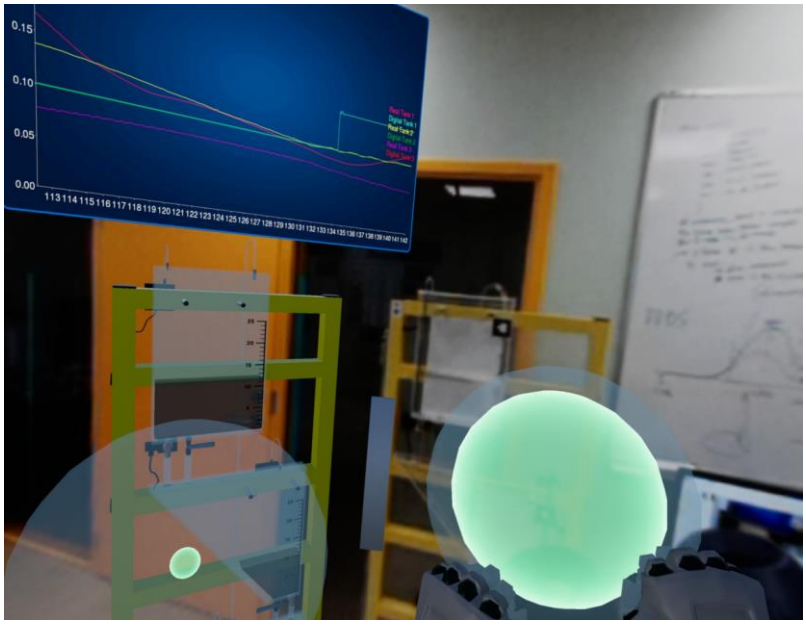


Рисунок 1.18 – Модель цифрового двійника заповненості контейнера водою на базі двигуна візуалізації Unreal Engine

Отже, на основі усіх розглянутих розробок можна приблизно сформулювати можливу технічну реалізацію ЦД комплексу розвантажувальних операцій на базі SCARA-руки. Але перед цим важливо розглянути патенти, та визначити їх переваги та недоліки.

1.5 Пошук патентів за напрямом досліджень та їх опис

Оскільки тема цифрових двійників дуже новітня, у українському полі патентів немає (при пошуці по базах даних UAPatents, SisNipoGovUa, Ukrpatent по запросу «цифровий двійник», «digital twin» нічого немає взагалі). Саме тому пошук патентів буде вестися по базам патентів Google.

– Назва патенту: Калібрування робота для доповненої реальності та цифрового двійника;

– Дата реєстрування патенту: 12 березня 2020 року;

– Номер патенту: US11396100B2.

Даний патент вирішує проблему, яка полягає у складності і тривалості процесу калібрування віртуальних пристроїв, доповненої та віртуальної реальності в робочих камерах промислових SCARA-роботів. Той спосіб, який

зараз використовується в індустрії (використання візуальних мішеней для визначення положення об’єкта) є занадто дорогим і, при цьому, не точним.

Візуальний маркер 120 (рис 1.19) за цим патентом розташований у фіксованому місці у робочій комірці. Опорна рамка 122 має початок координат і орієнтацію, що визначається відносно маркера 120.

Потім її положення визначається відносно SCARA-робота 112. Дана система визначення положення робота має три осі переміщення і три осі повороту. Загалом вона створена для того щоб уникнути неточності при відображенні об’єкту у системі двигуна візуалізації.

Патен показує як можна відкалібрувати двигун візуалізації з використанням маркерів реального положення SCARA-робота. Замість використання візуальних мішеней які є стандартом, використовується нова система відліку. Модель поверхні робота поєднується з окремими моделями його деталей для коректного відображення його у поточному положенні.

Плюсом є і те, що метод не потребує окремої візуальної мішені для калібрування, втім мінусом є те, що вона потребує заздалегідь прорахованих параметрів осей переміщення та повороту, що може потребувати великих ресурсів та складних моделей [38].

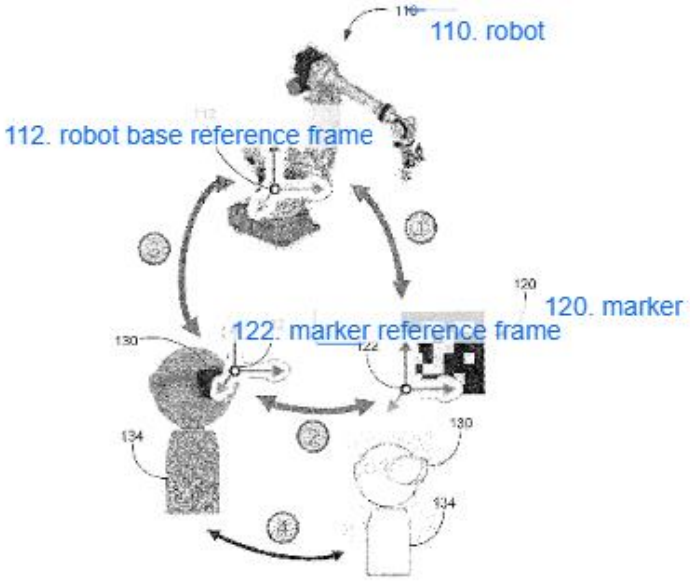


Рисунок 1.19 – Модель патента для калібрування SCARA-робота

- Назва патенту: Промисловий метод і система керування механічним маніпулятором-рукою на основі технології цифрових двійників;
- Дата реєстрування патенту: 30 вересня 2022 року;
- Номер патенту: CN115446867A.

Даний патент описує систему та метод керування механічною SCARA-рукою на основі створення цифрової моделі механічної руки з шістьома степенями свободи (на рисунку 1.20 вони зображені), а сама рука навчається та тренується за допомогою алгоритму машинного навчання з підкріпленням. Підкріплення використовується для отримання стратегії оптимізації процесу, та і для загального визначення траєкторії руху моделі. Після цього механічна рука виконує самоадаптивний режим на основі траєкторії руху моделі, а це дозволяє реалізувати інтелектуальне керування механічною рукою.

Перевагою є те, що використанні нейромереж та машинного навчання вирішує головну проблему, а саме нелінійності та невизначеності (які звісно можуть бути присутні в звичайних фіз. моделях). Забезпечення роботи алгоритму глибокого навчання дозволяє SCARA самонавчатися та підвищувати ступінь автоматизації, поступово реалізувати самоадаптований рух та обертання осей об’єкту.

Втім, у патенту є і мінуси, а саме те що він є складним та трудозатратним для реалізації такої цифрової моделі. Необхідно уважно врахуввати найдрібніші фактори, а саме тертя та знос. Забезпечення машинного навчання можуть зробити вимоги до до обчислювальних ресурсів та апаратного забезпечення занадто високими, як і зробити такими вимоги до забезпечення стабільного зв’язку між цифровою та фізичною системою [39].

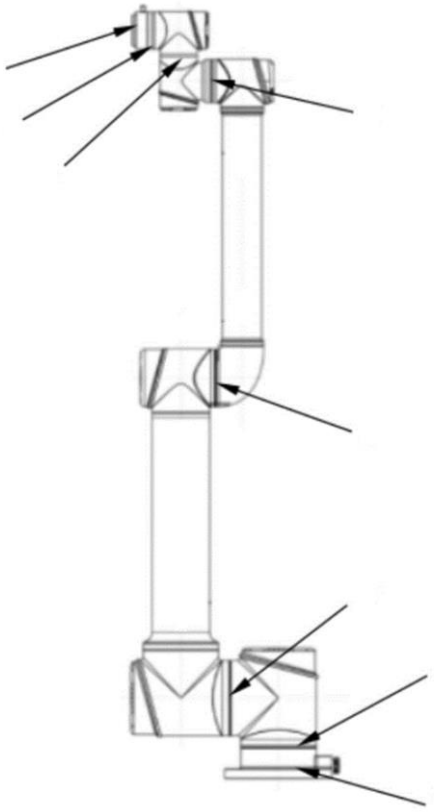


Рисунок 1.20 – Цифрова модель механічної руки з шістьома степенями свободи

– Назва патенту: Система цифрових двійників лінії складання комплексних продуктів з використанням механічних маніпуляторів;

– Дата реєстрування патенту: 17 березня 2020 року;

– Номер патенту: CN111413887B.

Даний патент описує систему цифрових двійників, яка складається з фізичного середовища, віртуального середовища та системи передачі даних між ними.

Система використовує дані з фізичних об’єктів для того щоб створити віртуальну модель відображення реального процесу збірки: робочі деталі, SCARA-руки (рис 1.21), циліндри.

Система також у реальному часі аналізує дані для виявлення та виправлення помилок на лінії збірки деталей.

Цей патент фокусується на модульності системи та категоризуванні даних за допомогою динамічних бібліотек. Ці бібліотеки складаються з

незалежних елементів. Патент має на меті створити новий режим управління обладнанням підриємств на основі синхронного відображення взаємодії фізичного та віртуального об’єкта. Це може, наприклад, забезпечити швидкий вияв несправностей.

При цьому мінусом патенту є те, що автор зазначає, що при роботі з великими масивами даних та обладнання система може мати проблеми з її модифікаціями, розширенням та безпекою, та загалом може бути дорогою у розробці [40].

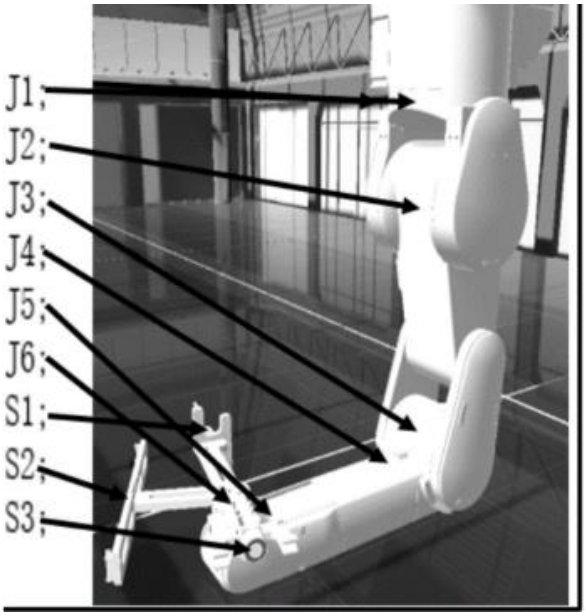


Рисунок 1.21 – Віртуальна модель механічного маніпулятора у цифровому середовищі

– Назва патенту: Цифровий робот-двійник для гнучкого процесу збірки автоматичного вимикача;

– Дата реєстрування патенту: 25 січня 2019 року;

– Номер патенту: CN109719730B.

Даний патент описує цифрового двійника SCARA, який знаходиться на моделі лінії збірки вимикачів (рис 1.22).

Цифровий двійник робота вводить модель механічної руки у середовище Unity, щоб правильно симулювати рух моделі. Завдяки цьому досягається синхронізація з реальним об’єктом.

Даний патент забезпечує синхронізацію фізичної та реальної моделі та дозволяє динамічно стежити за реальним процесом збірки. Завдяки ньому можна виконувати моніторинг несправностей, зберігати дані у базі даних, та завдяки алгоритмам обчислювати частоту несправностей за допомогою даних у історії дій ЦД.

Але при цьому якість роботи даної системи дуже залежить від самої якості обраних алгоритмів, та при цьому вимагає постійного оновлення та модифікації окремих компонентів. Система може бути обмеженою через розміри ресурсів обчислювальних машин, через що можуть виникати помилки у синхронізації [41].

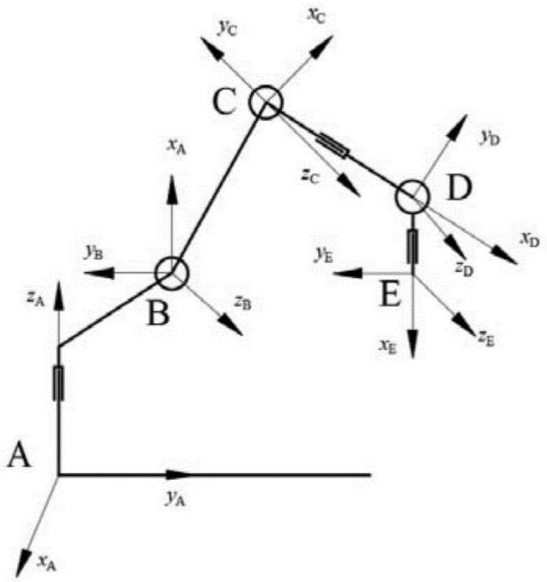


Рисунок 1.22 – Основні осі керування цифровим двійником SCARA-робота

– Назва патенту: Візуалізація сенсорів IoT у реальному часі із використанням доповненої реальності та цифрових війників;

– Дата реєстрування патенту: 26 травня 2019 року;

– Номер патенту: US12306220B2.

Даний патент описує передачу показань датчиків до ігрового двигуна візуалізації Unreal Engine. Для візуалізації вимірюваних значень використовуються модулі віртуальних діаграм Kantan Charts плагіна Unreal Motion Graphics, а для моніторінгу змінних діаграмми Time Series Plot, які

представляють дані з сенсорів, що містять інформацію про навколишнє середовище.

Дані з сенсорів потрапляють до ігрового двигуна (на рис. 1.23), після чого відбувається їх збереження, обчислення, і після цього відображення на HMI-інтерфейсі [42].

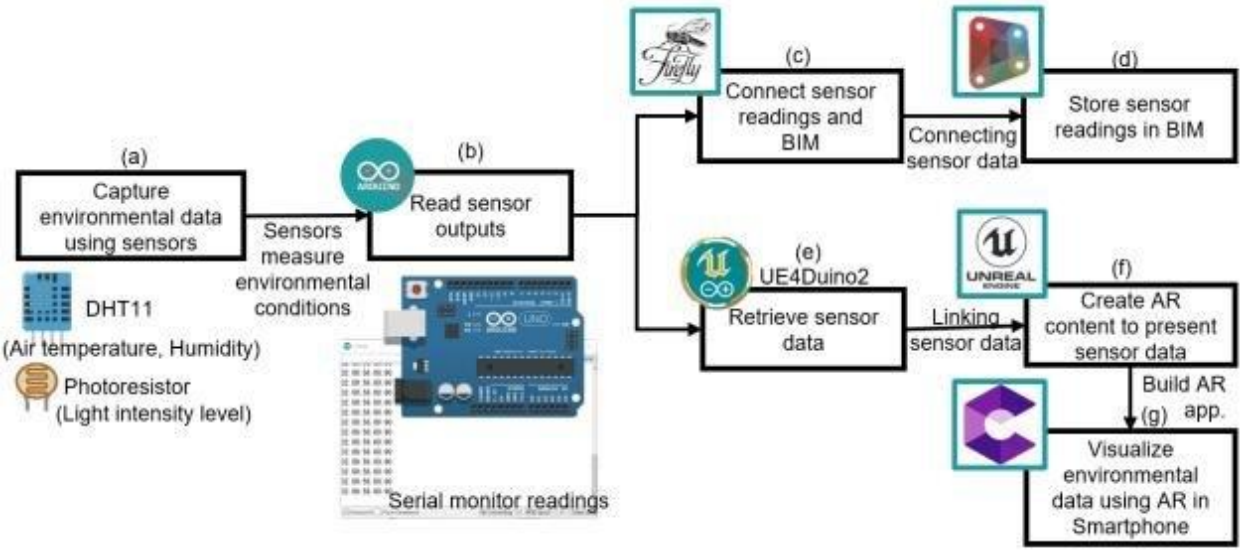


Рисунок 1.23 – Схема зі статті опису патента, яка розглядає приклад інтеграції патенту у реальні IoT-системи цифрових двійників

Висновки до розділу:

У цьому розділі було наведено загальні відомості про технологічний процес, описано модель транспортно-розвантажувальних операцій та цифрового двійника системи. Написана приблизна модель реалізації ЦД, та на базі яких програм це планується робити.

Після цього було розглянуто сам принцип створення цифрових двійників, їх переваги та недоліки, їх класифікацію та потенціал (який було сформовано на основі статей та новин щодо провідних компаній світу, а також статистики).

Після цього було проведено аналіз інформації щодо конкретної реалізації цифрових двійників та наявних технологій IoT, а також схожих моделей двопоточної передачі даних між цифровим двійником та реальним фізичним об’єктом.

Після визначення технології реалізації, а саме двигуна візуалізації Unreal Engine 5, було розглянуто аналогічні проекти реалізації цифрових двійників SCARA-руки на UE5 та сміжних технологіях, а також вивчено досвід інших людей щодо реалізації ЦД у цілому.

Проведено аналіз патентів за напрямком дослідження, виокремлено плюси та мінуси у їх реалізації.

РОЗДІЛ 2
СТВОРЕННЯ МОДЕЛІ СИСТЕМИ ЦИФРОВОГО ДВІЙНИКА, ЇЇ СТРУКТУРИ ТА ОПИС НАПРЯМКУ ДОСЛІДЖЕНЬ

2.1 Проектування загальної архітектури системи на базі еталонних архітектур Industry 4.0

При проектуванні загальної архітектури цифрового двійника транспортно-розвантажувальної системи на базі SCARA-робота, ми будемо спиратися на еталонну архітектуру RAMI 4.0 (Reference Architectural Model Industry 4.0) [43] (рис 2.1).

Цей стандарт був розроблений робочою групою Platformen Industrie 4.0 та визначає узгоджену архітектурну модель для реалізації кіберфізичних систем у промисловості.

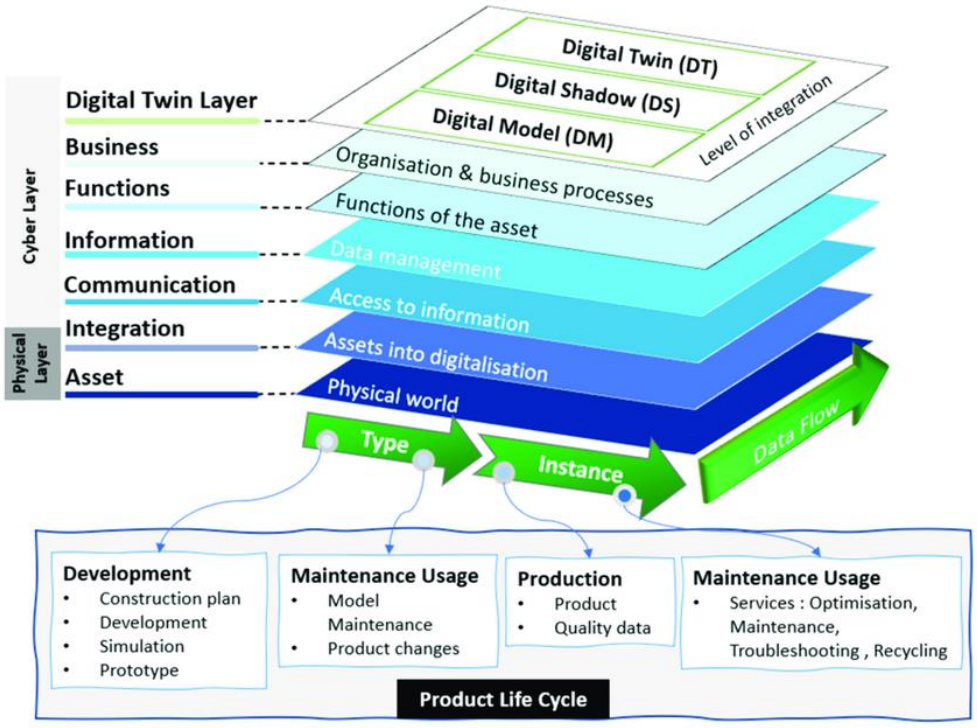


Рисунок 2.1 – Тривимірна модель RAMI 4.0 у контексті цифрових двійників (її опис наданий далі)

Архітектура RAMI 4.0 має структуру що складається з кількох ієрархічних шарів, що відображаються у трьох вимірах, які охоплюють життєвий цикл системи, ієрархію від найнижчого рівня пристроїв до підприємства в цілому, і різноманітні аспекти від функцій до комунікацій. Завдяки розділенню складних процесів на прості ієрархічні шари, забезпечується простота керування, безпека даних. Її основні виміри це:

Ієрархічні рівні (рис 2.2), або іноді – рівень потоку даних. Від пристрою/актуатора до станції, виробничої лінії, заводу та підприємства як єдиного цілого. Архітектура передбачає інтеграцію на всіх рівнях.

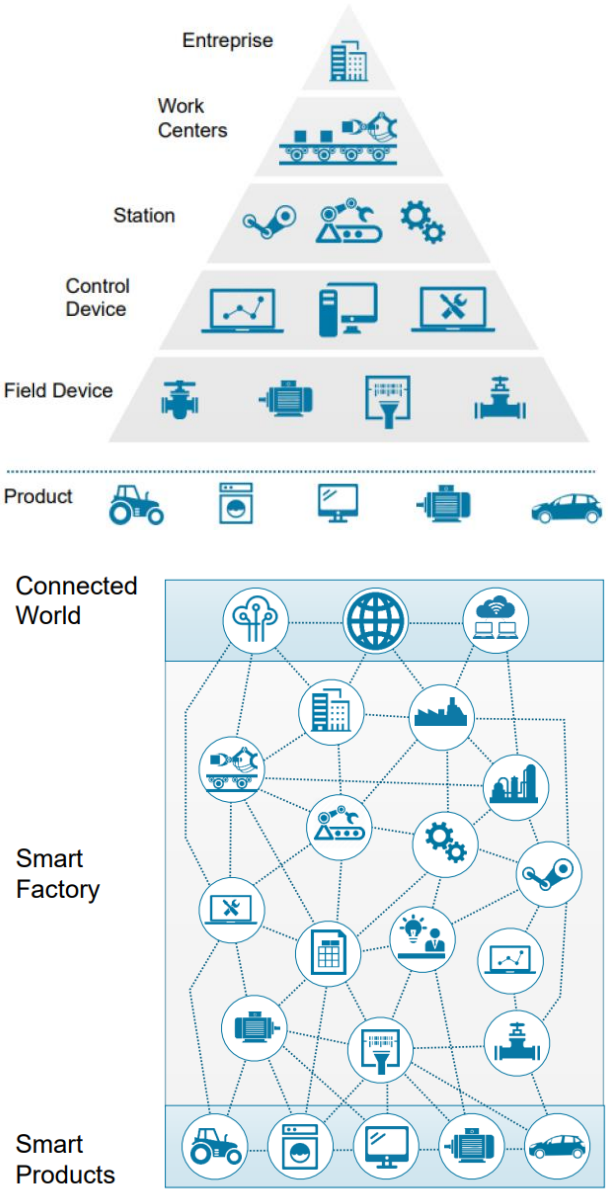


Рисунок 2.2 – Ієрархічний рівень (а – RAMI 3.0, б – RAMI 4.0)

Але важлива різниця між версією моделі RAMI у контексті Industry 3.0 та Industry 4.0. Стара модель більше спиралася на «залізо», навіть програмні функції та застосунки покладаються на hardware. Комунікація відбувалася суто за ієрархією, а продукт ізолювався від фактичних систем управління.

Industry 4.0 змінила підхід. Системи стали гнучкими та взаємопов'язаними у спільній мережі зв'язків, де кожний девайс обмінюється даними один з одним. І продукт перестав бути відділеним шаром комунікації, натомість об'єднавшись з хмарними сервісами та пристроями керування і атрибутами передачі даних.

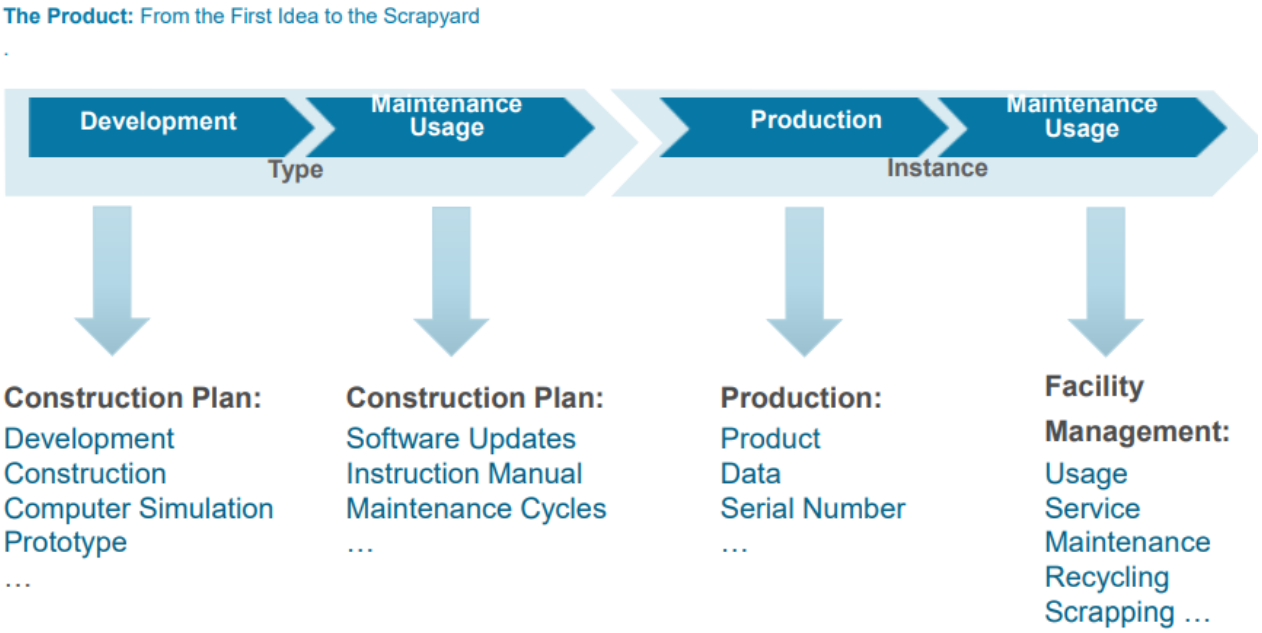


Рисунок 2.3 – Рівень життєвого циклу продукту RAMI 4.0

Другий вимір, шари життєвого циклу продукту (рис 2.3) – вони охоплюють усі етапи від концептуалізації та розробки виробу до його експлуатації та утилізації. Цифровий двійник повинен підтримувати цей безперервний цикл.

Цикл складається з чотирьох кроків (перші два – це кроки типу, останні два – це кроки окремих об'єктів).

– Розробка: Розробка. Збір системи. Комп'ютерна симуляція системи. Прототипування.

– Підтримка використання: Оновлення програм, інструкції користувачам та адміністрації, цикли підтримки.

– Продукція: Створення продуктів, оперування даними.

– Менеджмент: Використання продуктів, їх обслуговування та підтримка, за необхідності – оновлення запчастин, або утилізація.

Рисунок 2.4 – Рівень архітектури RAMI 4.0

Останній вимір – рівень архітектурних шарів (рис. 2.4), який на рисунку 2.1 відображає вертикальну вісь. Він відображає фактичну структуру майбутньої системи, від фізичного пристроя до цифрового двійника. Фактично це зв'язок «фізичний об'єкт – цифрові методи інтеграції – цифрові методи коммунікації – управління даними – виконання операцій над даними чи об'єктом – організаційні процеси – цифровий двійник + цифрова тінь (фактично симуляція роботи двійника) та цифрова модель.

Фізичний рівень буде представлений обладнанням SCARA-робота, конвеєрних ліній, датчиків тощо. Кожен компонент матиме свій цифровий двійник чи його аналог (наприклад, замість об'єктів конвеєра можуть бути умовні сірі коробки), що дозволить відстежувати його стан, параметри та взаємодію, а також стан усієї системи.

RAMI 4.0 гарантує інтеграцію фізичного та віртуального просторів на всіх рівнях промислової ієрархії від пристроїв до підприємства.

ЦД стане основою для моделювання, симуляції та оптимізації транспортно-розвантажувальних операцій, навіть з можливістю масштабування рішення відповідно до потреб бізнесу.

Загалом важливо ще наголосити важливість концепту цифрових двійників у структурі моделі Industry 4.0. Вони відіграють ключову роль у навчанні та підготовці персоналу в контексті Industry 4.0.

Віртуальні копії виробничих процесів та обладнання дозволяють співробітникам виконувати практику або навчатися у безпечному середовищі. Така можливість є особливо цінною для небезпечних або критично важливих операцій, де помилки можуть мати серйозні наслідки.

Завдяки своїй природі цифрові двійники полегшують співпрацю та віддалене управління в рамках Industry 4.0. Експерти можуть одночасно з кількох пристроїв взаємодіяти з цифровим двійником, вносити зміни у його модель, або навіть тестувати різні сценарії. Особливо це може бути корисно для міжнародних компаній або у випадках, коли фізичний доступ до обладнання є небажаним через певні причини.

Для забезпечення максимальної ефективності та сумісності цифрових двійників у промислових умовах треба дотримуватися стандартів та забезпечувати взаємодію між різними системами та платформами. Це і було описано у моделі RAMI 4.0.

2.2 Розробка структури комплексу технічних та програмних засобів для реалізації системи

Як і описувалося у розділі 2.1, технічний процес загалом є вже реалізованною моделю транспортно-розвантажувальних операцій на базі SCARA-робота, яка зроблена на базі мікроконтролера S7-1200, який живить модуль PM1207 та мікропроцесора CPU 1215c.

Мікроконтролер S7-1200 (рис. 2.5) – це компактний програмований логічний контролер для автоматизації малих та середніх систем. Має модульну конструкцію, що дозволяє гнучку конфігурацію вхідних/вихідних модулів.

- Процесор: 32-бітний RISC
- Робоча пам'ять: 50-150 Кбайт (залежно від моделі)
- Ємність завантаження програми: 1-4 Мбайт
- Дискретні входи/виходи: 6-1012
- Аналогові входи/виходи: 0-16

Рисунок 2.5 – Мікроконтролер S7-1200

Модуль живлення PM1207 (рис 2.6) – це однофазний блок живлення для контролерів S7-1200. Забезпечує стабільне живлення CPU та додаткових модулів.

- Вхідна напруга: 120/230 В змінного струму
- Вихідна напруга: 24 В постійного струму
- Вихідний струм: 2 А (постійний), 4 А (пусковий)
- Сертифікати: CE, cULus, FM, C-Tick

Рисунок 2.6 – Модуль живлення PM1207

Процесорний модуль CPU 1215C – це центральний процесорний модуль контролера S7-1200. Виконує керуючу програму, забезпечує зв'язок з периферією.

– Робоча пам'ять: 125 Кбайт (для програми), 60 Кбайт (для даних)

– Введення/виведення: 14 дискретних входів, 10 дискретних виходів, 2 аналогових входів

– Інтерфейси: PROFINET, PROFIBUS, промисловий Ethernet

– Тактова частота: 60 МГц

При цьому важливою частиною реалізації майбутньої системи є саме аналіз, розробка та реалізація саме програмних засобів системи, оскільки створення цифрового двійника – це складна задача, та при цьому поєднує у собі програмування, моделювання, симулювання, візуалізацію, створення інтерфейсу, налаштування потоку даних, налаштування фізичного об’єкту та оптимізацію тощо.

Для того, щоб зрозуміти, які технології та загалом який комплекс засобів необхідний для створення подібної системи, необхідно знову поглянути на базовий

принцип роботи цифрових двійників та проаналізувати можливі рішення на ринку технологій.

Як ми розглядали раніше, дослідники пропонують модель SLADTA (Six-Layer Architecture for Digital Twins with Aggregation), що складається з шости основних шарів:

1. Девайси та сенсори;
2. Джерела даних;
3. Локальні репозиторії даних;
4. IoT-мости;
5. Хмарні сервіси, що містять дані про об’єкт;
6. Емуляція, моделювання, симулювання, візуалізація.

Якщо проаналізувати схему (рис. 1.4), на якій зображено принцип роботи даного фреймворку та архітектури, то бачимо, що фактично саме так ми і планували реалізувати цифрового близнюка транспортної системи та SCARA-робота.

2.3 Розробка схеми мережних інформаційних потоків інтегрованої автоматизованої системи, її структура та напрямку досліджень

Фактично на основі усієї інформації можна сформувати структуру комплекса засобів, необхідного для реалізації системи цифрового двійника, а також схему цієї структури, яка відображає типи інформаційних потоків у даній інтегрованій автоматизованій системі (рис. 2.7).

Схема мережних інформаційних потоків має забезпечити безперебійний обмін даними в режимі реального часу, а також надійність та безпеку передачі інформації.

На нижньому рівні ієрархії знаходяться різноманітні датчики та виконавчі пристрої: ваговий датчик, лазерні датчики на конвеєрній стрічці, сервоприводи SCARA-руки та двигун конвеєра. Вони підключені до програмованого логічного контролера S7-1200, який збирає дані з датчиків через TIA та відправляє команди

керування на виконавчі механізми. Для передачі даних з контролера на веб-сервер доцільно використовувати промисловий протокол OPC UA, який забезпечує надійну та безпечну комунікацію.

Дані з OPC UA сервера на контролері S7-1200 передаватимуться через мережевий шлюз на сервер Node-RED. Цей програмний комплекс виконуватиме первинну обробку (наприклад, задання даних або базове читання даних) та агрегацію даних, а також буде відповідати за візуалізацію основних параметрів системи в зручному для користувача інтерфейсі. Для комунікації між контролером та Node-RED можна використовувати протокол MQTT, який є легким та ефективним для передачі даних в режимі реального часу. Окрім цього, великим плюсом є те, що завдяки очним заняттям, є великий досвід роботи з цим протоколом.

Наступним кроком буде можлива передача даних з Node-RED на хмарну платформу, де вони будуть зберігатися та оброблятися для моніторингу стану системи, виявлення аномалій та оптимізації процесів. Обмін даними між локальним сервером Node-RED та хмарою також доцільно здійснювати за допомогою протоколу MQTT через проміжний MQTT-брокер, наприклад, Mosquitto.

Важливою складовою системи є цифровий двійник, який буде реалізований у потужному двигуні візуалізації, такому як Unreal Engine 5. Він отримуватиме дані про стан фізичної системи з хмарної платформи за допомогою підписки на відповідні теми MQTT-брокера. На основі цих даних цифровий двійник відображатиме віртуальну 3D-модель транспортно-розвантажувальної системи, її поточний стан та дозволятиме керувати процесами у віртуальному середовищі.

Зворотний зв'язок від цифрового двійника до фізичної системи забезпечуватиметься шляхом публікації команд керування на відповідні теми MQTT-брокера. Ці команди будуть передаватися через хмарну платформу та Node-RED на OPC UA сервер контролера S7-1200, який виконуватиме необхідні дії на фізичному обладнанні.

Така схема мережних інформаційних потоків дозволить створити повноцінну інтегровану автоматизовану систему управління транспортно-

розвантажувальними операціями на базі SCARA-робота, яка поєднуватиме фізичний та віртуальний простори, забезпечуватиме моніторинг, аналіз та оптимізацію процесів, а також надаватиме можливість керування системою як з локальних інтерфейсів, так і віддалено через цифровий двійник. Тобто, усі умови, необхідні для правильного функціонування системи.

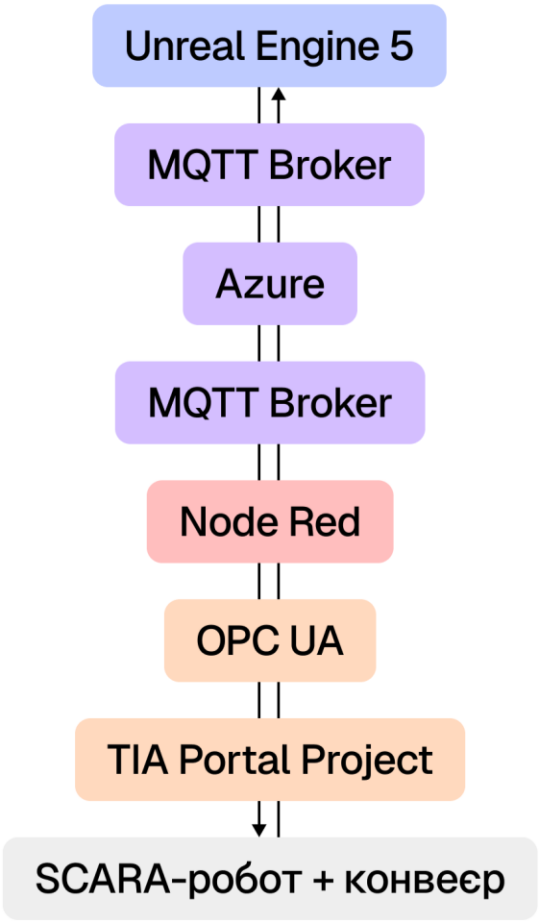


Рисунок 2.7 – Структура комплексу засобів цифрового двійника SCARA-робота

Але це лише один із варіантів можливої реалізації роботи. На кваліфікаційній роботі також необхідно провести дослідження щодо можливих альтернатив реалізації поєднання Node Red та Unreal Engine.

Серед можливих альтернатив буде використання протоколу WebSocket та розгортання його серверу (рис. 2.8). Цей протокол є новітньою альтернативою переліченим вище альтернативам (MQTT тощо) та дуже часто використовується

для з’єднання Unreal Engine з ігровими серверами, або для з’єднання з клієнтами що містять дані про цифрові двійники. Окрім протоколу WebSocket можно коротко розглянути потенціал використання HTTP, навести його переваги та мінуси.

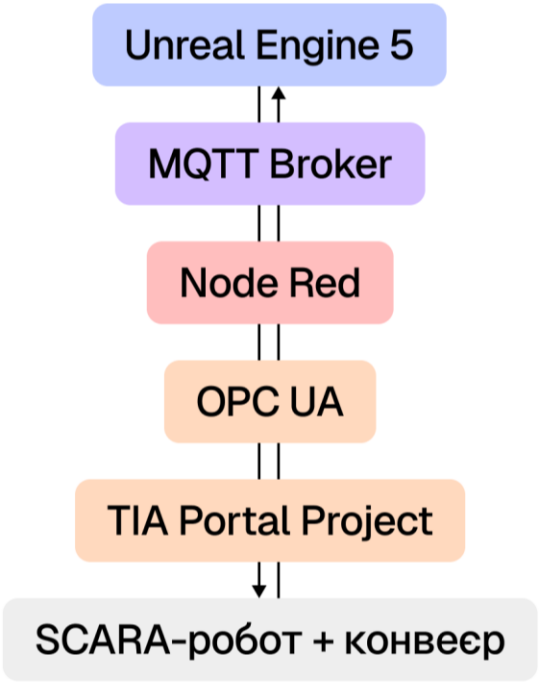


Рисунок 2.8 – Структура комплексу засобів цифрового двійника SCARA-робота, дослідження альтернативної реалізації 1

Також у дослідженні відбудеться вивчення функціоналу Azure, і за випадку, коли технічна реалізація задач буде неможливою, треба буде розглянути пряме з’єднання через MQTT брокера з можливим розглядом альтернатив хмарного сервісу (рис. 2.9).

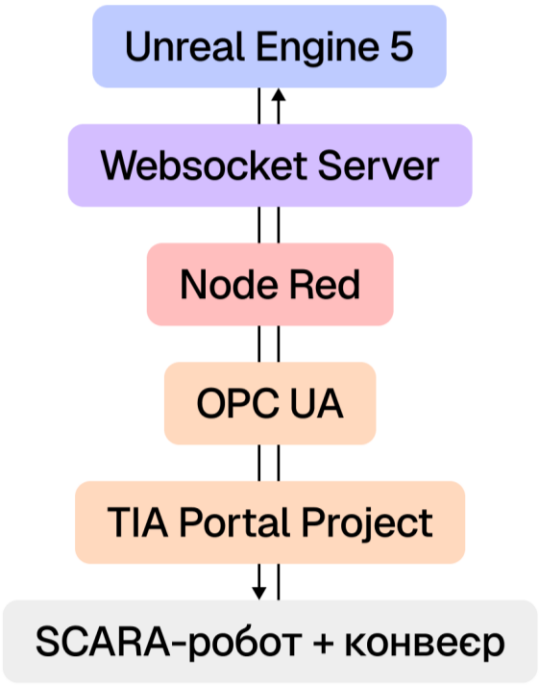


Рисунок 2.9 – Структура комплексу засобів цифрового двійника SCARA-робота, дослідження альтернативної реалізації 2

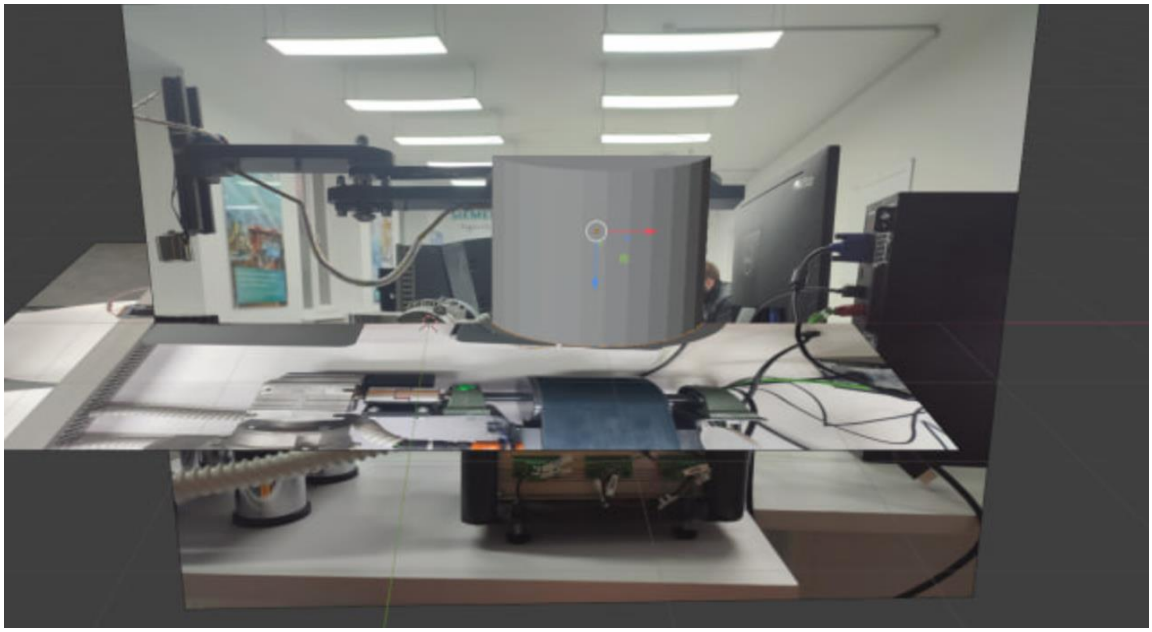
2.3 Опис налаштування та розгортання компонентів системи

Для того щоб забезпечити працюючий зв'язок даних з хмарною платформою (WebSocket/Azure) необхідно буде налаштувати MQTT-брокер, наприклад, Mosquitto, або відповідний WebSocket сервер, за гайдлайнами або допоміжними матеріалами. Node-RED повинен бути інтегрований з цим брокером/сервером для того, щоб дані коректно передавалися далі до UE5.

На хмарній платформі Microsoft Azure потрібно буде створити необхідні ресурси та, можливо, бази даних, налаштувати з'єднання з MQTT-брокерами у двосторонньому форматі, та забезпечити безпечний доступ до цих даних.

Нарешті, у двигуні візуалізації, такому як Unreal Engine 5, необхідно створити віртуальну 3D-модель транспортно-розвантажувальної системи (рис 2.10), налаштувати її параметри та забезпечити інтеграцію з MQTT-брокером для отримання даних з хмарної платформи.

а)



б)

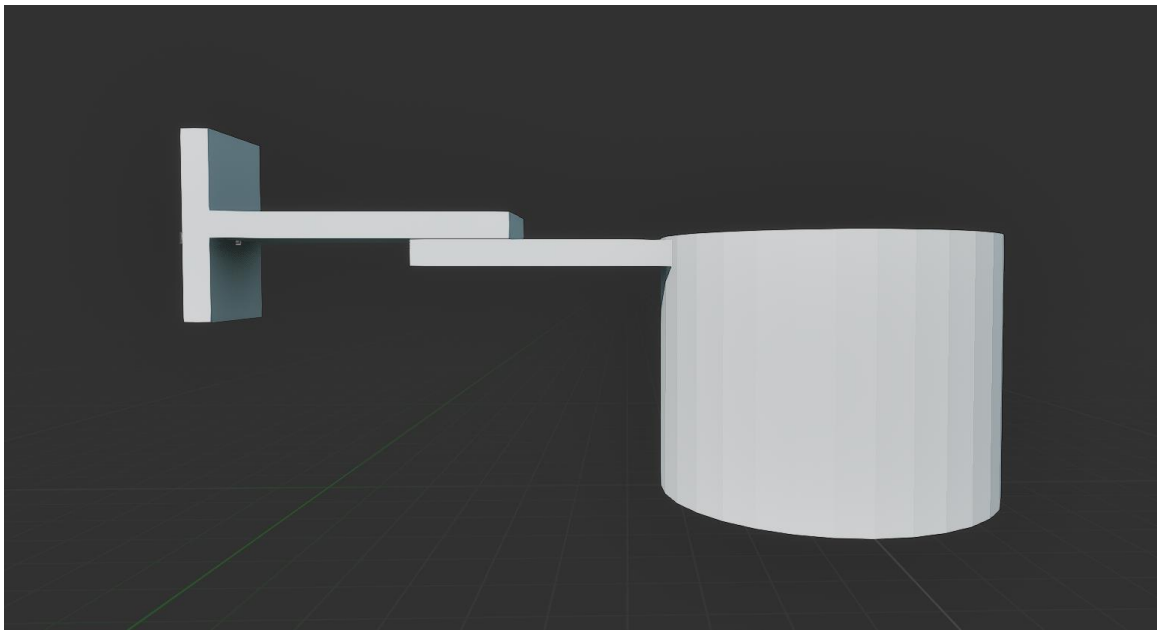


Рисунок 2.10 – Створення 3D моделі (а – використання референсних зображень для правильних пропорцій, б – зроблена 3D модель комплексу розвантажувальних операцій SCARA + конвеєр)

Важливо також і візуалізація коректного відображення стану фізичної системи на основі отриманих даних, а саме – забезпечити можливість керування системою через віртуальний інтерфейс. Основна частина роботи буде пов’язана саме с двигуном візуалізації, його програмуванням, плагінами та системами. Тому необхідно у візуальній програмі Figma створити скетч інтерфейсу, за яким буде потім створено реальний (рис 2.11).

Окрім цього, під час розгортання компонентів даної системи, необхідно уважно тестувати її поетапно, перевіряючи коректність передачі даних по інформаційному потоку, та при необхідності проводячи дебаггінг програм.

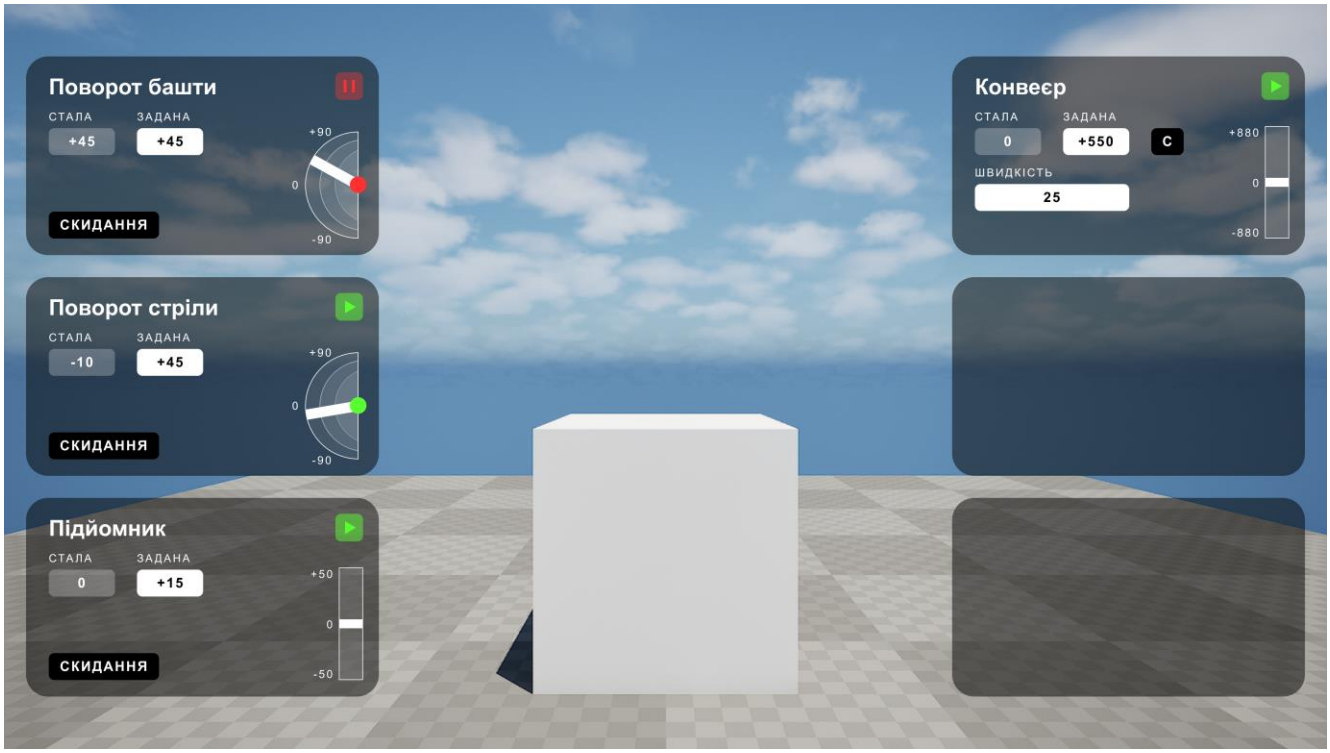


Рисунок 2.11 – Прототип інтерфейсу користувача цифрового двійника, зроблений у програмі Figma

Для перевірки можливості читання даних через OPC UA клієнт UaExpert (рис. 2.12) було виконано наступні кроки: запуск UaExpert, додавання серверу OPC UA, введення URL-адреси та порту серверу контролера, вибір кінцевої точки для підключення, прийняття сертифіката сервера. Після цього клієнт UaExpert отримав доступ до адресного простору сервера та відповідних тегів контролера, таких як стан електромагніту, задана швидкість обертання та поточна швидкість. Ці дані відображалися в області «Data Access View», що дозволяло переглядати та змінювати їх в режимі реального часу.

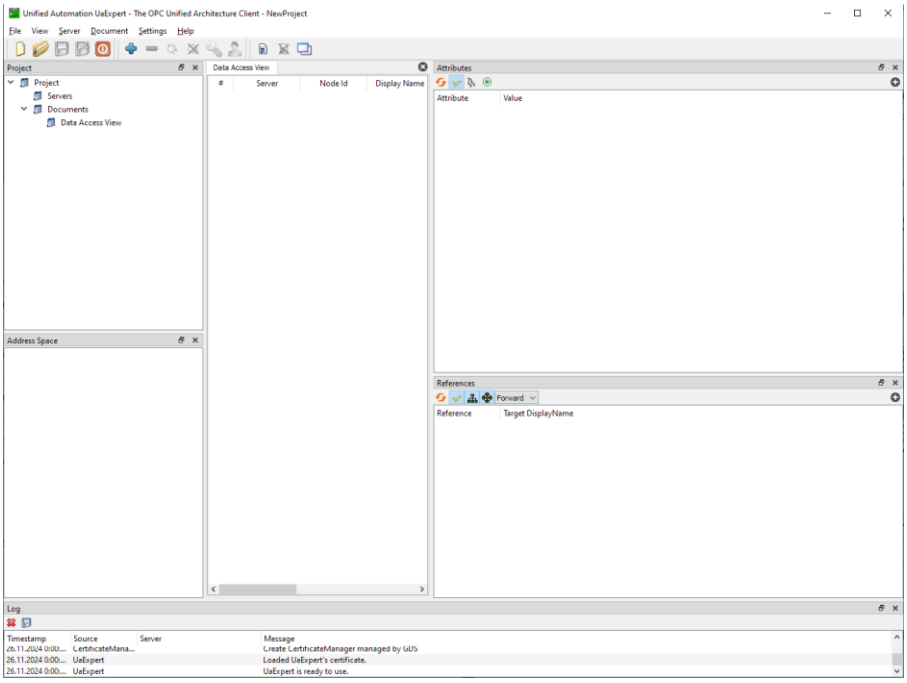


Рисунок 2.12 – Інтерфейс налаштованої та встановленної програми UaExpert

Отже, за умови виконання усіх зазначених дій розгортання компонентів, можна приступати до створення технічної реалізації проєкту. Втім, залишається важливим написати приблизну структуру інструкції користувача системи, яка задасть напрям для роботи нашого проєкту, а також описати важливий принцип з'єднання Node Red та Unreal Engine, який стане у нагоді при реалізації реального проєкту.

2.4 Інструкція користувача системи

– На даний момент для коректної роботи системи, треба запевнитися в тому, що SCARA-модель працює правильно та підключена до OPC UA серверу.

– При вході до інтерфейсу, можна за допомогою миші обрати текстбокси для зміни заданих значень, а також вимкнути чи увімкнути бажані елементи стенду. При затисканні середньої кнопки миші, програма переходить до режиму візуалізації, дозволяючи обертати камеру навколо 3д-моделі, яка відображає положення реального об’єкту у реальному часі.

– При зміні значень, можна запевнитися на прикладі реальної моделі, що відбувається зміна положення системи. Окрім цього, можна прослідкувати, як інформація проходить через інформаційний потік: статус інформації буде відображатися спочатку у консолі Node-Red, потім у інтерфейсі UA Expert, а потім у таблиці моніторингу TIA Portal.

– Або навпаки, якщо значення не задаються, а зчитуються (наприклад, у випадку зміни значень не через інтерфейс Node-Red, а через WinCC Unified у самій TIA Portal (рис 2.12).

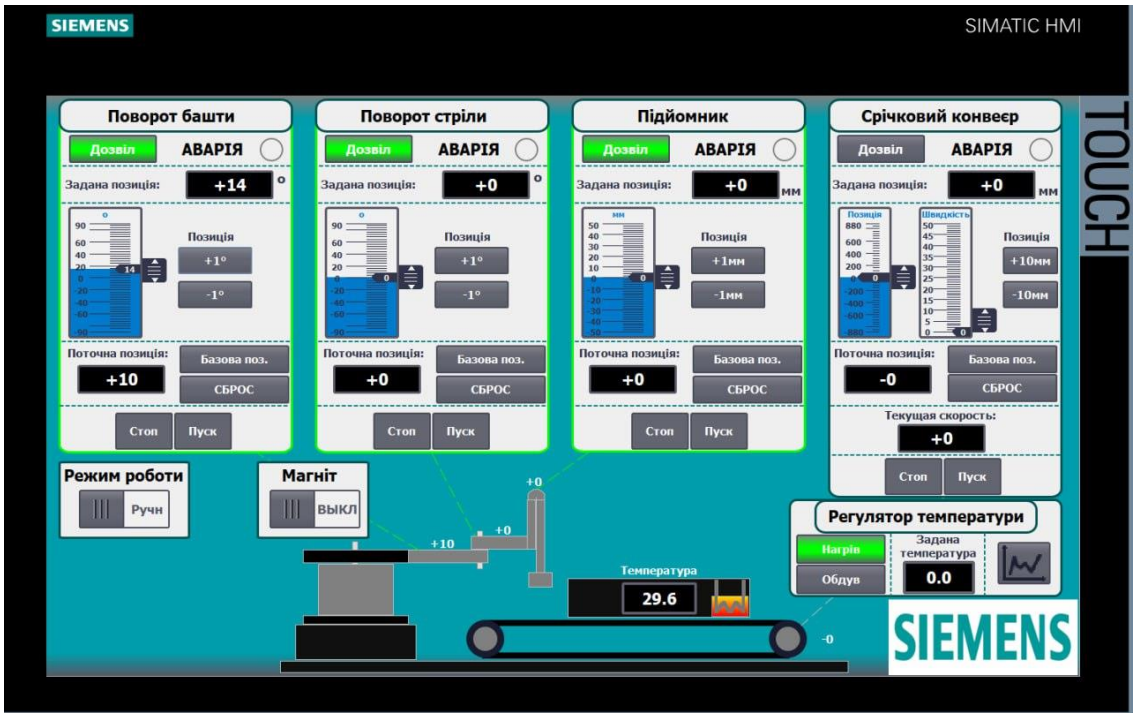


Рисунок 2.12 – Інтерфейс WinCC у TIA Portal

2.5 Структура та принцип з’єднання архітектур Node Red та Unreal Engine

Поєднання UE5 та Node Red за допомогою MQTT дозволить реалізувати динамічну систему обміну даними, яка так важлива для цифрових двійників. Node Red це потужний інструмент візуального програмування, який буде обробляти дані з обох сторін системи, і направляти, маршрутизувати повідомлення до UE5, який буде вже трьохвимірно візуалізовувати об’єкт.

Для встановлення зв'язку між ними використовується протокол MQTT (Message Queuing Telemetry Transport). Він працює за принципом "публікація-

підписка", і у випадку нашої системи є двосторонньою системою передачею даних (тобто відправляти і отримувати повідомлення будуть і Node-RED, і Unreal Engine).

Структура з'єднання починається з налаштування MQTT-брокера, який служить посередником між Node-RED та Unreal Engine. В Node-RED створюються вузли для публікації повідомлень на певні теми цифрового двійника MQTT. Це ми робили у попередніх пунктах.

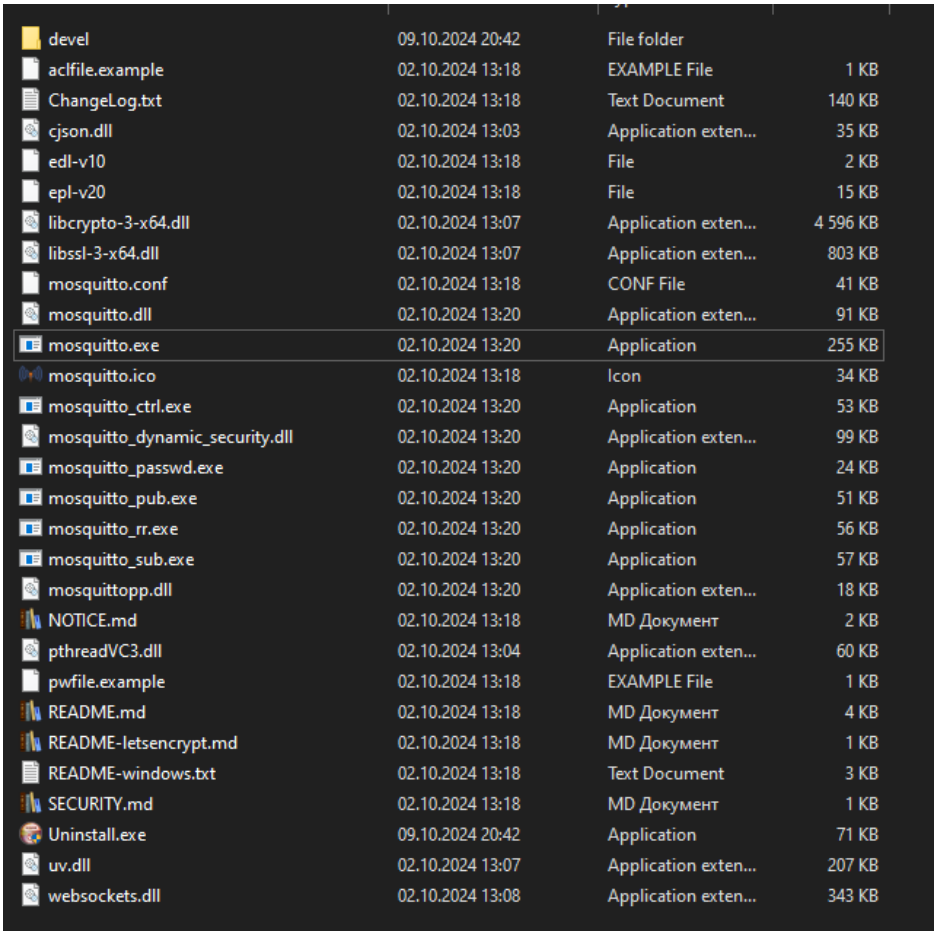


Рисунок 2.13 – Директорія MQTT-брокеру

Сам брокер можна використовувати локальний від Mosquitto. Після його завантаження через git, можна побачити з ним папку (рис. 2.13). Для його запуску та активації необхідно активувати .exe файл, попередньо налаштувавши .ini файли із вказуванням бажаних портів MQTT-брокера.

З боку Unreal Engine інтеграція MQTT може бути реалізована за допомогою спеціальних плагінів або власних інтегрованих реалізацій на основі сокетів (вбудована у власне двигун, але зі свома мінусами).

Наприклад, можна використовувати плагін RedTalaria, який розширює функціональність Unreal Engine для роботи з мережевими протоколами. В грі створюються відповідні Blueprint-вузли або C++ класи, які підписуються на теми MQTT та обробляють отримані повідомлення.

Для забезпечення надійності зв'язку можна реалізувати механізм підтвердження отримання повідомлень. Node-RED може очікувати підтвердження від Unreal Engine після кожного важливого оновлення. Якщо підтвердження не надходить протягом певного часу, система може спробувати повторно відправити дані або сповістити оператора про проблему зв'язку.

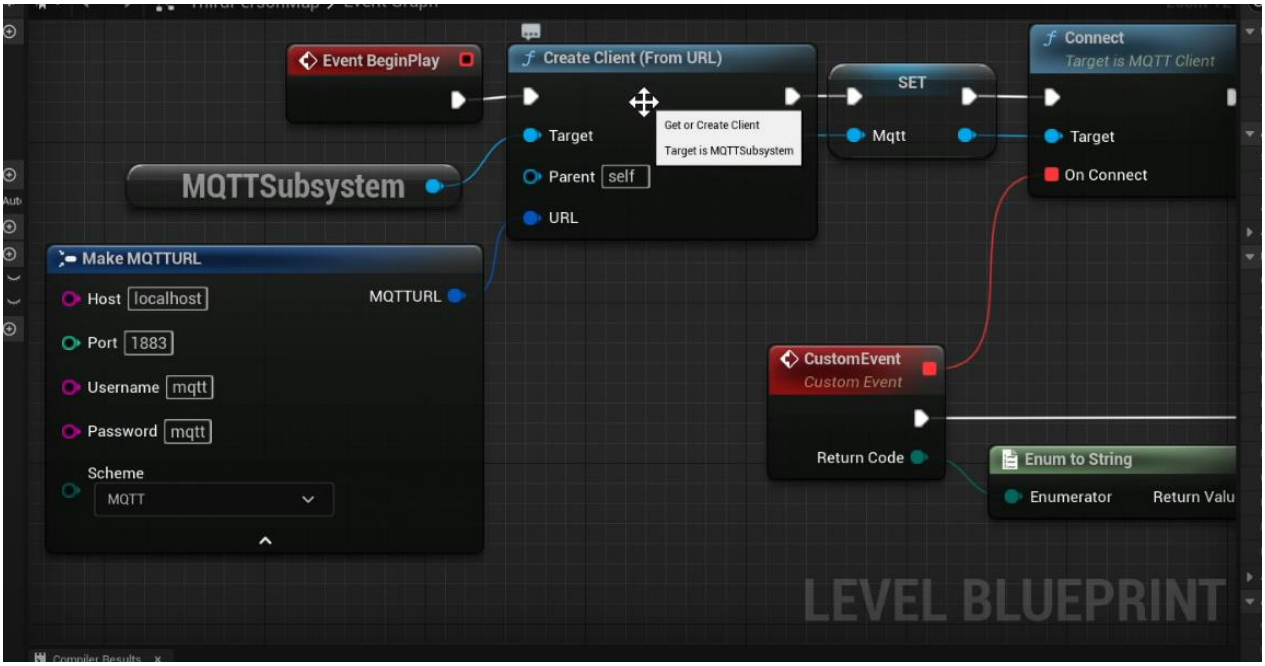


Рисунок 2.13 – MQTT-функції у візуальному редакторі скриптів Blueprints

Принцип з'єднання полягає в тому, що Node-RED відправляє дані через MQTT-брокер (наприклад Mosquitto), а Unreal Engine отримує ці дані та реагує на них відповідно до логіки програми. Наприклад, зміна стану реального об'єкта, зафіксована Node-RED, може призвести до візуальних змін у віртуальному середовищі Unreal Engine, рухаючи SCARA-робота або конвеєр.

Реалізація може виглядати наступним чином:

– Оброблені дані публікуються через MQTT на тему «ReadList» (тобто вхідні дані).

– В Unreal Engine MQTT Manager (плагін) підписується на відповідні теми та отримує дані.

– Після отримання нових даних, MQTT Manager викликає відповідні функції для оновлення стану віртуального середовища, оновлення UI, 3д-візуалізації.

– В Unreal Engine 5 через інтерфейс задається значення, наприклад зміна положення башні SCARA-руки.

– Ці дані формуються спеціальними функціями та передаються на MQTT-брокер на тему «WriteList» (тобто вихідні дані).

– Далі ці дані вже власне обробляються через Node-RED.

Створена архітектура поєднання Node-RED та UE5 загалом здатна забезпечити дуже високу гнучкість та масштабованість системи. У подальшому це дозволдить легко додавати нові датчики, розширювати функціональність віртуального середовища, масштабувати технологічний об'єкт, і все це без необхідності значних змін в основній структурі системи, тобто через банальні розширення функцій.

Висновки до розділу:

У цьому розділі було наведено структурну модель реалізації цифрового двійника моделі комплексу розвантажувальних операцій, а саме архітектуру RAMI 4.0, яка забезпечить максимальну ефективність та масштабованість системи.

Після цього було розроблено та розглянуто структуру комплексу технічних та програмних засобів для реалізації системи, порівняно її з аналогічною структурою SLADTA та сформульовано основні технічні шари розробляємої системи.

Ця структура була подробно описана, на основі неї було створено схему мережних інформаційних потоків інтегрованої автоматизованої системи. Втім, було зазначено, що необхідно провести дослідження альтернативних варіантів реалізації системи, а саме окрім MQTT-Azure ще і Websocket та MQTT-UE. Для них також було створено схему мережних інформаційних потоків.

Після цього було проведено налаштовування та розгортання компонентів системи, створення прототипів її інтерфейсу, створення 3Д-моделей. Було створено інструкцію користувача, яка задає приблизну структуру управління цифровим двійником у середовищі Unreal Engine 5. Проведено аналіз з’єднання компонентів Node Red та Unreal Engine 5 через різні протоколи передачі – Azure, MQTT, Websocket, HTTP. Подробно проаналізовано та описано за документацією принцип з'єднання Unreal Engine 5 та MQTT.

РОЗДІЛ 3
ПРОГРАМНО-ТЕХНІЧНА РЕАЛІЗАЦІЯ ЦИФРОВОГО ДВІЙНИКА КОМПЛЕКСУ МОДЕЛЮВАННЯ ТРАНСПОРТНО-РОЗВАНТАЖУВАЛЬНИХ ОПЕРАЦІЙ

3.1 Програмна реалізація з’єднання програмного забезпечення TIA Portal та OPC UA Server

Для створення повноцінної системи цифрового двійника, спочатку треба забезпечити двостороннє з’єднання ігрового двигуна та програмного забезпечення, яке управляє реальним об’єктом комплексу транспортно-розвантажувальних операцій.

У нашому випадку це Unreal Engine 5 та, відповідно, TIA Portal, у якому була заздалегідь реалізована програма керування виробничим об’єктом.

Для забезпечення цього з’єднання треба, відповідно до схем зображених на рис. 2.7-2.9, було реалізувати двосторонній зв’язок TIA Portal та UaExpert за допомогою серверу OPC UA.

Для цього у програмі TIA Portal спочатку необхідно було активувати OPC UA Server та обрати необхідну ліцензію для коректної роботи програми (рис 3.1).

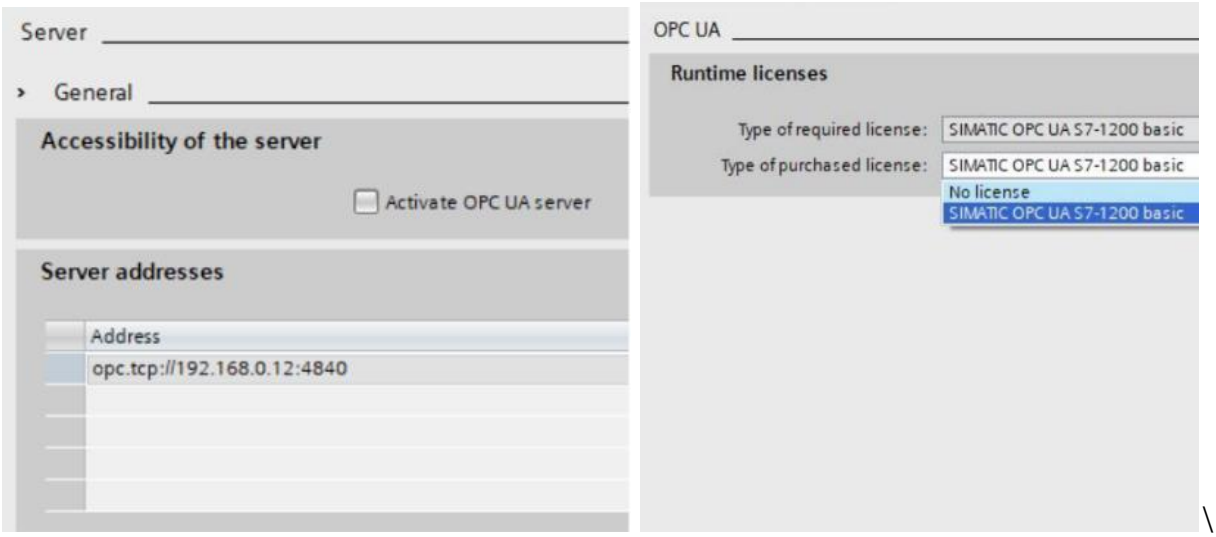


Рисунок 3.1 – Активація OPC UA Server (а – активація серверу, б – задання необхідної ліцензії для коректного функціонування)

Після цього програма була скомпільована. Зроблено перехід до розділу OPC UA communication. Там необхідно було створити новий інтерфейс серверу, який і буде надалі передаватися масивом значень до UE5, або навпаки, приймати керуюче значення.

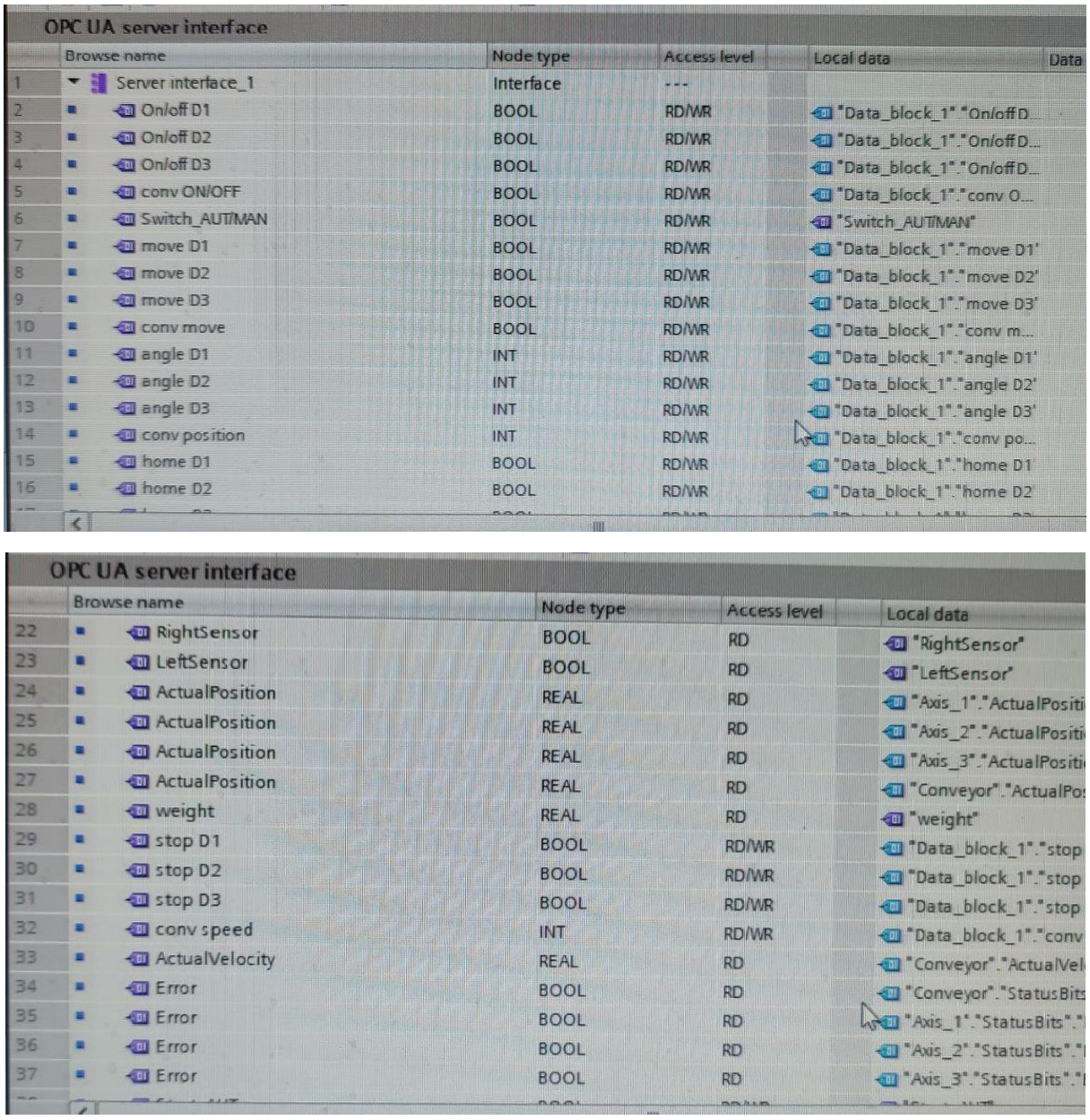
Втім, для створення інтерфейсу серверу необхідно визначити, які самі змінні необхідно використовувати для керування об'єктом, або навпаки, його візуалізації.

Для цього була створена таблиця у Microsoft Excel, у якій зазначено назву змінної у UE5, назву змінної у TIA Portal Database. Окрім цього, зазначено для чого використовується змінна, який формат вона має мати, та який тип вона має (Read – для тих, які тільки передаються на двигун для візуалізації, Write – для тих, які окрім передачі на двигун ще і призначені для керування об'єктом). Ця таблиця зображена на рисунку 3.2.

	A	B	C	D	E
1	HMI Name	PLC Name	Type	Function	W or R
2	Enabled_1	Data_block_1."On/off D1"	Bool	ВКЛ вежі	W
3	Enabled_2	Data_block_1."On/off D2"	Bool	ВКЛ стріли	W
4	Enabled_3	Data_block_1."On/off D3"	Bool	ВКЛ підйомника	W
5	Enabled_4	Data_block_1."conv ON/OFF"	Bool	ВКЛ конвеєра	W
6	AUTO/Manual	"Switch_AUT/MAN"	Bool	Авто/ручний	W
7	Start_1	Data_block_1."move D1"	Bool	Пуск вежі	W
8	Start_2	Data_block_1."move D2"	Bool	Пуск стріли	W
9	Start_3	Data_block_1."move D3"	Bool	Пуск підйомника	W
10	Start_4	Data_block_1."conv move"	Bool	Пуск конвеєра	W
11	Position1	Data_block_1."angle D1"	Int	Позиція вежі	W
12	Position2	Data_block_1."angle D2"	Int	Позиція стріли	W
13	Position3	Data_block_1."angle D3"	Int	Позиція підйомника	W
14	Position4	Data_block_1."conv position"	Int	Позиція конвеєра	W
15	Starting_position_1	Data_block_1."home D1"	Bool	Старт позиція вежі	W
16	Starting_position_2	Data_block_1."home D2"	Bool	Старт позиція стріли	W
17	Starting_position_3	Data_block_1."home D3"	Bool	Старт позиція підйомника	W
18	Starting_position_4	Data_block_1."conv home"	Bool	Старт позиція конвеєра	W
19	Conveyor_forward	Data_block_1."conv move"	Bool	Рух конвеєра	W
20	Conveyor_stop	Data_block_1."conv stop"	Bool	Стоп конвеєра	W
21	Magnet_ON	magnit	Bool	Магніт	W
22	Blank_1	RightSensor	Bool	Датчик 1	R
23	Blank_2	LeftSensor	Bool	Датчик 2	R
24	Actual_position_1	Axis_1.ActualPosition	Real	Факт Вежі	R
25	Actual_position_2	Axis_2.ActualPosition	Real	Факт Стріли	R
26	Actual_position_3	Axis_3.ActualPosition	Real	Факт Підйомника	R
27	Actual_position_4	Conveyor.ActualPosition	Real	Факт Конвеєра	R
28	Weight_blank	weight	Real	Вага	R
29	Stop_1	Data_block_1."stop D1"	Bool	Стоп вежі	W
30	Stop_2	Data_block_1."stop D2"	Bool	Стоп стріли	W
31	Stop_3	Data_block_1."stop D3"	Bool	Стоп підйомника	W
32	Data_block_1_convspeed	Data_block_1."conv speed"	Int	Швидкість конвеєра	W
33	Conveyor_ActualVelocity	Conveyor.ActualVelocity	Real	Актуальна швидкість конвеєра	R
34	Conveyor_StatusBits_Error	Conveyor.StatusBits.Error	Bool	Помилка конвеєра	R
35	Axis_1_StatusBits_Error	Axis_1.StatusBits.Error	Bool	Помилка вежі	R
36	Axis_2_StatusBits_Error	Axis_2.StatusBits.Error	Bool	Помилка стріли	R
37	Axis_3_StatusBits_Error	Axis_3.StatusBits.Error	Bool	Помилка підйомника	R
38	Start_AUT	Start_AUT	Bool	Старт АВТ	W
39	Stop_AUT	Stop_AUT	Bool	Стоп АВТ	W

Рисунок 3.2 – Таблиця Microsoft Excel з усіма необхідними параметрами системи

Маючи таблицю з усіма необхідними параметрами системи, було створено інтерфейс серверу, що містив у собі усі ці значення (рис 3.3).



OPC UA server interface

	Browse name	Node type	Access level	Local data	Data
1	Server interface_1	Interface	---		
2	On/off D1	BOOL	RD/WR	"Data_block_1"."On/off D...	
3	On/off D2	BOOL	RD/WR	"Data_block_1"."On/off D...	
4	On/off D3	BOOL	RD/WR	"Data_block_1"."On/off D...	
5	conv ON/OFF	BOOL	RD/WR	"Data_block_1"."conv O...	
6	Switch_AUTMAN	BOOL	RD/WR	"Switch_AUTMAN"	
7	move D1	BOOL	RD/WR	"Data_block_1"."move D1"	
8	move D2	BOOL	RD/WR	"Data_block_1"."move D2"	
9	move D3	BOOL	RD/WR	"Data_block_1"."move D3"	
10	conv move	BOOL	RD/WR	"Data_block_1"."conv m...	
11	angle D1	INT	RD/WR	"Data_block_1"."angle D1"	
12	angle D2	INT	RD/WR	"Data_block_1"."angle D2"	
13	angle D3	INT	RD/WR	"Data_block_1"."angle D3"	
14	conv position	INT	RD/WR	"Data_block_1"."conv po...	
15	home D1	BOOL	RD/WR	"Data_block_1"."home D1	
16	home D2	BOOL	RD/WR	"Data_block_1"."home D2	

OPC UA server interface

	Browse name	Node type	Access level	Local data
22	RightSensor	BOOL	RD	"RightSensor"
23	LeftSensor	BOOL	RD	"LeftSensor"
24	ActualPosition	REAL	RD	"Axis_1"."ActualPositi
25	ActualPosition	REAL	RD	"Axis_2"."ActualPositi
26	ActualPosition	REAL	RD	"Axis_3"."ActualPositi
27	ActualPosition	REAL	RD	"Conveyor"."ActualPos
28	weight	REAL	RD	"weight"
29	stop D1	BOOL	RD/WR	"Data_block_1"."stop
30	stop D2	BOOL	RD/WR	"Data_block_1"."stop
31	stop D3	BOOL	RD/WR	"Data_block_1"."stop
32	conv speed	INT	RD/WR	"Data_block_1"."conv
33	ActualVelocity	REAL	RD	"Conveyor"."ActualVel
34	Error	BOOL	RD	"Conveyor"."StatusBits
35	Error	BOOL	RD	"Axis_1"."StatusBits"."I
36	Error	BOOL	RD	"Axis_2"."StatusBits"."I
37	Error	BOOL	RD	"Axis_3"."StatusBits"."I

Рисунок 3.3 – Інтерфейс серверу із значеннями системи керування

Після цього через UaExpert було перевірено з’єднання з сервером OPC UA. Проведено тестування за допомогою HMI-інтерфейсу об’єкту, змінювання значень та перевірка їх актуальності у клієнті UA Expert. За якоїсь причини, масив змінних типу REAL, а саме: ActualPosition 1, Actual Position 2, Actual Position 3, Actual Position 4 (нередаговані позиції вежі, стріли, підйомника та конвеєра відповідно) – увесь цей масив викликав помилку, при якій значення змінних не зчитувалося. Для запобігання цьому у головному програмному блоці Main було створено ще одну

гілку, у якій було створено SCL блок з кодом-інструкцією, який присваював значення цього масиву ActualPosition новим змінним з дуже схожими назвами і ідентичними типами Real/RD. І вже ці змінні передавалися до OPC UA, на заміну тим, що містяться у датабазах об'єкту. Завдяки цьому помилку було виправлено, і всі змінні почали відображатися коректно.

Отже, зв'язок TIA Portal та OPC UA зроблено, однак для повноцінної реалізації програми необхідно зробити задання значень до TIA через OPC UA. І для цього необхідно реалізувати програмний код у Node Red.

3.2 Програмна реалізація програми у Node Red та його з'єднання з TIA Portal та Unreal Engine

3.2.1 Програмна реалізація зв'язку Node Red та TIA Portal

Для правильного масштабування проєкту та доступу з будь-якої системи, проєкт Node Red було створено локально, за допомогою завантаження дистрибутиву Node Red за допомогою git на локальний SSD-диск. Там також були збережений і сам проєкт у форматі .json (рис 3.4).

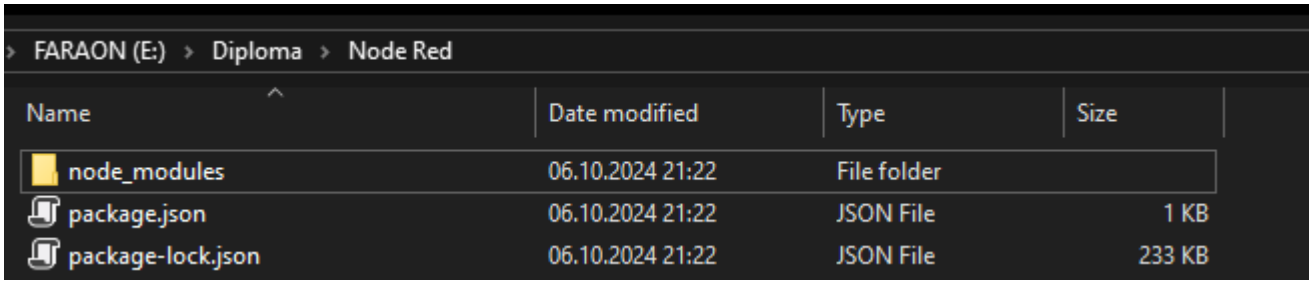


Рисунок 3.4 – Дистрибутив Node Red

Після цього створюється код для зчитування змінних з OPC UA. Спочатку цей функціонал був реалізований за допомогою великого масиву функцій (рис 3.5), які формували запит на читання змінних, потім присваювали їх відповідним глобальним змінним, і після цього формували єдиний .json payload, який вже передавався далі за інформаційним потоком.

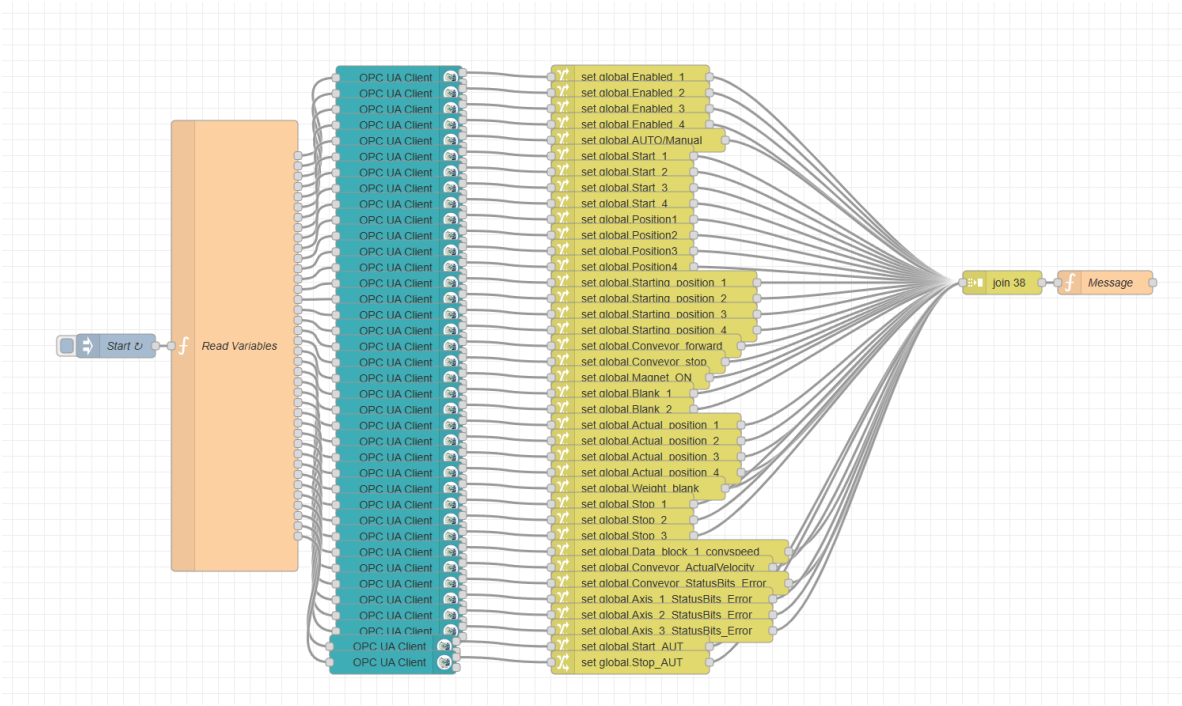


Рисунок 3.5 – Нодовий код для зчитування масиву значень OPC UA

```
let msg35 = {};
let msg36 = {};
let msg37 = {};
let msg38 = {};

msg1.topic = `ns=4;s=Data_block_1."On/off D1"`;
msg2.topic = `ns=4;s=Data_block_1."On/off D2"`;
msg3.topic = `ns=4;s=Data_block_1."On/off D3"`;
msg4.topic = `ns=4;s=Data_block_1."conv ON/OFF"`;
msg5.topic = `ns=4;s=Switch_AUT/MAN`;
msg6.topic = `ns=4;s=Data_block_1."move D1"`;
msg7.topic = `ns=4;s=Data_block_1."move D2"`;
msg8.topic = `ns=4;s=Data_block_1."move D3"`;
msg9.topic = `ns=4;s=Data_block_1."conv move"`;
msg10.topic = `ns=4;s=Data_block_1."angle D1"`;
msg11.topic = `ns=4;s=Data_block_1."angle D2"`;
msg12.topic = `ns=4;s=Data_block_1."angle D3"`;
msg13.topic = `ns=4;s=Data_block_1."conv position"`;
msg14.topic = `ns=4;s=Data_block_1."home D1"`;
msg15.topic = `ns=4;s=Data_block_1."home D2"`;
msg16.topic = `ns=4;s=Data_block_1."home D3"`;
msg17.topic = `ns=4;s=Data_block_1."conv home"`;
msg18.topic = `ns=4;s=Data_block_1."conv move"`;
msg19.topic = `ns=4;s=Data_block_1."conv stop"`;
msg20.topic = `ns=4;s=magnit`;
msg21.topic = `ns=4;s=RightSensor`;
msg22.topic = `ns=4;s=LeftSensor`;
msg23.topic = `ns=4;s=Axis_1.ActualPosition`;
msg24.topic = `ns=4;s=Axis_2.ActualPosition`;
msg25.topic = `ns=4;s=Axis_3.ActualPosition`;
msg26.topic = `ns=4;s=Conveyor.ActualPosition`;
msg27.topic = `ns=4;s=weight`;
msg28.topic = `ns=4;s=Data_block_1."stop D1"`;
msg29.topic = `ns=4;s=Data_block_1."stop D2"`;
msg30.topic = `ns=4;s=Data_block_1."stop D3"`;
msg31.topic = `ns=4;s=Data_block_1."conv speed"`;
msg32.topic = `ns=4;s=Conveyor.ActualVelocity`;
msg33.topic = `ns=4;s=Conveyor.StatusBits.Error`;
msg34.topic = `ns=4;s=Axis_1.StatusBits.Error`;
msg35.topic = `ns=4;s=Axis_2.StatusBits.Error`;
msg36.topic = `ns=4;s=Axis_3.StatusBits.Error`;
msg37.topic = `ns=4;s=Start_AUT`;
msg38.topic = `ns=4;s=Stop_AUT`;

return [msg1, msg2, msg3, msg4, msg5, msg6, msg7, msg8, msg9,
```

Рисунок 3.6 – Програмний код для зчитування масиву значень OPC UA

Втім, цей метод отримання даних виявився неактуальним, оскільки може бути виконаний на базі потужних OPC UA серверів. Шляхом дослідження було виявлено, що фізичний комплекс управління розвантажувальними операціями та

його сервер підтримує одночасне з’єднання лише чотирьох (включно) клієнтів. Через це у процесі дослідження було прийняте рішення реструктуризувати програму та створити нову реалізацію, яка задіює лише два клієнти, а саме – читання значень та запис значень.

У результаті фінальна реалізація з'єднання зображена на рисунку 3.7.

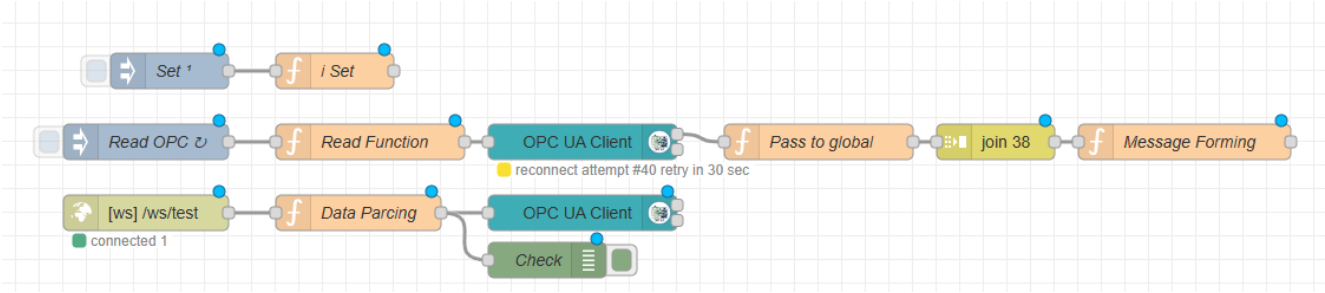


Рисунок 3.7 – Нодовий код для зчитування масиву значень OPC UA, фінальна реалізація

Необхідно розглянути принцип його роботи. Спочатку при запуску роботи Node-Red клієнту автоматично створює глобальну змінну i, яка робить відлік запросів до OPC UA серверу.

```
1   let i = global.get('i');
2
3   for (i; i<39; i++) {
4       msg.topic = `ns=3,s=${i}`;
5       msg.payload = i;
6       if (i < 38) global.set('i', i+1);
7       if (i > 37) global.set('i', 1);
8       node.send(msg);
9   }
10
11  return msg;
```

Рисунок 3.8 – Код формування запросів до OPC UA

Кожні 0.1 секунду робиться виклик до OPC UA серверу та формується код запросу функцією Read Function (рис 3.8). Отримується значення зі змінної i та отримується відповідне наступне значення пакету даних. Після цього змінна i

збільшується на 1, або скидається. OPC UA Client налаштований на режим Read Multiple.

Після цього робиться запрос до OPC UA, який видає бажане значення змінної. Відповідно до поточного положення змінної i, значення записується у відповідну глобальну змінну функцією Pass to global (рис 3.9).

```
let i = global.get('i');

switch (i) {
        case 1:
            global.set('Enabled_1', msg.payload);
            break;
        case 2:
            global.set('Enabled_2', msg.payload);
            break;
        case 3:
            global.set('Enabled_3', msg.payload);
            break;
        case 4:
            global.set('Enabled_4', msg.payload);
            break;
        case 5:
            global.set('AUTO_Manual', msg.payload);
            break;
        case 6:
            global.set('Start_1', msg.payload);
            break;
        case 7:
            global.set('Start_2', msg.payload);
            break;
        case 8:
            global.set('Start_3', msg.payload);
            break;
        case 9:
            global.set('Start_4', msg.payload);
```

Рисунок 3.9 – Запис значення до відповідної глобальної змінної (частково)

Після цього сигнал направлений на блок join 38, який фактично слугує лічильником. Коли він налічує 38 значень, то він дає сигнал на блок Message Forming. Цей блок на основі глобальних змінних формує .json повідомлення (рис 3.10), яке далі буде передаватися через інтернет-протокол до UE5.

```
var Enabled_1 = global.get('Enabled_1');
var Enabled_2 = global.get('Enabled_2');
var Enabled_3 = global.get('Enabled_3');
var Enabled_4 = global.get('Enabled_4');
var AUTO_Manual = global.get('AUTO/Manual');
var Start_1 = global.get('Start_1');
var Start_2 = global.get('Start_2');
var Start_3 = global.get('Start_3');
var Start_4 = global.get('Start_4');
var Position1 = global.get('Position1');
var Position2 = global.get('Position2');
var Position3 = global.get('Position3');
var Position4 = global.get('Position4');

msg.payload = {
    "Enabled_1": Enabled_1,
    "Enabled_2": Enabled_2,
    "Enabled_3": Enabled_3,
    "Enabled_4": Enabled_4,
    "AUTO_Manual": AUTO_Manual,
    "Start_1": Start_1,
    "Start_2": Start_2,
    "Start_3": Start_3,
    "Start_4": Start_4,
    "Position1": Position1,
    "Position2": Position2,
    "Position3": Position3,
    "Position4": Position4,
    "Starting_position_1": Starting_position_1,
    "Starting_position_2": Starting_position_2,
    "Starting_position_3": Starting_position_3,
    "Starting_position_4": Starting_position_4,
    "Conveyor_forward": Conveyor_forward,
    "Conveyor_stop": Conveyor_stop,
    "Magnet ON": Magnet ON,
```

Рисунок 3.9 – Формування .json повідомлення (частково)

Коли Node Red отримує значення від UE5, воно передається у форматі «назва_змінної: значення». Для того, щоб відправити його на OPC UA та задати змінну, треба спочатку сформувати запит за допомогою парсингу отриманого повідомлення. Парсинг відбувається завдяки блоку Data Parcing (рис. 3.10).

```
const topicMapping = {
    "Enabled_1": `ns=4;s=1"`,
    "Enabled_2": `ns=4;s=2"`,
    "Enabled_3": `ns=4;s=3"`,
    "AUTO_Manual": `ns=4;s=4`,
    "Start_1": `ns=4;s=5"`,
    "Start_2": `ns=4;s=6"`,
    "Start_3": `ns=4;s=7"`,
    "Position1": `ns=4;s=8"`,
    "Position2": `ns=4;s=9"`,
    "Position3": `ns=4;s=10"`,
    "Starting_position_1": `ns=4;s=11`,
    "Starting_position_2": `ns=4;s=12`,
    "Starting_position_3": `ns=4;s=13`,
    "Conveyor_forward": `ns=4;s=14`,
    "Conveyor_stop": `ns=4;s=15`,
    "Magnet_ON": `ns=4;s=16`,
    "Stop_1": `ns=4;s=17`,
    "Stop_2": `ns=4;s=18`,
    "Stop_3": `ns=4;s=19`,
};


const input = msg.payload.trim();
const [key, value] = input.split(":").map(str => str.trim());


if (topicMapping[key]) {
    msg.topic = topicMapping[key];
    msg.payload = value;
    return msg;
} else {
    node.error(`Невідомий ключ: ${key}`);
    return null;
}
```

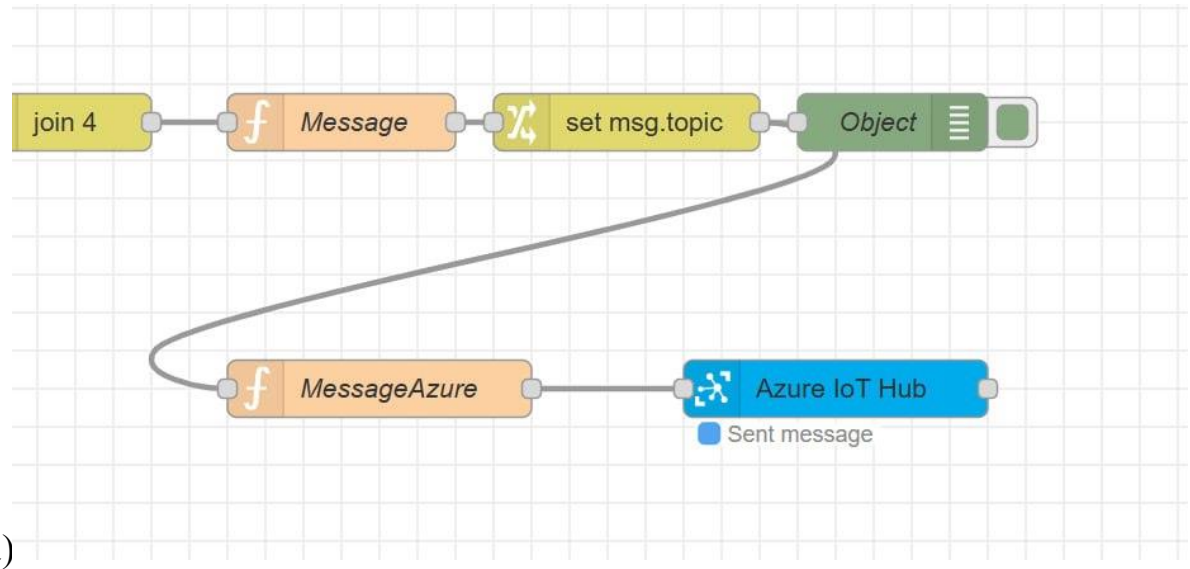
Рисунок 3.10 – Парсинг повідомлення з UE5

Після цього відбувається запит до OPC UA та встановлюється відповідне значення змінної.

За допомогою тестування на реальному об’єкті переконалися, що дана програмна реалізація дійсно працює і є робочим фінальним варіантом у контексті кваліфікаційної роботи. Втім, щоб реалізувати повну систему, необхідно ще забезпечити двосторонній зв’язок Unreal Engine 5 та Node Red, і з цією метою необхідно провести відповідне дослідження.

3.2.2 Дослідження програмної реалізації зв’язку Node Red та Unreal Engine 5 через зв’язку Azure-MQTT

Спочатку було зареєстровано акаунт у Azure Hub та обрано необхідну ліцензію, яка дозволяє використовувати IoT Hub та передавати туди дані. У Node Red реалізовано тестову гілку, яка передає значення до Azure IoT Hub за допомогою кастомної функції Message Azure (рис 3.11 а, б).



а)

```
1  msg.payload = {
2      "deviceId":"SCARA",
3      "key":"vyRneTwT0uHfg7f[redacted]Fde6nKEEosmd5Q=",
4      "protocol":"mqtt",
5      "data":msg.payload
6  }
7  return msg;
```

б)

Рисунок 3.11 – (а) Тестова гілка передачі даних до Azure IoT Hub через MQTT, (б) Функція формування повідомленя для коректної передачі даних

У результаті дослідження переконалися, що дані передаються коректно (рис 3.12).

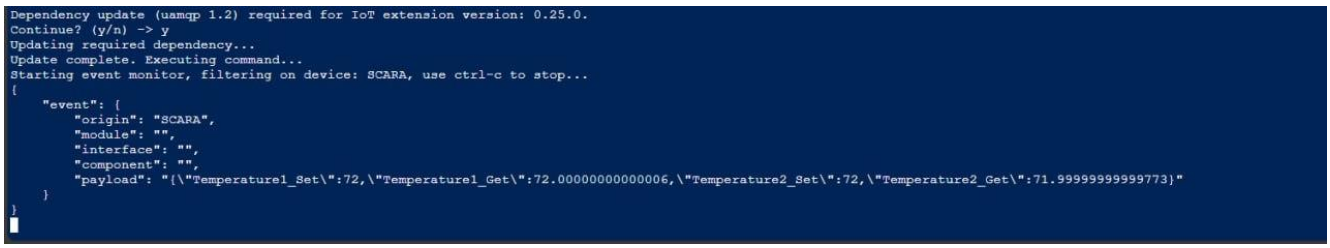


Рисунок 3.12 – Отримане повідомлення із даними

Втім, після детального вивчення функціоналу Azure IoT Hub, було зроблено висновок, що він не має зручного та ефективного функціоналу для автоматичного форвардингу MQTT-повідомлень між адресантами з повним збереженням контейнерів повідомлень.

У Azure існує функціонал логічних графічних функцій, тобто створення логічних дерев подій. Втім, він дуже перевантаженний та новітній, і у мережі немає ніяких аналогів чи згадувань реалізації форвардингу MQTT за допомогою цього функціоналу [44].

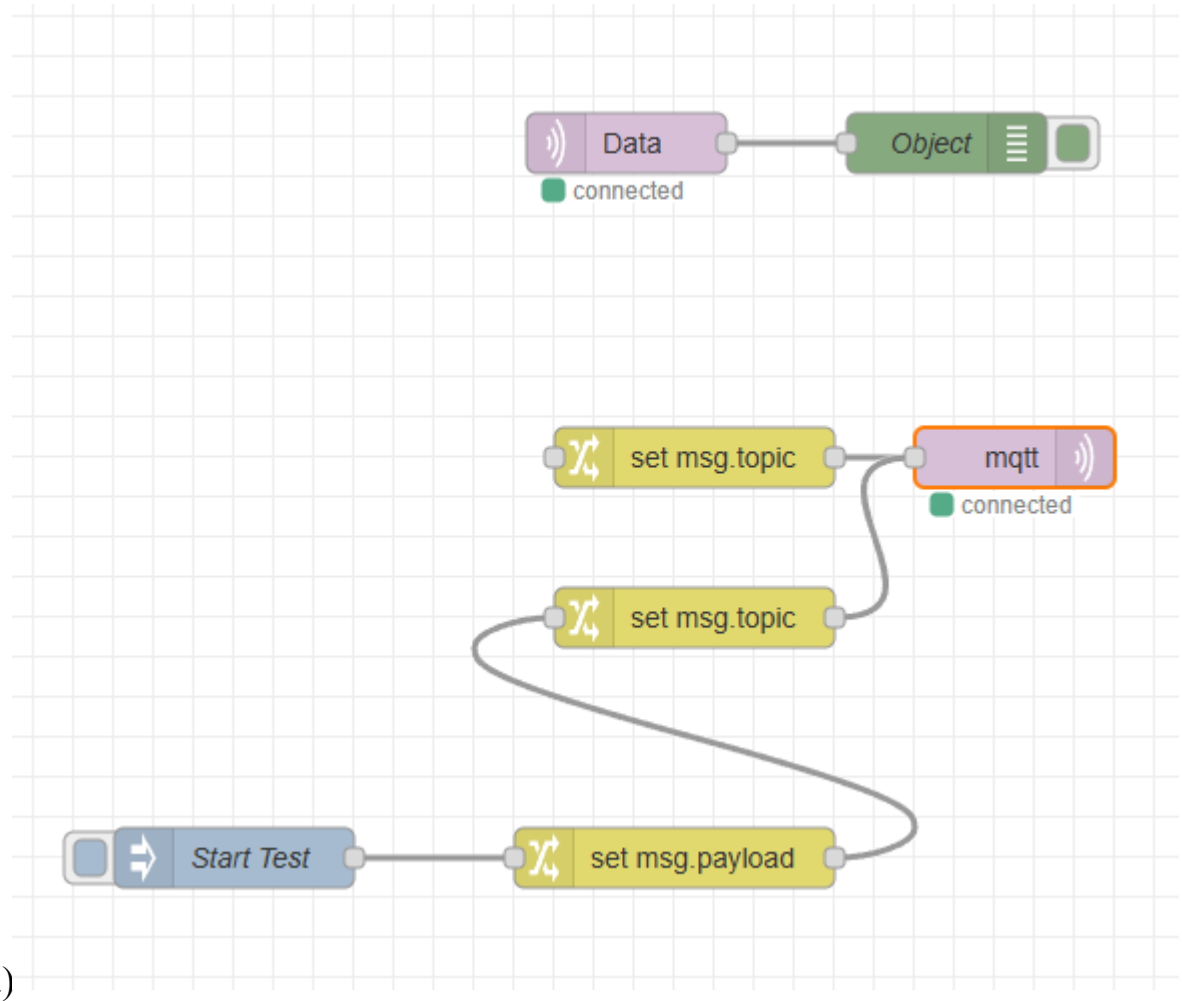
У Azure також існує функціонал MQTT-брокера (загалом те що потрібно для device-to-device з'єднання). Втім, він зараз у стадії альфа-тестування, не має навіть базового функціоналу, та дуже часто крашиться у користувачів. Окрім цього, за цим брокером у мережі немає документації. І як надалі буде зроблено висновки, кінцевий варіант роботи не буде мати MQTT-з'єднання взагалі через обмеження UE5. Це буде аргументовано надалі.

Отже, у результаті дослідження зв'язку Azure-MQTT було зроблено висновок, що дана технологія не підходить для реалізації у поточному проєкті, через що було зроблено пошук альтернатив, і у якості такої обрано з'єднання Node Red – MQTT Server – Unreal Engine 5.

3.2.3 Дослідження програмної реалізації зв'язку Node Red та Unreal Engine 5 через MQTT Server

Оскільки у пункті 2.5 була зроблена підготовка до реалізації MQTT-передачі даних від Node Red до UE5, можна почати реалізовувати цей функціонал (рис 3.13).

На Unreal Engine 5 подається тестовий пейлоад через MQTT із значеннями. MQTT налаштований на localhost із портом Node Red, але можливе і масштабування з переносом на повноцінний онлайн-сервер.



а)

(б)

Рисунок 3.13 – Схема передачі та отримання даних через MQTT (а), Отримані дані у UE5 через MQTT (б)

На стороні Unreal Engine 5 код приймання реалізований наступним чином. При запуску програми створюється об’єкт клієнту MQTT на основі MQTT URL, після чого відбувається підключення до серверу. Після успішного підключення відбувається підписка за темою ReadData (вхідні дані). Для того, щоб власне

отримати вхідні дані, використовується модуль MessageHandler, який делегує отримане повідомлення на створену подію Message, яка переводить отримане повідомлення у тип даних String (рис 3.14).

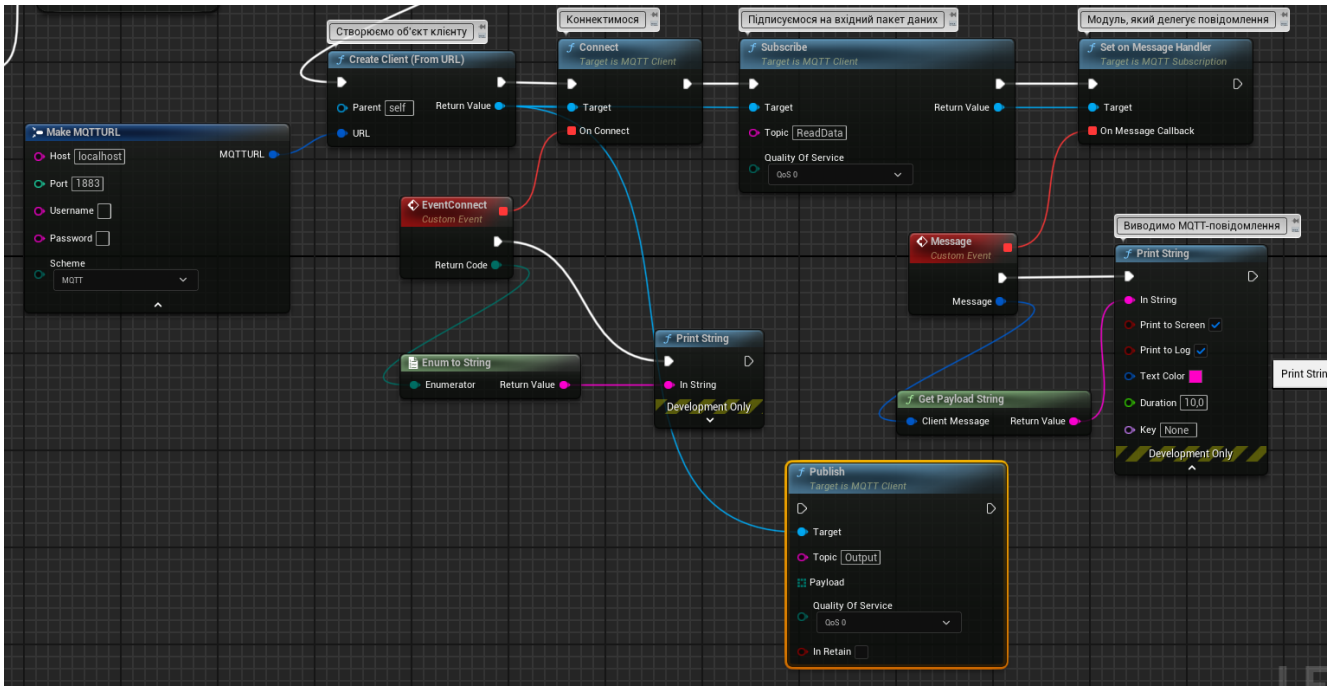


Рисунок 3.14 – Програмний код функціоналу MQTT у Unreal Engine 5

Під час виконання дослідження реалізації даного зв'язку виявилася дуже велика проблема, яка загалом поставила під сумнів можливість даної реалізації у середовищі UE5. Під час передачі довгого повідомлення (а у нас повідомлення-пейлод, який має передаватися раз на 0.1 секунду має 38 параметрів) або дуже частого повідомлення (х < 1 секунду), Unreal Engine 5 крашиться (рис 3.15). Відбувається це через помилку вбудованого експериментального модуля Blueprints в UE5, який власне і відповідає за роботу цих функцій-нодів, що забезпечують зв'язок з MQTT-сервером (рис 3.15).

Отже, шляхом дослідження виявилося, що використання MQTT-протоколу не є можливим, оскільки він не задовольняє потребі, сформульовані при проєктуванні системи цифрового двійника.

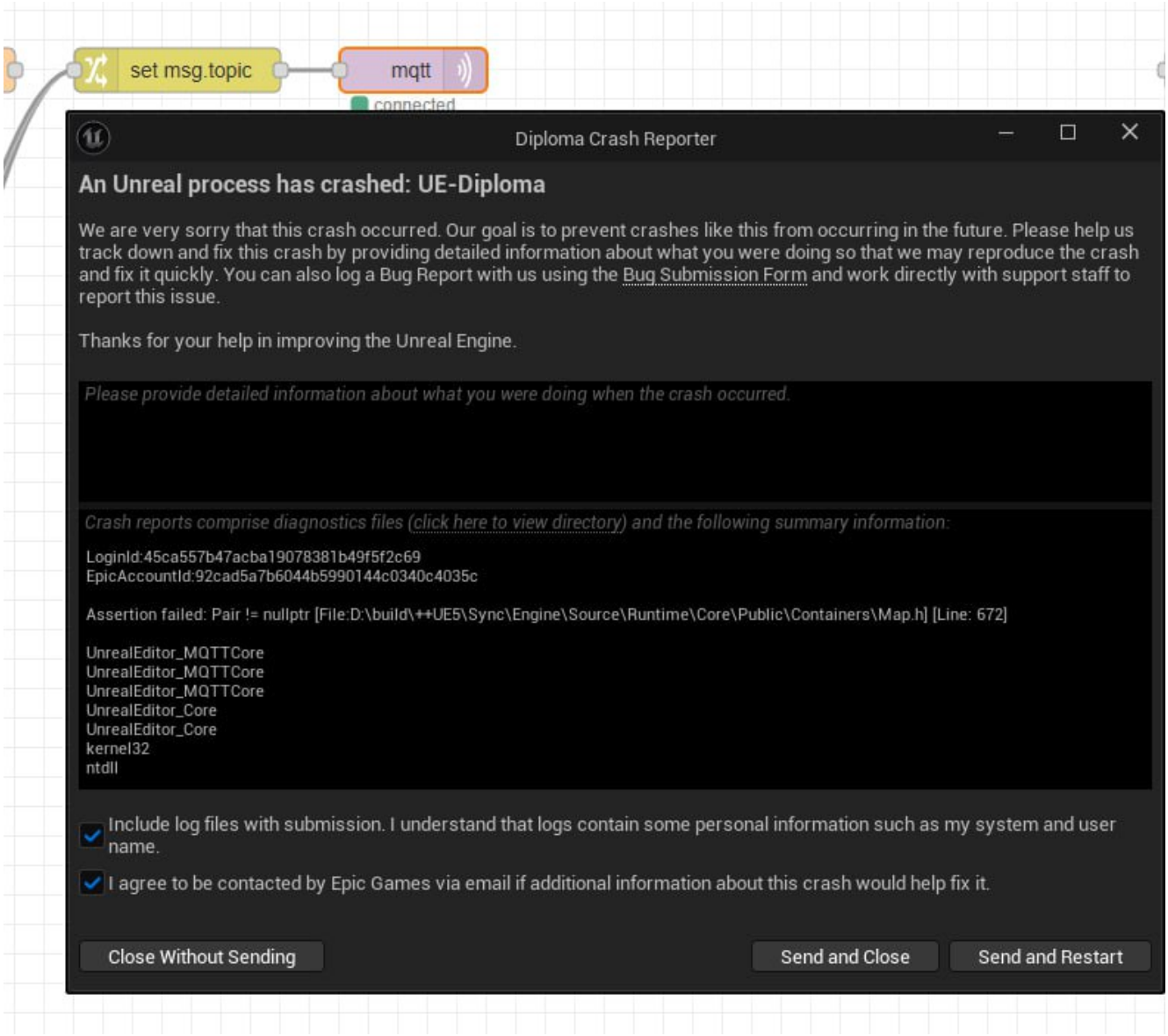


Рисунок 3.15 – Фатальна помилка UE5 у зв’язку з нарушенням присвоювання покажчика у модулі MQTT

3.2.3 Дослідження програмної реалізації зв’язку Node Red та Unreal Engine 5 через Websocket

Для реалізації на Node Red необхідно лише після модуля Message Forming (рис. 3.7) додати нод Websocket In, а перед Data Parcing – Websocket Out.

Реалізація на Unreal Engine 5 є більш складною задачею. Оскільки не існує безкоштовних плагінів на реалізацію даного зв’язку, а ліцензії коштують 60+ доларів, було прийняте рішення написати кастомну бібліотеку, яка буде містити усі необхідні функції для зв’язку UE5 та WebSocket.

Оскільки проєкт кваліфікаційної роботи має тип Blueprints, а нам необхідний повний функціонал C++, було створено клас-бібліотека C++ з вбудованим шаблоном Blueprint-Ready (щоб потім можна було використовувати ці функції в нодовому коді).

Фактично ця бібліотека UWebSocketTestGameInstance є не тільки бібліотекою функцій, а окремим інстанс-класом, який глобально у собі зберігає усі змінні, отримані з Websocket серверу, і звідки потім їх беруть відповідні функції.

Для того, щоб мати доступ до C++ функціоналу з серверами Websocket, а також до реалізації майбутньої функції парсингу даних, необхідно у коді проєкту UE5 додати параметри “WebSockets”, “Json” та “JsonUtilities” (рис 3.16).

Рисунок 3.16 – Параметри проєкту UE5 було модифіковано для реалізації необхідного функціоналу

Після цього у самому хедер-файлі бібліотеки було прописано список усіх необхідних функцій, функцій-блюпрінтів (функції парсингу вхідних даних, функції передачі даних на WebSocket сервер, функції формування даних для передачі), а також створені змінні, які можна буде потім використовувати у Blueprints (рис. 3.17).

Основний код програми, відповідно, наведено на рисунку 3.18.

```
public:
    virtual void Init() override;
    virtual void Shutdown() override;
    void TryReconnect();
    void ParseDataString(const FString& DataString);


    UPROPERTY(BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    FString ServerMessage;

    UFUNCTION(BlueprintCallable, Category = "String")
    static void NotifyServer(const FString& WString, UWebSocketTestGameInstance* GameInstance);

    UFUNCTION(BlueprintCallable, Category = "String")
    static FString StringBuilder(const FString& Name, const FString& Value);

    UPROPERTY(BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    bool Enabled_1;
    UPROPERTY(BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    bool Enabled_2;
    UPROPERTY(BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    bool Enabled_3;
    UPROPERTY(BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    bool Enabled_4;
```

Рисунок 3.17 – Хедер-файл UWebSocketTestGameInstance з оголошенням функцій та змінних проєкту

При ініціалізації програми інстанс створює з'єднання з WebSocket сервером та повідомляє про це у консолі, якщо з'єднання успішне. Якщо з'єднання неможливе, або розірване, програма намагається знову з'єднатися кожні 5 секунд.

Коли програма отримує повідомлення від WebSocket серверу, вона надсилає його до парсинг-функції, яка власне вже розподіляє все по необхідним змінним.

```
void UWebSocketTestGameInstance::Init() {
    Super::Init();

    if (!FModuleManager::Get().IsModuleLoaded("WebSockets")) {
        FModuleManager::Get().LoadModule("WebSockets");
    }

    WebSocket = FWebSocketsModule::Get().CreateWebSocket("ws://127.0.0.1:1880/ws/test");

    WebSocket->OnConnected().AddLambda([]() {
        GEngine->AddOnScreenDebugMessage(-1, 1.0f, FColor::Green, "Successfully connected");
        UE_LOG(LogTemp, Warning, TEXT("Successfully connected"));
        });

    WebSocket->OnConnectionError().AddLambda([this](const FString& Error) {
        GEngine->AddOnScreenDebugMessage(-1, 1.0f, FColor::Red, Error);
        this->TryReconnect();
        });

    WebSocket->OnClosed().AddLambda([](int32 StatusCode, const FString& Reason, bool bWasClean) {
        GEngine->AddOnScreenDebugMessage(-1, 1.0f, bWasClean ? FColor::Green : FColor::Red, "Connection closed " + Reason);
        });

    WebSocket->OnMessage().AddLambda([this](const FString& MessageString) {
        ServerMessage = MessageString;
        GEngine->AddOnScreenDebugMessage(-1, 1.0f, FColor::Cyan, "Received message: " + MessageString);

        ParseDataString(ServerMessage);
        });

    WebSocket->OnMessageSent().AddLambda([](const FString& MessageString) {
        GEngine->AddOnScreenDebugMessage(-1, 1.0f, FColor::Yellow, "Sent message: " + MessageString);
        });

    this->TryReconnect();
}
```

Рисунок 3.18 – Основний файл UWebSocketTestGameInstance

Давайте розглянемо основні функції бібліотеки UWebSocketTestGameInstance. Окрім вищезазначених, функція NotifyServer надсилає до серверу повідомлення формату String (яке вже потім парситься в Node Red). Перед відправкою, задля запобігання крашам, функція перевіряє існування інстансу та підключення до серверу) (рис 3.19). Функція буде використовуватися у середовищі програмування Blueprints.

```
void UWebSocketTestGameInstance::NotifyServer(const FString& WString, UWebSocketTestGameInstance* GameInstance)
{
    if (GameInstance)
    {
        if (GameInstance->WebSocket->IsConnected())
        {
            GameInstance->WebSocket->Send(WString);
        }
    }
}
```

Рисунок 3.19 – Функція NotifyServer

Функція StringBuilder (рис 3.20) будує повідомлення, яке потім направляється до NotifyServer. Оскільки парсинг у Node Red налаштований за схемою «Змінна: значення», то відповідно і FString має мати відповідне форматування. Функція також використовується у Blueprints.

```
FString UWebSocketTestGameInstance::StringBuilder(const FString& Name, const FString& Value)
{
    FString Notify = Name + ": " + Value;
    return Notify;
}
```

Рисунок 3.20 – Функція StringBuilder

Функція TryReconnect (рис. 3.21) перевіряє, чи є з’єднання з сервером, і якщо немає, намагається його установити кожні п’ять секунд.

```
void UWebSocketTestGameInstance::TryReconnect() {
    if (!WebSocket->IsConnected()) {
        UE_LOG(LogTemp, Warning, TEXT("Trying to reconnect..."));
        WebSocket->Connect();
        GetWorld()->GetTimerManager().SetTimer(ReconnectTimer, this, &UWebSocketTestGameInstance::TryReconnect, 5.0f, false);
    }
}
```

Рисунок 3.21 – Функція TryReconnect

Функція ParseDataString (рис. 3.22) автоматично викликається після отримання даних, конвертує отримані дані у json-об'єкт та далі парсить його та розподіляє значення по відповідним параметрам у інстансі.

```
void UWebSocketTestGameInstance::ParseDataString(const FString& DataString)
{
    UE_LOG(LogTemp, Warning, TEXT("Parsing data string: %s"), *DataString);

    // Десеріалізуємо строку JSON в об'єкт
    TSharedPtr<FJsonObject> JsonObject;
    TSharedRef<TJsonReader<TCHAR>> Reader = TJsonReaderFactory<TCHAR>::Create(DataString);

    if (FJsonSerializer::Deserialize(Reader, JsonObject) && JsonObject.IsValid())
    {
        UE_LOG(LogTemp, Warning, TEXT("Successfully parsed JSON"));

        if (JsonObject->HasField("Enabled_1"))
        {
            Enabled_1 = JsonObject->GetBoolField("Enabled_1");
        }
        if (JsonObject->HasField("Enabled_2"))
        {
            Enabled_2 = JsonObject->GetBoolField("Enabled_2");
        }
        if (JsonObject->HasField("Enabled_3"))
        {
            Enabled_3 = JsonObject->GetBoolField("Enabled_3");
        }
```

Рисунок 3.22 – Функція ParseDataString

Після тестування функціоналу даної бібліотеки та передачі даних до інтерфейсу, було зроблено висновок, що усе працює коректно (рис 3.23). Функції у UE5 в Blueprints-середовищі будуть розглянуті далі.

Рисунок 3.23 – Успішна передача даних від Node Red до UE5

Після передачі у UE5 було випробувано функціонал передачі даних у зворотньому порядку. Задача була протестована успішно (рис 3.24).

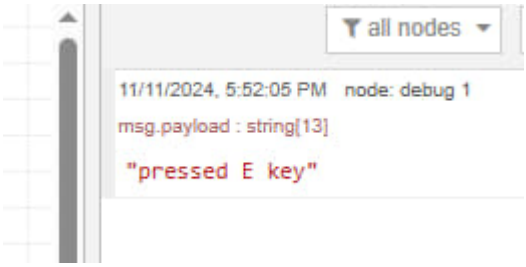


Рисунок 3.24 – Успішна передача даних від UE5 до Node Red

3.2.4 Висновки щодо проведених досліджень встановлення з’єднання Node Red та Unreal Engine 5

Після проведеного дослідження можна зробити наступні висновки: зв’язок Azure-MQTT не підходить для реалізації у поточному проєкті через те, що він не має необхідного ефективного функціоналу для форвардингу device-to-device повідомлень, малу документацію, та через загальну експериментальність модуля MQTT-брокера, який дуже часто крашиться.

Зв'язок MQTT не підходить через те, що при надсиланні великих повідомлень або повідомлень з високою частотою, викликає фатальні краші UE5, і виправити це не можна, оскільки до помилки призводить експериментальний MQTT-плагін, доступ до коду якого закритий.

Отже, найкращим варіантом для реалізації зв’язку Node Red та Unreal Engine 5 можна вважати WebSocket, який окрім доволі простої реалізації ще і забезпечує передачу даних із частотою навіть 0.01 секунд, а також повідомлень будь-якої довжини, без будь-яких затримок.

3.3 Розробка візуального інтерфейсу у середовищі Unreal Engine 5

Оскільки проєкт кваліфікаційної роботи має тип Blueprints, нам необхідно створити Blueprint клас з типом Widget. Він буде відображатися поверх усієї ігрової сцени та слугувати ігровим інтерфейсом.

Фінальний інтерфейс зображений на рисунку 3.25.

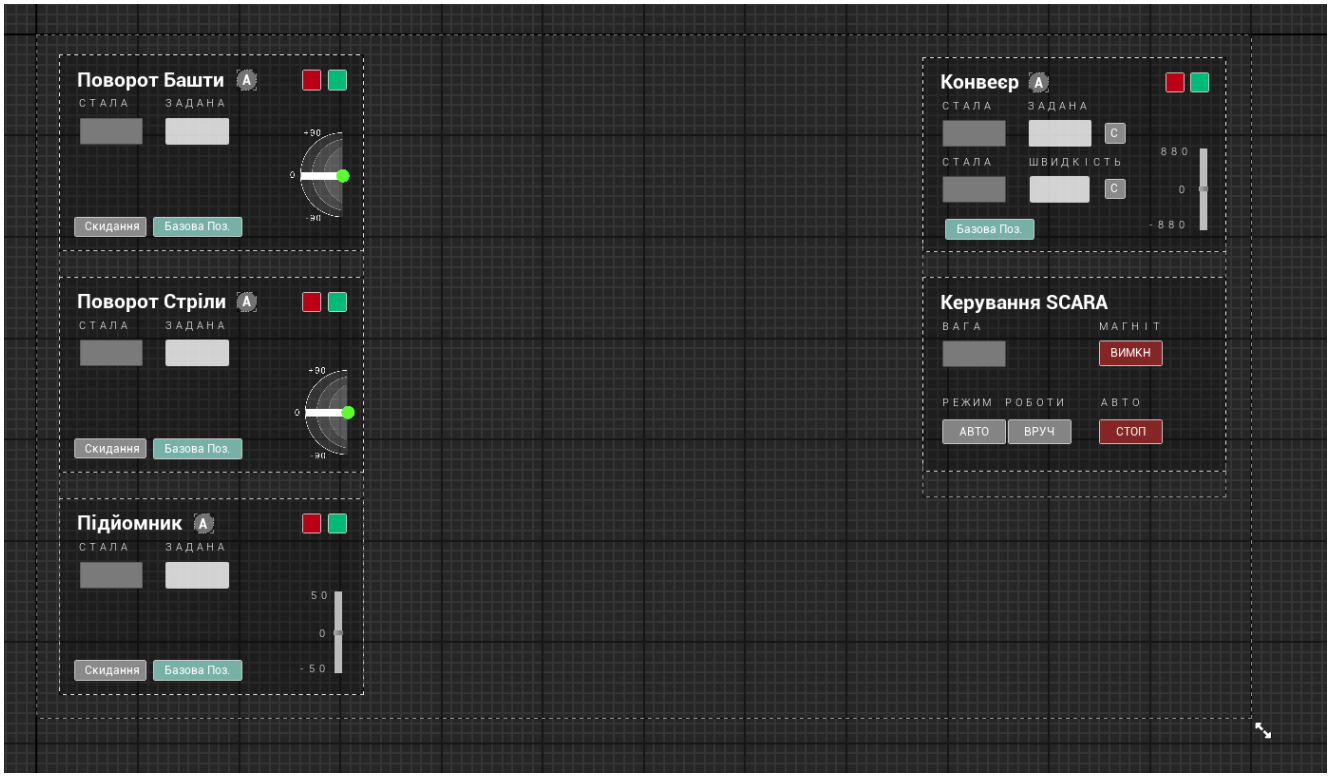


Рисунок 3.25 – Фінальний вигляд інтерфейсу цифрового двійника

Необхідно розглянути його функціонал більш подробно.

У кожної секції є значення позиції «стала» — тобто та, яка є актуальною зараз, та значення «задана», яку можна задати, натиснувши та ввівши необхідне значення.

Кнопка скидання скидає задане значення до базової позиції (нульового значення), а кнопка базової позиції задає цю базову позицію.

Червона та зелена кнопки справа вгорі є індікаторами роботи заданої частини. Горить тільки одна з них, якщо червона, то частина SCARA або конвеєр не працює.

Керування SCARA дозволяє обрати режим Авто або Вручну, та зупинити або увімкнути режим АВТО. Можна увімкнути чи вимкнути магніт, та побачити вагу об’єкта, що стоїть на вагах.

Візуалізація башти, стріли реалізована за допомогою двух .png картинок, значення обертів яких фактично відповідають заданому значенню (рис. 3.26).

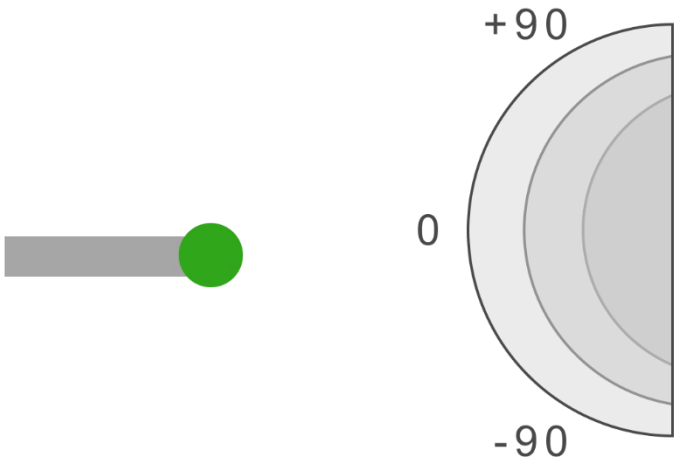


Рисунок 3.26 – Реалізація візуалізації положення башні

Візуалізація позиції підйомника та конвеєра зроблена через неактивний слайдер та зміну його значення.

3.4 Розробка програмного коду цифрового двійника у середовищі Blueprints

Для повноцінної реалізації цифрового двійника необхідно розробити програмний код у середовищі Unreal Engine 5. Спочатку, у інтерфейсі Widgets (рис 3.25) кожному полю текстовому, яке відображає параметри значення, прив'язати відповідну змінну з UWebSocketTestGameInstance яка отримується з контролеру. Це робиться за допомогою функції Bind.

Після цього можна приступати до створення коду для інтерфейсу, SCARA-руки, сенсорів та конвеєру. На початку коду кожного з перелічених елементів системи цифрового двійника відбувається каст до ігрового інстансу, який власне і дозволяє отримати доступ до функцій та змінних UWebSocketTestGameInstance (рис 3.27).

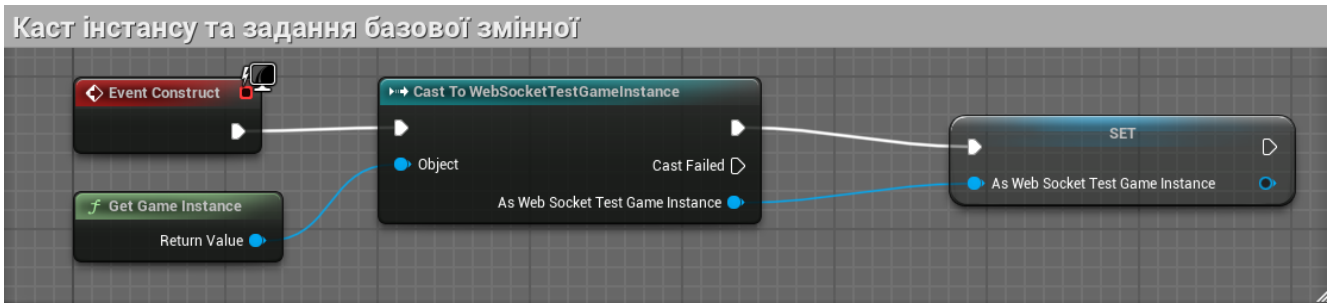


Рисунок 3.27 – Каст до інстансу UWebSocketTestGameInstance

Кожний зарендерений кадр програма перевіряє статус змінних ActualPosition 1, 2, 3 (башти, руки та підйомника), і встановлює відповідні значення до параметрів руки (рис. 3.28). Сама 3D-модель реалізована за допомогою комбінації елементів StaticMesh та PhysicsConstraint (які слугують осями обертання чи руху).

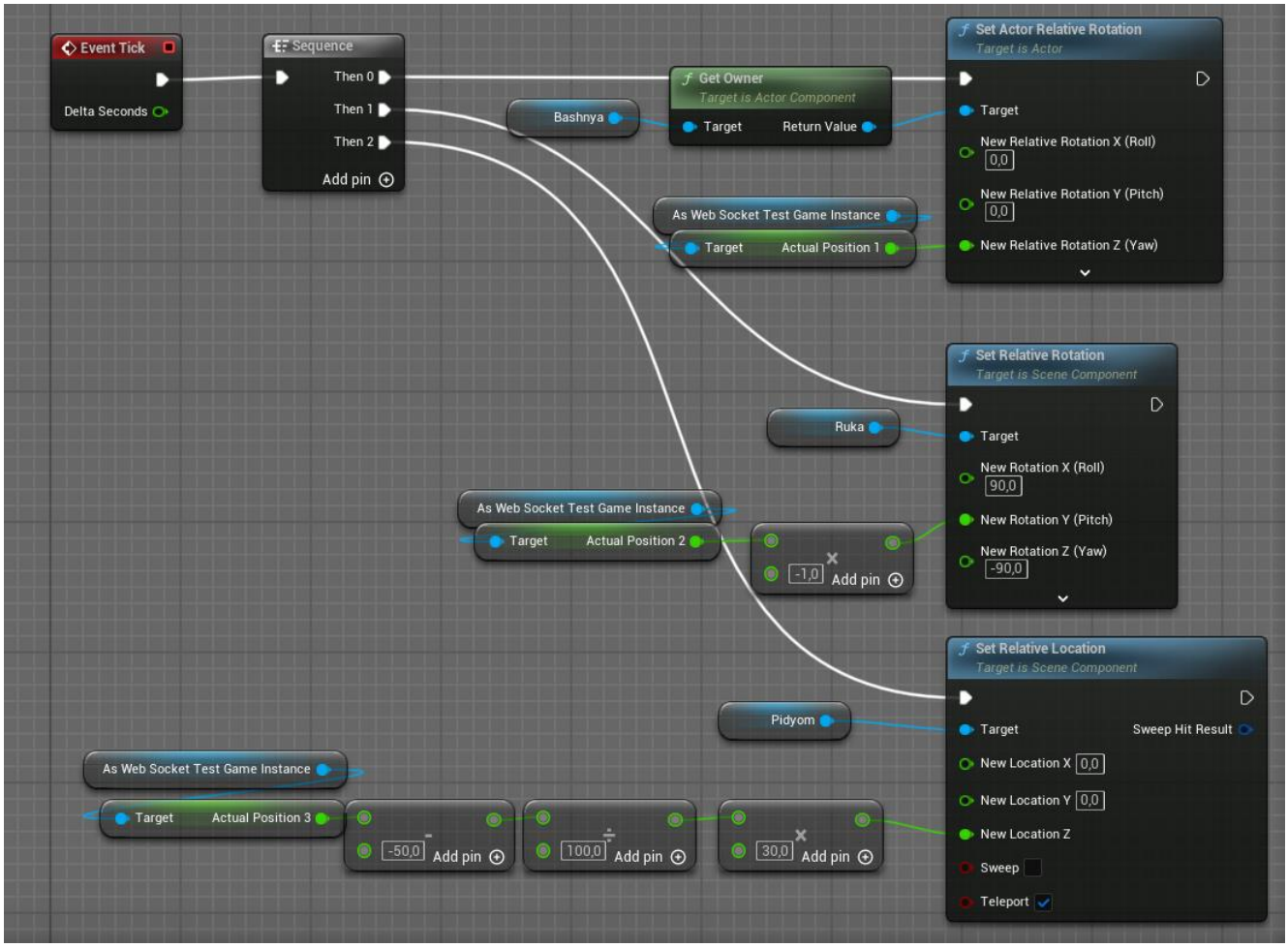


Рисунок 3.28 – Код цифрового двійника SCARA-руки

Для обертання башти значення обертів ідентичне, тобто 0 це значення 0 у змінній, 50 – це 50 відповідно. Для обертання руки значення ідентичне, але зворотнє. Тобто 90 у змінній мають бути -90 у осі обертання Z, 75 – -75.

Для руху підйомника ситуація була більш складною, оскільки у програмі вона рухається у діапазоні від 0 до 30, а на стенді від -50 до 50. Для правильного відображення довелося вичислити апроксімацію і прийти до формули $y = \frac{(x+50)*30}{100}$.

Для правого та лівого сенсору код програми ідентичний (рис. 3.29), за виключенням різних отриманих змінних (RightSensor та LeftSensor). Кожен кадр програма перевіряє статус змінної bool. Якщо вона має значення True, об’єкту сенсору встановлюється матеріал (умовно кажучи, текстура) зеленого кольору з

підсвіткою, якщо False – тоді базова біла текстура, яка сигналізує про те, що сенсор не ідентифікує жодних об’єктів на своєму шляху.

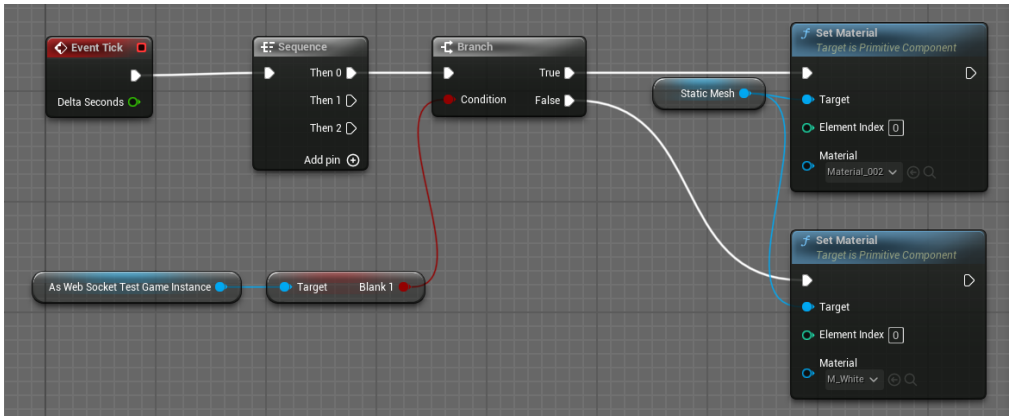


Рисунок 3.29 – Код цифрового двійника датчику руху

Оскільки код інтерфейсу доволі складний та комплексний за своєю кількістю, розбирати його будемо за окремими прикладами. Наприклад, у нас є чотири кнопки вимикання, але ми будемо розглядати лише одну з них, у інших аналогічна реалізація, але лише з іншою змінною.

Для того щоб кнопки вмикання-вимикання змінювали свій колір при отриманні даних, було зроблено систему із двох if-Branch, які перевіряють статус змінної та задають відповідний колір кнопці, при чому роблячи іншу неактивною (рис. 3.30).

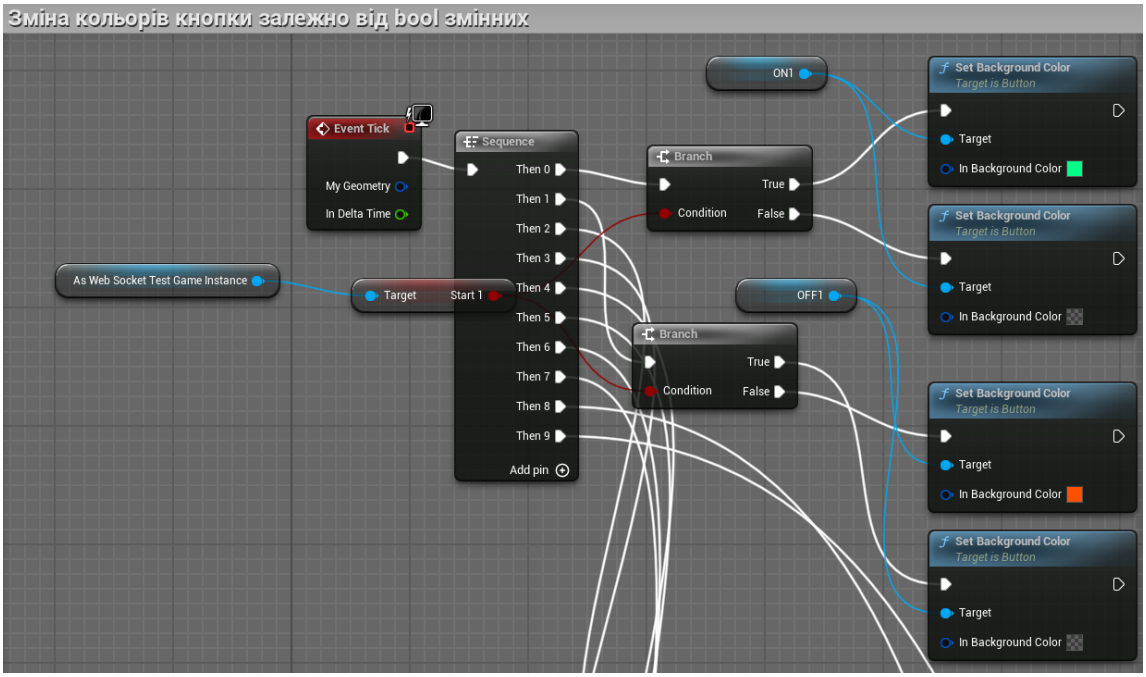


Рисунок 3.30 – Код зміни кольору кнопок (частина)

Усі кнопки скиду до базового значення мають наступну структуру (рис. 3.31): після натискання на кнопку, будується String з відповідною змінною та значенням 0 за допомогою функції UWebSocketTestGameInstance::String Builder, після чого сформований WString надсилається на сервер за допомогою UWebSocketTestGameInstance::Notify Server.

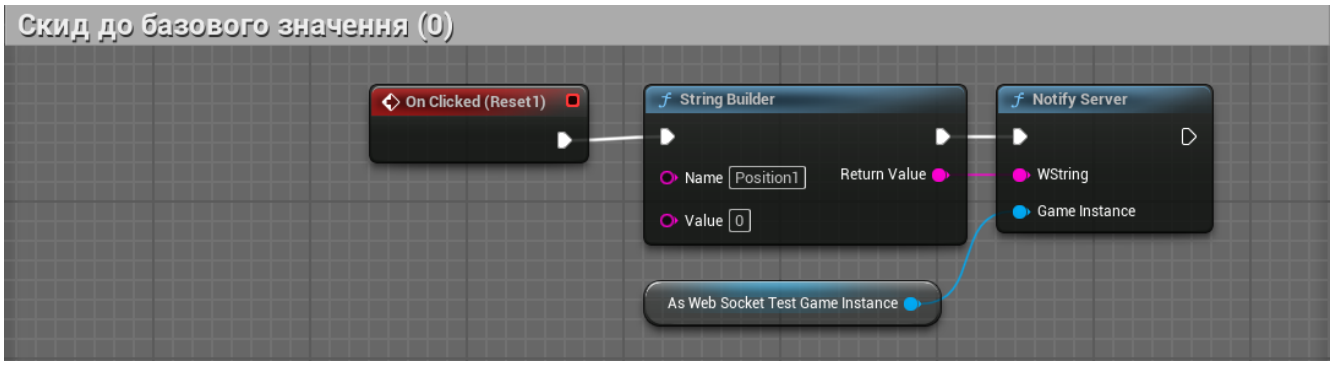


Рисунок 3.31 – Код кнопки скиду до базового значення (частина)

Код установлення базової позиції чимось схожий, але на відміну від скиду, він надсилає одиницю булевої змінної Starting_Position на сервер та одразу її скидає, за рахунок чого поточна позиція елементу SCARA-робота задається базовою (рис. 3.32).

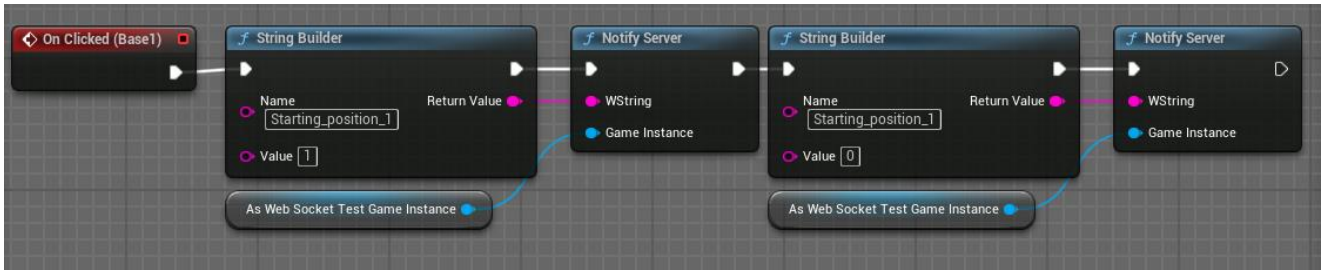


Рисунок 3.32 – Код задання базової позиції (частина)

Код задання аварійного сигналу перевіряє поточний статус змінної, що за це відповідає, і якщо вона має значення «1», тоді встановлює жовтий колір як індикатор аварії (рис. 3.33).

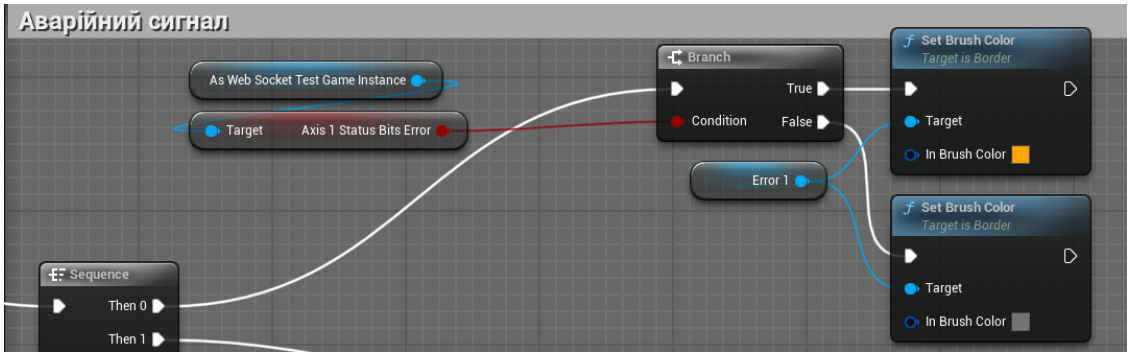


Рисунок 3.33 – Код аварійного сигналу (частина)

Код відправлення значень на вебсокет спочатку перевіряє як саме було введене значення у поле зміни значення (наприклад, якщо було клікнуто поза полем, тоді значення не приймається, а приймається лише тоді, коли натиснуто Enter). Після цього перевіряється, чи значення розташоване у відповідних допустимих межах (щоб не призвести до аварійної ситуації). Після цього будується String та надсилається на WebSocket сервер (рис. 3.34).

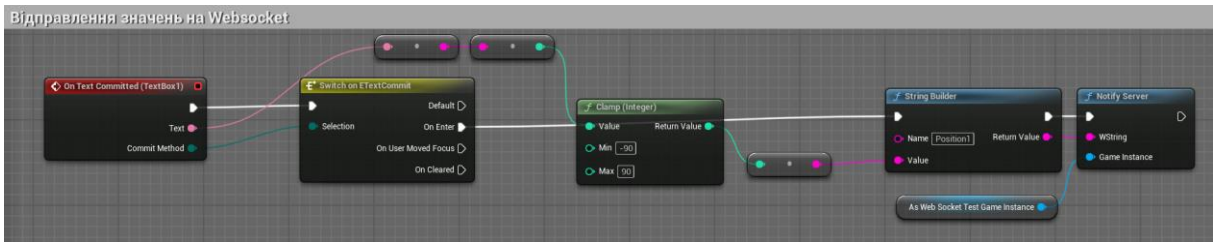


Рисунок 3.34 – Код задання керуючої змінної (частина)

Код кнопок вмикання та вимикання будує відповідний String та відправляє його на сервер (рис. 3.35).

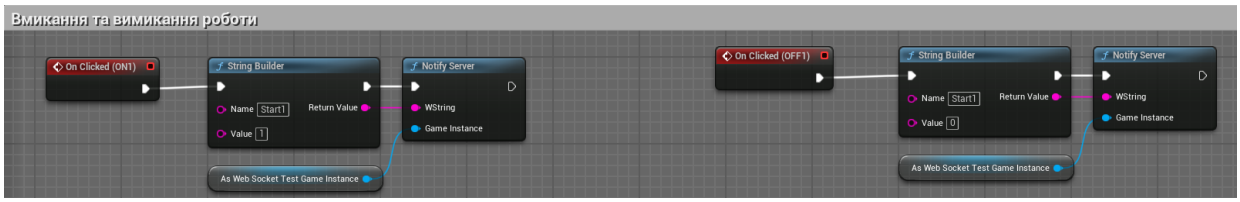


Рисунок 3.35 – Код вмикання та вимикання елементів ЦД (частина)

Код керування візуалізаціями (рис. 3.36) отримує відповідні значення актуальних позицій башти, руки, підйомника та конвеєра та задає їх до індикатора положення (рис. 3.26) або до слайдеру положення (рис. 3.25).

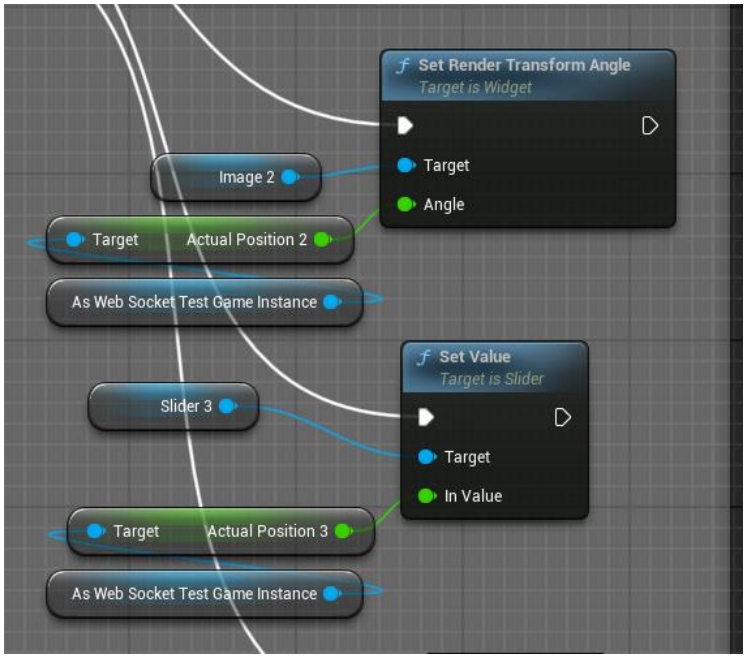


Рисунок 3.35 – Код керування візуалізаціями-індикаторами (частина)

Керування кнопками Auto/Manual відбувається за схожим принципом із кнопками увімкнення та вимкнення (рис. 3.36).

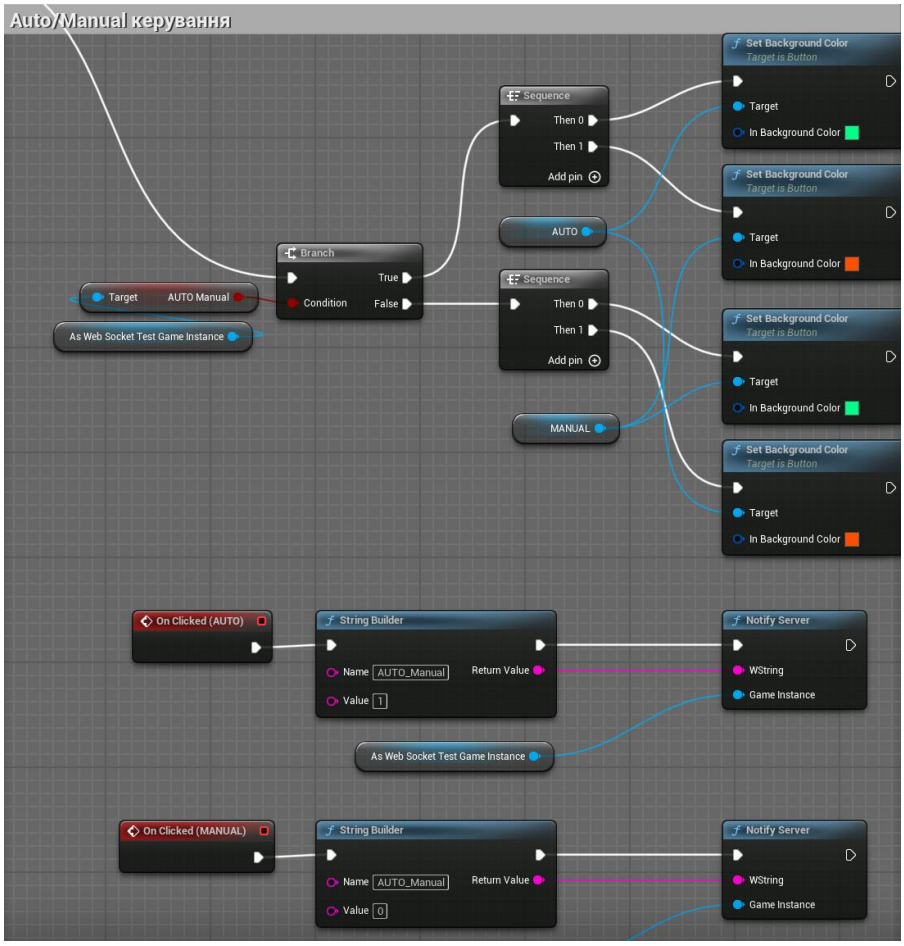


Рисунок 3.36 – Код керування Auto/Manual (частина)

Керування кнопками магніту та вимкнення/увімкнення АВТО режиму відбувається теж за схожим принципом, але окрім кольору там ще задається і надпис ON/OFF (рис 3.37).

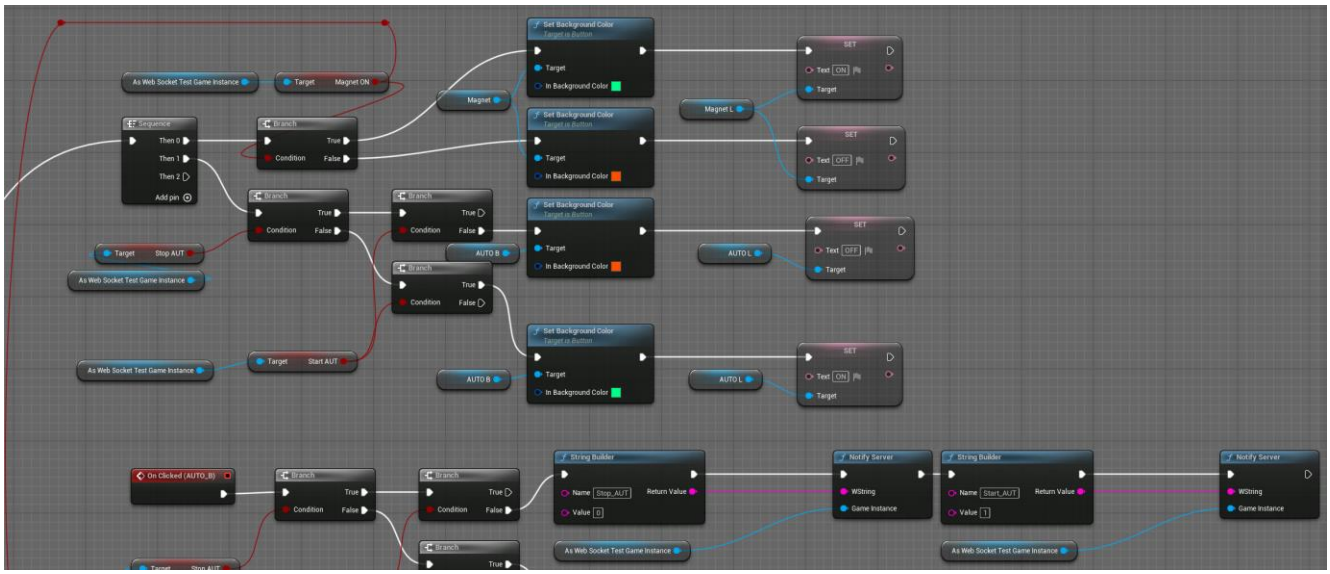


Рисунок 3.37 – Код керування магнітом та кнопкою вмикання-вимикання автоматичного режиму (частина)

Отже, код цифрового двійника та його інтерфейсу реалізовано коректно, і залишається реалізувати код, який дозволяє керувати камерою.

3.5 Розробка програмного коду камери цифрового двійника

Для керування камерою було створено окремий Blueprint класс BP_Camera, і зроблено саму структуру камери, а саме – створено сцену для неї, додано елемент SpringArm (який дозволяє зробити так, щоб камера оберталась навколо певної точки чи об'єкту), і у ньому власне саму камеру (рис. 3.38)

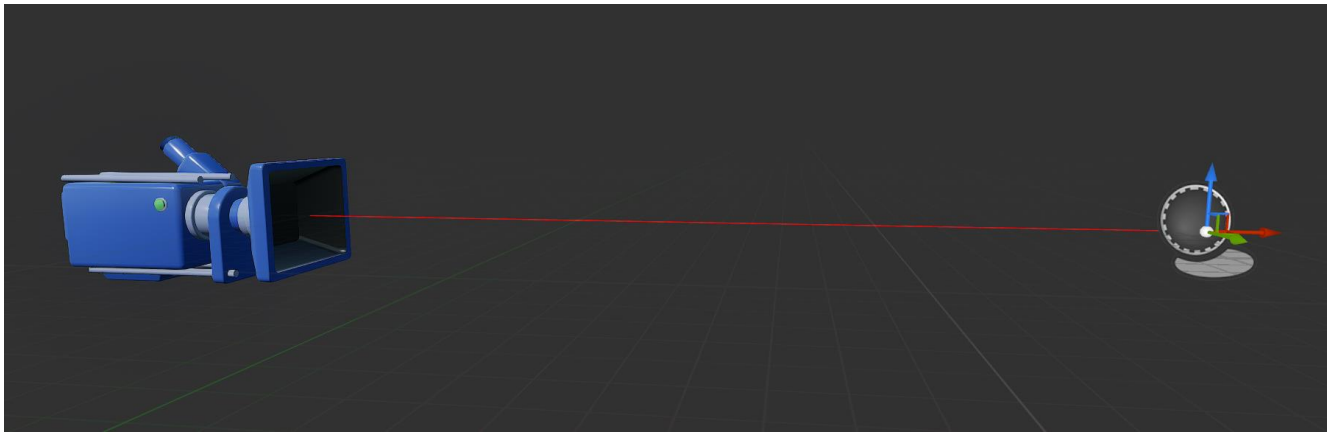


Рисунок 3.38 – Структура камери цифрового двійника

Код працює наступним чином. Після затискання середньої кнопки задається значення змінної IsMiddle та прибирається курсор. Осі переміщення миші X та Y задають Yaw та Pitch обертання камери. При цьому рух камери по вертикалі обмежений до значень від 0 до 90, оскільки нижче 0 камера почне виходити за межі підлоги (рис. 3.39).

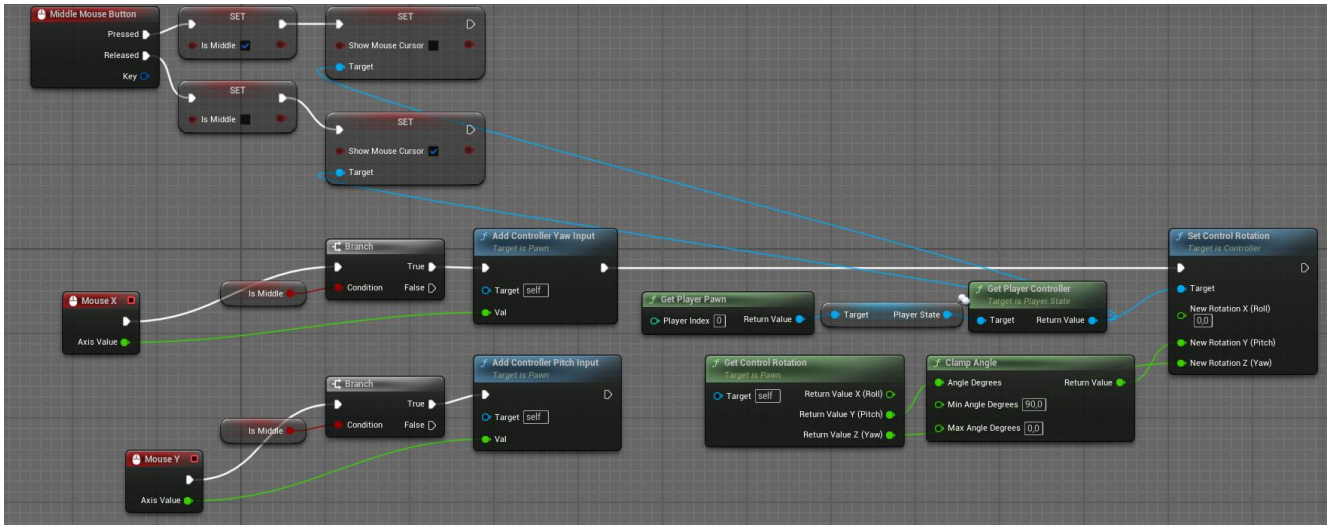


Рисунок 3.39 – Код камери цифрового двійника

3.6 Тестування роботи цифрового двійника

Давайте протестуємо програму цифрового двійника. Спочатку необхідно ініціалізувати саму програму та отримати значення з контролера через ланцюг з’єднання (рис. 3.40).

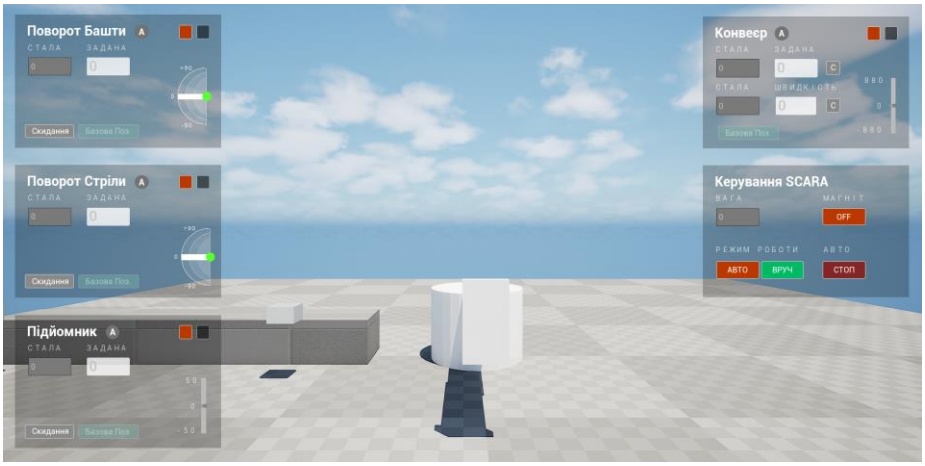


Рисунок 3.40 – Ініціалізований цифровий двійник без отриманих значень

Починаємо відправляти значення з контролера та бачимо, що програма працює абсолютно коректно та відображає поточний стан комплексу моделювання транспортно-розвантажувальних операцій (рис. 3.41). Сигнали про аварію сигналізують про те, що сталася аварійна ситуація, що ми і бачемо на екрані: SCARA-рука уткнулася у детектор руху та не може рухатися далі. Детектор, у свою чергу, показує, що бачить об’єкт перед ним.

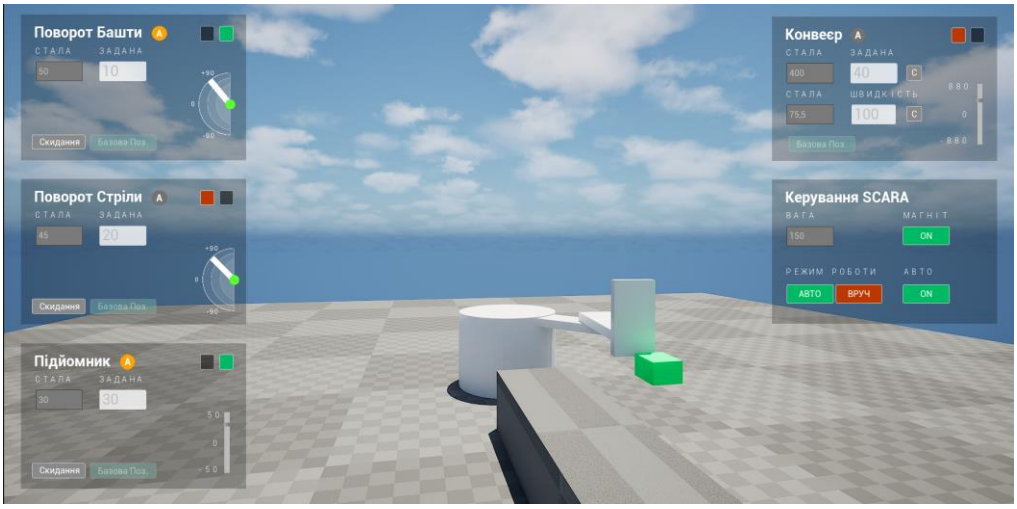


Рисунок 3.41 – Демонстрація роботи цифрового двійника

Давайте вирішимо аварійну ситуацію та задамо позицію башти 90, а стріли 0 (рис. 3.42). Перемкнемо заздалегідь режим на ВРУЧ, а потім знову на АВТО після закінчення роботи. Як ми бачимо, завдання виконане вірно, і квалифікаційну роботу можна вважати правильно виконаною.

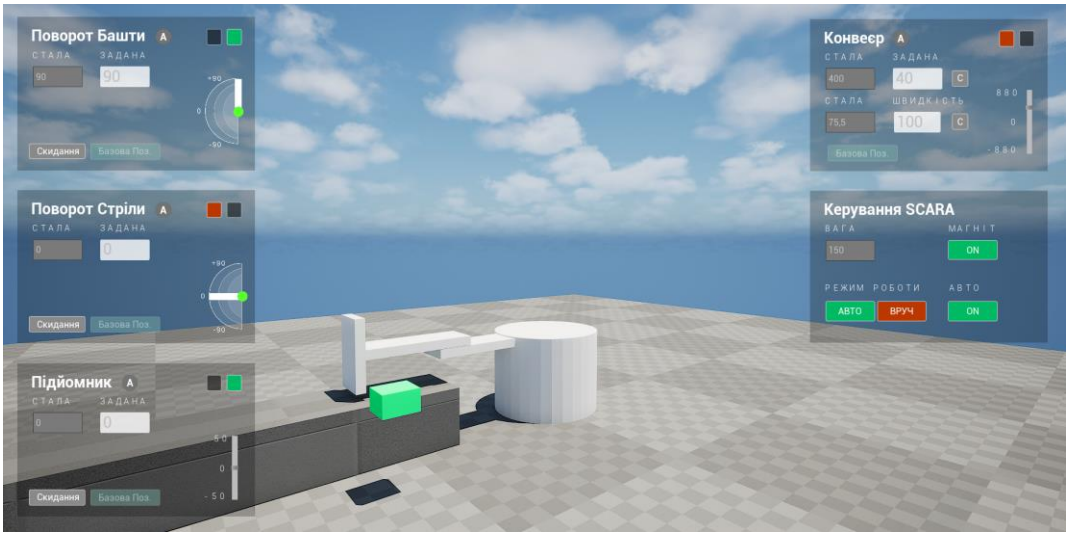


Рисунок 3.42 – Демонстрація роботи цифрового двійника

Висновки до розділу:

У цьому розділі було повністю зроблено програмну реалізацію кваліфікаційної роботи. Спочатку було встановлено зв'язок між TIA Portal та OPC UA та реалізовано програмний код в Node Red для двосторонньої передачі даних. Виконане змістовне дослідження щодо обрання правильної технології для встановлення зв'язку між Node Red та Unreal Engine 5, і у результаті обрано варіант з використанням серверу Websocket, який дозволив з великою частотою та довжиною повідомлення передавати пакети даних.

Після цього був реалізований власне код самої програми цифрового двійника, його інтерфейсу користувача, камери, описано структуру проєкту, описано код написанної бібліотеки функцій та значень UWebSocketTestGameInstance.

ВИСНОВКИ

У цій кваліфікаційній роботі магістра було розглянуто, досліджено та реалізовано розробку цифрового двійника комплексу моделювання транспортно-розвантажувальних операцій з використанням рушіїв візуалізації.

У першому розділі було описано сам технологічний процес, а також модель системи. Було розказано про основні принципи створення цифрових двійників. Після цього розглянуто їх класифікацію, переваги, недоліки та перспективи розвитку. Проведено аналіз технологій та патентів для реалізації SCARA-руки в Unreal Engine 5.

У другому розділі була створена структурна модель системи на базі архітектури RAMI 4.0, після чого сформульовано та розроблено схеми інформаційних потоків системи. Проведено порівняння технологій передачі даних. У кінці було розгорнуто початкові компоненти системи у якості підготовки до реалізації, описано принципи інтеграції Node-RED та Unreal Engine 5 через різні мережеві протоколи, а також створено прототип інтерфейсу користувача та 3D-модель SCARA-руки.

У третьому розділі було виконано повну програмну реалізацію системи. Зроблена інтеграція TIA Portal та Node-RED через OPC, та Node Red і Unreal Engine 5 через WebSocket. Реалізовано інтерфейс цифрового двійника, бібліотеку функцій, нодовий код програми, описано структуру проєкту. Після тестування цифрового двійника було зроблено висновок, що робота виконана правильно та програма функціонує коректно.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Цифрові двійники на варті безпеки та екологічності. Інноваційна технологія, що дає безліч спроб перед початком роботи: веб-сайт. URL: https://dia.dp.gov.ua/cifrovi-bliznyuki-na-varti-bezpeki-ta-ekologichnosti-innovacijna-texnologiya-shho-daye-bezlich-sprob-pered-pochatkom-roboti/ (дата звернення: 09.12.2024).

2. Ваш цифровий двійник допоможе вам залишатися здоровим: веб-сайт. URL: https://techtoday.in.ua/reviews/vash-tsyfrovyj-blyznyuk-dopomozhe-vam-zalyshatysya-zdorovym-161267.html (дата звернення: 09.12.2024).

3. Жмай О.В. ВПЛИВ ПАНДЕМІЇ НА ПРОМИСЛОВИЙ СВІТ: ЯК ОЦИФРОВКА І АВТОМАТИЗАЦІЯ РОБЛЯТЬ ВИРОБНИЦТВО БЕЗПЕЧНИМ ДЛЯ МАЙБУТНЬОГО: Матеріали конференції. Одеса, 2022. 178 с.

4. Цифрові двійники – майбутнє моделювання та моделювання: веб-сайт. URL: https://hashdork.com/uk/digital-twins/ (дата звернення: 09.12.2024).

5. Types of Digital Twins: веб-сайт. URL: https://www.twinspiration.in/what-are-different-types-of-digital-twins/ (дата звернення: 09.12.2024).

6. How Does a Digital Twin Work? A Comprehensive Guide: веб-сайт. URL: https://www.twinspiration.in/how-does-a-digital-twin-works/ (дата звернення: 09.12.2024).

7. Digital Twin Market Size: веб-сайт. URL: https://www.fortunebusinessinsights.com/digital-twin-market-106246 (дата звернення: 09.12.2024).

8. Jacoby Michael. Digital Twin and Internet of Things—Current Standards Landscape: Review. Germany, 2020. 21 с.

9. MQTT Vs. HTTP for IoT: веб-сайт. URL: https://www.hivemq.com/article/mqtt-vs-http-protocols-in-iot-iiot/ (дата звернення: 09.12.2024).

10. Importance of Real-Time Integration in Digital Twin: веб-сайт. URL: https://www.toobler.com/blog/real-time-integration-in-digital-twins (дата звернення: 09.12.2024).

11. MQTT vs AMQP for IoT: веб-сайт. URL: https://www.hivemq.com/article/mqtt-vs-amqp-for-iot/ (дата звернення: 09.12.2024).

12. Digital Twin Data Pipeline Using MQTT in SLADTA: веб-сайт. URL: https://www.researchgate.net/figure/The-digital-twin-data-pipeline-mapped-onto-the-SLADTA-framework_fig2_349750145 (дата звернення: 09.12.2024).

13. Predictive Digital Twins for Autonomous Ships: веб-сайт. URL: https://www.researchgate.net/figure/Predictive-digital-twin-architecture-for-autonomous-ships_fig1_371853873 (дата звернення: 09.12.2024).

14. Can we benefit from game engines to develop digital twins for planning the deployment of photovoltaics?: веб-сайт. URL: https://energyinformatics.springeropen.com/articles/10.1186/s42162-022-00222-7 (дата звернення: 09.12.2024).

15. Importance of Real-Time Integration in Digital Twin: веб-сайт. URL: https://www.toobler.com/blog/real-time-integration-in-digital-twins (дата звернення: 09.12.2024).

16. The Shifting Landscape of Industrial Simulation: Unity, Unreal Engine, and Omniverse: веб-сайт. URL: https://www.linkedin.com/pulse/shifting-landscape-industrial-simulation-unity-unreal-thomas-strigl/ (дата звернення: 09.12.2024).

17. Як Unity позиціонує себе для демократизації метавсесвіту: веб-сайт. URL: https://zephyrnet.com/uk/how-unity-is-positioning-itself-to-democratize-the-metaverse/ (дата звернення: 09.12.2024).

18. ADT Link Plugin: веб-сайт. URL: https://github.com/Azure-Samples/azure-digital-twins-unreal-integration/blob/main/docs/adt-link-plugin-ue.md (дата звернення: 09.12.2024).

19. Unreal Engine and Azure Digital Twins integration demo: веб-сайт. URL: https://github.com/Azure-Samples/azure-digital-twins-unreal-integration (дата звернення: 09.12.2024).

20. Microsoft Azure Digital Twins : Everything You Need To Know: веб-сайт. URL: https://xmpro.com/microsoft-azure-digital-twins-everything-you-need-to-know/ (дата звернення: 09.12.2024).

21. Simulate the IoT Devices: веб-сайт. URL: https://github.com/Azure-Samples/azure-digital-twins-unreal-integration/blob/main/docs/simulate-iot-devices.md (дата звернення: 09.12.2024).

22. THE AUTOMOTIVE FIELD GUIDE: Epic Games. US, 2020. 108 с.

23. Bidirectional Linkage Robot Digital Twin System Based on ROS: веб-сайт. URL: https://ieeexplore.ieee.org/document/10076497 (дата звернення: 09.12.2024).

24. Importance of Digital Twins in Industry 4.0: веб-сайт. URL: https://www.twinspiration.in/importance-of-digital-twins-in-industry-4-0/ (дата звернення: 09.12.2024).

25. What is a Digital Twin?: веб-сайт. URL: https://www.twinspiration.in/what-is-a-digital-twin/ (дата звернення: 09.12.2024).

26. Digital Twins – Unreal Engine: веб-сайт. URL: https://www.twinspiration.in/digital-twins-unreal-engine/ (дата звернення: 09.12.2024).

27. HMI Dojo initiative: a digital twin development in Unreal Engine 5: веб-сайт. URL: https://spyro-soft.com/blog/hmi/hmi-dojo-initiative-digital-twin-unreal-engine-5 (дата звернення: 09.12.2024).

28. A VIRTUAL CLONE OF ADELAIDE FOR DIGITAL TWIN AND SMART CITY APPLICATIONS: веб-сайт. URL: https://www.twinmotion.com/en-US/spotlights/a-virtual-clone-of-adelaide-for-digital-twin-and-smart-city-applications (дата звернення: 09.12.2024).

29. The SCARA robot based on the idea of digital twins: веб-сайт. URL: https://github.com/kholodilinivan/ (дата звернення: 09.12.2024).

30. Design of a Multi-robot Digital Twin System with Bidirectional Motion Synchronization Capabilities: веб-сайт. URL: https://link.springer.com/chapter/10.1007/978-981-99-6504-5_26 (дата звернення: 09.12.2024).

31. Virtual SCARA Robot with Reduced Dynamics in a Real-Time Simulation Scheme for Robotics and Control Engineering Education: веб-сайт. URL: https://ieeexplore.ieee.org/document/9698437 (дата звернення: 09.12.2024).

32. MTAB Automation SCARA-robot: веб-сайт. URL: https://mtabautomation.com/ (дата звернення: 09.12.2024).

33. A robot arm digital twin utilising reinforcement learning: веб-сайт. URL: https://www.sciencedirect.com/science/article/abs/pii/S009784932100011X (дата звернення: 09.12.2024).

34. Design of a Digital Twin Training Centre for an Industrial Robot Arm: веб-сайт. URL: https://www.mdpi.com/2076-3417/12/17/8862 (дата звернення: 09.12.2024).

35. Potentials of game engines for wind power digital twin development: an investigation of the Unreal Engine: Research. Denmark, 2022. 31 с.

36. Wind generator digital twin development: an investigation of the Unreal Engine: веб-сайт. URL: https://www.researchgate.net/figure/The-wind-power-architecture_fig1_366463254 (дата звернення: 09.12.2024).

37. DEVELOPMENT OF DIGITAL TWIN IN EXTENDED REALITY WITH UNREAL ENGINE 4: Research. Thesis, 2020. 75 с.

38. Пат. US11396100B2 США: веб-сайт. URL: https://eureka.patsnap.com/view/#/fullText'figures/?patentId=991074bb-abb0-4b7e-b308-5147cd88efee (дата звернення: 09.12.2024).

39. Пат. CN115446867A Китай: веб-сайт. URL: https://patents.google.com/patent/CN115446867A/en (дата звернення: 09.12.2024).

40. Пат. CN111413887B Китай: веб-сайт. URL: https://patents.google.com/patent/CN111413887B/en (дата звернення: 09.12.2024).

41. Пат. CN109719730B Китай: веб-сайт. URL: https://patents.google.com/patent/CN109719730B/en (дата звернення: 09.12.2024).

42. Пат. US12306220B2 США: веб-сайт. URL: https://www.researchgate.net/figure/Sensor-readings-transfer-to-BIM-and-game-engine-

43-Connecting-IoT-sensor-outputs-with_fig3_333458074 (дата звернення: 09.12.2024).

43. RAMI 4.0 ARCHITECTURE: веб-сайт. URL: https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference_ architectural_model_industrie_4.0_rami_4.0.pdf (дата звернення: 09.12.2024).

44. A first look at Azure Event Grid MQTT support: веб-сайт. URL: https://sandervandevelde.wordpress.com/2023/10/14/a-first-look-at-azure-eventgrid-mqtt-support/ (дата звернення: 09.12.2024).

45. Маринич І. А., Тронь В. В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 151 “Автоматизація та комп’ютерно-інтегровані технології”. Кривий Ріг : Видавничий центр КНУ, 2022. 50с.

46. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

47. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

48. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові.

49. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація)

50. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).