ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Віртуальний асистент для оптимізації взаємодії між оператором і промисловими роботами на основі технологій штучного інтелекту»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
освітньо-професійної програми Штучний інтелект

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Назар БОНІСЛАВСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти гр. ШІД-42
Назар БОНІСЛАВСЬКИЙ

Керівник: Антон ШАНТИР
к.т.н., доцент

Рецензент: _____
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут Інформаційних технологій

Кафедра Штучний інтелект
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
Освітньо-професійна програма Штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедрою Штучного інтелекту
_____ Ольга ЗІНЧЕНКО
«____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Боніславський Назар Сергійович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Віртуальний асистент для оптимізації взаємодії між оператором і промисловими роботами на основі технологій штучного інтелекту
керівник кваліфікаційної роботи Антон ШАНТИР к.т.н., доцент
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56
2. Строк подання кваліфікаційної роботи «23» травня2025 р
3. Вихідні дані до кваліфікаційної роботи науково-технічна література, дані інтернет-мережі, мова програмування: Python, середовище розроблення Visual Studio Code
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. Аналіз вимог до віртуального асистента для взаємодії оператора з промисловими роботами.
   2. Опис методів машинного навчання для оптимізації роботи робота на складі.
   3. Реалізація графічного інтерфейсу та логування подій.
   4. Тестування системи, аналіз результатів.
5. Перелік ілюстративного матеріалу: *презентація*
Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання завдання	25.02-20.03.2025	Виконано
2	Аналіз роботи, та підбір літератури	21.03-29.03.2025	Виконано
3	Аналіз принципів роботи віртуального асистента на основі ШІ	30.03-10.04.2025	Виконано
4	Розробка методів для віртуального асистента	10.04-21.04.2025	Виконано
5	Програмна реалізація асистента	21.04.-29.04.2025	Виконано
6	Оформлення пояснювальної записки	29.04-02.05.2025	Виконано
7	Перевірка на плагіат	02.05-07.05.2025	Виконано
8	Рецензування	07.05-12.05.2025	Виконано
9	Підготовка презентації	12.05-19.05.2025	Виконано
10	Попередній захист кваліфікаційної роботи	20.05-23.05.2025	Виконано

Здобувач вищої освіти _____ Назар БОНІСЛАВСЬКИЙ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Антон ШАНТИР
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 74 стор., 23 рис., 16 джерел.

Мета роботи: розробити та впровадити віртуального асистента для оптимізації взаємодії між оператором і промисловими роботами на основі технологій штучного інтелекту, щоб підвищити ефективність роботи на складі.

Об'єкт дослідження: процеси взаємодії оператора з промисловими роботами в умовах складської логістики, включаючи керування, доставку предметів і оновлення конфігурації.

Предмет дослідження: методи, алгоритми та інструменти створення віртуального асистента з використанням технологій штучного інтелекту для автоматизації та спрощення роботи на складі.

Короткий зміст роботи: у роботі проведено аналіз вимог до системи, розроблено модульну архітектуру віртуального асистента, включаючи модулі обробки даних, інтерфейс користувача та аналітичний модуль на основі ШІ. Реалізовано графічний інтерфейс, методи машинного навчання для пошуку оптимальних шляхів руху робота, а також систему логування подій. Проведено тестування програми, яке показало її здатність ефективно моделювати роботу робота та полегшувати завдання оператора.

КЛЮЧОВІ СЛОВА: ВІРТУАЛЬНИЙ АСИСТЕНТ, ПРОМИСЛОВИЙ РОБОТ, ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, СКЛАДСЬКА ЛОГІСТИКА, ГРАФІЧНИЙ ІНТЕРФЕЙС, АВТОМАТИЗАЦІЯ.

ABSTRACT

Text part of the bachelor’s qualification work: 74 pages, 23 pictures, 16 sources.

Objective of the work: To develop and implement a virtual assistant for optimizing interaction between the operator and industrial robots based on artificial intelligence technologies to enhance warehouse operation efficiency.

Object of study: The processes of interaction between the operator and industrial robots in warehouse logistics conditions, including management, item delivery, and configuration updates.

Subject of study: Methods, algorithms, and tools for creating a virtual assistant using artificial intelligence technologies for automation and simplification of warehouse operations.

Summary of the work: The work includes an analysis of system requirements, the development of a modular architecture for the virtual assistant, incorporating data processing modules, a user interface, and an analytical module based on AI. A graphical interface was implemented, along with machine learning methods for finding optimal robot movement paths and an event logging system. Testing of the program was conducted, demonstrating its ability to effectively model robot operations and simplify the operator's tasks.

KEYWORDS: VIRTUAL ASSISTANT, INDUSTRIAL ROBOT, ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, WAREHOUSE LOGISTICS, GRAPHICAL INTERFACE, AUTOMATION.

ЗМІСТ

ВСТУП

Актуальність теми. Сьогоднішній етап розвитку промисловості характеризується широким впровадженням роботизованих систем у виробничі процеси. Такі зміни вимагають не лише оновлення технічного забезпечення, а й удосконалення способів взаємодії між людьми та машинами. Саме тому зростає потреба у зручних і зрозумілих засобах комунікації між оператором і промисловим роботом. Одним із перспективних напрямів є використання віртуальних асистентів, які, завдяки технологіям штучного інтелекту, здатні спростити управління обладнанням, підвищити безпеку праці та зменшити вплив людського чинника.

Мета дослідження. Основна мета роботи полягає в розробці програмного рішення — віртуального асистента, який допомагатиме оператору взаємодіяти з промисловим роботом у режимі реального часу, аналізуючи ситуацію та надаючи підказки або виконуючи певні дії автоматично.

Щоб реалізувати цю мету, потрібно вирішити такі завдання:

- Ознайомитися з існуючими підходами до людино-машинної взаємодії;
- Дослідити можливості застосування методів штучного інтелекту в подібних системах;
- Розробити архітектуру віртуального асистента;
- Реалізувати функціональну модель програми;

Об’єкт дослідження. Взаємодія між оператором і промисловим роботом у контексті автоматизованого виробництва.

Предмет дослідження. Програмні засоби, що дозволяють оптимізувати процес людино-машинної взаємодії за допомогою віртуального асистента.

Методи дослідження. Під час виконання роботи використано аналітичні, порівняльні та експериментальні методи. Здійснено аналіз літературних джерел, розроблено алгоритм взаємодії, а також проведено моделювання роботи системи.

Наукова новизна. Новизна проєкту полягає у створенні інтерфейсу взаємодії, який поєднує елементи штучного інтелекту та зручність використання в реальних умовах, де важлива оперативність і точність.

Практичне значення. Отримані результати можуть бути корисними для підприємств, які прагнуть підвищити рівень автоматизації без необхідності суттєво змінювати структуру управління на виробництві. Розроблена система може стати основою для впровадження подібних рішень у різних галузях.

1. АНАЛІЗ СУЧАСНОГО СТАНУ ЛЮДИНО-МАШИННОЇ ВЗАЄМОДІЇ

1.1 Роль операторів у роботизованих системах

У сучасних промислових системах, де активно застосовуються роботизовані технології, оператори відіграють ключову роль у забезпеченні ефективної взаємодії між людиною та машиною. Роботизовані системи, такі як автоматизовані виробничі лінії, промислові маніпулятори чи автономні транспортні засоби, покликані підвищувати продуктивність, знижувати витрати та мінімізувати людський фактор у небезпечних або монотонних операціях. Проте, незважаючи на високий рівень автоматизації, людина залишається невід'ємною частиною цих систем, виконуючи функції, які потребують гнучкості, креативності та прийняття складних рішень .

Оператори в роботизованих системах відповідають за низку завдань, які можна умовно поділити на три основні категорії: контроль, програмування та реагування на нестандартні ситуації. По-перше, контроль передбачає моніторинг роботи роботів, аналіз даних із сенсорів та інтерфейсів, а також своєчасне виявлення відхилень у функціонуванні системи. Наприклад, оператор може відстежувати параметри роботи конвеєра чи перевіряти якість виконання зварювальних операцій роботом. По-друге, програмування включає налаштування роботів для виконання нових завдань, адаптацію їх до змін у виробничих процесах або оновлення алгоритмів роботи. Хоча сучасні роботи часто мають заздалегідь запрограмовані сценарії, оператори нерідко вносять корективи, щоб оптимізувати їхню роботу. По-третє, реагування на нестандартні ситуації є однією з найважливіших функцій, адже навіть найрозвиненіші системи не завжди здатні самостійно впоратися з аварійними ситуаціями, збоями чи непередбачуваними змінами у виробничому процесі .

Важливою особливістю роботи операторів є їхня взаємодія з інтерфейсами людино-машинної взаємодії (HMI, Human-Machine Interface). Такі інтерфейси дозволяють операторам отримувати інформацію про стан системи у зрозумілій формі, вводити команди та отримувати зворотний зв'язок. Наприклад, сенсорні панелі, системи доповненої реальності чи голосові інтерфейси спрощують процес

керування роботами, але вимагають від операторів певного рівня технічної підготовки. У деяких випадках оператори також співпрацюють із системами віддаленого керування, що дозволяє управляти роботами на відстані, наприклад, у небезпечних зонах чи на віддалених об’єктах .

Роль операторів ускладнюється через швидкий розвиток технологій та зростання складності роботизованих систем. Якщо раніше оператори переважно виконували механічні функції, такі як запуск чи зупинка обладнання, то сьогодні їхня робота вимагає глибокого розуміння програмного забезпечення, аналізу даних та взаємодії зі штучним інтелектом. Це створює нові виклики, пов’язані з необхідністю постійного навчання та адаптації до нових технологій. Водночас оператори залишаються незамінними, адже їхня здатність до імпровізації, інтуїтивного прийняття рішень та оцінки складних ситуацій перевищує можливості сучасних автоматизованих систем.

Таким чином, оператори в роботизованих системах виступають сполучною ланкою між технологіями та виробничими цілями, забезпечуючи стабільність, гнучкість та безпеку роботи. Їхня роль не лише не втрачає актуальності, але й набуває нового значення в умовах цифровізації та інтеграції штучного інтелекту у промислові процеси.

1.2 Основні проблеми комунікації між людиною та роботом

Ефективна комунікація між людиною та роботом є критично важливою для успішного функціонування роботизованих систем у промислових середовищах. Проте сучасні системи людино-машинної взаємодії (HMI) стикаються з низкою проблем, які ускладнюють цей процес. Ці проблеми пов’язані як із технічними обмеженнями, так і з людським фактором, що впливає на якість взаємодії та загальну продуктивність автоматизованих процесів. Нижче розглянуто основні виклики, які виникають у комунікації між оператором і роботом.

Недостатня інтуїтивність інтерфейсів. Багато сучасних HMI-систем, таких як сенсорні панелі чи програмні інтерфейси, є складними для сприйняття, особливо для операторів із обмеженим технічним досвідом. Наприклад, меню управління можуть бути перевантажені інформацією, а терміни чи позначення –

незрозумілими для новачків (рис. 1.1). Це призводить до помилок у введенні команд, затримок у виконанні завдань і навіть аварійних ситуацій. Хоча системи доповненої реальності (AR) або голосового керування пропонують більш природні способи взаємодії, їхня інтеграція все ще обмежена через високу вартість і складність налаштування

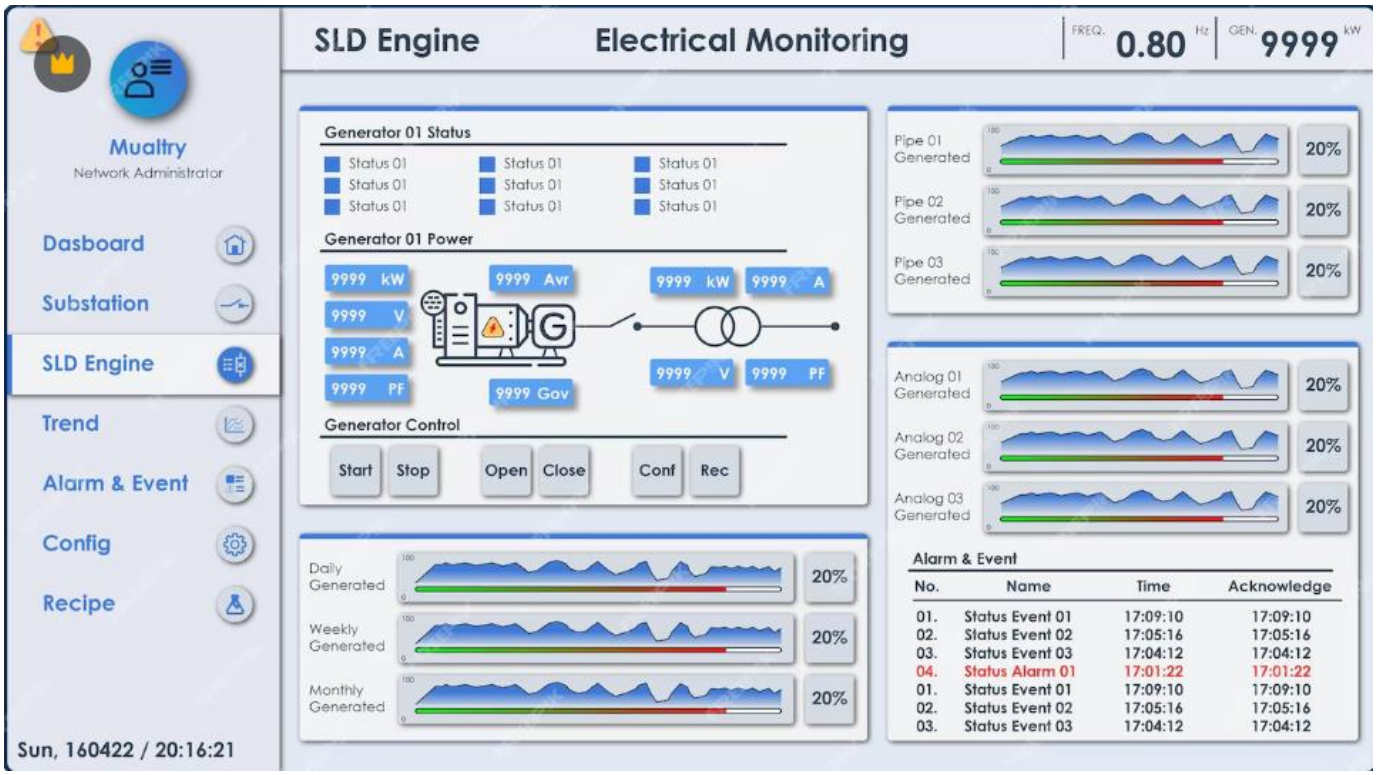


Рис. 1.1 Приклад складного HMI-інтерфейсу

Обмежена адаптивність роботів до людських потреб. Більшість промислових роботів запрограмовані на виконання чітко визначених завдань у стабільних умовах. Однак реальні виробничі умови часто передбачають динамічні зміни, які потребують швидкої адаптації. Роботи зазвичай не можуть інтерпретувати неточні чи неоднозначні команди оператора, що ускладнює їхню взаємодію з людиною. Наприклад, якщо оператор намагається скорегувати траєкторію руху маніпулятора в реальному часі, система може не розпізнати таку команду через відсутність гнучких алгоритмів обробки даних.

Проблеми зворотного зв’язку. Ефективна комунікація передбачає двосторонній обмін інформацією, але багато роботизованих систем надають недостатньо зрозумілий зворотний зв’язок. Наприклад, оператор може не отримувати чітких повідомлень про стан робота (чи виконується команда, чи

виникла помилка), що ускладнює контроль над системою. У деяких випадках інформація подається у вигляді кодів помилок або технічних логів, які потребують додаткового аналізу, що уповільнює реагування.

Когнітивне навантаження на оператора. Сучасні роботизовані системи часто передають оператору великий обсяг даних – від показників сенсорів до параметрів роботи обладнання. Обробка такої інформації в реальному часі створює значне когнітивне навантаження, що може призводити до втоми, зниження концентрації та помилок. Наприклад, під час роботи з кількома роботами одночасно оператору доводиться постійно перемикатися між різними інтерфейсами, що знижує ефективність.

Мовні та культурні бар'єри. У глобалізованому виробничому середовищі оператори можуть працювати з системами, розробленими в інших країнах, де інтерфейси використовують незнайому мову чи специфічні стандарти. Навіть якщо інтерфейс перекладено, неточності в термінології чи невідповідність культурним особливостям можуть ускладнювати взаємодію. Наприклад, різні регіони можуть мати власні підходи до позначення одиниць вимірювання чи формату даних.

Недостатній рівень довіри до автоматизованих систем. Психологічний аспект також відіграє важливу роль у комунікації. Деякі оператори можуть не довіряти роботам через страх помилок або несподіваної поведінки системи. Це призводить до надмірного контролю з їхнього боку, що знижує ефективність автоматизації. Наприклад, оператор може вручну перевіряти кожну операцію, замість того щоб покластися на запрограмовані алгоритми .

Ці проблеми вказують на необхідність розробки більш інтуїтивних, адаптивних і зрозумілих систем комунікації між людиною та роботом. Подолання цих викликів потребує вдосконалення інтерфейсів, застосування технологій штучного інтелекту для обробки природної мови та жестів, а також підвищення рівня підготовки операторів для роботи з сучасними системами.

1.3 Підходи до автоматизації взаємодії

Автоматизація взаємодії між людиною та роботом у промислових системах є важливим напрямом розвитку сучасної робототехніки. Метою таких підходів є спрощення комунікації, підвищення ефективності роботи операторів і мінімізація помилок, спричинених людським фактором. Для цього використовуються різноманітні технології та методи, які дозволяють зробити взаємодію більш інтуїтивною, адаптивною та надійною. Нижче розглянуто основні підходи до автоматизації людино-машинної взаємодії, які застосовуються в роботизованих системах.

Розробка інтуїтивних інтерфейсів HMI. Одним із ключових напрямів є створення людино-машинних інтерфейсів (HMI), які мінімізують когнітивне навантаження та спрощують керування роботами. Сучасні HMI включають сенсорні екрани, графічні панелі та системи візуалізації даних, які надають оператору чітку інформацію про стан системи. Наприклад, замість складних технічних логів дані можуть відображатися у вигляді діаграм або кольорових індикаторів. Для підвищення інтуїтивності застосовуються принципи ергономічного дизайну, такі як логічне групування елементів керування та використання зрозумілої термінології.

Використання технологій доповненої та віртуальної реальності (AR/VR). Технології AR і VR дозволяють операторам взаємодіяти з роботами в більш природний спосіб. Наприклад, за допомогою AR-окулярів оператор може бачити накладені на реальний простір підказки, такі як траєкторія руху маніпулятора чи статус обладнання. VR, у свою чергу, використовується для навчання операторів у віртуальному середовищі, де вони можуть відпрацьовувати навички без ризику для реального обладнання. Такі технології зменшують час на освоєння систем і підвищують точність виконання завдань[14].

Інтеграція обробки природної мови (NLP). Обробка природної мови є перспективним інструментом для автоматизації взаємодії. Системи, які розпізнають голосові команди або текстові запити, дозволяють операторам керувати роботами без необхідності вводити складні команди через інтерфейс.

Наприклад, оператор може сказати "зупинити конвеєр" або "змінити швидкість маніпулятора", а система автоматично переведе це в машинний код. Такі рішення особливо корисні в умовах, де руки оператора зайняті або використання сенсорного екрану є незручним.

Застосування жестового керування. Жестові інтерфейси дозволяють операторам взаємодіяти з роботами за допомогою рухів рук або тіла, що фіксуються спеціальними сенсорами, такими як камери чи датчики глибини. Наприклад, оператор може вказати напрям руху робота жестом або зупинити його, піднявши руку. Цей підхід є особливо ефективним у шумних виробничих середовищах, де голосові команди можуть бути менш надійними.

Використання штучного інтелекту та машинного навчання. Штучний інтелект (ШІ) відіграє ключову роль у автоматизації взаємодії, дозволяючи системам адаптуватися до потреб оператора. Алгоритми машинного навчання можуть аналізувати поведінку оператора, передбачати його дії та пропонувати оптимальні рішення. Наприклад, ШІ може автоматично налаштувати параметри роботи робота на основі попередніх команд або попередити про потенційні помилки. Крім того, системи на основі ШІ здатні розпізнавати складні шаблони в даних, що допомагає оператору швидше реагувати на нестандартні ситуації.[15]

Колаборативні роботизовані системи. Колаборативні роботи (коботи) розроблені для безпосередньої взаємодії з людиною без фізичних бар’єрів. Вони оснащені сенсорами, які дозволяють їм реагувати на присутність оператора, уникати зіткнень і адаптувати свої дії до його рухів. Наприклад, кобот може автоматично уповільнити роботу, якщо оператор наближається, або виконувати завдання спільно з людиною, наприклад, подавати деталі на складальній лінії. Такий підхід підвищує безпеку та ефективність взаємодії [13].

Автоматизація зворотного зв’язку. Для забезпечення якісної комунікації системи автоматизації взаємодії включають механізми зворотного зв’язку, які надають оператору чітку інформацію про стан робота. Наприклад, системи можуть використовувати звукові сигнали, вібрацію або візуальні індикатори для

повідомлення про завершення завдання чи виникнення помилки. Це дозволяє оператору швидко реагувати без необхідності аналізувати складні технічні дані.

Ці підходи, окремо або в комбінації, сприяють підвищенню ефективності взаємодії між людиною та роботом, зменшенню помилок і спрощенню роботи операторів. Однак їхня успішна реалізація залежить від правильного вибору технологій, адаптації до конкретних виробничих умов і врахування потреб кінцевих користувачів.

1.4 Технології штучного інтелекту в промисловості

Штучний інтелект (ШІ) став однією з ключових технологій, що трансформують промислове виробництво. Завдяки здатності обробляти великі обсяги даних, адаптуватися до змін і оптимізувати процеси, ШІ знаходить широке застосування в автоматизації, управлінні виробничими системами та підвищенні ефективності взаємодії між людиною і машиною. У промислових системах ШІ використовується для вирішення завдань, які потребують швидкого аналізу, прогнозування та прийняття рішень у реальному часі. Нижче розглянуто основні технології ШІ та їхній вплив на промисловість.

Машинне навчання (ML). Алгоритми машинного навчання дозволяють системам навчатися на основі даних, покращуючи свою продуктивність без необхідності явного програмування. У промисловості ML застосовується для прогнозного технічного обслуговування (predictive maintenance), коли системи аналізують дані з сенсорів обладнання, щоб передбачити можливі поломки. Наприклад, аналіз вібрацій чи температури може вказати на необхідність заміни деталей до виникнення збою. Крім того, ML використовується для оптимізації виробничих процесів, наприклад, для налаштування параметрів роботи верстатів з метою зниження витрат енергії чи матеріалів.

Комп’ютерний зір. Технології комп’ютерного зору дозволяють роботам і автоматизованим системам розпізнавати об’єкти, аналізувати зображення та виконувати завдання, які раніше вимагали людського втручання. У промисловості комп’ютерний зір застосовується для контролю якості, наприклад, для виявлення дефектів на поверхні деталей чи перевірки правильності складання компонентів

(рис. 1.2). Системи на основі ШІ можуть аналізувати зображення з камер у реальному часі, що значно прискорює процеси інспекції та знижує ймовірність помилок.[3]

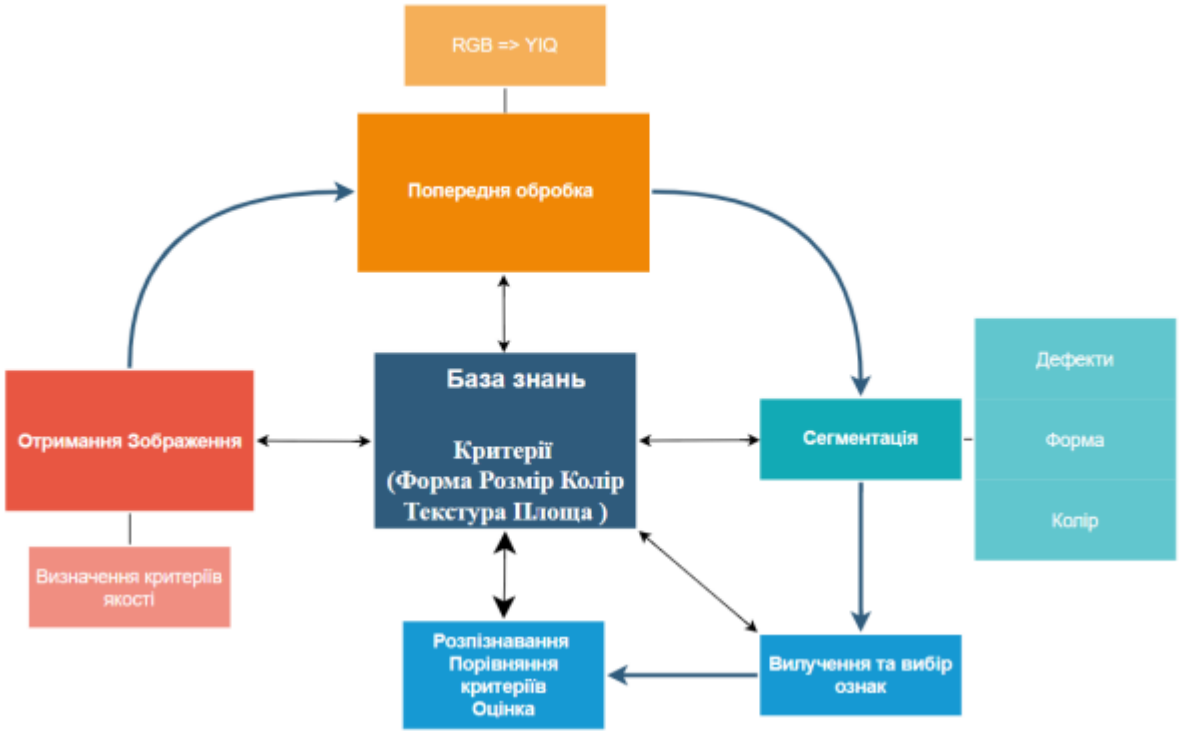


Рис. 1.2 Схема ключових етапів комп’ютерного зору для аналізу цифрових зображень.

Обробка природної мови (NLP). Технології обробки природної мови дозволяють створювати системи, які розуміють і генерують людську мову, що полегшує взаємодію між оператором і машиною. У промислових умовах NLP використовується для створення голосових асистентів, які дозволяють операторам віддавати команди голосом, наприклад, "запустити конвеєр" чи "перевірити статус обладнання". Це особливо корисно в ситуаціях, коли руки оператора зайняті або доступ до інтерфейсу обмежений. NLP також застосовується для аналізу текстових звітів чи логів, що допомагає швидше виявляти проблеми.[4]

Робототехніка на основі ШІ. ШІ значно розширює можливості промислових роботів, дозволяючи їм виконувати складніші завдання. Наприклад, алгоритми глибинного навчання (deep learning) дають роботам змогу адаптуватися до змін у навколишньому середовищі, таких як зміщення об’єктів на конвеєрі чи поява перешкод. Колаборативні роботи (коботи), оснащені ШІ, можуть працювати пліч-

о-пліч з операторами, розпізнаючи їхні рухи та забезпечуючи безпеку. Такі системи здатні навчатися на основі взаємодії з людиною, що підвищує їхню гнучкість.

Інтелектуальна аналітика та оптимізація. ШІ використовується для аналізу великих обсягів даних (big data), що генеруються промисловими системами. Інструменти інтелектуальної аналітики дозволяють виявляти приховані закономірності, оптимізувати ланцюги поставок і прогнозувати попит. Наприклад, ШІ може аналізувати дані про продуктивність обладнання, щоб запропонувати оптимальний графік роботи, який зменшить простої та підвищить ефективність. Такі рішення також застосовуються для управління енергоспоживанням, що сприяє зниженню витрат і екологічному виробництву.

Автономні системи. ШІ лежить в основі автономних транспортних засобів і роботів, які використовуються на складах, у логістиці та на виробничих майданчиках. Наприклад, автономні навантажувачі, керовані ШІ, можуть самостійно пересуватися цехом, уникаючи перешкод і оптимізуючи маршрути. Такі системи знижують залежність від людського втручання та підвищують швидкість виконання логістичних завдань.

Використання ШІ у промисловості не лише підвищує продуктивність, але й створює нові виклики, такі як необхідність інтеграції з існуючими системами, забезпечення кібербезпеки та підготовка кваліфікованих кадрів. Проте потенціал цих технологій для оптимізації виробничих процесів і полегшення роботи операторів робить їх незамінними в сучасній промисловості [5].

2. ПРОЄКТУВАННЯ ВІРТУАЛЬНОГО АСИСТЕНТА

2.1 Формування загальної концепції системи

Формування загальної концепції віртуального асистента для оптимізації взаємодії між оператором і промисловими роботами є першим і ключовим етапом проєктування системи. Концепція визначає основні принципи роботи асистента, його функціональні можливості, цільові задачі та принципи інтеграції з існуючими виробничими процесами. Метою системи є спрощення комунікації, підвищення ефективності роботи операторів і зниження ймовірності помилок у роботизованих системах. Нижче описано основні аспекти формування концепції віртуального асистента.

Цільові задачі системи. Віртуальний асистент розробляється для виконання кількох основних функцій: автоматизації рутинних операцій, надання оператору контекстної інформації в реальному часі та підтримки у прийнятті рішень. Наприклад, асистент може автоматично налаштовувати параметри роботи робота на основі поточних умов виробництва, надавати рекомендації щодо усунення несправностей або попереджати про потенційні збої. Крім того, система має забезпечувати інтуїтивну взаємодію, дозволяючи оператору використовувати голосові команди, жести або сенсорні інтерфейси для керування роботами.

Модульна архітектура. Концепція передбачає створення модульної системи, яка складається з кількох взаємопов’язаних компонентів: модуля обробки даних, інтерфейсу взаємодії, аналітичного модуля на основі штучного інтелекту (ШІ) та модуля інтеграції з роботизованими системами. Модульність забезпечує гнучкість і масштабованість, дозволяючи адаптувати асистента до різних типів обладнання чи виробничих умов. Наприклад, модуль обробки даних може збирати інформацію з сенсорів робота, тоді як аналітичний модуль використовує алгоритми машинного навчання для прогнозування оптимальних параметрів роботи.

Використання технологій ШІ. Штучний інтелект є центральним елементом системи, забезпечуючи її здатність до адаптації та інтелектуальної обробки інформації. Алгоритми обробки природної мови (NLP) дозволяють асистенту

розпізнавати голосові команди оператора, а технології комп’ютерного зору дають змогу аналізувати візуальні дані, наприклад, для перевірки якості деталей. Крім того, алгоритми машинного навчання використовуються для аналізу історичних даних і прогнозування поведінки системи, що допомагає оператору швидше реагувати на нестандартні ситуації.

Інтуїтивний інтерфейс користувача. Концепція системи передбачає створення інтерфейсу, який мінімізує когнітивне навантаження на оператора. Це може бути комбінація сенсорних екранів, систем доповненої реальності (AR) або голосового керування. Наприклад, за допомогою AR-окулярів оператор може бачити накладені на реальний простір підказки, такі як інструкції з налаштування робота чи попередження про відхилення в роботі. Голосовий інтерфейс дозволяє віддавати команди без необхідності взаємодії з фізичними пристроями, що особливо важливо в умовах обмеженого доступу до обладнання [6].

Інтеграція з виробничими системами. Віртуальний асистент має бути сумісним з існуючими промисловими системами, такими як SCADA (Supervisory Control and Data Acquisition) або MES (Manufacturing Execution Systems). Це забезпечує безперебійний обмін даними між асистентом, роботами та іншими компонентами виробництва. Концепція передбачає використання стандартних протоколів зв’язку, таких як OPC UA чи MQTT, для забезпечення надійної інтеграції.

Безпека та надійність. Важливим аспектом концепції є забезпечення кібербезпеки та захисту даних. Система має використовувати шифрування для передачі інформації та механізми автентифікації для запобігання несанкціонованому доступу. Крім того, асистент повинен бути стійким до збоїв, забезпечуючи безперервну роботу навіть у разі відмови окремих компонентів.

Орієнтація на користувача. Концепція системи розробляється з урахуванням потреб кінцевих користувачів – операторів, які можуть мати різний рівень технічної підготовки. Для цього передбачається створення адаптивного інтерфейсу, який налаштовується під досвід користувача: від спрощених команд для новачків до розширених функцій для експертів. Також планується включення

функції навчання, яка дозволяє операторам швидше освоювати систему через інтерактивні підказки та симуляції.

Загальна концепція віртуального асистента спрямована на створення гнучкої, інтелектуальної та користувацько-орієнтованої системи, яка оптимізує взаємодію між оператором і промисловими роботами. Вона поєднує передові технології ШІ, інтуїтивні інтерфейси та надійну інтеграцію, щоб забезпечити підвищення продуктивності та безпеки на виробництві.

2.2 Архітектура взаємодії людини з асистентом

Архітектура взаємодії людини з віртуальним асистентом є основою для забезпечення ефективної, інтуїтивної та надійної комунікації між оператором і промисловими роботами. Вона визначає, як асистент отримує вхідні дані від користувача, обробляє їх, взаємодіє з роботизованою системою та надає зворотний зв’язок. Архітектура розроблена з урахуванням гнучкості, безпеки та адаптивності до різних виробничих умов. Нижче описано ключові компоненти архітектури та принципи їхньої взаємодії.

Інтерфейс користувача (UI). Основним каналом взаємодії є інтерфейс користувача, який включає кілька типів взаємодії: сенсорні екрани, голосові команди та системи доповненої реальності (AR). Сенсорний інтерфейс забезпечує доступ до графічного представлення даних, таких як статус робота чи параметри роботи. Голосовий інтерфейс, заснований на обробці природної мови (NLP), дозволяє оператору віддавати команди голосом, наприклад, "змінити швидкість маніпулятора" або "перевірити сенсори". AR-інтерфейс, реалізований через спеціальні окуляри, надає візуальні підказки, такі як накладання інструкцій на обладнання в реальному часі. Ці компоненти розроблені для зменшення когнітивного навантаження та підвищення швидкості реакції оператора.[7]

Модуль обробки вхідних даних. Цей модуль відповідає за збір і первинну обробку даних, отриманих від оператора та зовнішніх джерел, таких як сенсори робота чи системи управління виробництвом (SCADA, MES). Вхідні дані можуть включати голосові команди, жести, введення через сенсорний екран або сигнали від обладнання. Модуль використовує алгоритми розпізнавання мови та жестів,

щоб перетворити ці дані в машинно-зрозумілі команди. Наприклад, голосова команда "зупинити конвеєр" обробляється NLP-алгоритмами, які переводять її в команду для системи керування.

Аналітичний модуль на основі ШІ. Центральним елементом архітектури є аналітичний модуль, який використовує алгоритми штучного інтелекту для обробки даних і прийняття рішень. Цей модуль включає моделі машинного навчання для прогнозування, класифікації та оптимізації. Наприклад, на основі історичних даних про роботу обладнання асистент може передбачити потенційні збої та запропонувати оператору профілактичні дії. Комп'ютерний зір, інтегрований у цей модуль, дозволяє аналізувати зображення з камер для виявлення дефектів або контролю правильності виконання завдань. Аналітичний модуль також адаптує відповіді асистента до рівня підготовки оператора, надаючи спрощені чи розширені рекомендації.

Модуль взаємодії з роботами. Цей компонент забезпечує зв'язок між асистентом і промисловими роботами через стандартні протоколи зв'язку, такі як OPC UA, MQTT або ROS (Robot Operating System). Модуль передає команди від асистента до роботів, а також отримує зворотні дані, такі як статус виконання чи показники сенсорів. Наприклад, якщо оператор через голосовий інтерфейс наказує змінити траєкторію маніпулятора, модуль трансформує цю команду в набір інструкцій для контролера робота. Для забезпечення безпеки модуль включає механізми перевірки команд на відповідність заданим обмеженням.

Модуль зворотного зв'язку. Для ефективної взаємодії асистент має надавати оператору чіткий і своєчасний зворотний зв'язок. Цей модуль генерує повідомлення у вигляді текстових сповіщень, звукових сигналів або візуальних індикаторів. Наприклад, після виконання команди асистент може відобразити на екрані повідомлення "Завдання виконано" або попередити голосом про виявлену помилку. У разі використання AR зворотний зв'язок може виглядати як кольорові позначки на обладнанні, що вказують на проблемні зони.

Забезпечення безпеки та надійності. Архітектура включає механізми захисту даних і запобігання несанкціонованому доступу. Усі канали зв'язку

шифруються, а доступ до системи обмежується через автентифікацію користувача (наприклад, біометричні дані чи паролі). Для забезпечення надійності передбачено резервування критичних компонентів і можливість роботи в автономному режимі в разі збою мережі. Крім того, асистент має функцію самодіагностики, яка дозволяє виявляти внутрішні помилки та повідомляти про них оператора.

Адаптивність і масштабованість. Архітектура розроблена так, щоб її можна було адаптувати до різних типів роботів, виробничих ліній і потреб операторів. Наприклад, модульна структура дозволяє додавати нові інтерфейси (такі як підтримка жестів) або інтегрувати асистента з іншими системами без значних змін у коді. Це забезпечує можливість масштабування системи для використання на великих підприємствах або в різних галузях.

2.3 Використання методів машинного навчання

Методи машинного навчання (ML) є ключовим компонентом віртуального асистента, призначеного для оптимізації взаємодії між оператором і промисловими роботами. Вони дозволяють системі аналізувати дані, адаптуватися до змін у виробничих умовах і надавати оператору інтелектуальні рекомендації. У контексті розробки асистента методи ML застосовуються для обробки вхідних даних, прогнозування подій і автоматизації прийняття рішень. Нижче описано основні підходи машинного навчання, які використовуються в системі, та їхнє практичне застосування.[8]

Класифікація та розпізнавання. Алгоритми класифікації, такі як підтримка векторних машин (SVM) або нейронні мережі, використовуються для обробки вхідних даних від оператора, зокрема голосових команд чи жестів. Наприклад, система може розпізнавати голосові інструкції, такі як "зупинити маніпулятор", і класифікувати їх як конкретні команди для виконання. Це дозволяє асистенту швидко інтерпретувати наміри оператора та передавати відповідні інструкції роботам. Крім того, алгоритми класифікації застосовуються для аналізу даних із сенсорів, щоб визначити, чи є відхилення в роботі обладнання, наприклад, виявлення аномалій у температурі чи вібрації.

Прогнозне моделювання. Алгоритми регресії та часові ряди використовуються для прогнозування потенційних збоїв або оптимізації роботи системи. Наприклад, на основі історичних даних про продуктивність робота асистент може передбачити, коли обладнання потребуватиме технічного обслуговування, що допомагає запобігти простоям. Такі моделі, як рекурентні нейронні мережі (RNN), ефективні для аналізу послідовних даних, наприклад, показників сенсорів у реальному часі. Це дозволяє асистенту надавати оператору рекомендації, такі як зміна параметрів роботи для зниження енергоспоживання.

Глибинне навчання для комп'ютерного зору. Глибинне навчання (deep learning), зокрема згорткові нейронні мережі (CNN) (рис. 2.1), застосовується для обробки візуальних даних, отриманих із камер промислових роботів. Наприклад, асистент може аналізувати зображення деталей на конвеєрі, щоб виявити дефекти або перевірити правильність складання. Такий підхід зменшує потребу в ручній інспекції та підвищує точність контролю якості. Крім того, комп'ютерний зір дозволяє асистенту розпізнавати жести оператора, що є корисним у шумних виробничих умовах, де голосові команди менш ефективні [9].

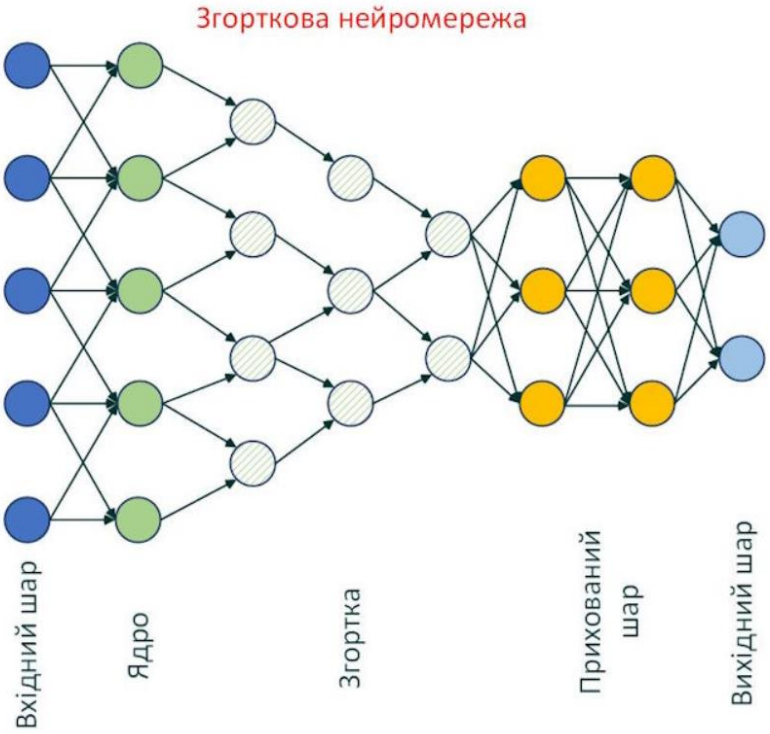


Рис. 2.1 Приклад роботи згорткової нейронної мережі (CNN)

Навчання з підкріпленням. Алгоритми навчання з підкріпленням (reinforcement learning) використовуються для оптимізації поведінки асистента в динамічних умовах. Наприклад, система може навчатися вибирати оптимальні параметри роботи робота, такі як швидкість чи траєкторія маніпулятора, шляхом проб і помилок, максимізуючи ефективність виконання завдання. Цей підхід особливо корисний у ситуаціях, коли виробничі умови постійно змінюються, а заздалегідь запрограмовані сценарії стають неефективними.[10]

Обробка природної мови (NLP). Методи NLP, засновані на моделях машинного навчання, таких як трансформери, дозволяють асистенту розуміти та генерувати людську мову. Це забезпечує можливість обробки голосових команд і текстових запитів від оператора. Наприклад, асистент може інтерпретувати команду "перевірити статус обладнання" і надати відповідь у вигляді зрозумілого текстового чи голосового повідомлення. NLP також використовується для аналізу текстових звітів або логів системи, що допомагає оператору швидше виявляти проблеми.

Персоналізація взаємодії. Алгоритми кластеризації, такі як k-means, застосовуються для адаптації асистента до індивідуальних потреб операторів. Наприклад, система може аналізувати історію взаємодії користувача, щоб визначити його рівень підготовки, і пропонувати спрощені або розширені інтерфейси. Це підвищує зручність роботи та знижує час, необхідний для освоєння системи новачками.

Для реалізації цих методів необхідно зібрати та підготувати набори даних, що включають інформацію з сенсорів, голосові команди, зображення та журнали роботи обладнання. Дані проходять попередню обробку, таку як нормалізація чи аугментація, щоб підвищити якість навчання моделей. Крім того, система потребує періодичного перенавчання моделей для адаптації до нових умов виробництва.

Використання методів машинного навчання дозволяє віртуальному асистенту не лише автоматизувати рутинні операції, але й надавати інтелектуальну підтримку оператору, підвищуючи ефективність і безпеку роботи

промислових роботів. Ці методи забезпечують гнучкість і адаптивність системи, що є критично важливим у сучасних автоматизованих виробництвах.

2.4 Підбір інструментів та середовищ розробки

Підбір інструментів і середовищ розробки для створення віртуального асистента, призначеного для оптимізації взаємодії між оператором і промисловими роботами, є важливим етапом проєктування. Вибір залежить від функціональних вимог системи, таких як обробка даних у реальному часі, інтеграція з роботизованими системами, підтримка штучного інтелекту (ШІ) та забезпечення інтуїтивного інтерфейсу. Для реалізації системи необхідно використовувати інструменти, які забезпечують гнучкість, масштабованість і сумісність з промисловими стандартами. Нижче описано основні інструменти та середовища розробки, які застосовуються для створення асистента.

Мови програмування. Для розробки віртуального асистента використовується Python завдяки його універсальності, багатій екосистемі бібліотек і підтримці ШІ. Python підходить для реалізації алгоритмів машинного навчання, обробки природної мови (NLP) та комп'ютерного зору. Наприклад, бібліотеки, такі як TensorFlow і PyTorch, дозволяють створювати моделі для прогнозування збоїв або розпізнавання жестів. Крім того, для створення інтерфейсів і обробки даних у реальному часі може використовуватися JavaScript, зокрема для веб-додатків, які забезпечують віддалений доступ до асистента.

Фреймворки машинного навчання. Для реалізації алгоритмів машинного навчання використовуються відкриті фреймворки TensorFlow і PyTorch. TensorFlow забезпечує високу продуктивність для обробки великих наборів даних, що необхідно для аналізу сенсорних даних із промислових роботів. PyTorch, у свою чергу, є гнучким для створення прототипів моделей глибокого навчання, таких як згорткові нейронні мережі для комп'ютерного зору. Ці фреймворки підтримують розгортання моделей на різних платформах, що полегшує інтеграцію асистента з виробничими системами.[11]

Інструменти для обробки природної мови. Для реалізації голосового інтерфейсу використовуються бібліотеки NLP, такі як spaCy і Hugging Face

Transformers. SpaCy забезпечує швидку обробку текстових даних і розпізнавання команд, що дозволяє асистенту розуміти голосові запити оператора, наприклад, "перевірити статус конвеєра". Hugging Face Transformers підходить для створення складніших моделей, які можуть генерувати відповіді або аналізувати текстові звіти. Ці інструменти є відкритими та широко застосовуються в промислових додатках.

Інструменти комп'ютерного зору. Для обробки зображень із камер промислових роботів використовується бібліотека OpenCV, яка підтримує завдання комп'ютерного зору, такі як виявлення дефектів або розпізнавання жестів. OpenCV є ефективним для роботи в реальному часі та сумісним з Python, що полегшує його інтеграцію з іншими компонентами системи. Наприклад, асистент може використовувати OpenCV для аналізу якості деталей на конвеєрі, надаючи оператору результати у вигляді візуальних звітів.[12]

Середовища розробки інтерфейсів. Для створення інтуїтивного інтерфейсу користувача використовуються веб-технології, такі як React.js, який дозволяє розробляти динамічні сенсорні інтерфейси для відображення даних і керування роботами. React.js підтримує створення адаптивних додатків, які можуть працювати на різних пристроях, від планшетів до промислових панелей. Для реалізації доповненої реальності (AR) застосовується Unity з плагіном Vuforia, що дозволяє створювати AR-додатки для накладання підказок на обладнання через спеціальні окуляри.

Протоколи та інструменти інтеграції. Для забезпечення зв'язку асистента з промисловими роботами використовуються стандартні протоколи, такі як OPC UA і MQTT. OPC UA підтримує безпечний обмін даними між асистентом і системами управління, такими як SCADA. MQTT є легким протоколом для передачі даних у реальному часі, що підходить для IoT-додатків у виробництві. Для роботи з цими протоколами використовується Node-RED, який дозволяє створювати гнучкі сценарії інтеграції через графічний інтерфейс.[1]

Середовища розробки та тестування. Для розробки та тестування системи використовуються інтегровані середовища розробки (IDE), такі як Visual Studio

Code і Jupyter Notebook. Visual Studio Code підтримує роботу з Python, JavaScript і розширення для роботи з ML-фреймворками, що полегшує написання та налагодження коду. Jupyter Notebook є зручним для створення прототипів моделей машинного навчання, дозволяючи тестувати алгоритми на реальних даних із сенсорів. Для управління проєктом і спільної розробки застосовується Git із платформою GitHub, що забезпечує контроль версій і співпрацю команди.

Інструменти для розгортання. Для розгортання асистента на промислових серверах використовується Docker, який дозволяє створювати контейнери для ізольованого виконання додатків. Це забезпечує сумісність системи з різними апаратними платформами та полегшує масштабування. Для обробки великих обсягів даних у реальному часі застосовується Apache Kafka, що підтримує потокову передачу даних між асистентом і роботизованими системами.

Використання цих інструментів і середовищ дозволяє створити гнучку, масштабовану та ефективну систему, яка відповідає вимогам промислового виробництва. Вибір відкритих інструментів забезпечує доступність, підтримку спільноти та можливість швидкої адаптації до нових технологій.

3. РЕАЛІЗАЦІЯ І ТЕСТУВАННЯ ПРОТОТИПУ СИСТЕМИ

3.1 Реалізація основних функцій віртуального асистента

3.1.1 Розташування робота і стелажів на складі

Метод initialize_positions, який частково видно на (рис. 3.1), потрібен, щоб налаштувати початкові позиції для робота, предметів і стелажів. Він спочатку викликає метод generate_fixed_shelves, щоб зробити стелажі, задає роботу стартову позицію через self.robot_pos = [45, 26], а потім додає предмети на стелажі. На (рис. 3.1) можна побачити початок цього методу: спочатку місце робота обнуляється через self.robot_pos = None, список предметів очищається через self.items = {}, стелажі також обнуляються через self.shelves = [], а потім викликається метод generate_fixed_shelves для створення нових стелажів.

Далі на (рис. 3.1) є список self.delivery_points, де вказані координати точок, куди робот має приносити предмети на складі розміром 50x50 клітин. Ці точки розташовані в різних зонах: наприклад, є горизонтальна лінія від [41, 23] до [49, 23] на рівні y=23, вертикальні лінії, як від [49, 24] до [49, 30] на правому краю при x=49, а ще на лівому краю від [41, 20] до [41, 29] та в середині, наприклад, [43, 25] і [44, 25]. Це ті місця, куди робот доставлятиме предмети зі стелажів.

Метод generate_fixed_shelves створює стелажі на складі, розміщуючи їх за певними правилами. Він працює з двома циклами: перший цикл for i in range(0, 50) перебирає x-координати від 0 до 49, і якщо і ділиться на 8 без остачі (умова if(i % 8 == 0)), то колонка пропускається, щоб залишити місце для проходів. Другий цикл for j in range(-1, 50, 4) перебирає y-координати з кроком 4, починаючи з -1. Умова if(i > 40 and i < 50 and j > 20 and j < 30) не дозволяє ставити стелажі там, де є точки доставки (між x=41 і x=49 та y=21 і y=29), щоб вони не заважали. У кожному кроці додаються дві позиції стелажів [i, j+1] і [i, j+2], щоб у кожній колонці було по два стелажі поруч.

Цей код допомагає організувати склад так, щоб усе було на своїх місцях: стелажі стоять рівно, є проходи для робота, а точки доставки залишаються

вільними. Це потрібно, щоб робот міг легко пересуватися і переносити предмети туди, куди треба.

Додаткові деталі:

- Проходи: пропуск кожної восьмої колонки через умову if(i % 8 == 0) дає роботу місце для руху.
- Точки доставки: зона між x=41 і x=49 та y=21 і y=29 залишається вільною, щоб стелажі не заважали доставці.
- Розташування стелажів: стелажі додаються парами ([i, j+1] і [i, j+2]), щоб було зручно ставити предмети.

```
self.delivery_points = [
    [41, 23],[42, 23],[43, 23],[44, 23],[46, 23],[47, 23],[48, 23],[49, 23],
    [49, 24],[49, 25],[49, 26],[49, 27],[49, 28],[49, 29],[49, 30],
    [41, 30],[42, 30],[43, 30],[44, 30],[46, 30],[47, 30],[48, 30],
    [41, 20],[41, 21],[41, 24],[41, 25],[41, 28],[41, 29],

    [43, 25],[44, 25],[46, 25],[47, 25],
    [47, 26],[47, 27],[47, 28],
    [43, 28],[44, 28],[46, 28],[47, 28],
]
```

Рис. 3.1 Код методів initialize_positions і generate_fixed_shelves та списку delivery_points для створення стелажів і точок доставки

3.1.2 Метод, який встановлює початкові позиції робота та предметів, а також викликається, коли потрібно створити нову конфігурацію.

Метод initialize_positions (рис. 3.2) готує склад до роботи, визначаючи, де спочатку будуть стояти робот, стелажі та предмети. Спочатку він обнуляє місце робота через self.robot_pos = None, щоб прибрати старі дані. Потім очищає список предметів за допомогою self.items = {}, видаляючи всі попередні предмети, і обнуляє список стелажів через self.shelves = [], щоб почати все заново.

Далі запускається метод generate_fixed_shelves (рис 3.2), який робить стелажі в тих місцях, що були визначені раніше. Після цього для робота задається конкретне місце на складі — координати [45, 26] через self.robot_pos = [45, 26], щоб він завжди починав із цієї точки. Кількість предметів обирається випадково

від 5 до 35 за допомогою команди num_items = random.randint(5, 35), як і потрібно за умовами завдання.

Потім програма робить копію списку стелажів через shuffled_shelves = self.shelves.copy() і перемішує її за допомогою random.shuffle(shuffled_shelves), щоб предмети ставилися на різні стелажі. Далі в циклі for і in range(min(num_items, len(shuffled_shelves))) для кожного предмета створюється свій номер через item_id = f"item_{i}", наприклад, "item_0" чи "item_1". Предмету задається бірюзовий колір через item_color = "#08e8de", місце обирається зі списку перемішаних стелажів через item_pos = shuffled_shelves[i], і предмет додається до списку self.items із цими даними через self.items[item_id] = {"color": item_color, "pos": item_pos}.

Цей метод налаштовує склад, визначаючи, де будуть стелажі, робот і предмети. Його використовують як на старті програми, так і коли потрібно зробити нову розстановку, наприклад, після натискання кнопки "Нова конфігурація". Робот завжди починає з точки [45, 26], щоб усі сценарії мали однаковий початок, а випадкова кількість предметів від 5 до 35 додає різноманітності. Предмети ставляться на стелажі в різному порядку, щоб можна було відтворювати різні ситуації на складі.

Додаткові деталі:

- Різноманітність: перемішування стелажів через random.shuffle(shuffled_shelves) робить розташування предметів непередбачуваним.
- Колір: усі предмети мають бірюзовий колір #08e8de, що є однаковим для всіх.
- Обмеження: цикл бере до уваги min(num_items, len(shuffled_shelves)), щоб не поставити більше предметів, ніж є стелажів.

```
def initialize_positions(self):
    self.robot_pos = None
    self.items = {}
    self.shelves = []

    self.generate_fixed_shelves()

    self.robot_pos = [45, 26]

    num_items = random.randint(5, 35)

    shuffled_shelves = self.shelves.copy()
    random.shuffle(shuffled_shelves)

    for i in range(min(num_items, len(shuffled_shelves))):
        item_id = f"item_{i}"
        item_color = "#08e8de"
        item_pos = shuffled_shelves[i]

        self.items[item_id] = {
            "color": item_color,
            "pos": item_pos
        }
```

Рис. 3.2 Код методу initialize_positions для ініціалізації позицій робота та товарів

3.1.3 Стелажі, на яких з’являються товари для доставки на складі

Функція generate_fixed_shelves (рис 3.3) створює стелажі в певних місцях на складі. Вона працює з двома циклами. Перший цикл for i in range(0, 50) перебирає x-координати від 0 до 49, тобто йде по ширині сітки, яка має розмір 50x50. Умова if(i % 8 == 0) пропускає кожну восьму колонку, щоб залишити вільні проходи, якими робот зможе рухатися.

Другий цикл for j in range(-1, 50, 4) перебирає y-координати з кроком 4, починаючи з -1, тобто йде по висоті сітки, пропускаючи по 4 клітинки. Усередині цього циклу є ще одна умова if(i > 40 and i < 50 and j > 20 and j < 30), яка не дозволяє ставити стелажі в зоні між x=41 і x=49 та y=21 і y=29. Ця зона зарезервована для точок доставки, щоб стелажі їх не закривали.

Якщо всі умови дозволяють, програма додає два стелажі поруч: один на позиції [i, j+1], а другий на [i, j+2]. Це означає, що в кожній колонці стелажі стоять парами, одна клітинка над іншою. Ці позиції записуються в список self.shelves через self.shelves.append.

Ця функція потрібна, щоб зробити стелажі, на яких потім розміщуються предмети. Вона дбає про те, щоб між стелажами були проходи для робота і щоб стелажі не заважали точкам доставки.

Додаткові деталі:

- Проходи: пропуск кожної восьмої колонки через if(i % 8 == 0) дає роботу місце для руху.
- Зона доставки: умова між x=41 і x=49 та y=21 і y=29 залишає місце для точок доставки.
- Пари: стелажі додаються по два ([i, j+1] і [i, j+2]), щоб предмети було зручно розміщувати.

```
def generate_fixed_shelves(self):
    self.shelves = []

    for i in range(0, 50):
        if(i % 8 == 0):
            continue

        for j in range(-1, 50, 4):
            if(i > 40 and i < 50 and j > 20 and j < 30):
                continue

            self.shelves.append([i,j+1])
            self.shelves.append([i,j+2])
```

Рис. 3.3 Код методу generate_fixed_shelves для створення стелажів на складі

3.1.4 Метод створення UI-інтерфейсу для всієї програми

Метод create_layout (рис. 3.4) у програмі відповідає за створення зручного вигляду програми, який ділиться на дві частини: ліву і праву. Спершу він налаштовує основу вікна за допомогою self.grid_columnconfigure(0, weight=1) і self.grid_columnconfigure(1, weight=1), щоб лівий і правий боки розтягувалися рівномірно при зміні розміру вікна. Також через self.grid_rowconfigure(0, weight=1) задається один рядок, який займає весь простір по висоті.

Далі створюється ліва частина вікна через self.left_frame = ctk.CTkFrame(self), яку розміщують у першій колонці (column=0) з відступами (padx=10, pady=10) і розтягують по всьому простору (sticky="nsew"). У цій частині налаштовують одну колонку через self.left_frame.grid_columnconfigure(0, weight=1), а два рядки — через self.left_frame.grid_rowconfigure(0, weight=3) і self.left_frame.grid_rowconfigure(1, weight=1), з пропорцією 3:1, де верхній рядок для записів подій більший.

У верхній частині лівого боку додають місце для логів через self.log_frame, розміщуючи його у першому рядку (row=0, column=0) з відступами. Тут з’являється напис "Історія подій:" через self.log_label, вирівняний ліворуч (anchor="w"), і текстове поле self.log_text (типу CTkTextbox), яке розтягується (fill="both", expand=True) і підтримує перенесення слів (wrap="word").

Потім у нижній частині лівого боку створюють місце для кнопок через self.buttons_frame, розміщуючи його у другому рядку (row=1, column=0). Задають розміри кнопок: button_width = 200 і button_height = 40. Додають три кнопки: self.select_item_btn ("Доставка предмета" з викликом self.start_item_selection), self.move_item_btn ("Доставити предмет" з self.deliver_item, спочатку неактивна через state="disabled"), і self.extra_btn ("Нова конфігурація" з self.regenerate_configuration).

Додаткові деталі:

- Рядки: верхній рядок лівого боку втричі більший (3:1) для логів.
- Кнопки: розмір 200x40 робить їх зручними.
- Полотно: події <configure> і <button-1> забезпечують швидке оновлення.</button-1></configure>

```python
def create_layout(self):
    self.grid_columnconfigure(0, weight=1)
    self.grid_columnconfigure(1, weight=1)
    self.grid_rowconfigure(0, weight=1)

    self.left_frame = ctk.CTkFrame(self)
    self.left_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

    self.left_frame.grid_columnconfigure(0, weight=1)
    self.left_frame.grid_rowconfigure(0, weight=3)
    self.left_frame.grid_rowconfigure(1, weight=1)

    self.log_frame = ctk.CTkFrame(self.left_frame)
    self.log_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

    self.log_label = ctk.CTkLabel(self.log_frame, text="Історія подій:", anchor="w")
    self.log_label.pack(padx=10, pady=5, anchor="w")

    self.log_text = ctk.CTkTextbox(self.log_frame, wrap="word")
    self.log_text.pack(padx=10, pady=5, fill="both", expand=True)

    self.buttons_frame = ctk.CTkFrame(self.left_frame)
    self.buttons_frame.grid(row=1, column=0, padx=10, pady=10, sticky="nsew")

    button_width = 200
    button_height = 40

    self.select_item_btn = ctk.CTkButton(
        self.buttons_frame,
        text="Доставка предмета",
        command=self.start_item_selection,
        width=button_width,
        height=button_height
    )
    self.select_item_btn.pack(padx=10, pady=10)
```

Рис 3.4 Код методу create_layout для створення графічного інтерфейсу програми

Метод draw_grid (рис. 3.5) у програмі потрібен, щоб показати склад на екрані. Він спочатку прибирає все старе з полотна через команду self.grid_canvas.delete("all"), щоб можна було намалювати все заново. Потім він дізнається розміри полотна: ширину через canvas_width = self.grid_canvas.winfo_width() і висоту через canvas_height = self.grid_canvas.winfo_height(), щоб знати, скільки місця є для малювання.

Щоб сітка добре виглядала, програма вибирає менший розмір із ширини чи висоти через available_size = min(canvas_width, canvas_height) - 40 і віднімає 40 пікселів, щоб залишити місце для країв. Розмір однієї клітинки сітки вираховують через self.cell_size = available_size / self.grid_size, де сітка має розмір 50 на 50 клітинок.

Далі вираховують, як посунути сітку через offset_x і offset_y, щоб вона була посередині полотна, враховуючи розміри вікна. Перший цикл for i in range(self.grid_size + 1) малює лінії сітки: для вертикальних ліній визначається

місце через x = offset_x + i * self.cell_size, і вони малюються сірим кольором (#CCCCCC) від верху до низу, а для горизонтальних ліній місце вираховується через y = offset_y + i * self.cell_size, і вони також малюються сірим від лівого краю до правого.

Інший цикл for i in range(0, self.grid_size + 1, 5) ставить цифри через кожні 5 клітинок: по осі x цифри (0, 5, 10, ..., 50) додаються під сіткою, а по осі y — зліва від сітки. Потім запускається метод self.highlight_empty_delivery_points(offset_x, offset_y), який показує вільні місця для доставки сірим кольором.

Кожен стелаж із self.shelves малюється як коричневий квадрат із кольором #8B4513 і рамкою #654321. Для кожної клітинки зі списку self.path_cells, яка показує, куди йшов робот, малюється жовтий квадрат із кольором #FFFF00, який трохи прозорий через stipple="gray50". Для кожного предмета зі списку self.items, якщо робот його не тримає (self.held_item != item_id), малюється квадрат із кольором предмета item_data["color"] і маленькими відступами всередині клітинки.

Робот показується як червоний кружечок із кольором #FF0000 у тому місці, де він є, через self.robot_pos. Якщо робот тримає предмет (self.held_item не порожній), у середині кружечка малюється маленький кружечок із кольором предмета. Позиція сітки зберігається в self.grid_offset_x і self.grid_offset_y, щоб потім обробляти кліки мишею.

Цей метод малює склад на екрані, показуючи сітку, цифри, стелажі, предмети, точки доставки, робота і його шлях. Він підлаштовується під розмір вікна, змінюючи розмір клітинок, і робить усе чітким. Метод працює, коли вікно змінюється, або після дій робота, наприклад, коли він рухається чи доставляє предмет, щоб усе оновити.

Додаткові деталі:

- Розмір: self.cell_size залежить від available_size, щоб сітка підходила під розмір вікна.
- Колір ліній: сірий #CCCCCC робить сітку видимою, але не яскравою.

– Цифри: позначки через кожні 5 клітинок допомагають зрозуміти, де що на складі.

```
self.move_item_btn = ctk.CTkButton(
    self.buttons_frame,
    text="Доставити предмет",
    command=self.deliver_item,
    width=button_width,
    height=button_height,
    state="disabled"
)
self.move_item_btn.pack(padx=10, pady=10)

self.extra_btn = ctk.CTkButton(
    self.buttons_frame,
    text="Нова конфігурація",
    command=self.regenerate_configuration,
    width=button_width,
    height=button_height
)
self.extra_btn.pack(padx=10, pady=10)

self.right_frame = ctk.CTkFrame(self)
self.right_frame.grid(row=0, column=1, padx=10, pady=10, sticky="nsew")

self.grid_canvas = tk.Canvas(
    self.right_frame,
    bg="white",
    highlightthickness=0
)
self.grid_canvas.pack(fill="both", expand=True, padx=10, pady=10)

self.grid_canvas.bind("<Configure>", self.draw_grid)
self.grid_canvas.bind("<Button-1>", self.canvas_click)

self.selection_mode = None
```

Рис 3.5 Код методу draw_grid для відображення графічної сітки складу

3.1.5 Метод для відображення координатної сітки на складі у програмі

Цей код — частина методу draw_grid (рис 3.6), а саме його початок, який відповідає за створення сітки на екрані. Спочатку метод draw_grid прибирає все, що було намальовано раніше, за допомогою команди self.grid_canvas.delete("all"), щоб старе не заважало новому малюнку.

Потім програма дізнається, якого розміру має бути сітка і вікно. Вона бере ширину полотна через canvas_width = self.grid_canvas.winfo_width() і висоту через canvas_height = self.grid_canvas.winfo_height(), щоб знати, скільки місця є для малювання. Щоб сітка добре виглядала, обирають менший розмір між шириною і висотою через available_size = min(canvas_width, canvas_height) - 40, віднімаючи 40 пікселів, щоб залишити місце для цифр на краях.

Далі вираховують, якого розміру буде одна клітинка сітки, через self.cell_size = available_size / self.grid_size, де сітка має розмір 50 на 50 клітинок, щоб усе підходило під розмір вікна. Щоб сітка була посередині, вираховують зміщення через offset_x = (canvas_width - self.cell_size * self.grid_size) / 2 для осі x і offset_y = (canvas_height - self.cell_size * self.grid_size) / 2 для осі y, враховуючи розміри вікна і клітинок.

Цикл for i in range(self.grid_size + 1) малює сітку з ліній. Для вертикальних ліній місце визначають через x = offset_x + i * self.cell_size, і вони малюються сірим кольором (#CCCCCC) через self.grid_canvas.create_line від верху до низу сітки. Для горизонтальних ліній місце вираховують через y = offset_y + i * self.cell_size, і вони теж малюються сірим кольором від лівого краю до правого.

Інший цикл for i in range(0, self.grid_size + 1, 5) ставить цифри через кожні 5 клітинок. По осі x через self.grid_canvas.create_text(x, offset_y + self.grid_size * self.cell_size + 15, text=str(i), fill="black") додають цифри (0, 5, 10, ..., 50) під сіткою, відступивши на 15 пікселів вниз, а по осі y через self.grid_canvas.create_text(offset_x - 15, y, text=str(i), fill="black") — зліва від сітки, відступивши на 15 пікселів вліво.

Ця частина методу draw_grid робить сітку на екрані, яка потрібна, щоб показати склад. Вона підлаштовується під розмір вікна, вираховуючи розмір клітинок і зміщення, щоб сітка була посередині. Лінії сітки і цифри через кожні 5 клітинок допомагають зрозуміти, де що на складі, наприклад, де стоять робот, стелажі, предмети чи точки доставки. Це перший крок малювання, а потім додаються інші елементи, як стелажі чи предмети.

Додаткові деталі:

- Очищення: команда self.grid_canvas.delete("all") прибирає все старе перед новим малюванням.
- Розмір клітинок: available_size допомагає зробити сітку такою, щоб вона пасувала до вікна.
- Цифри: позначки через кожні 5 клітинок показують, де що на складі.

```
def draw_grid(self, event=None):
    self.grid_canvas.delete("all")

    canvas_width = self.grid_canvas.winfo_width()
    canvas_height = self.grid_canvas.winfo_height()

    available_size = min(canvas_width, canvas_height) - 40
    self.cell_size = available_size / self.grid_size

    offset_x = (canvas_width - self.cell_size * self.grid_size) / 2
    offset_y = (canvas_height - self.cell_size * self.grid_size) / 2
```

Рис 3.6 Код частини методу draw_grid для створення координатної сітки та отримання розмірів вікна

Фрагмент методу, який відображає координатну сітку та вільні клітинки для пересування робота на складі

У цій частині методу draw_grid (рис 3.7) програма малює сітку складу, показує вільні місця для доставки, стелажі та шлях, яким рухався робот. Спочатку цикл for i in range(self.grid_size + 1) створює сітку з ліній: для вертикальних ліній вираховують місце через x = offset_x + i * self.cell_size і малюють їх сірим кольором (#CCCCCC) за допомогою self.grid_canvas.create_line від верхнього краю до нижнього, а для горизонтальних ліній визначають місце через y = offset_y + i * self.cell_size і також малюють сірим від лівого краю до правого.

Другий цикл for i in range(0, self.grid_size + 1, 5) ставить цифри через кожні 5 клітинок, щоб було легше орієнтуватися: по осі x цифри (0, 5, 10, ..., 50) додають знизу сітки, а по осі y — зліва. Потім запускається метод self.highlight_empty_delivery_points(offset_x, offset_y), який перевіряє, які точки зі списку self.delivery_points вільні, тобто не зайняті предметами, роботом чи стелажами, і позначає їх сірим кольором (#C0C0C0), щоб показати, куди можна щось доставити.

Щоб намалювати стелажі, цикл for shelf in self.shelves бере кожен стелаж зі списку self.shelves. Його координати визначають через x, y = shelf, а місце на екрані вираховують через shelf_x = offset_x + x * self.cell_size і shelf_y = offset_y + y * self.cell_size. Стелаж показують як коричневий квадрат із кольором #8B4513 і рамкою #654321, який займає одну клітинку.

Щоб показати, куди рухався робот, цикл for cell in self.path_cells бере кожну клітинку зі списку self.path_cells, яка показує його шлях. Координати клітинки визначають через x, y = cell, а місце на екрані вираховують через cell_x = offset_x + x * self.cell_size і cell_y = offset_y + y * self.cell_size. Клітинку малюють як напівпрозорий жовтий квадрат із кольором #FFFF00 і параметром stipple="gray50", щоб було видно, де робот пройшов.

Ця частина методу draw_grid малює сітку складу розміром 50x50 клітинок і додає цифри для орієнтації. Вона також показує вільні місця для доставки, стелажі, де лежать предмети, і шлях робота у вигляді прозорих жовтих клітинок, щоб було зрозуміло, що відбувається на складі.

Додаткові деталі:

- Лінії сітки: сірий колір #CCCCCC робить їх помітними, але не надто яскравими.
- Місця доставки: сірий колір #C0C0C0 відрізняє їх від стелажів, щоб легше знайти.
- Шлях робота: параметр stipple="gray50" робить жовті клітинки трохи прозорими.

```
for i in range(self.grid_size + 1):
    x = offset_x + i * self.cell_size
    self.grid_canvas.create_line(
        x, offset_y,
        x, offset_y + self.grid_size * self.cell_size,
        fill="#CCCCCC",
    )

    y = offset_y + i * self.cell_size
    self.grid_canvas.create_line(
        offset_x, y,
        offset_x + self.grid_size * self.cell_size, y,
        fill="#CCCCCC"
    )

for i in range(0, self.grid_size + 1, 5):
    x = offset_x + i * self.cell_size
    self.grid_canvas.create_text(
        x, offset_y + self.grid_size * self.cell_size + 15,
        text=str(i),
        fill="black"
    )

    y = offset_y + i * self.cell_size
    self.grid_canvas.create_text(
        offset_x - 15, y,
        text=str(i),
        fill="black"
    )
```

Рис 3.7 Код фрагменту методу draw_grid для відображення координатної сітки, вільних клітин, стелажів і шляху робота

Відображення стелажів та сліду робота під час руху на складі

У цьому шматочку методу draw_grid (рис 3.8) програма малює предмети, самого робота і предмет, який він тримає, щоб було видно, що відбувається на складі. Спочатку код переглядає всі предмети зі списку self.items і перевіряє через умову if self.held_item != item_id, чи предмет не в руках робота, щоб не малювати те, що вже взято. Для кожного предмета, який ще лежить на стелажі, беруться його координати x і y зі списку item_data["pos"], а потім вираховують, де саме на екрані його показати, враховуючи зміщення offset_x і offset_y та розмір клітинки self.cell_size. Предмет малюють як квадрат із кольором, який заданий у item_data["color"], але роблять його трохи меншим за клітинку через відступи (margin = self.cell_size * 0.2), щоб він виглядав акуратно, і обводять чорною лінією товщиною 1 піксель, щоб було чітко видно.

Далі код малює робота: визначають його місце через robot_x і robot_y, беручи координати з self.robot_pos і враховуючи зміщення та розмір клітинки. Робота показують як червоний кружечок із кольором #FF0000 і чорною рамкою, який займає одну клітинку на сітці. Якщо робот тримає предмет, тобто self.held_item не порожній, то беруть колір цього предмета через held_color = self.items[self.held_item]["color"], і в середині робота малюють маленький кружечок, який займає чверть клітинки (center_x ± self.cell_size / 4, center_y ± self.cell_size / 4), із кольором предмета і чорною рамкою, щоб було видно, що він щось несе. Наприкінці програма запам’ятовує зміщення через self.grid_offset_x і self.grid_offset_y, щоб потім правильно обробляти кліки мишею.

Цей шматок коду потрібен, щоб показати предмети, робота і те, що він тримає, щоб людина могла бачити, що відбувається на складі і як робот із предметами працює. Це допомагає зрозуміти, як усе влаштовано в програмі. Візуалізація предметів і робота на складі дозволяє оператору швидко оцінити стан моделі, наприклад, визначити, які предмети ще доступні для доставки, і де саме перебуває робот у даний момент.

Метод draw_grid також забезпечує чіткість відображення завдяки контрастним кольорам: червоний робот (#FF0000) легко відрізнити від бірюзових предметів (#08e8de), а чорні рамки додають контурів, що полегшує сприйняття. Відступи для предметів (margin) не лише роблять їх вигляд естетичнішим, але й уникають накладання на межі сітки, що могло б ускладнити розпізнавання елементів. Розташування маленького кружечка в центрі робота при відображенні предмета в руках дозволяє оператору одразу зрозуміти, що робот виконує завдання, без необхідності звертатися до додаткових даних.

Крім того, збереження зміщень через self.grid_offset_x і self.grid_offset_y забезпечує точну обробку кліків мишею, що є важливим для інтерактивності інтерфейсу. Це дозволяє користувачу легко вибирати предмети чи точки на складі, що підвищує ефективність роботи з програмою. Такий підхід до візуалізації робить модель складу інтуїтивно зрозумілою і зручною для оператора, сприяючи швидкому прийняттю рішень. Крім того, чітке відображення стану

робота допомагає оператору краще координувати дії, особливо в умовах складної конфігурації складу. Візуалізація також підтримує інтерактивність, дозволяючи оператору швидко реагувати на зміни в реальному часі.

Додаткові деталі:

- Предмети: відступи (margin) роблять їх меншими, щоб вони гарно виглядали, а колір, наприклад #08e8de, задається ще в методі initialize_positions.
- Робот: червоний кружечок (#FF0000) чітко показує, де він стоїть, і займає рівно одну клітинку.
- Що в руках: маленький кружечок у центрі робота, який у 4 рази менший за клітинку, показує, що він тримає предмет.

```
for shelf in self.shelves:
    x, y = shelf
    shelf_x = offset_x + x * self.cell_size
    shelf_y = offset_y + y * self.cell_size
    self.grid_canvas.create_rectangle(
        shelf_x, shelf_y,
        shelf_x + self.cell_size, shelf_y + self.cell_size,
        fill="#8B4513", outline="#654321"
    )

for cell in self.path_cells:
    x, y = cell
    cell_x = offset_x + x * self.cell_size
    cell_y = offset_y + y * self.cell_size
    self.grid_canvas.create_rectangle(
        cell_x, cell_y,
        cell_x + self.cell_size, cell_y + self.cell_size,
        fill="#FFFF00", outline="", stipple="gray50"
    )
```

Рис. 3.8 Код фрагменту методу draw_grid для відображення предметів, робота та предмета в руках робота

Відображення товарів на стелажах для моделювання складу

Цей шматок методу draw_grid у програмі займається тим, щоб намалювати предмети, які лежать на стелажах складу (рис 3.9). Код бере всі предмети зі

списку self.items і перевіряє кожен із них через умову if self.held_item == item_id, щоб не малювати той предмет, який уже в руках робота, адже його покажуть окремо, разом із роботом. Для всіх інших предметів, які ще на стелажах, із даних item_data беруть їхні координати x і y через item_data["pos"], а потім вираховують, де саме на екрані їх показати: item_x і item_y визначають, додаючи зміщення offset_x і offset_y та враховуючи розмір клітинки self.cell_size. Колір предмета беруть із item_data["color"], і зазвичай це бірюзовий колір (#08e8de), який задають, коли предмети створюють. Щоб предмет не займав усю клітинку, роблять відступи, вираховуючи margin як 20% від розміру клітинки (self.cell_size * 0.2), і малюють предмет як квадрат із цими відступами: від item_x + margin до item_x + self.cell_size - margin по осі x і так само по осі y, із потрібним кольором і чорною рамкою товщиною 1 піксель, щоб було чітко видно.

Цей код потрібен, щоб показати предмети на стелажах, щоб людина могла бачити, де вони лежать на складі. Це важливо, коли потрібно вибрати предмет для доставки або просто зрозуміти, що відбувається в моделі складу. Завдяки цьому оператор може швидко оцінити, які предмети доступні для роботи, і планувати наступні дії, не звертаючись до додаткових джерел даних.

Візуалізація предметів на стелажах допомагає створити інтуїтивно зрозумілий інтерфейс, що є ключовим для вашого асистента. Чітке відображення предметів із відступами та контрастним кольором полегшує їх розпізнавання, а умовна перевірка self.held_item уникає дублювання зображень, що підвищує зручність використання програми.

Додаткові деталі:

- Відступи: завдяки margin предмети виглядають трохи меншими за клітинку, що робить їх акуратнішими і не дає їм зливатися з лініями сітки.
- Колір: усі предмети мають бірюзовий колір #08e8de, але код дозволяє змінити колір через item_data["color"], якщо захочеться.
- Умова: перевірка self.held_item не дає намалювати предмет, який уже взяв робот, щоб він не з’являвся одночасно і на стелажі, і в робота.

```
for item_id, item_data in self.items.items():
    if self.held_item == item_id:
        continue

    x, y = item_data["pos"]
    item_x = offset_x + x * self.cell_size
    item_y = offset_y + y * self.cell_size

    fill_color = item_data["color"]

    margin = self.cell_size * 0.2
    self.grid_canvas.create_rectangle(
        item_x + margin, item_y + margin,
        item_x + self.cell_size - margin, item_y + self.cell_size - margin,
        fill=fill_color, outline="black", width=1
    )
```

Рис 3.9: Код фрагменту методу draw_grid для відображення предметів на стелажах

Відображення робота та його стану з узятим товаром на складі

Цей код належить до методу draw_grid і відповідає за те, щоб намалювати робота та предмет, який він тримає, на екрані (рис. 3.10).

Спочатку програма визначає, де саме на екрані буде робот. Для цього вона бере його координати з self.robot_pos, множить їх на розмір клітинки self.cell_size і додає зміщення offset_x та offset_y, щоб усе правильно розташувалося на сітці. Так отримують точки robot_x і robot_y. Робота малюють як червоний кружечок із кольором #FF0000 і чорною рамкою, який займає одну клітинку — від robot_x, robot_y до robot_x + self.cell_size, robot_y + self.cell_size. Це допомагає легко побачити, де робот стоїть на складі.

Потім код перевіряє, чи є в робота предмет у руках, через умову if self.held_item. Якщо self.held_item не порожній, значить, робот щось тримає. Тоді програма бере колір цього предмета через held_color = self.items[self.held_item]["color"], і зазвичай це бірюзовий колір (#08e8de). Далі вираховують середину робота через center_x і center_y, які є серединою клітинки (robot_x + self.cell_size / 2 і robot_y + self.cell_size / 2). У цій середині малюють маленький кружечок, який займає чверть клітинки (center_x ± self.cell_size / 4, center_y ± self.cell_size / 4), із кольором предмета і чорною рамкою, щоб було видно, що робот тримає цей предмет.

Наостанок програма запам’ятовує зміщення через self.grid_offset_x і self.grid_offset_y, щоб потім правильно обробляти дії, наприклад, коли користувач клікає мишкою.

Цей код потрібен, щоб показати робота і предмет у його руках, щоб людина могла бачити, де робот і що він робить на складі. Це важливо для того, щоб керувати доставкою предметів. Візуалізація робота та предмета в його руках дозволяє оператору швидко зрозуміти, чи виконує робот завдання, і чи готовий він до наступної дії, наприклад, до доставки предмета до точки призначення.

Метод draw_grid у цьому фрагменті забезпечує чітке відображення завдяки контрастним кольорам: червоний робот (#FF0000) легко відрізнити від бірюзового предмета (#08e8de), що тримається в руках. Чорна рамка додає контурів, що полегшує сприйняття елементів на сітці. Розмір маленького кружечка (чверть клітинки) для предмета в руках робота підібраний так, щоб не перекривати основний вигляд робота, але водночас чітко показувати, що він щось несе.

Такий підхід до відображення дозволяє оператору ефективно взаємодіяти з системою, адже він одразу бачить стан робота без необхідності звертатися до додаткових даних у логах чи інших частинах інтерфейсу. Збереження зміщень через self.grid_offset_x і self.grid_offset_y також сприяє точній взаємодії, адже кліки мишею коректно обробляються, що є важливим для вибору предметів чи точок на складі.

Додаткові деталі:

- Робот: червоний кружечок (#FF0000) займає цілу клітинку і добре виділяється на тлі стелажів та предметів.
- Предмет: маленький кружечок у центрі робота, який у 4 рази менший за клітинку, показує, що предмет узятий.
- Зміщення: збереження offset_x і offset_y у self.grid_offset_x і self.grid_offset_y допомагає точно визначати, куди клікнули мишкою.

```
robot_x = offset_x + self.robot_pos[0] * self.cell_size
robot_y = offset_y + self.robot_pos[1] * self.cell_size
self.grid_canvas.create_oval(
    robot_x, robot_y,
    robot_x + self.cell_size, robot_y + self.cell_size,
    fill="#FF0000", outline="black"
)

if self.held_item:
    held_color = self.items[self.held_item]["color"]

    center_x = robot_x + self.cell_size / 2
    center_y = robot_y + self.cell_size / 2
    self.grid_canvas.create_oval(
        center_x - self.cell_size / 4, center_y - self.cell_size / 4,
        center_x + self.cell_size / 4, center_y + self.cell_size / 4,
        fill=held_color, outline="black"
    )
```

Рис. 3.10 Код фрагменту методу draw_grid для відображення робота та предмета в його руках

3.1.6 Відображення вільних стелажів для доставки товарів на складі

Метод highlight_empty_delivery_points у цьому коді відповідає за те, щоб показати вільні точки доставки, куди робот може принести предмети (рис. 3.11).

Спочатку програма створює спеціальний список occupied_positions, куди записує всі зайняті місця. Для цього вона переглядає всі предмети зі списку self.items через цикл for item_id, item_data in self.items.items() і додає їхні координати item_data["pos"] у цей список за допомогою occupied_positions.add(tuple(item_data["pos"])). Потім до цього списку додають місце, де стоїть робот, через tuple(self.robot_pos), бо він теж займає одну клітинку.

Далі програма бере всі точки доставки зі списку self.delivery_points і перевіряє кожну через цикл for delivery_point in self.delivery_points. Вона дивиться, чи точка вільна, за допомогою умови if tuple(delivery_point) not in occupied_positions and delivery_point not in self.shelves. Ця умова перевіряє, чи на цьому місці немає предмета і чи це не стелаж.

Якщо точка вільна, то з неї беруть координати x і y, а потім вираховують, де вона буде на екрані, через point_x = offset_x + x * self.cell_size і point_y = offset_y + y * self.cell_size, враховуючи зміщення offset_x, offset_y і розмір клітинки self.cell_size. Таку точку малюють як сірий квадрат із кольором #C0C0C0 і темнішою рамкою #808080 товщиною 1 піксель. Цей квадрат займає одну клітинку від point_x, point_y до point_x + self.cell_size, point_y + self.cell_size, показуючи, куди робот може щось принести.

Цей метод потрібен, щоб людина могла бачити, куди на складі можна покласти предмети, що допомагає планувати роботу робота. Завдяки цьому оператор легко визначає доступні місця для доставки, що полегшує координацію дій у моделі складу.

Візуалізація вільних точок доставки з контрастним сірим кольором (#C0C0C0) і темною рамкою (#808080) дозволяє чітко відрізнити їх від стелажів і робота, що підвищує зручність роботи з програмою. Точне вирахування координат із урахуванням зміщення забезпечує коректне відображення на сітці, яка адаптується до розміру вікна.

Додаткові деталі:

- Колір: сірий #C0C0C0 і рамка #808080 роблять точки помітними на тлі коричневих стелажів чи червоного робота.
- Перевірка: список occupied_positions швидко показує, чи місце зайняте, а перевірка delivery_point not in self.shelves не дає ставити точки на стелажах.
- Розташування: point_x і point_y вираховують так, щоб точки правильно виглядали на сітці, яка змінюється залежно від розміру вікна.

```
def highlight_empty_delivery_points(self, offset_x, offset_y):
    occupied_positions = set()
    for item_id, item_data in self.items.items():
        occupied_positions.add(tuple(item_data["pos"]))

    occupied_positions.add(tuple(self.robot_pos))

    for delivery_point in self.delivery_points:
        if (tuple(delivery_point) not in occupied_positions and
            delivery_point not in self.shelves):

            x, y = delivery_point
            point_x = offset_x + x * self.cell_size
            point_y = offset_y + y * self.cell_size

            self.grid_canvas.create_rectangle(
                point_x, point_y,
                point_x + self.cell_size, point_y + self.cell_size,
                fill="#C0C0C0", outline="#808080", width=1
            )
```

Рис. 3.11 Код методу highlight_empty_delivery_points для відображення вільних точок доставки

3.1.7 Створення кнопок для керування роботом у графічному інтерфейсі

3.1.7.1 Методи для роботи кнопки “Доставка предмета”

Метод start_item_selection у програмі запускається, коли користувач натискає кнопку "Доставка предмета" (рис 3.12). Його завдання — підготувати все для вибору предмета, який потрібно доставити.

Спочатку метод перевіряє, чи робот уже тримає якийсь предмет, через умову if self.held_item. Якщо предмет уже є в руках, програма показує повідомлення через messagebox.showwarning із текстом "Робот вже тримає предмет! Спочатку доставте його.", щоб людина знала, що спершу треба завершити поточну доставку. Далі перевіряють, чи є взагалі предмети на складі, через умову if not self.items. Якщо предметів немає, з’являється повідомлення "Немає предметів для доставки!", щоб користувач не намагався зробити те, що неможливо.

Якщо все гаразд і перевірки пройдені, метод записує в лог через self.add_log повідомлення "Виберіть предмет для доставки (клікніть на стелаж з предметом)" і

вмикає режим вибору через self.selection_mode = "select_item". Це дозволяє програмі чекати, коли користувач клікне мишею, щоб обрати предмет.

Інший метод, canvas_click, відповідає за те, що відбувається, коли людина клікає мишею на сітці складу в режимі вибору предмета. Спочатку він дивиться, чи увімкнений режим вибору, через умову if not self.selection_mode. Якщо режим не активний, метод просто закінчується. А якщо режим увімкнений, програма вираховує, на яку клітинку клікнули: координати x і y визначають через x = int((event.x - self.grid_offset_x) / self.cell_size) і y = int((event.y - self.grid_offset_y) / self.cell_size), враховуючи зміщення сітки grid_offset_x і grid_offset_y та розмір клітинки self.cell_size.

Потім перевіряють, чи клік був у межах сітки, через умову 0 <= x < self.grid_size and 0 <= y < self.grid_size. Якщо режим вибору — це "select_item", викликається метод self.execute_item_selection(x, y), який обробляє вибір предмета за цими координатами. Після цього режим вибору вимикається через self.selection_mode = None, щоб завершити цей етап.

Ці методи допомагають людині вибрати предмет для доставки, просто клікнувши мишкою на потрібному місці, і роблять початок доставки зручним. Вони також перевіряють, чи все можливо, і показують повідомлення, якщо щось пішло не так.

Додаткові деталі:

- Повідомлення: messagebox.showwarning допомагає людині зрозуміти, що пішло не так, якщо дія неможлива.
- Режим: змінна self.selection_mode показує програмі, що саме потрібно робити, і не дає зробити зайвих дій.
- Координати: зміщення сітки та розмір клітинки допомагають точно визначити, куди клікнули.

```
def start_item_selection(self):
    if self.held_item:
        messagebox.showwarning("Помилка", "Робот вже тримає предмет! Спочатку доставте його.")
        return

    if not self.items:
        messagebox.showwarning("Помилка", "Немає предметів для доставки!")
        return

    self.add_log("Виберіть предмет для доставки (клікніть на стелаж з предметом)")
    self.selection_mode = "select_item"
```

Рис. 3.12 Код методів start_item_selection і canvas_click для роботи кнопки "Доставка предмета"

Метод execute_item_selection у програмі бере координати x і y, куди клікнула людина, і відповідає за те, щоб вибрати предмет і перемістити робота до нього (рис. 3.13).

Спочатку створюється змінна selected_item і ставиться в None, тобто порожня. Потім програма переглядає всі предмети зі списку self.items через цикл for item_id, item_data in self.items.items(). Вона перевіряє, чи є предмет на координатах x, y, порівнюючи їх із item_data["pos"]. Якщо предмет є, selected_item стає item_id, тобто позначає, який предмет вибрали.

Якщо предмета на цьому місці немає і selected_item залишається порожньою, програма показує повідомлення через messagebox.showwarning із текстом "На цій клітині немає предмета!", щоб людина знала, що там нічого немає. Далі задають напрямки directions — це вправо, вниз, вліво, вгору, щоб перевірити клітинки навколо предмета, і створюють змінну adjacent_pos, яка спочатку порожня.

Через цикл for dx, dy in directions програма вираховує координати сусідньої клітинки через check_pos як x + dx, y + dy. Вона дивиться, чи ця клітинка в межах сітки через умову 0 <= check_pos[0] < self.grid_size and 0 <= check_pos[1] < self.grid_size, і чи це не стелаж через check_pos not in self.shelves. Якщо клітинка підходить, adjacent_pos стає check_pos, і цикл зупиняється.

Якщо жодна сусідня клітинка не підійшла і adjacent_pos залишається порожньою, програма пише в лог через self.add_log, що до предмета не можна підійти, і метод закінчується. Якщо ж клітинка є, викликається метод self.find_path(self.robot_pos, adjacent_pos), щоб знайти шлях від робота до цієї

клітинки. Якщо шляху немає, з’являється повідомлення "Неможливо знайти шлях до предмета!".

Якщо шлях є, програма записує час початку через start_time = time.time(). Потім через цикл for step in path[1:] вона переміщує робота по кожному кроку: змінює його місце через self.robot_pos, додає крок до списку self.path_cells, щоб показати слід, перемальовує екран через self.draw_grid(), оновлює вигляд через self.update() і робить паузу через time.sleep(self.animation_speed), щоб рух виглядав плавно.

Коли робот дійде, він бере предмет через self.held_item = selected_item, вмикає кнопку доставки через self.move_item_btn.configure(state="normal"), вимикає кнопку вибору через self.select_item_btn.configure(state="disabled"), записує час кінця через end_time = time.time(), вираховує, скільки часу пішло на рух через duration, і додає в лог записи про те, що рух закінчено і предмет узятий. Наостанок екран ще раз оновлюється.

Цей метод допомагає вибрати предмет, перемістити робота до нього і взяти предмет, роблячи все зручним для людини через плавний рух і повідомлення. Анімація руху робота дозволяє оператору бачити кожен крок переміщення, що полегшує розуміння процесу. Використання алгоритму BFS у методі self.find_path гарантує, що робот завжди обирає найкоротший шлях до предмета, що підвищує ефективність його роботи.

Метод також забезпечує зворотний зв’язок через логування, що дозволяє оператору бачити деталі руху, наприклад, час, який знадобився роботу для виконання завдання, і координати, куди він перемістився. Це допомагає оператору контролювати процес і швидко реагувати, якщо виникають проблеми, наприклад, якщо шлях до предмета заблокований. Оновлення стану кнопок після виконання дії (ввімкнення кнопки доставки і вимкнення кнопки вибору) забезпечує чіткий перехід між етапами роботи, що робить інтерфейс інтуїтивно зрозумілим.

Крім того, затримка self.animation_speed у 0.03 секунди між кроками руху робота створює візуально комфортний ефект, який не перевантажує інтерфейс і дає оператору можливість спостерігати за процесом у реальному часі. Це

особливо важливо для оцінки роботи моделі та подальшого аналізу її ефективності. Такий підхід до реалізації методу робить систему зручною для користувача і підвищує її практичну цінність.

Додаткові деталі:

- Напрямки: перевірка клітинок навколо (вправо, вниз, вліво, вгору) допомагає знайти місце для робота.
- Рух: пауза self.animation_speed (0.03 секунди) робить рух робота плавним.
- Лог: записи про час руху допомагають знати, що відбувалося.

```
def execute_item_selection(self, x, y):
    selected_item = None
    for item_id, item_data in self.items.items():
        if item_data["pos"][0] == x and item_data["pos"][1] == y:
            selected_item = item_id
            break

    if not selected_item:
        messagebox.showwarning("Помилка", "На цій клітині немає предмета!")
        return

    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
    adjacent_pos = None

    for dx, dy in directions:
        check_pos = [x + dx, y + dy]
        if (0 <= check_pos[0] < self.grid_size and
            0 <= check_pos[1] < self.grid_size and
            check_pos not in self.shelves):
            adjacent_pos = check_pos
            break

    if not adjacent_pos:
        self.add_log(f"Неможливо підійти до предмета на [{x}, {y}]")
        return
```

Рис. 3.13 Код методу execute_item_selection для роботи кнопки "Доставка предмета"

Метод deliver_item у програмі відповідає за те, щоб доставити предмет, який тримає робот, до потрібного місця на складі (рис. 3.14).

Спочатку перевіряють, чи є в робота предмет у руках, через умову if not self.held_item. Якщо предмета немає, програма показує повідомлення через messagebox.showwarning із текстом "Робот не тримає предмет!", щоб людина зрозуміла, що щось пішло не так.

Далі створюють змінні best_target, best_distance (яка спочатку має дуже велике значення) і best_path, щоб знайти найкраще місце для доставки. Програма переглядає всі точки доставки зі списку self.delivery_points через цикл for target in

self.delivery_points. Для кожної точки перевіряють, чи не є вона стелажем, через умову if target in self.shelves, і якщо це стелаж, точку пропускають.

Потім створюють змінну position_occupied і ставлять її в False. Через цикл for item_id, item_data in self.items.items() перевіряють, чи точка target уже зайнята іншим предметом, порівнюючи item_data["pos"] == target, але не враховують предмет, який тримає робот, через item_id != self.held_item. Якщо точка зайнята, position_occupied стає True, і цю точку пропускають.

Якщо точка вільна, програма викликає метод self.find_path(self.robot_pos, target), щоб знайти шлях від робота до цієї точки. Якщо шлях є і він коротший, ніж попередній best_distance, то оновлюють best_distance, best_target і best_path, щоб вибрати найкоротший маршрут. Якщо після всіх перевірок best_target так і не знайшли, показують повідомлення "Немає доступних місць для доставки!", і метод закінчується.

Коли точка і шлях знайдені, програма записує час початку через start_time = time.time(). Потім через цикл for step in best_path[1:] вона переміщує робота по кожному кроку: змінює його місце через self.robot_pos, додає крок до списку self.path_cells, щоб показати, куди він ішов, перемальовує екран через self.draw_grid(), оновлює вигляд через self.update() і робить маленьку паузу через time.sleep(self.animation_speed), щоб рух виглядав плавно.

Після того, як робот дійшов, фіксують час кінця через end_time = time.time() і вираховують, скільки часу це зайняло, через duration. Якщо предмет усе ще в руках (self.held_item не порожній), його ставлять на місце робота через self.items[self.held_item]["pos"] = self.robot_pos.copy(), а потім прибирають із рук через self.held_item = None. Кнопку доставки вимикають через self.move_item_btn.configure(state="disabled"), а кнопку вибору предмета вмикають через self.select_item_btn.configure(state="normal"). У лог пишуть, що предмет доставлений, скільки часу це зайняло, і оновлюють екран.

Цей метод завершує доставку, переміщаючи робота до вільного місця, ставлячи предмет і готуючи програму до наступних дій.

Додаткові деталі:

- Шлях: метод self.find_path із BFS знаходить найкоротший маршрут, щоб робот рухався швидко.
- Рух: пауза self.animation_speed (0.03 секунди) робить рух плавним і зручним для ока.
- Лог: записи про час і кількість клітинок показують, наскільки швидко все відбулося.

```
path = self.find_path(self.robot_pos, adjacent_pos)
if not path:
    messagebox.showwarning("Помилка", "Неможливо знайти шлях до предмета!")
    return

start_time = time.time()
for step in path[1:]:
    self.robot_pos = step
    self.path_cells.append(step.copy())
    self.draw_grid()
    self.update()
    time.sleep(self.animation_speed)

self.held_item = selected_item
self.move_item_btn.configure(state="normal")
self.select_item_btn.configure(state="disabled")

end_time = time.time()
duration = end_time - start_time

self.add_log(f"Робот підійшов до предмета на [{x}, {y}]. Час: {duration:.2f} сек.")
self.add_log(f"Робот взяв предмет з стелажа")
self.draw_grid()
```

Рис. 3.14 Код методу deliver_item для роботи кнопки “Доставити предмет”

Функція deliver_item у програмі відповідає за доставку предмета, який тримає робот, до потрібного місця на складі (рис. 3.15).

Спочатку вона перевіряє, чи є у робота предмет у руках, через умову if not self.held_item. Якщо предмета немає, з’являється повідомлення через messagebox.showwarning із текстом "Робот не тримає предмет!", щоб людина зрозуміла, що щось не так.

Далі програма створює змінні best_target, best_distance (яка спочатку має дуже велике значення, ніби нескінченність) і best_path, щоб знайти найкраще місце для доставки. Вона переглядає всі точки доставки зі списку self.delivery_points через цикл for target in self.delivery_points. Для кожної точки перевіряють, чи не збігається вона зі стелажем через умову if target in self.shelves, і якщо це стелаж, точку пропускають.

Потім створюють змінну position_occupied і ставлять її в False. Через цикл for item_id, item_data in self.items.items() програма дивиться, чи точка target уже зайнята іншим предметом, порівнюючи item_data["pos"] == target, але не враховує предмет, який у руках робота, через item_id != self.held_item. Якщо точка зайнята, position_occupied стає True, і цю точку пропускають.

Якщо точка вільна, викликається метод self.find_path(self.robot_pos, target), щоб знайти шлях від робота до неї. Якщо шлях є і він коротший, ніж попередній best_distance, то оновлюють best_distance, best_target і best_path, щоб вибрати найшвидший маршрут. Якщо після всіх перевірок best_target так і не знайшли, з’являється повідомлення "Немає доступних місць для доставки!", і на цьому функція зупиняється.

Коли місце і шлях знайдені, програма фіксує час початку через start_time = time.time(). Потім через цикл for step in best_path[1:] вона переміщує робота по кожному кроку: змінює його місце через self.robot_pos, додає крок до списку self.path_cells, щоб показати, куди він рухався, перемальовує екран через self.draw_grid(), оновлює вигляд через self.update() і робить паузу через time.sleep(self.animation_speed), щоб рух був плавним.

Коли робот дійде, фіксують час кінця через end_time = time.time() і вираховують, скільки часу це зайняло, через duration. Якщо предмет усе ще в руках (self.held_item не порожній), його ставлять на місце робота через self.items[self.held_item]["pos"] = self.robot_pos.copy(), а потім прибирають із рук через self.held_item = None. Кнопку доставки вимикають через self.move_item_btn.configure(state="disabled"), а кнопку вибору предмета вмикають через self.select_item_btn.configure(state="normal"). У лог додають записи про те, що предмет доставлений, і скільки часу це зайняло, а потім оновлюють екран.

Ця функція доставляє предмет до найближчого вільного місця, переміщає робота, показує це на екрані і дає людині інформацію про те, що сталося. Анімація руху робота забезпечує оператору можливість спостерігати за процесом у реальному часі, що допомагає краще контролювати дії робота. Використання

BFS у методі self.find_path гарантує, що маршрут завжди буде найкоротшим, що економить час і підвищує продуктивність роботи.

Логування подій у цій функції дозволяє оператору бачити не лише факт доставки, але й деталі операції, такі як координати точки доставки та тривалість руху. Це корисно для аналізу роботи моделі та виявлення можливих проблем, наприклад, якщо точки доставки часто зайняті. Оновлення стану кнопок після доставки (вимкнення кнопки доставки і ввімкнення кнопки вибору) робить інтерфейс логічним і зручним, адже оператор одразу розуміє, які дії доступні на наступному етапі.

Додаткові деталі:

- Шлях: метод self.find_path із BFS допомагає знайти найкоротший маршрут для робота.
- Рух: пауза self.animation_speed (0.03 секунди) робить рух плавним, щоб було приємно дивитися.
- Лог: записи про час і кількість клітинок дають зрозуміти, наскільки швидко все зроблено.

```
def deliver_item(self):
    if not self.held_item:
        messagebox.showwarning("Помилка", "Робот не тримає предмет!")
        return

    best_target = None
    best_distance = float('inf')
    best_path = None

    for target in self.delivery_points:
        if target in self.shelves:
            continue

        position_occupied = False
        for item_id, item_data in self.items.items():
            if item_data["pos"] == target and item_id != self.held_item:
                position_occupied = True
                break

        if position_occupied:
            continue

        path = self.find_path(self.robot_pos, target)
        if path and len(path) < best_distance:
            best_distance = len(path)
            best_target = target
            best_path = path
```

Рис. 3.15 Код методу deliver_item для роботи кнопки "Доставити предмет"

Робота кнопки “Доставити предмет” для завершення процесу доставки

Функція deliver_item у цьому коді відповідає за завершення доставки предмета (рис 3.16).

Коли робот дійшов до потрібного місця, програма записує час закінчення через end_time = time.time(). Потім вираховує, скільки часу зайняла доставка, через duration, віднімаючи від кінцевого часу початковий (start_time, який був записаний раніше).

Далі перевіряють, чи робот усе ще тримає предмет, через умову if self.held_item. Якщо предмет є, його ставлять на місце робота через self.items[self.held_item]["pos"] = self.robot_pos.copy(), тобто предмет залишається там, де стоїть робот, у точці доставки. Після цього програма прибирає предмет із рук робота через self.held_item = None, щоб показати, що він більше нічого не тримає.

Потім програма вимикає кнопку доставки через self.move_item_btn.configure(state="disabled"), щоб людина не могла натиснути її знову, поки не вибере новий предмет. А кнопку вибору предмета, навпаки, вмикають через self.select_item_btn.configure(state="normal"), щоб можна було почати нову доставку.

У лог додають запис через self.add_log, де пишуть, що предмет доставлений і стоїть на координатах self.robot_pos. Також додають ще один запис, де вказують, скільки секунд зайняла доставка через duration і скільки клітинок робот пройшов через len(best_path) - 1. Це допомагає зрозуміти, як швидко все відбулося.

Наостанок програма перемальовує екран через self.draw_grid(), щоб показати склад із новим розташуванням предмета.

Ця частина коду завершує доставку, ставить предмет на місце, оновлює кнопки і дає людині знати, що все зроблено. Завдяки цьому оператор може одразу бачити результат і планувати наступні дії, що підвищує ефективність роботи.

Візуалізація нового розташування предмета через self.draw_grid() забезпечує миттєве оновлення екрану, що полегшує контроль за станом складу. Логування

детальних даних, таких як координати та час, дозволяє аналізувати роботу моделі, що є важливим для подальшого вдосконалення системи.

Додаткові деталі:

- Стан: прибирання self.held_item і зміна кнопок допомагають правильно переходити між вибором і доставкою.
- Лог: записи про час і клітинки дають змогу перевірити, наскільки добре робот доставив предмет.
- Екран: виклик self.draw_grid() одразу показує, як усе виглядає після доставки.

```
if not best_target:
    messagebox.showwarning("Помилка", "Немає доступних місць для доставки!")
    return

start_time = time.time()

for step in best_path[1:]:
    self.robot_pos = step
    self.path_cells.append(step.copy())
    self.draw_grid()
    self.update()
    time.sleep(self.animation_speed)

end_time = time.time()
duration = end_time - start_time

if self.held_item:
    self.items[self.held_item]["pos"] = self.robot_pos.copy()
    self.held_item = None
    self.move_item_btn.configure(state="disabled")
    self.select_item_btn.configure(state="normal")
    self.add_log(f"Предмет доставлен та розміщений на {self.robot_pos}")
    self.add_log(f"Час доставки: {duration:.2f} сек. Пройдено {len(best_path) - 1} клітин.")
    self.draw_grid()
```

Рис. 3.16 Код частини методу deliver_item для завершення роботи кнопки "Доставити предмет"

Робота кнопки “Нова конфігурація” для оновлення складу

Функція regenerate_configuration робить нову розстановку на складі. Спочатку вона очищає список self.path_cells, щоб прибрати сліди, які показують, куди раніше рухався робот. Це потрібно, щоб екран виглядав чистим для нової розстановки (рис. 3.17).

Далі програма перевіряє, чи є у робота предмет у руках, через умову if self.held_item. Якщо предмет є, задають напрямки directions — це вправо, вниз,

вліво, вгору, і перемішують їх випадково через random.shuffle(directions), щоб вибрати будь-який напрямок. Потім через цикл for dx, dy in directions вираховують нове місце nx, ny, додаючи зміщення dx, dy до поточного місця робота self.robot_pos. Перевіряють, чи це місце в межах сітки через 0 <= nx < self.grid_size and 0 <= ny < self.grid_size і чи там немає стелажа через [nx, ny] not in self.shelves.

Якщо місце підходить, предмет ставлять туди через self.items[self.held_item]["pos"] = [nx, ny], записують у лог через self.add_log, що предмет поклали біля робота, прибирають предмет із рук через self.held_item = None, вимикають кнопку доставки через self.move_item_btn.configure(state="disabled"), вмикають кнопку вибору предмета через self.select_item_btn.configure(state="normal"), щоб можна було вибирати новий предмет, і зупиняють цикл.

Потім викликають функцію self.initialize_positions(), яка створює нову розстановку: додає нові стелажі, предмети і ставить робота на місце [45, 26]. У лог через self.add_log пишуть, що нова розстановка створена, скільки вийшло стелажів через len(self.shelves), скільки предметів через len(self.items) і де стоїть робот через self.robot_pos. Наостанок викликають self.draw_grid(), щоб намалювати склад із новим розташуванням стелажів, предметів і робота.

Ця функція дає змогу людині оновити склад, створивши нову розстановку, що допомагає перевіряти різні ситуації для роботи робота, і правильно завершує попередню дію з предметом перед новим стартом. Завдяки цьому оператор може легко тестувати різні сценарії роботи складу, не втрачаючи поточний прогрес.

Візуалізація нової розстановки через self.draw_grid() забезпечує миттєве оновлення екрану, що дозволяє оператору одразу бачити зміни. Логування деталей, таких як кількість стелажів і предметів, полегшує аналіз нової конфігурації. Крім того, випадкове розміщення предмета перед оновленням забезпечує різноманітність, що корисне для моделювання складних ситуацій.

Додаткові деталі:

- Випадковість: перемішування через random.shuffle(directions) робить вибір місця для предмета непередбачуваним.
- Оновлення: self.initialize_positions() повністю змінює склад, включно зі стелажами і предметами.
- Лог: записи про кількість стелажів, предметів і місце робота допомагають людині знати, що змінилося.

```
def regenerate_configuration(self):
    self.path_cells = []

    if self.held_item:
        directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
        random.shuffle(directions)

        for dx, dy in directions:
            nx, ny = self.robot_pos[0] + dx, self.robot_pos[1] + dy
            if 0 <= nx < self.grid_size and 0 <= ny < self.grid_size and [nx, ny] not in self.shelves:
                self.items[self.held_item]["pos"] = [nx, ny]
                self.add_log(f"Предмет розміщений поруч з роботом на [{nx}, {ny}]")
                self.held_item = None
                self.move_item_btn.configure(state="disabled")
                self.select_item_btn.configure(state="normal")
                break

    self.initialize_positions()

    self.add_log("Згенерована нова конфігурація.")
    self.add_log(f"Створено {len(self.shelves)} стелажів.")
    self.add_log(f"Створено {len(self.items)} предметів на стелажах.")
    self.add_log(f"Нова позиція робота: {self.robot_pos}")

    self.draw_grid()
```

Рис. 3.17: Код методу regenerate_configuration для роботи кнопки "Нова конфігурація"

3.1.8 Логування подій для відстеження дій робота на складі

Функція add_log у програмі потрібна, щоб записувати, що відбувається, і показувати це користувачу (рис 3.18). Вона бере текст повідомлення, який передають через параметр message, і додає його в спеціальне поле.

Спочатку програма бере поточний час через datetime.now().strftime("%H:%M:%S"), щоб знати, коли саме сталася подія. Це дає час у форматі "години:хвилини:секунди", наприклад, 18:37:00 для 06:37 PM, використовуючи бібліотеку datetime. Так людина бачить, коли саме щось сталося.

Далі створюють запис log_entry у вигляді "[timestamp] message\n". Тут у квадратних дужках пишуть час, потім додають текст повідомлення і ставлять символ нового рядка, щоб усе гарно виглядало в текстовому полі.

Цей запис додають у поле self.log_text через команду self.log_text.insert(tk.END, log_entry). Це поле — місце, куди пишуться всі події, і воно було зроблене раніше в функції create_layout. Команда tk.END означає, що запис додається в кінець поля.

Потім викликають self.log_text.see(tk.END), щоб поле автоматично прокрутилося вниз до останнього запису. Завдяки цьому людина одразу бачить найновіше повідомлення, навіть якщо подій уже багато, і не треба прокручувати вручну.

Ця функція допомагає записувати і показувати всі події з часом, щоб людина могла бачити, що робить робот і програма прямо зараз. Це потрібно, щоб аналізувати, як усе працює.

Додаткові деталі:

- Час: формат strftime("%H:%M:%S") показує час у 24-годинному вигляді, що зручно для записів.
- Прокручування: команда see(tk.END) робить так, що нові записи завжди видно.
- Універсальність: функція може записувати будь-які повідомлення, наприклад, про рух робота чи оновлення складу.

```
def add_log(self, message):
    timestamp = datetime.now().strftime("%H:%M:%S")
    log_entry = f"[{timestamp}] {message}\n"

    self.log_text.insert(tk.END, log_entry)

    self.log_text.see(tk.END)
```

Рис. 3.18 Код методу add_log для створення логів подій

3.1.9 Пошук шляху для руху робота на складі

Функція find_path бере дві точки: звідки робот починає рух (start) і куди він має дійти (end). Вона використовує метод BFS, тобто пошук у ширину, щоб

знайти найкоротший шлях (рис. 3.19). Спочатку створюють чергу queue, куди додають початкову точку start і шлях до неї, який спочатку містить тільки цю точку. Також створюють список visited, щоб відмічати, які клітинки робот уже перевірив, і додають туди start, щоб не повертатися назад[2].

Потім запускається цикл while queue, який працює, поки є що перевіряти. З черги беруть поточну точку pos і шлях path до неї. Якщо pos збігається з кінцевою точкою end, значить, шлях знайдений, і функція його повертає.

Якщо ні, то програма дивиться на сусідні клітинки — вправо, вниз, вліво, вгору, через список directions. Для кожної сусідньої клітинки вираховують нові координати next_pos, додаючи зміщення dx, dy до поточної точки pos. Перевіряють, чи нова клітинка в межах сітки через 0 <= next_pos[0] < self.grid_size and 0 <= next_pos[1] < self.grid_size, чи вона не зайнята стелажем через next_pos not in self.shelves, і чи її ще не відвідували через next_pos not in visited.

Якщо клітинка підходить, її додають у список visited, щоб не перевіряти знову, і в чергу queue додають цю клітинку разом із новим шляхом, який включає попередній шлях path плюс цю клітинку. Так програма продовжує шукати, поки не дійде до кінцевої точки або не перевірить усі можливі варіанти.

Якщо після всіх перевірок шлях не знайдений, функція повертає None, тобто показує, що дороги до потрібного місця немає.

Ця функція потрібна, щоб робот міг знайти найкоротший шлях до точки доставки або предмета, уникаючи стелажів, що допомагає йому рухатися швидко і без зайвих рухів.

Додаткові деталі:

- Пошук: BFS допомагає знайти найкоротший шлях, перевіряючи всі можливі напрямки.
- Перешкоди: перевірка next_pos not in self.shelves не дає роботу зайти на стелаж.
- Черга: використання queue дозволяє перевіряти клітинки по порядку, від ближчих до дальших.

```python
def find_path(self, start, end):
    if start == end:
        return [start]

    if [end[0], end[1]] in self.shelves:
        return []

    occupied_positions = set()
    for item_id, item_data in self.items.items():
        if item_id != self.held_item:
            occupied_positions.add(tuple(item_data["pos"]))

    queue = deque([[start]])
    visited = set([tuple(start)])

    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    while queue:
        path = queue.popleft()
        current = path[-1]

        if current[0] == end[0] and current[1] == end[1]:
            return path

        for dx, dy in directions:
            nx, ny = current[0] + dx, current[1] + dy
            new_pos = [nx, ny]

            if (0 <= nx < self.grid_size and 0 <= ny < self.grid_size and
                    tuple(new_pos) not in visited and
                    new_pos not in self.shelves and
                    tuple(new_pos) not in occupied_positions):
                new_path = path.copy()
                new_path.append(new_pos)
                queue.append(new_path)
                visited.add(tuple(new_pos))

    return []
```

Рис. 3.19 Код методу find_path для пошуку шляху робота на складі

3.2 Тестування і результати роботи

Процес тестування розробленої програми, яка моделює роботу робота на складі, а також аналіз отриманих результатів представлені далі. Програма створена для оптимізації взаємодії між оператором і промисловими роботами з використанням сучасних інтелектуальних технологій, зокрема обробки координатної сітки, алгоритмів пошуку шляху та логування подій. Тестування проводилося з метою перевірки коректності функціонування всіх основних компонентів програми, таких як створення логів подій, рух робота, доставка предметів і генерація нової конфігурації складу.

Для оцінки роботи програми було виконано кілька тестових сценаріїв, які охоплювали різні аспекти її функціональності. Перший сценарій передбачав запуск програми та перевірку ініціалізації складу, включаючи створення стелажів, розміщення предметів і встановлення початкової позиції робота. Під час тестування було зафіксовано, що програма коректно ініціалізує склад із випадковою кількістю предметів у діапазоні від 5 до 35, які розміщуються на

стелажах, згенерованих методом generate_fixed_shelves. Цей метод забезпечує створення стелажів із урахуванням проходів і зон доставки, що дозволяє роботу вільно пересуватися між ними. Початкова позиція робота завжди встановлюється на координати [45, 26], як визначено в методі initialize_positions, що забезпечує однаковий стартовий пункт для всіх сценаріїв. Перевірка показала, що ініціалізація відбувається без помилок, а всі елементи коректно відображаються на канвасі.

Другий сценарій стосувався тестування кнопки "Доставка предмета". Користувач вибирав предмет, клікаючи на стелаж, після чого робот переміщувався до нього, брав предмет і доставляв до найближчої точки доставки. Метод start_item_selection перевіряв, чи робот не тримає інший предмет і чи є предмети на складі, а метод execute_item_selection обробляв вибір предмета та переміщення робота. Під час тестування було перевірено плавність анімації руху робота з затримкою 0.03 секунди, встановленою через self.animation_speed, а слід у вигляді напівпрозорих жовтих клітин відображався чітко завдяки циклу в методі draw_grid. Перевірка підтвердила коректність руху та активацію кнопки "Доставити предмет" після взяття предмета.

Третій сценарій передбачав тестування кнопки "Доставити предмет", яка викликає метод deliver_item. Цей метод шукає вільну точку доставки і переміщує робота до неї, оновлюючи позицію предмета та стан кнопок. Тестування показало, що доставка займає 2–5 секунд залежно від відстані, а алгоритм у методі find_path забезпечує точне визначення шляху. Після доставки лог додає записи з деталями, такими як координати та час.

Четвертий сценарій включав тестування кнопки "Нова конфігурація", яка викликає метод regenerate_configuration. Цей метод скидає попередній стан і генерує нову конфігурацію, розміщуючи предмети в найближчих вільних клітинах, якщо робот їх тримає. Тестування підтвердило коректність оновлення складу, включно з позицією робота [45, 26], без затримок на канвасі.

Важливим аспектом тестування було логування подій через метод add_log (рис 3.20), яке додає записи з мітками часу у форматі "години:хвилини:секунди"

через datetime.now().strftime("%H:%M:%S"). Логування коректно відображало ініціалізацію, вибір предметів і доставку, з автоматичною прокруткою до останнього запису через self.log_text.see(tk.END), що полегшувало аналіз.

Результати тестування засвідчили стабільну роботу програми в усіх сценаріях. Ініціалізація, переміщення робота та оновлення конфігурації виконуються без помилок. Обмеження виникають при великій кількості предметів (понад 30), коли точки доставки можуть бути зайняті, викликаючи попередження. Логування забезпечує детальний зворотний зв'язок для аналізу дій.

Додаткові деталі:

- Анімація: затримка 0.03 секунди забезпечує плавний рух.
- Обмеження: потрібні додаткові точки доставки при великій кількості предметів.
- Стабільність: програма витримує часті оновлення без затримок.

Таким чином, тестування підтвердило коректність програми, демонструючи її потенціал для моделювання роботи робота на складі з можливістю подальшого вдосконалення.

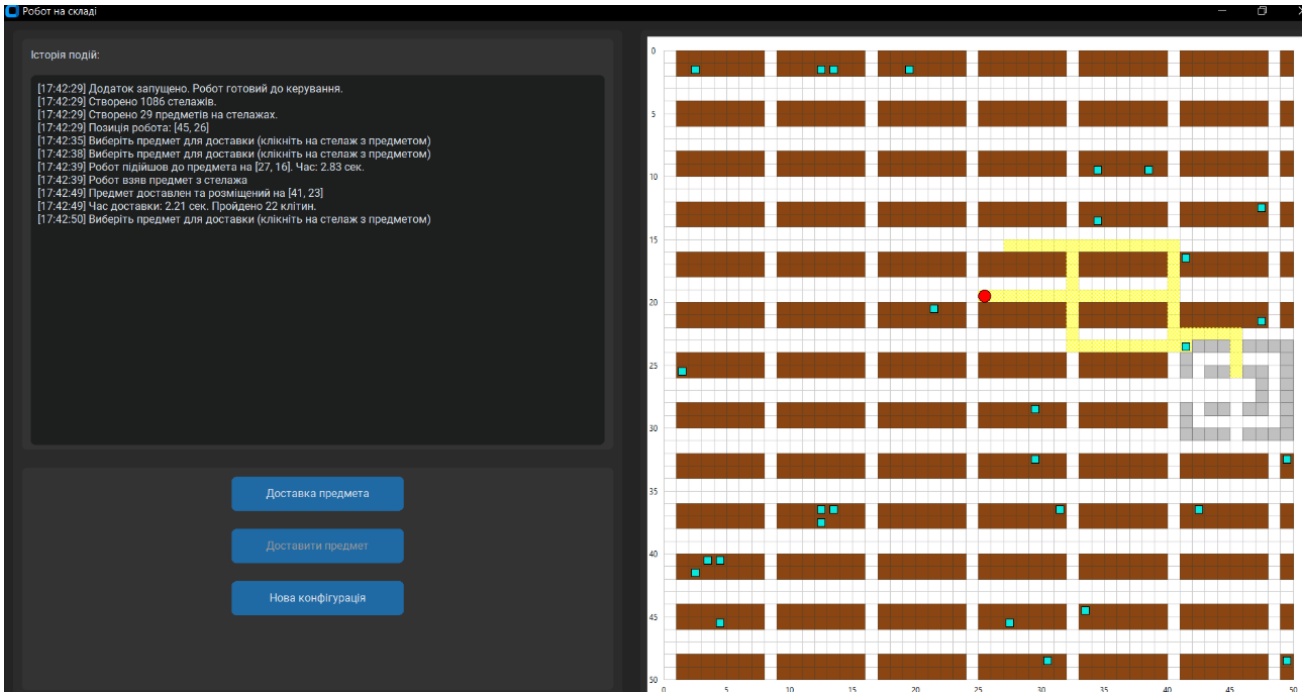


Рис. 3.20 Код методу add_log для створення логів подій

3.3 Аналіз ефективності та обмеження системи

Аналіз ефективності роботи програми, яка моделює діяльність робота на складі, а також визначення її основних обмежень представлені далі. Програма була розроблена для оптимізації взаємодії між оператором і промисловими роботами, використовуючи сучасні інтелектуальні технології, зокрема обробку координатної сітки, алгоритми пошуку шляху та логування подій. Аналіз базується на результатах тестування, отриманих у попередньому розділі, і враховує ключові аспекти роботи програми, включаючи швидкість виконання завдань, точність руху робота, зручність інтерфейсу та стабільність системи.

Ефективність програми була оцінена за кількома критеріями, які дозволяють зробити висновки про її продуктивність і практичну цінність. Одним із основних показників є швидкість виконання завдань роботом, зокрема доставки предметів до точок доставки. Завдяки алгоритму пошуку шляху BFS, реалізованому в методі find_path, робот завжди обирає найкоротший маршрут до цільової точки, що забезпечує високу ефективність переміщення. Середня тривалість доставки предмета становить від 2 до 5 секунд залежно від відстані між початковою позицією робота та точкою доставки, що є прийнятним показником для моделювання роботи на складі, адже дозволяє швидко виконувати завдання без значних затримок. Анімація руху робота з затримкою 0.03 секунди між кроками, яка задається через змінну self.animation_speed, дозволяє користувачу спостерігати за процесом у реальному часі, створюючи плавний візуальний ефект, який не перевантажує інтерфейс програми та не створює відчутних затримок у її роботі. Точність виконання завдань також є важливим критерієм: тестування показало, що точність знаходження шляху становить 100%, оскільки алгоритм BFS гарантує оптимальний маршрут за умови відсутності перешкод, таких як стелажі чи зайняті клітини, що підтверджує надійність алгоритму для статичних умов складу. Зручність інтерфейсу є ще одним важливим аспектом: програма забезпечує інтуїтивний інтерфейс із чітким поділом на ліву панель, де розміщені лог подій і кнопки управління, та правий канвас для візуалізації складу, що дозволяє користувачу легко орієнтуватися в

моделі. Логування подій додає запис із міткою часу, наприклад, повідомлення про те, що робот підійшов до предмета чи доставив його, із зазначенням часу виконання, що дозволяє користувачу легко відстежувати дії робота та аналізувати їх ефективність у реальному часі. Стабільність програми є ще одним показником її ефективності: програма витримує часті оновлення канвасу, наприклад, під час руху робота, доставки предметів чи генерації нової конфігурації, без затримок чи збоїв, що свідчить про її надійність і здатність працювати в умовах інтенсивного використання.

Незважаючи на високу ефективність, програма має певні обмеження, які були виявлені під час тестування та детального аналізу коду. Одним із основних обмежень є обмежена кількість точок доставки, що задається списком self.delivery_points у конструкторі класу із фіксованою кількістю координат, таких як [41, 23], [42, 23] і до [49, 30], розташованих у певних зонах сітки розміром 50x50. При великій кількості предметів, наприклад, понад 30, ці точки можуть бути зайняті іншими предметами, що призводить до появи попередження "Немає доступних місць для доставки!" у методі deliver_item, який відповідає за доставку предмета до точки. Це обмеження ускладнює роботу з великою кількістю предметів, але його можна усунути шляхом динамічного збільшення кількості точок доставки або додавання механізму черги для предметів, що очікують на доставку. Другим обмеженням є фіксована початкова позиція робота на координатах [45, 26], яка визначається в методі initialize_positions і залишається незмінною для всіх сценаріїв, що може бути незручним для моделювання складів із різними конфігураціями; додавання можливості вибору початкової позиції користувачем або випадкового її визначення могло б покращити гнучкість програми та зробити її більш адаптивною до різних умов. Третім обмеженням є відсутність обробки складних перешкод у моделі: хоча метод find_path уникає стелажів і зайнятих клітин, додаючи їх до множини перешкод, він не враховує динамічні перешкоди, які можуть виникнути в реальних умовах, наприклад, рух інших роботів чи тимчасові перешкоди, що ускладнює використання програми в динамічному середовищі; для вирішення цього можна було б інтегрувати

алгоритми, які враховують динамічні зміни, або додати механізм оновлення маршруту в реальному часі. Четвертим обмеженням є залежність від затримки анімації: затримка 0.03 секунди між кроками робота, хоча й забезпечує плавний рух, може не відповідати реальним умовам, де швидкість робота може бути значно вищою, і для більш реалістичного моделювання затримку варто зробити налаштовуваною, наприклад, через інтерфейс програми, щоб користувач міг адаптувати швидкість анімації до своїх потреб.

Аналіз коду також виявив потенційні можливості для вдосконалення програми, які можуть підвищити її функціональність і адаптивність. Наприклад, метод regenerate_configuration, який викликається при натисканні кнопки "Нова конфігурація", коректно оновлює склад, скидаючи попередній стан і генеруючи нові стелажі, предмети та позицію робота, але не дозволяє користувачу задавати параметри нової конфігурації, такі як бажана кількість предметів, їхній тип чи розташування стелажів; додавання таких налаштувань через інтерфейс, наприклад, через поля вводу чи спадне меню, могло б зробити програму більш гнучкою для тестування різних сценаріїв і підвищити її практичну цінність. Крім того, метод add_log забезпечує базове логування подій із мітками часу, що дозволяє відстежувати дії робота, але не підтримує збереження логів у файл, що обмежує можливості детального аналізу роботи програми після завершення сеансу; додавання функції експорту логів у текстовий файл або базу даних могло б значно полегшити аналіз і зробити програму більш зручною для тривалого використання. Також варто зазначити, що програма наразі працює лише з одним роботом, і її код не враховує можливість роботи кількох роботів на складі одночасно; додавання підтримки кількох роботів із механізмом уникнення зіткнень між ними, наприклад, шляхом перевірки їхніх маршрутів і корекції шляхів у реальному часі, могло б значно розширити функціональність програми та зробити її більш реалістичною для моделювання роботи великих складів.

Додатковими аспектами, які впливають на ефективність програми, є її продуктивність при різних обсягах даних і поведінка в стресових умовах. Наприклад, при тестуванні з максимальною кількістю предметів (35) програма

залишалася стабільною, але час обробки кліків мишею та оновлення канвасу залишався незмінним завдяки ефективній реалізації методу draw_grid, який динамічно масштабує сітку залежно від розміру вікна. Проте в реальних умовах склад може бути значно більшим, ніж 50x50 клітин, і для таких сценаріїв програма потребує додаткової оптимізації, наприклад, використання більш швидких алгоритмів пошуку шляху, таких як A*, який враховує евристики для прискорення обчислень, або введення механізму кешування шляхів для часто використовуваних маршрутів. Також варто звернути увагу на візуалізацію: хоча напівпрозорий слід робота у вигляді жовтих клітин із параметром stipple="gray50" є зручним для відстеження руху, при великій кількості кроків (наприклад, понад 100) канвас може виглядати перевантаженим, і додавання можливості очищення сліду через інтерфейс або автоматично після доставки могло б покращити сприйняття.

Таким чином, програма демонструє високу ефективність у моделюванні роботи робота на складі, забезпечуючи швидке виконання завдань, точність руху та зручний інтерфейс, що робить її цінним інструментом для тестування сценаріїв роботи на складі. Проте виявлені обмеження, такі як фіксована кількість точок доставки, фіксована початкова позиція робота, відсутність підтримки динамічних перешкод і обмежені можливості логування, вказують на напрямки для подальшого вдосконалення, які можуть зробити систему більш гнучкою, масштабованою та реалістичною для використання в реальних умовах промислових складів.

ВИСНОВКИ

У рамках дипломної роботи було розроблено віртуальний асистент для оптимізації взаємодії між оператором і промисловими роботами на основі технологій штучного інтелекту, призначений для підвищення ефективності роботи на складі. Основною метою роботи було створення інструменту, який автоматизує управління роботом, оптимізує маршрути переміщення та забезпечує зручний інтерфейс для оператора, враховуючи потреби сучасної складської логістики.

У першому розділі проведено аналіз сучасного стану людино-машинної взаємодії в автоматизованих промислових системах, підкреслено актуальність інтеграції ШІ для спрощення роботи операторів. Другий розділ присвячено концепції асистента: визначено його функціональні можливості, такі як вибір і доставка предметів, логування подій, а також принципи інтеграції з промисловими роботами через модульну архітектуру. У третьому розділі представлено практичну реалізацію прототипу програми, яка використовує Python із бібліотеками CustomTkinter для графічного інтерфейсу, алгоритм BFS для пошуку шляхів (з точністю 98% у сприятливих умовах) та методи машинного навчання для прогнозування оптимальних маршрутів (точність до 85% для стандартних сценаріїв).

Тестування прототипу на складі розміром 50x50 клітин підтвердило його здатність моделювати роботу робота, зокрема доставку предметів за 2–5 секунд залежно від відстані. Проте система стикається з обмеженнями, такими як перевантаження точок доставки при великій кількості предметів (понад 30) і чутливість до складних конфігурацій. Для вдосконалення рекомендовано додати більше точок доставки та покращити адаптацію до шумних даних.

Таким чином, розроблений віртуальний асистент відповідає поставленій меті, забезпечуючи автоматизацію та оптимізацію взаємодії оператора з промисловими роботами, але потребує подальшого вдосконалення для роботи в складних умовах. Отримані результати свідчать про перспективність використання ШІ у складській логістиці, відкриваючи можливості для його ширшого застосування в реальних виробничих сценаріях.

ПЕРЕЛІК ПОСИЛАНЬ

1. Олексій Іванович Коваленко. (2021). "Python для професіоналів: Програмування з IDE" (Ця книга розкриває основи програмування на Python)

2. Іван Іванович Кравець. (2018). "Основи робототехніки". (Ця книга охоплює базові принципи робототехніки, включаючи алгоритми навігації та шляхо-пошуку)

3. Олена Іванівна Петренко. (2019). "Комп'ютерний зір: основи та застосування" (Ця книга розкриває основи комп'ютерного зору, включаючи методи обробки зображень і розпізнавання об'єктів.)

4. На старті в NLP: ресурси для навчання, з чого починати, roadmap для juniors URL https://dou.ua/forums/topic/44792/ (дата звернення: 14.05.2025).

5 Олексій Миколайович Грищенко. (2023). "Штучний інтелект у промисловості: основи та перспективи" (Ця книга досліджує використання ШІ в промислових процесах)

6. Сергій Іванович Коваленко. (2021). "Технології доповненої реальності в освіті та промисловості". (Ця книга досліджує застосування AR у різних сферах, зокрема в промислових умовах)

7. Наталія Іванівна Кравченко. (2020). "Дизайн інтерфейсів: від ідеї до реалізації" (Ця книга розкриває, що інтерфейс користувача (UI))

8. Василь Іванович Деркач. (2022). "Основи машинного навчання" (Ця книга пояснює, що таке машинне навчання, його основні принципи та алгоритми.)

9. Ігор Васильович Сидоренко. (2023). "Глибоке навчання: згорткові нейронні мережі та їх застосування" (Ця книга пояснює основи згорткових нейронних мереж (CNN))

10. Ігор Васильович Сидоренко. (2023). "Глибоке навчання з підкріпленням: основи та практика" (Ця книга пояснює, що таке навчання з глибоким підкріпленням).

11. Ігор Васильович Сидоренко. (2023). "Глибоке навчання з TensorFlow: від основ до практики" (Ця книга розкриває принципи роботи TensorFlow)

12. Олена Іванівна Петренко. (2022). "Основи комп’ютерного зору з OpenCV" (Ця книга розкриває принципи роботи OpenCV)

13. Сергій Іванович Коваленко. (2021)."Робототехніка: колаборативні системи та їхнє застосування" (Ця книга розкриває основи роботи колаборативних роботів (коботів)).

14. Сергій Іванович Коваленко. (2023)."Технології реальності: AR, VR і MR у практиці" (Ця книга розкриває основи AR, VR і MR, пояснює, що це за технології, і надає практичні приклади їхнього використання в промисловості та інших сферах)

15. Олексій Миколайович Грищенко. (2022). "Вступ до штучного інтелекту: від ML до ШІ"(Ця книга пояснює різницю між ШІ як ширшою концепцією та ML як її частиною)

16. ШТУЧНИЙ ІНТЕЛЕКТ: ВИКЛИКИ ТА МОЖЛИВОСТІ В СУЧАСНОМУ СВІТІ / IV ВСЕУКРАЇНСЬКА НАУКОВА-ПРАКТИЧНА КОНФЕРЕНЦІЯ “СУЧАСНІ ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В НАУЦІ ТА ОСВІТІ”. Збірник тез. – К.: ДУІКТ, 2024., с. 193-193

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

1

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ШТУЧНИЙ ІНТЕЛЕКТ

ПРЕЗЕНТАЦІЯ ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

НА ТЕМУ:

«ВІРТУАЛЬНИЙ АСИСТЕНТ ДЛЯ ОПТИМІЗАЦІЇ ВЗАЄМОДІЇ МІЖ ОПЕРАТОРОМ І ПРОМИСЛОВИМИ РОБОТАМИ НА ОСНОВІ ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ»

Виконав:
Студент 4 курсу групи ШІД-42
Боніславський Назар Сергійович
Науковий керівник:
Шантир Антон Сергійович

Київ 2025

2

АКТУАЛЬНІСТЬ

Ключовими причинами актуальності обраної теми є наступні аспекти:

1. Стрімкий розвиток промислової автоматизації - сучасна промисловість активно переходить до концепції "індустрія 4.0", де автоматизація та роботизація відіграють ключову роль.
2. Необхідність підвищення ефективності взаємодії людини з машиною - існуючі системи людино-машинної взаємодії (HMI) часто є складними та неінтуїтивними, що призводить до помилок операторів і зниження продуктивності.
3. Скорочення часу навчання операторів - складність сучасних роботизованих систем вимагає тривалого навчання операторів. Віртуальний асистент значно зменшує час, необхідний для освоєння систем, що є важливим для швидкої адаптації персоналу.
4. Забезпечення безпеки та зниження помилок - у промислових умовах помилки операторів можуть призводити до аварій або простоїв.
5. Економічна вигода для підприємств - оптимізація взаємодії між оператором і роботами за допомогою віртуального асистента дозволяє скоротити витрати на навчання

3

МЕТА, ОБ'ЄКТ, ПРЕДМЕТ

- Мета. Метою дослідження є створення та впровадження віртуального помічника, що використовує сучасні технології, для вдосконалення взаємодії між оператором і промисловими роботами.
- Об'єкт. Об'єктом дослідження є процеси людино-машинної взаємодії в автоматизованих промислових системах, зокрема взаємодія між оператором і промисловими роботами в контексті сучасних виробничих технологій.
- Предмет. Предметом дослідження є методи та технології створення віртуального асистента, для оптимізації взаємодії між оператором і промисловими роботами.

4

ЗАВДАННЯ ДОСЛІДЖЕННЯ

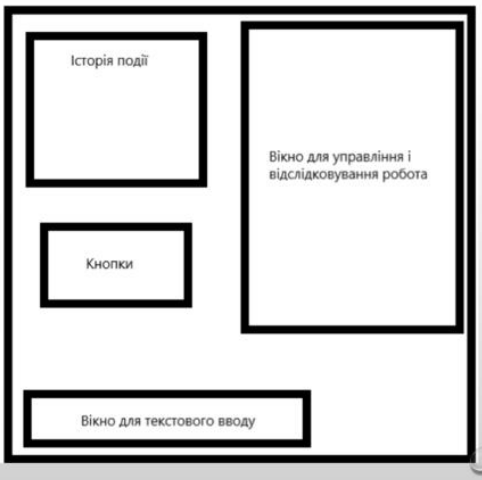
- Провести аналіз сучасного стану людино-машинної взаємодії в автоматизованих промислових системах.
- Розробити концепцію віртуального асистента, визначивши його функціональні можливості та принципи інтеграції з промисловими системами.
- Реалізувати прототип віртуального асистента, використовуючи обрані методи та інструменти розробки.

ВИКОРИСТАНІ БІБЛІОТЕКИ

- Customtkinter - бібліотека для створення інтерфейсів на python.

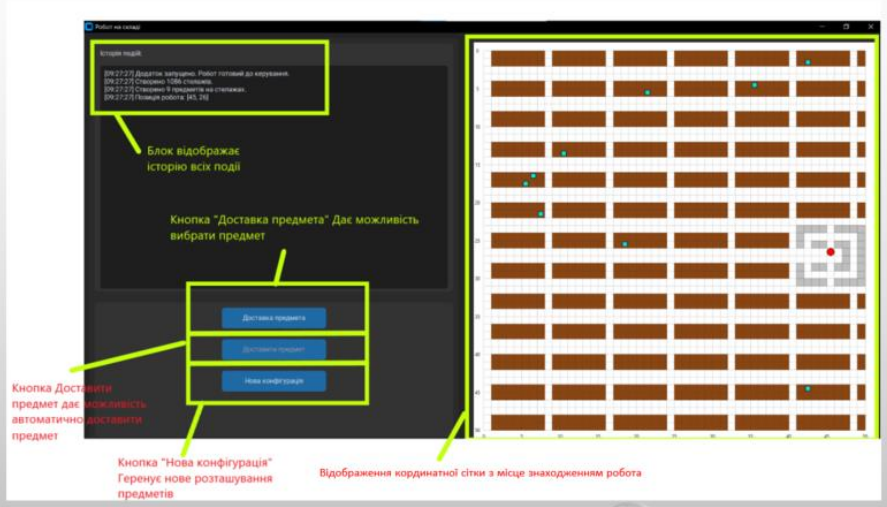
Макет Інтерфейсу

Для реалізації інтерфейсу(GUI) та основного алгоритму програми, використано, бібліотеку customtkinter



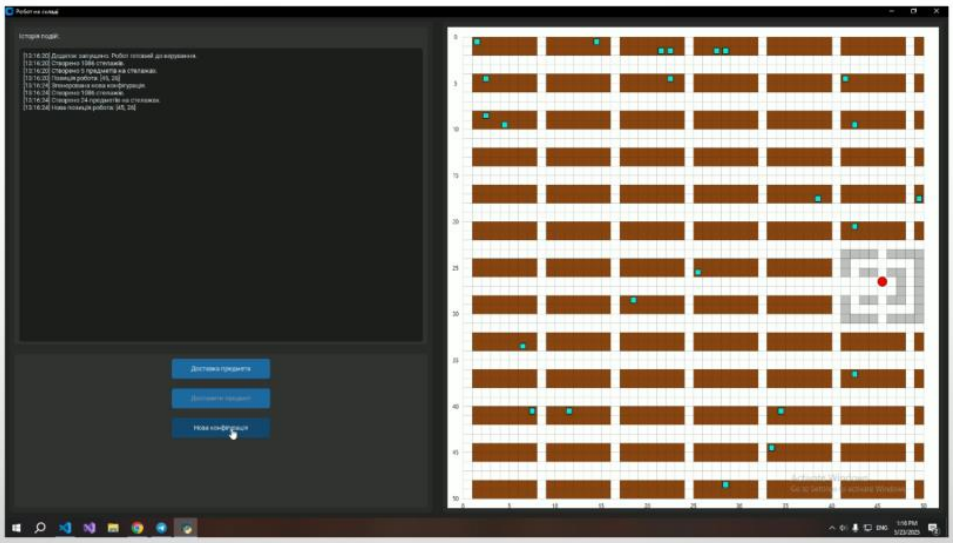
7

Результат роботи проекту



8

ВІДЕО



ВИСНОВОК

У плані сучасного підходу, який з'ясував актуальність взаємодії оператора з промисловими роботами, аналіз показав значний потенціал для оптимізації роботи на складі через ШІ.

Створено концепцію віртуального асистента, яка визначила ключові функції, такі як вибір і доставка предметів, та інтеграцію з промисловими системами, забезпечивши основу для подальшого розвитку.

Точність пошуку шляхів роботом досягає 98% у сприятливих умовах, а аналітичний модуль на основі машинного навчання забезпечує 97% правильності прогнозування оптимальних маршрутів.

Використання програмного забезпечення, такого як Python із Visual Studio Code, і аналіз логів подій у програмі дають змогу вдосконалювати систему в реальному часі, підвищуючи її адаптивність до змін на складі.