Шифр *«**СКАРЗСП**»*

Система керування автоматизованим роботом збору замовлення складських приміщень

ЗМІСТ

ВСТУП

Широке використання автоматизованих систем керування обумовлене бажанням людства автоматизувати свою роботу з метою оптимізації та полегшення праці. Сучасні підприємства та установи застосовують автоматизовані системи, як для управління підприємства, так, і для автоматизації праці.

Дані системи повинні бути корисними в своїй сфері, та відповідати усім стандартам. Зокрема, їх використання дозволяє домогтися зниження чисельності управлінського персоналу, підвищити якість функціонування об'єкта управління і самого управління. В даний час ми можемо побачити різноманітні типи автоматизованої техніки, яка призначена для вузького кола робіт та побуту.

Метою даної роботи є створення системи керування автоматизованим роботом збору замовлення складських приміщень, а саме алгоритмічно-програмних засобів, які повинні отримувати інформацію від глобальної мережі Інтернет, забезпечувати її опрацювання та передачу до апаратних засобів, які у свою чергу є виконавчими елементами системи.

Одним із основних завдань у цій роботі, є реалізація мультиагентної системи із алгоритмом пошуку найоптимальнішого шляху до продукції. Також під час організації інфраструктури системи, потрібно враховувати можливості гнучкого розширення та нарощування функціоналу.

Для досягнення вище наведеної мети необхідно виконати ряд задач, в тому числі:

- проаналізувати сучасне становище подібних систем на ринку;
- дослідити алгоритми пошуку найкоротшого шляху на графах, внести модифікації для досягнення найкращої швидкодії;
- реалізувати підпрограму, яка виконуватиме вибірку продукції із стеку;
- дослідити можливість виникнення конфліктів у мультиагентній системі, обчислити оптимальну кількість роботів у системі, для підвищення ефективності її роботи;

- обрати елементну базу для реалізації програмної частини
- розробити електричну схему виконавчого елемента системи, яка буде слугувати базовим взірцем.

Сучасні роботизовані системи збору замовлень є інноваційним рішенням, що забезпечує повну конфігурацію та гнучкість при логістиних операціях підприємства.

Дана тема є актуальною в наш час та варта уваги для подальшого дослідження та удосконалення.

РОЗДІЛ 1. ОСНОВНЕ ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ

У своїй діяльності практично будь-яка компанія стикається з завданням створення і підтримки функціонування складу на належному рівні.

У сучасних умовах потрібне ефективне складування всіх видів матеріалів: комплектуючих, запчастин, готової продукції та навіть документів. Це завдання може бути вирішено на базі технічних рішень, та за допомогою різних спеціалізованих шаф елеваторного або карусельного типу, призначених для зберігання малогабаритних та середньогабаритних вантажів. Автоматизовані склади з інформаційно-пошуковою системою широко застосовуються у всіх галузях промисловості, на підприємствах сфери обслуговування і торгівлі, де необхідна швидка доставка матеріалів на робочі місця операторів.

Провівши аналіз структурних рішень для організації складів, врахувавши всі плюси та мінуси конструктивного плану, було прийнято рішення розробляти систему керування роботом автоматизованого збору замовлення на базі вертикального складування. Основними перевагами таких систем є висока щільність зберігання, на одиницю займаної площі. Продукція зберігається у невеликих контейнерах, які щільно розташовуються один до другого. Розумне використання площі складського приміщення, забезпечує безпеку при зберіганні продукції, надійний облік і зручність отримання необхідних матеріалів. Порівнюючи з приміщення у яких використовуються навантажувачі, дані рішення економлять близько 40-60% простору. Немало важливою характерною ознакою є низьке енергоспоживання. Під час нормальної роботи системи не потрібно використовувати освітлення. Також достатньо близько 50% персоналу для виконання операцій: збору замовлень, сортування та обліку. Завдяки цьому, значно знижуються витрати праці та підвищується точність операцій порівняно з традиційним способом.

На Рис. 1.1 та 1.2 наведено загальну структуру складу, вигляд збоку та зверху відповідно. Конструктивно система автоматизованого збору замовлення має комірчасту структуру у вигляді тривимірної матриці (X, Y, Z). Продукція

розміщуться у вигляді стеку (у певному шарі Z), який знаходиться за координатами (X, Y).

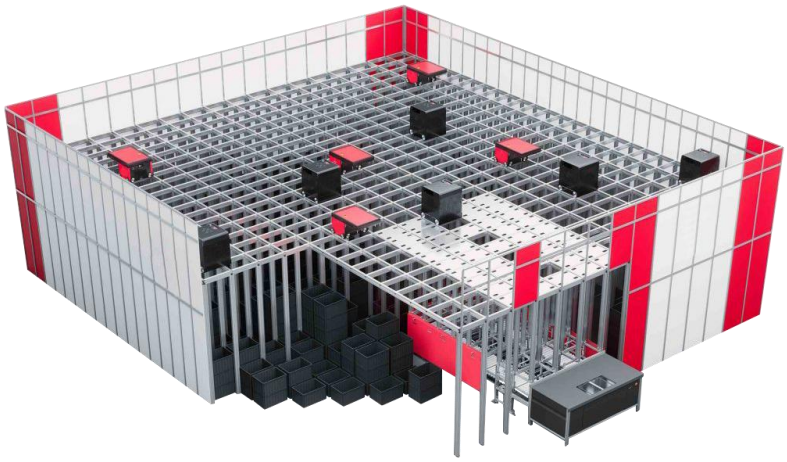


Рис 1.1 Загальна структура системи автоматизованого збору замовлення (вигляд збоку).

Дане конструктивне рішення дозволяє без проблеми розширювати систему як у ширину так і висоту. Обмеження встановлюються розмірами самого приміщення. На верхньому рівні по рейковій системі переміщаються виконавчі елементи (роботи збору замовлення), а також ділянка для видачі знайденої продукції операторові.

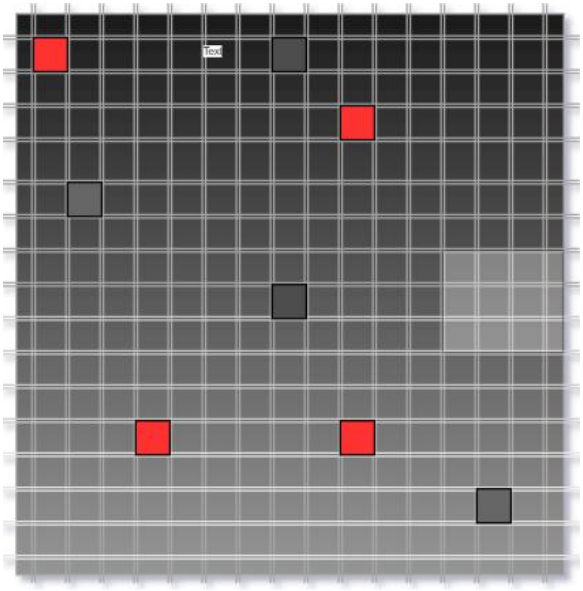


Рис 1.2 Загальна структура системи автоматизованого збору замовлення (вигляд зверху).

Одним із основних завдань у цій роботі, є реалізація мультиагентної системи із алгоритмом пошуку найоптимальнішого шляху до продукції.

Важливою і актуальною проблемою розробки мультиагентних систем є управління переміщеннями агентів в реальному часі. Агенти, можуть виконувати завдання, пов'язані з пересуванням з однієї точки середовища в іншу. Також агенти підкоряються певним обмеженням, серед яких можна виділити самих агентів, які є динамічними перешкодами один для одного. Під мультиагентною системою управління переміщеннями в середовищі будемо розуміти певний набір засобів, що забезпечує пересування агентів з однієї точки середовища в іншу без зіткнення з іншими агентами.

При розробці мультиагентної системи управління переміщеннями в першу чергу була проаналізована проблема планування шляху - прокладки маршруту для агента в середовищі без урахування присутності в цьому середовищі інших агентів. Це пояснюється тим, що при переміщенні агента з деякої початкової точки віртуального середовища в кінцеву (цільову) агент повинен дотримуватися найкоротшого шляху між початковим і кінцевим пунктом з урахуванням нерухомих перешкод для більш швидкого і ефективного виконання своїх завдань. При цьому планування найкоротшого шляху, якого повинен дотримуватися агент, не залежить від інших агентів через те, що довгострокове прогнозування зіткнень агента з іншими вимагає великої кількості обчислювальних ресурсів, що є неприйнятним при роботі системи, в якій одночасно бере участь безліч агентів в реальному часі. У зв'язку з цим було вирішено використовувати графи, що характеризують прохідні ділянки віртуального середовища, на яких буде здійснюватися пошук найкоротшого шляху. Після того, як був знайдений найкоротший шлях, виникає проблема трасування шляху - відстеження можливості пересування агента зі свого поточного місцезнаходження безпосередньо до кожної вершини графу, а також проблема запобігання зіткнень при локальній взаємодії між агентами.

Програмна частина розбивається на дві складові. Перша складова реалізує собою клієнт-серверну архітектуру з використанням веб технологій. Переваги

даного способу в тому, що спроектована система може легко розширюватись та оновлюватись. Також організаціям, які інтегрують дану систему не потрібно володіти стороннім програмним забезпеченням, оскільки всі потужності та ресурси, надаються індивідуально для кожного.

На Рис. 1.3 та 1.4 зображено UML діаграму та структурну схему клієнт-серверного застосунку відповідно.

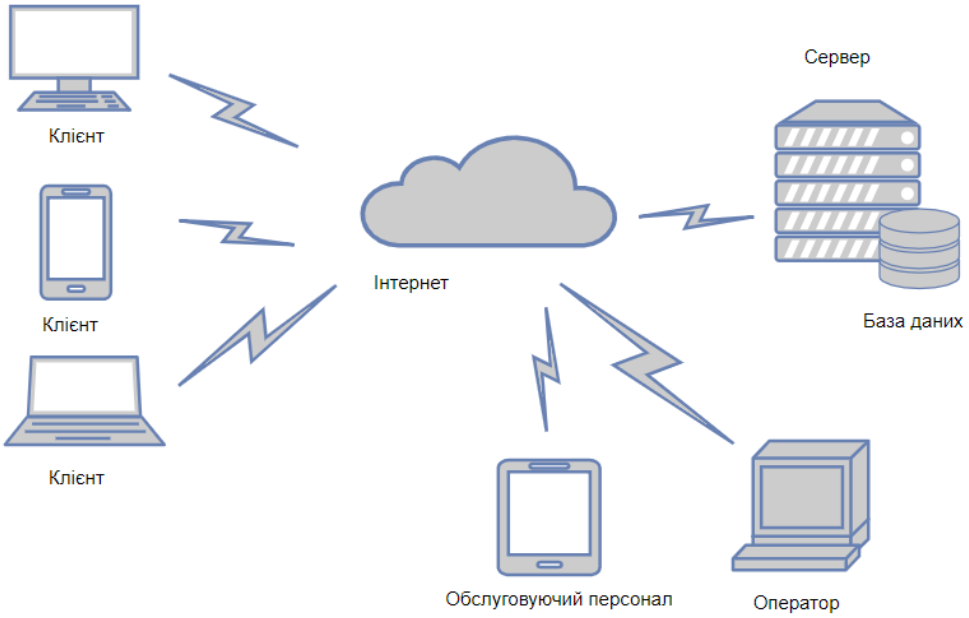


Рис 1.3 UML діаграма клієнт-серверної частини.

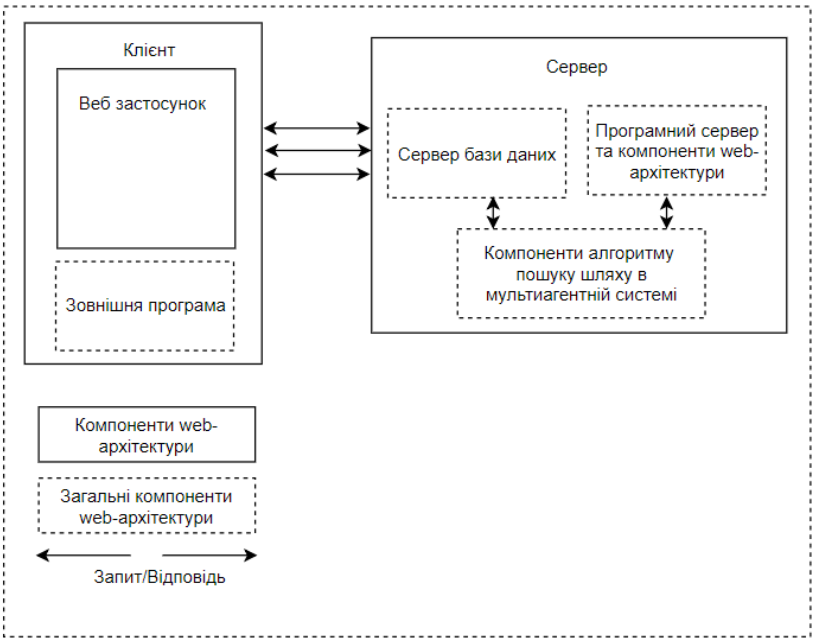


Рис 1.4 Структурна схема клієнт-серверного застосунку

Другою складовою являється сам алгоритм для виконання пошуку шляху, уникнення конфліктів та вибірки продукції. Для реалізації цієї частини, досліджувались алгоритми пошуку найкоротшого шляху на графах. Найбільш дієвими серед розглянутих виявились алгоритм A* та хвильовий алгоритм.

Алгоритм A* - це алгоритм пошуку по першому найкращому збігу на графі, який знаходить найкоротший шлях від однієї вершини (початкової) до іншої (цільової, кінцевої).

Це інформований метод пошуку, тобто порядок обходу вершин визначається евристичною функцією, яка є сумою двох інших: функцій, визначену довжину шляху, знайдену на попередніх етапах алгоритму, з початкової вершини до даної вершини, і евристичної оцінки довжини найкоротшого шляху з даної вершини в кінцеву.

Алгоритм A* розглядається в даній роботі з тієї причини, що цей алгоритм є допустимим, оскільки він ніколи не упустить можливості мінімізувати довжину шляху, повним (в тому сенсі, що він завжди знаходить рішення, якщо таке існує) і обходить при цьому мінімальну кількість вершин.

Хвильовий алгоритм - це алгоритм пошуку найкоротшого шляху на графі, заснований на методах пошуку в ширину. У зв'язку з тим, що сусідні клітинки вибираються з околу Мура, в розглянутому алгоритмі виникають неоднозначності при відновленні шляху після поширення хвилі. Це загрожує знаходженню шляху, який не є найкоротшим, тому необхідна конкретизація. При виникненні такої ситуації будемо встановлювати найвищий пріоритет у ортогональних сусідів поточного вузла при відновленні шляху. Такий пріоритет буде забезпечувати знаходження найкоротшого шляху виходячи з того, що відстань між діагональними сусідами перевищує відстань між ортогональними сусідами, а також виходячи з правил поширення хвилі, визначених даним алгоритмом.

Практичне порівняння ефективності цих алгоритмів на сітці однакової розмірності показали, що хвильовий алгоритм знаходить найкоротший шлях швидше, ніж алгоритм A*.

В спроектованій системі, алгоритм відповідає за пошук необхідної продукції та будує оптимальний шлях до неї. Даний алгоритм був вибраний мною для реалізації пошуку шляху в системі керування автоматизованим роботом. Дана система використовує дві осі координат (X, Y) для пошуку шляху до вказаного стовбця, та додаткову третю координату для пошуку потрібної комірки по осі Z.

Для обчислення потрібно отримати дані про точки старт та фініш. В даному випадку за поле пошуку шляху береться матриця (двомірний масив) int. Якщо значення комірки = 1, то вважається, що вона є зайнятою іншим роботом, якщо = 0, то шлях вільний. Об'єкт може рухатися в 4-х напрямках. Робота алгоритмного циклу починається зі старту. При кожній ітерації, проводиться порівнянна на вільну комірку. Якщо значення істинне, тоді записуємо +1 крок, якщо існує перешкода, здійснюється пошук в іншу сторону і так поки не досягнеться кінцева точка. Крім цього, шукається не єдиний шлях, а всі можливі і при порівнянні вибираєтья найкоротший. Після того як робот пройде шлях до потрібної комірки, для вибірки замовленої продукції, було реалізовано підпрограму, яка працює з декількома варіантами вибірки. В подальшому, вісь Z буде згадуватись як стек.

Варіанти вибірки продукції:

1) якщо продукція знаходиться на вершині стеку, все що потрібно це вибрати її і доставити до терміналу з оператором. В такому випадку, будується шлях за тим же принципом, тільки в зворотню сторону;

2) якщо продукція розміщується в середині або на дні стеку і до неї потрібно добратись, тоді алгоритм працює таким чином, що вибирає всю продукцію поступово та переміщає тимчасово в інші комірки. Сервер містить додаткову інформацію про продукцію, а саме індекс, який характеризує частоту замовлення даної одиниці. Чим дане число менше, тим нижче розташована дана продукція. Розташування з врахуванням індексу частоти замовлення пришвидшує роботу системи. Після того як робот доступився до потрібного товару, він виконує роботу, яка була описана в пункті №1.

Було проведено додаткове дослідження, яке стосується ситуації з виникнення конфліктів. Перш за все пропонується обчислити можливу оптимальну кількість роботів у системі, на основі габаритів складського приміщення (довжина та ширина), навколо кожного робота умовно малюється коло з радіусом, який дозволить йому розминутись із іншим таким же роботом. Площа усіх радіусів у сумі не повинна перевищувати загальну площу системи. Тоді можна стверджувати, що конфлікти зіткнень будуть малоімовірними, так як завжди буде знайдено шлях обходу перешкоди.

РОЗДІЛ 2. АПАРАТНА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ

В даному розділі розробляється блок схеми алгоритму системи збору замовення складських приміщень, та реалізація програмної частини. Також розроблено загальну електричну принципову схему розробленого пристрою яка розміщується в додатку В.

Дана схема складається з наступних вузлів:

- Модуля WI-FI (1);
- Flash пам‘яті (2);
- Вузла скиду (3);
- Вузла синхронізації (4);
- Вузла драйвера крокових двигунів (5);
- Вузла давача лінії (6);
- Вузла USB-UART конвертера (7);
- Вузла живлення (8);

За допомогою спільного функціонування усіх вузлів, отримано надійний та високопродуктивний виконавчий елемент системи, який є базою для функціонування системи.

Координування роботи здійснюється за допомогою мікроконтроллера з бездротовим інтерфейсом Wi-Fi. Для того щоб впевнено приймати та передавати радіосигнал, система оснащена антеною. Після того, як сигнал опрацьовується модулем Wi-Fi, потрібна команда надходить на драйвер крокового двигуна. Через простий інтерфейс здійснюється управління напрямком і кроком обертання електродвигуна. Крковий привід дозволяє отримувати точне позиціонування без використання зворотного зв'язку від давачів кутового положення, та точно здійснювати рух по вказаному шляху.

Для кращого позиціонування робота у системі, використовується давач лінії. Основне його призначення в даному пристрої, це корегування руху при подоланні дистанції. Так як робот здійснює рух у трьох напрямках і

направляється по рейковій системі, йому потрібно корегувати дані про полеження за допомогою міток.

За збір вихідних даних із пристрою керування автоматизованим роботом відповідає Flash память. На ній логується інформація про стан та можливі збої в роботі.

Наступним етапом є розробка програмної частини спроектованої системи.

На Рис. 2.1. зображено блок-схему алгоритму програми виконавчого елемента (робота)

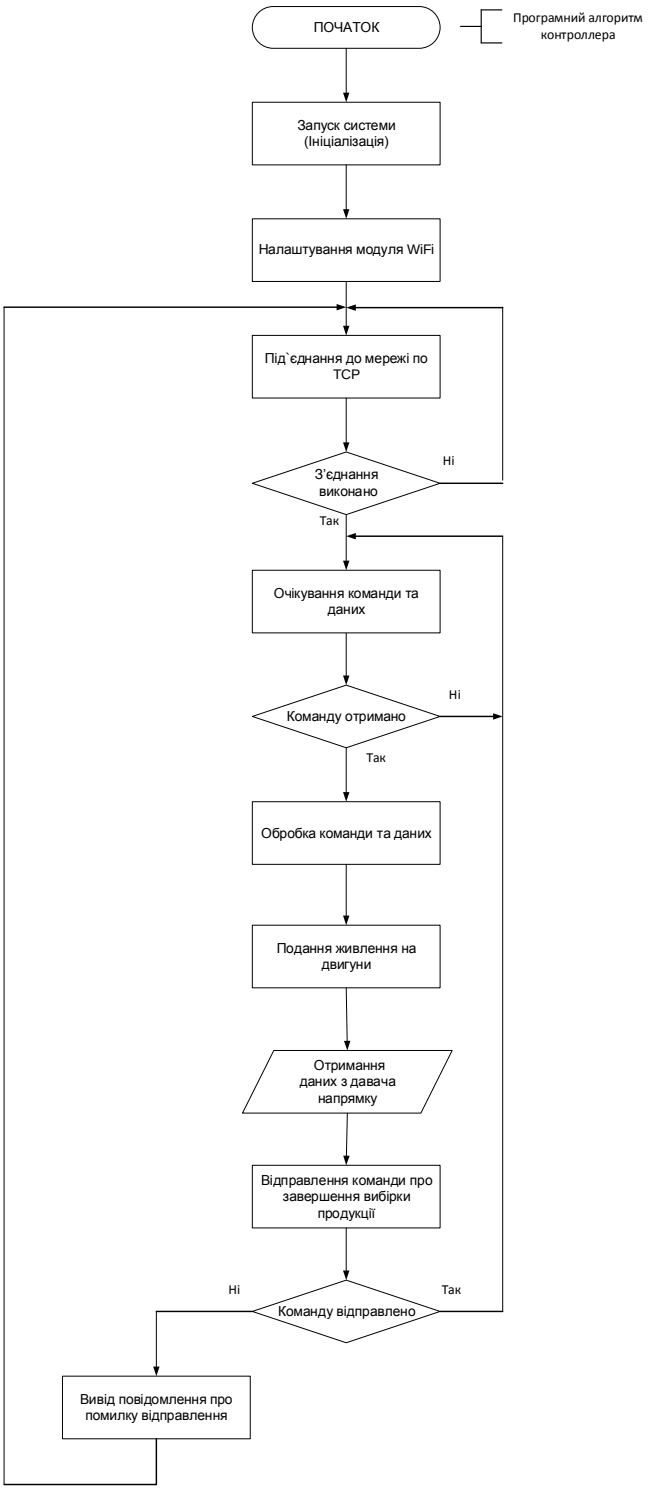


Рис. 2.1 Блок-схема алгоритму програмної частини виконавчого елемента

На Рис. 2.2 зображено блок-схему, яка описує загальний принцип функціонування програмної частини сервера.

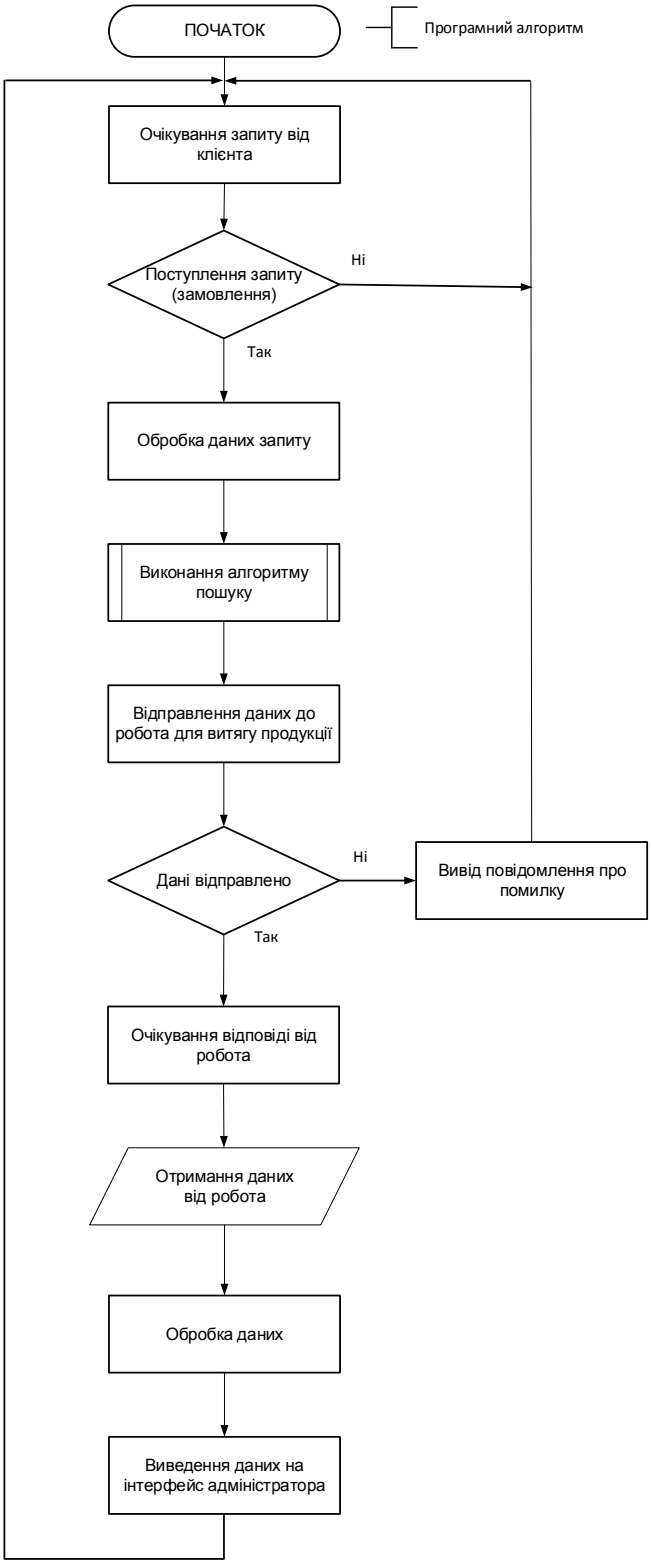


Рис. 2.2 Блок-схема алгоритму програмної частини сервера

На Рис 2.3 зображено детальний опис алгоритму пошуку найкоротшого маршруту. На етапі знаходження усіх можливих маршрутів, алгоритм зображається в окремій блоксхемі.

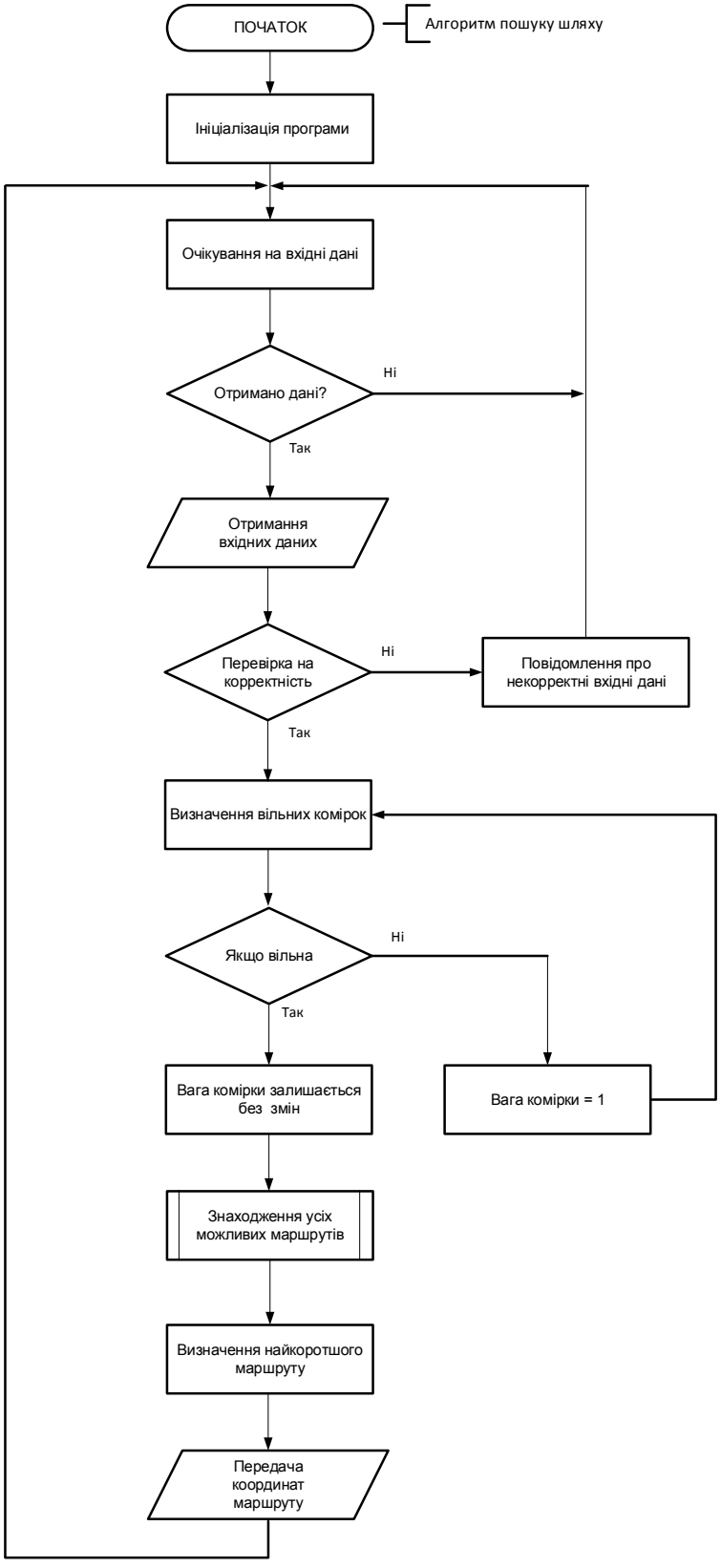


Рис. 2.3 Блок-схема алгоритму пошуку найкоротшого маршруту

Основна ціль такого підходу, це знайти усі можливі шляхи які існують до заданої продукції, після чого вибирається той, в якого найменша довжина.

При написанні програмного забезпечення мікроконтроллера керування автоматизованим роботом збору замовлень на складських приміщень, було використано бібліотеку <SoftwareSerial.h>.

За допомогою бібліотеки програмно створюються кілька послідовних портів, що працюють на швидкості до 115200 бод. Пристрої, що працюють з інвертованим сигналом, в бібліотеці передбачається відповідний параметр, який вміщує інвертування.

В програмі використовувалась дана бібліотека для реалізації послідовного інтерфейсу на 16 та 17 цифрових виводах.

```
#include <SoftwareSerial.h>
…
SoftwareSerial mySerial(10, 11);  де 10 – RX(Recieve) , 11 – TX(Transmit)
…
```

Наведений уривок коду демонструє перевірку на під’єднання до точки доступу Wi-Fi по мережному протоколу TCP.

```
#include <WiFi.h>
const char* ssid     = "Place Wifi name here";
const char* password = "Place your Wifi password here";
int stepper=15; // stepper delay
WiFiServer server(80);
…
Serial.println();
   Serial.print("Connecting to ");
   Serial.println(ssid);
   WiFi.begin(ssid, password);
   while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
   }
   server.begin();
```

Ідентифікатори SSID, PASSWORD та PORT містять внесені дані про точку доступу, які потрібні для підключення, це назва точки доступу, пароль та порт. За допомогою функції connectWiFi() здійснюється встановлення з'єднання з роутером та очікування на відповідь, при успішному зєднанні повертається "true", в іншому випадку "false" після чого на послідовний порт виводиться повідомлення про успішне підключення або помилку.

Наступний код відповідає за ініціалізацію та прийом команд та даних з серверної частини, подачі імпульсів управління на двигуни та зворотній зв'язок з сервером.

```
…
HTTP.onNotFound(handleNotFound);
HTTP.on("/", handleRoot);
HTTP.on("/restart", handle_Restart);
…
pinMode(5, OUTPUT);
pinMode(38, OUTPUT);
pinMode(41, OUTPUT);
pinMode(35, OUTPUT);
pinMode(5, OUTPUT);
…
if (currentLine.endsWith("GET /goForOrder:{xy}")) {
        if(xy!=null){
                for (int i=0; i <= 137 ; i++){
                        digitalWrite(41, HIGH);
                        digitalWrite(5, HIGH);
                }
        }
}
```

Java Server був зібраний за допомогою Maven у середовищі розробки IntelliJ IDEA. Даний засіб автоматизації роботи з програмними проектани є досить поширеним та легким у використанні. Тестування написаної програми проводилось за допомогою Unit тестів, під час яких враховувались різноманітні ситуаціїї проходження сценаріїв роботи системи.

При розробці програми використовувався імпорт наступних пакетів:

- java IO

  У java.io існують, так звані, потоки введення та виведення (InputStream and OutputStream). Вони поділяються на байтові та символьні потоки.

Для даної програми java.io був призначений для запису і читання даних:

1) у файл;
2) Робота з мережним підключенням;
3) System.err, System.in, System.out;

- Collection

  Цей пакет використовується для створення так званих колекцій java.

  Вони працюють за прикладом масивів та здатні зберігати велику кількість даних в одній змінній. Основною особливістю є те, що у них явно не задається розмір, він збільшується автоматично у півтори рази при повному заповненні.

В наступному коді проводиться ініціалізація матриці, по які здійснюють рух робота. Також відбувається ініціалізація початкової та кінцевої точок маршруту, після чого вони передаються як вхідні дані алгоритму пошуку шляху:

```
…
GetIndexMethod indexMethod = new GetIndexMethod();
   int[][] labyrinth = {
         {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
         {0, 1, 1, 1, 1, 1, 1, 1, 1, 0},
         {0, 1, 0, 1, 1, 0, 1, 0, 0, 0},
         {0, 1, 1, 1, 1, 1, 1, 1, 1, 0},
```

```
            {0, 1, 1, 1, 1, 0, 0, 0, 1, 0},
            {0, 1, 0, 1, 1, 0, 1, 0, 1, 0},
            {0, 1, 1, 1, 1, 1, 1, 0, 1, 0},
            {0, 1, 1, 0, 1, 1, 1, 0, 1, 0},
            {0, 1, 1, 1, 1, 1, 1, 1, 1, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
    };
    PathFinder pathFinder = new PathFinder(labyrinth);
    PathFinder.Point start = pathFinder.new Point(getX(), getY());
    PathFinder.Point end = pathFinder.new Point(getZ(), getK()
    PathFinder.Point[] path = new PathFinder.Point[0];
    try {
        path = pathFinder.find(start, end);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    for (PathFinder.Point p : path) {
         return String.valueOf(p.getX() + "" + p.getY());
    }
    return null;
}
…
```

При створенні нового замовлення, створюється окремий потік, в якому виконується алгоритм пошуку шляху за вхідними даними, після чого інформація про маршрут зберігається та пересилається на мікроконтролер. В даному випадку на графічний тестовий варіант роботи системи. Нижче наведений фрагмент програми створення потоку:

```
…
@Override
public void run() {
```

```
    try {
        dataInit();
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        System.out.println();
    }
}
…
```

При отримані даних про замовлення, починається виконання алгоритму пошуку маршруту. Це відбувається при кожному запиті та триває доти, поки не отримується повний шлях.

Нижче зображено фрагмент коду, за допомогою якого отримується запит від користувача, та передається на вхід виконання алгоритму пошуку шляху:

```
public String[] dataWay(){
    int k = 1;
    int a = 1;
    List<Basket> basketList = wayBuilder.findAll();
    String way[] = new String[100];
    if(!basketList.isEmpty()) {
        for (Basket basket : basketList) {
            a++;
            k++;
            way = Stream.concat(Arrays.stream(way), Arrays.stream(new Example("" +
basket.getId(), basket.getX(), basket.getY(), basket.getZ(), k,
a).dataInit())).toArray(String[]::new);
        }
    }
```

```
        return way;
}
```

Шлях коригується щоразу при проходженні нової комірки, та порівнюється із іншими маршрутами що є в системі. Даний фрагмент демонструє частину логіки написаного алгоритму:

```
   n = fillmap[p.getY()][p.getX()] + labyrinth[p.getY()][p.getX()];
   if ((p.getY() + 1 < labyrinth.length) && labyrinth[p.getY() + 1][p.getX()] != 0)
      push(new Point(p.getX(), p.getY() + 1), n);
   if ((p.getY() - 1 >= 0) && (labyrinth[p.getY() - 1][p.getX()] != 0))
      push(new Point(p.getX(), p.getY() - 1), n);
   if ((p.getX() + 1 < labyrinth[p.getY()].length) && (labyrinth[p.getY()][p.getX() +
1] != 0))
      push(new Point(p.getX() + 1, p.getY()), n);
   if ((p.getX() - 1 >= 0) && (labyrinth[p.getY()][p.getX() - 1] != 0))
      push(new Point(p.getX() - 1, p.getY()), n);
}
if (fillmap[end.getY()][end.getX()] == Integer.MAX_VALUE) {
   System.err.println("Шляху не існує !!!");
   return null;
…
```

В коді вище виконується пошук усіх можливих шляхів до вказаної точки. Якщо під час пошуку наступної комірки зустрічається перешкода, тобто інший робот, то здійснюється перевірка на сусідні незайняті комірки. Координати кожного проходу зберігаються у масиві, для уникнення повторюваності.

Після того, як всі шляхи знайдено та збережено, з них вибирається найкоротший.

Наступний фрагмент коду демонструє прохід по знайденому шляху.

```
   System.out.println("Пошук завершено, прохід по ньому !!!");
List path = new ArrayList();
```

```
path.add(end);
int x = end.getX();
int y = end.getY();
n = Integer.MAX_VALUE;
while ((x != start.getX()) || (y != start.getY())) {
    if (fillmap[y + 1][x] < n) {
        tx = x;
        ty = y + 1;
        t = fillmap[y + 1][x];
    }
    if (fillmap[y - 1][x] < n) {
        tx = x;
        ty = y - 1;
        t = fillmap[y - 1][x];
    }

    if (fillmap[y][x + 1] < n) {
        tx = x + 1;
        ty = y;
        t = fillmap[y][x + 1];
    }
    if (fillmap[y][x - 1] < n) {
        tx = x - 1;
        ty = y;
        t = fillmap[y][x - 1];
    }
    x = tx;
    y = ty;
    n = t;
```

```
    path.add(new Point(x, y));
}
```

Після того як шлях знайдено, масив координат за допомогою TCP/IP протоколу відправляється на спроектований виконавчий елемент, для початку здійснення руху по заданих координатах.

Наступним етапом є вибірка заданої продукції із комірки. Система зазделегідь містить інформацію про кількість продукції, та розміщення контейнера із продукцією у стеку. Якщо продукцію знаходиться на вершині стеку, то будується зворотній шлях до терміналу за тим же принципом. При необхідності вибірки продукції із середини або дна стеку, робота відбувається покроково. Після того як витягується непотрібний контейнер, алгоритм будує шлях до найближчої комірки, куди його можна поставити і так доти, поки не досягнеться ціль вибірки.

РОЗДІЛ 3. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

Розроблено клієнт-серверне програмне забезпечення на базі веб-технологій, за допомогою якого можна продемонструвати взаємодію різних частини аплікації. На Рис. 3.1 зображено сторінку адміністратора, з якої здійснюється додавання продукції до бази даних.

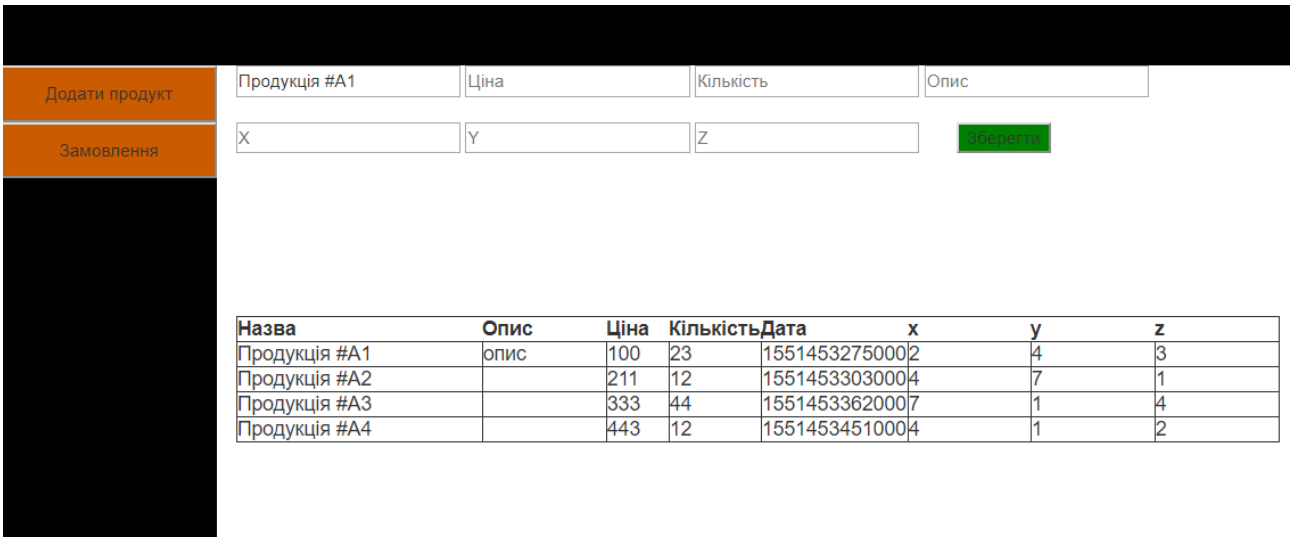


Рис. 3.1 Веб сторінка адміністратора

Після заповнення полів, головним завданням постає прив’язка продукції до конкретної адреси в системі. Для цього надається три координати X, Y та Z.

Клієнт із сторінки клієнтського застосунку може здійснювати замовлення будь якої продукції, яка присутня в базі даних. Для цього потрібно лише вибрати потрібну йому, та за допомогою кнопки “Купити”, відправити запит на сервер.

На Рис 3.2. Зображено тестовий вигляд сторінки клієнтського веб застосунку.

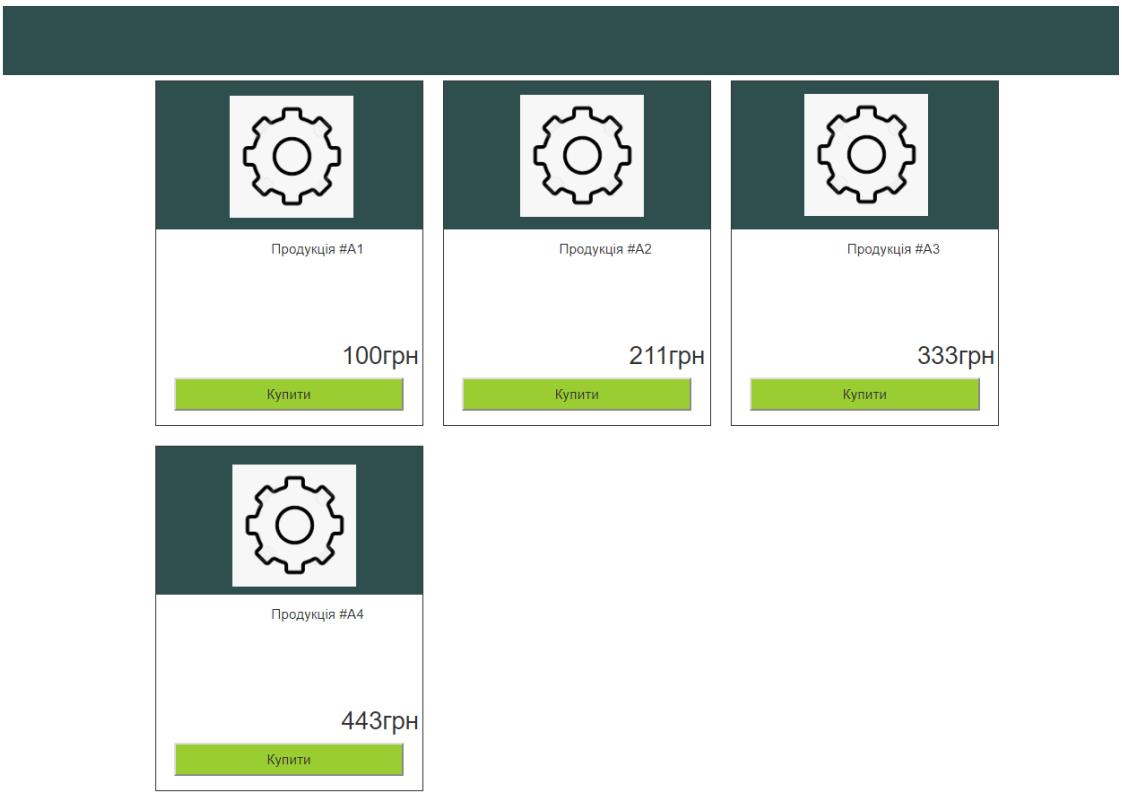


Рис. 3.2 Клієнтська сторінка веб застосунку

Після того як запит було відправлено, він опрацьовується, та передається на алгоритм побудови шляху.

Нижче на Рис. 3.3 та Рис. 3.4 наведено зображення роботи системи зі сторони оператора. Для цього спроектовано тестову програму, яка відображає траєкторію руху роботів у системі. Індекси які зображені на кожній комірці, є унікальними, до них прив’язується конкретна продукція.

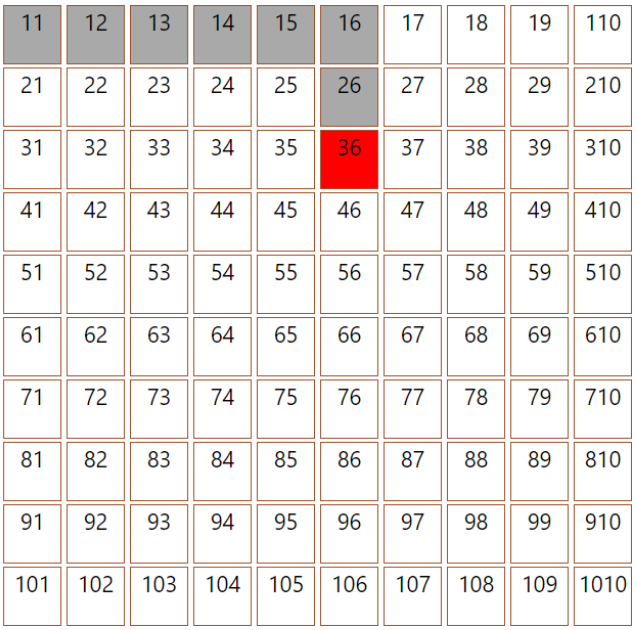


Рис. 3.3 Траєкторія руху робота у системі (один робот у системі)

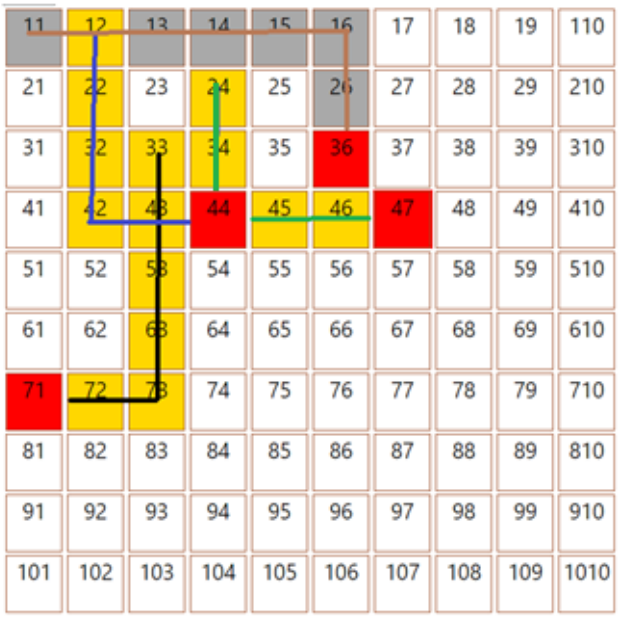


Рис. 3.4 Траєкторія руху роботів у системі (кілька роботів у системі)

На зображенні, червоні квадрати відповідають за позицію виконавчого елемента. Жовті – це шлях, який потрібно пройти до місця розміщення потрібної продукції. Сірим кольором, позначено пройдений шлях.

ВИСНОВКИ ТА ОСНОВНІ НАУКОВІ РЕЗУЛЬТАТИ РОБОТИ

За результатами досліджень розв'язана наукова задача з розробки системи керування автоматизованим роботом збору замовлення складських приміщень, яка повністю відповідає поставленим вимогам. Вирішено проблеми, які виникають при розробці мультиагентних систем управління переміщеннями. В першу чергу була проаналізована проблема планування шляху, відстеження можливості пересування агента зі свого поточного місцезнаходження безпосередньо до кожної вершини графу, а також проблема запобігання зіткнень при локальній взаємодії між агентами. Обрано елементну базу і наведено основні вузли системи. Розроблено схему електричну принципову, граф-схему алгоритму роботи програмної та апаратної частин та їх реалізацію.

Було створено алгоритмічно-програмні засоби, які отримують інформацію від глобальної мережі Інтернет, забезпечують її опрацювання та передачу до апаратних засобів, які у свою чергу є виконавчими елементами системи.

Розроблено клієнт-серверне програмне забезпечення на базі веб-технологій, яке надає можливість координувати роботу усіх складових частин системи.

Продемонстровано роботу веб застосунків зі сторони користувача (клієнта) та оператора. Спроектовано тестову програму, яка графічно відображає траєкторію руху роботів у системі. Проведено тестування системи за допомогою Unit тестів, під час яких враховувались різноманітні ситуаціїї проходження сценаріїв роботи системи.

СПИСОК ПОСИЛАНЬ

1. Алгоритми пошуку шляхів у графі [Електронний ресурс], 2018 – Режим доступу:http://evgavrilenko.ucoz.ru/DS/LEKCIYA_6-7.pdf, вільний. (3.05.2018).
2. Джон Росс. «Wi-Fi. Бездротові мережі. Установка. Конфігурація. Використання». 2005. № 31. С.34-89.
3. Вілл В.В. «Мікропроцесорні системи». 2009. № 7. С.24-28.
4. Багатопоточність в Java – керівництво [Електронний ресурс], 2018 – Режим доступу http://www.internet-technologies. ru/articles/mnogopotochnost-v-java-rukovodstvo-s-primerami.html, вільний. – (6.06.2018).

АНОТАЦІЯ

Стрімкий розвиток науки та техніки вимагає прогресивних рішень та пошуку нових моделей, методів, засобів та технологій побудови сучасних автоматизованих та комп’ютеризованих систем керування. У рамках розробки системи керування автоматизованим роботом збору замовлення на складських приміщеннях, постають проблеми планування шляху роботів в системі, запобігання зіткнень при їхній взаємодії, розрахунку оптимальної їх кількості для максимального пришвидшення роботи. Одним із основних завдань у цій роботі, є реалізація мультиагентної системи із алгоритмом пошуку найоптимальнішого шляху до продукції. Також під час організації інфраструктури системи, потрібно враховувати можливості гнучкого розширення та нарощування функціоналу, без внесення значних змін, інтегрування у існуючі системи складського обліку.

Метою цієї роботи є створення алгоритмічно-програмних засобів, які повинні отримувати інформацію від глобальної мережі Інтернет, забезпечувати її опрацювання та передачу до апаратних засобів, які у свою чергу є виконавчими елементами системи.

Для досягнення вище наведеної мети були виконані наступні завдання:

- проаналізувано сучасне становище подібних систем на ринку, обрано оптимальну схему організації складського приміщення, враховано основні аспекти органіцазії для подальшої розробки;
- досліджено алгоритми пошуку найкоротшого шляху на графах, наведені результати практичного порівняння ефективності цих алгоритмів, а також представлена розроблена модифікація алгоритму хвильового трасування, що демонструє перевагу у швидкодії;
- реалізовано підпрограму, яка працює з декількома варіантами вибірки продукції із стеку;

- проведено додаткове дослідження виникнення конфліктів, обчислено можливу оптимальну кількість роботів у системі, що підвищує ефективність її роботи;
- розроблено функціональну та електричну схеми виконавчого елемента системи із оптимальною елементною базою, описано основні вузли та їх призначення.

Розроблено клієнт-серверне програмне забезпечення на базі веб-технологій, яке надає можливість координувати роботу усіх складових частин системи. Перевагами даного архітектурного рішення є:

- мобільність та кросплатформеність;
- відсутність потреби встановлення зайвого програмного забезпечення;
- інтеграція з існуючими системами управління, та обліку;
- відсутність проблеми з підтримкою старих версій програм та зворотньою сумісністю.

Продемонстровано роботу веб застосунків зі сторони користувача (клієнта) та оператора. Спроектовано тестову програму, яка графічно відображає траєкторію руху роботів у системі.

Наведено UML – діаграми роботи реалізованого програмного забезпечення, блок схеми алгоритмів програмної та апаратної реалізації, а також пояснено фрагменти підпрограм.

Дана тема є актуальною в наш час та варта уваги для подальшого удосконалення, тому що тільки автоматизований склад може забезпечити конкурентну перевагу на сучасному ринку. Автоматизація, дозволяє прискорити логістичний процес, спростити роботу з інформацією, зменшити трудомісткість роботи людини і покращити загальний контроль на складі, отримати в режимі реального часу дані про товарні залишки і стадії обробки замовлення, а також значно скоротити витрати.

Ключові слова: система керування, мультиагентна система, автоматизація процесів, роботизовані системи збору замовлень.

Додаток А

Лістинг програми пошуку маршруту

PathFinder.jar

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
class PathFinder {
   class Point {
      private int x;
      private int y;
      Point(int x, int y) {
         this.x = x;
         this.y = y;
      }
      synchronized public int getX() {
         return x;
      }
      synchronized public int getY() {
         return y;
      }
      @Override
      synchronized public boolean equals(Object o) {
         if (!(o instanceof Point)) return false;
         return (((Point) o).getX() == x) && (((Point) o).getY() == y);
      }
      @Override
      synchronized public int hashCode() {
         return Integer.valueOf(x) ^ Integer.valueOf(y);
      }
```

```
    @Override
    synchronized  public String toString() {
        return "x: " + Integer.valueOf(x).toString() + " y:" + Integer.valueOf(y).toString();
    }
}

int[][] fillmap = new int[10][10];
int[][] labyrinth;
List buf = new ArrayList();
PathFinder(int[][] labyrinth) {
    this.labyrinth = labyrinth;
}
synchronized void  push(Point p, int n) {
    if (fillmap[p.getY()][p.getX()] <= n) return;
    fillmap[p.getY()][p.getX()] = n;
    buf.add(p);
}
synchronized Point pop() {
    if (buf.isEmpty()) return null;
    return (Point) buf.remove(0);
}
synchronized  Point[] find(Point start, Point end) throws InterruptedException {
    int tx = 0, ty = 0, n = 0, t = 0;
    Point p;
    for (int i = 0; i < fillmap.length; i++)
        Arrays.fill(fillmap[i], Integer.MAX_VALUE);
    push(start, 0);
    while ((p = pop()) != null) {
        if (p.equals(end)) {
            System.out.print("Знайдено шлях довжиною ");
            System.out.println(n);
        }
        n = fillmap[p.getY()][p.getX()] + labyrinth[p.getY()][p.getX()];
        if ((p.getY() + 1 < labyrinth.length) && labyrinth[p.getY() + 1][p.getX()] != 0)
            push(new Point(p.getX(), p.getY() + 1), n);
```

```
    if ((p.getY() - 1 >= 0) && (labyrinth[p.getY() - 1][p.getX()] != 0))
        push(new Point(p.getX(), p.getY() - 1), n);
    if ((p.getX() + 1 < labyrinth[p.getY()].length) && (labyrinth[p.getY()][p.getX() + 1] != 0))
        push(new Point(p.getX() + 1, p.getY()), n);
    if ((p.getX() - 1 >= 0) && (labyrinth[p.getY()][p.getX() - 1] != 0))
        push(new Point(p.getX() - 1, p.getY()), n);
}
if (fillmap[end.getY()][end.getX()] == Integer.MAX_VALUE) {
    System.err.println("Шляху не існує !!!");
    return null;
} else
    System.out.println("Пошук завершено, прохід по ньому !!!");
List path = new ArrayList();
path.add(end);
int x = end.getX();
int y = end.getY();
n = Integer.MAX_VALUE;
while ((x != start.getX()) || (y != start.getY())) {
    if (fillmap[y + 1][x] < n) {
        tx = x;
        ty = y + 1;
        t = fillmap[y + 1][x];
    }
    if (fillmap[y - 1][x] < n) {
        tx = x;
        ty = y - 1;
        t = fillmap[y - 1][x];
    }
    if (fillmap[y][x + 1] < n) {
        tx = x + 1;
        ty = y;
        t = fillmap[y][x + 1];
    }
    if (fillmap[y][x - 1] < n) {
        tx = x - 1;
```

```
            ty = y;
            t = fillmap[y][x - 1];
        }
        x = tx;
        y = ty;
        n = t;

        path.add(new Point(x, y));
        System.out.println(x+" "+y);
        Thread.sleep(1000);
    }
    Point[] result = new Point[path.size()];
    t = path.size();
    for (Object point : path)
        result[--t] = (Point) point;
    return result;
}
```

Init.jar

```
public  class Example implements Runnable {
    String robotNumber;
    int x;
    int y;
    int z;
    int k;
    int a;
    Thread thread;
    public Example(String robotNumber, int x, int y, int z, int k, int a) {
        this.robotNumber = robotNumber;
        this.x = x;
        this.y = y;
        this.z = z;
        this.k = k;
        this.a = a;
        this.thread = new Thread(this, String.valueOf(robotNumber));
        this.thread.start();
```

```
}
public Example() {
}
public String getRobotNumber() {
   return robotNumber;
}

public void setRobotNumber(String robotNumber) {
   this.robotNumber = robotNumber;
}
public Thread getThread() {
   return thread;
}
public void setThread(Thread thread) {
   this.thread = thread;
}
public int getX() {
   return x;
}
public void setX(int x) {
   this.x = x;
}
public int getY() {
   return y;
}
public void setY(int y) {
   this.y = y;
}
public int getZ() {
   return z;
}
public void setZ(int z) {
   this.z = z;
}
public int getK() {
```

```
    return k;
}
public void setK(int k) {
    this.k = k;
}
public int getA() {
    return a;
}
@Override
public void run() {
    try {
        dataInit();
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        System.out.println();
    }
}
synchronized public  String dataInit() {
    GetIndexMethod indexMethod = new GetIndexMethod();
    int[][] labyrinth = {
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 1, 1, 1, 1, 1, 1, 1, 1, 0},
            {0, 1, 0, 1, 1, 0, 1, 0, 0, 0},
            {0, 1, 1, 1, 1, 1, 1, 1, 1, 0},
            {0, 1, 1, 1, 1, 0, 0, 0, 1, 0},
            {0, 1, 0, 1, 1, 0, 1, 0, 1, 0},
            {0, 1, 1, 1, 1, 1, 1, 0, 1, 0},
            {0, 1, 1, 0, 1, 1, 1, 0, 1, 0},
            {0, 1, 1, 1, 1, 1, 1, 1, 1, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
    };
    PathFinder pathFinder = new PathFinder(labyrinth);
    PathFinder.Point start = pathFinder.new Point(getX(), getY());
```

```
        PathFinder.Point end = pathFinder.new Point(getZ(), getK());
        PathFinder.Point[] path = new PathFinder.Point[0];
        try {
            path = pathFinder.find(start, end);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        for (PathFinder.Point p : path) {
             return String.valueOf(p.getX() + "" + p.getY());
        }
        return null;
    }
}
```

Лістинг програми мікроконтролера

WebServer.c

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
IPAddress apIP(192, 168, 4, 1);
ESP8266WebServer HTTP(80);
String _ssid     = "home";
String _password = "i12345678";
String _ssidAP = "WiFi";
String _passwordAP = "";
void setup() {
  Serial.begin(115200);
  Serial.println("");
  Serial.println("Start 1-WIFI");
  WIFIinit();
  Serial.println("Start 2-WebServer");
  HTTP_init();
}
void loop() {
  HTTP.handleClient();
  delay(1);
```

```
}
```

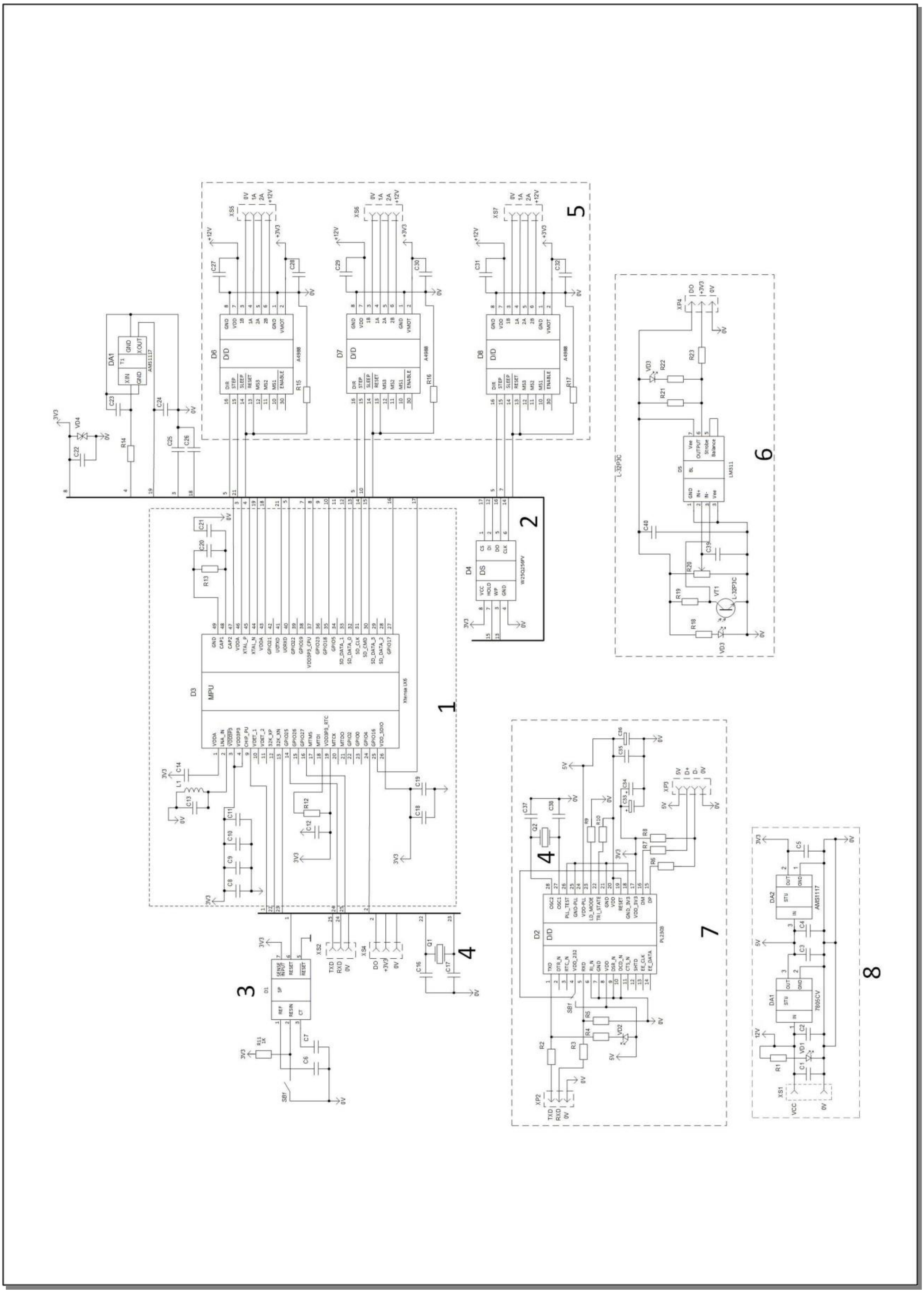
WiFi.c

```
void WIFIinit() {
  WiFi.mode(WIFI_STA);
  byte tries = 11;
  WiFi.begin(_ssid.c_str(), _password.c_str());
  while (--tries && WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(1000);
  }
  if (WiFi.status() != WL_CONNECTED)
  {
    Serial.println("");
    Serial.println("WiFi up AP");
    StartAPMode();
  }
  else {
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
  }
}
bool StartAPMode()
{
  WiFi.disconnect();
  WiFi.mode(WIFI_AP);
  WiFi.softAPConfig(apIP, apIP, IPAddress(255, 255, 255, 0));
  WiFi.softAP(_ssidAP.c_str(), _passwordAP.c_str());
  return true;
}
```

HTTP_INIT.jar

```
void HTTP_init(void) {
  HTTP.onNotFound(handleNotFound);
  HTTP.on("/", handleRoot);
  HTTP.on("/restart", handle_Restart);
  HTTP.begin();
}
void handleNotFound(){
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += HTTP.uri();
  message += "\nMethod: ";
  message += (HTTP.method() == HTTP_GET)?"GET":"POST";
  message += "\nArguments: ";
  message += HTTP.args();
  message += "\n";
  for (uint8_t i=0; i<HTTP.args(); i++){
    message += " " + HTTP.argName(i) + ": " + HTTP.arg(i) + "\n";
  }
  HTTP.send(404, "text/plain", message);
}
void handleRoot() {
  HTTP.send(200, "text/plain", "hello from esp8266!");
}
void handle_Restart() {
  String restart = HTTP.arg("device");
  if (restart == "ok") ESP.restart();
  HTTP.send(200, "text/plain", "OK");
}
```

Додаток В


1
2
3
4
5
6
7
8
D3
MPU
D4
DS
W25Q256FV
D6
D7
D8
A4988
DA1
DA2
AMS1117
7805CV
D2
PL2303
D5
LM311