ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА

(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій

(повне найменування факультету)

Кафедра комп'ютерних систем та технологій

(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

«Розробка системи орієнтування та будування карти навколишнього середовища для автономного робота»

(тема кваліфікаційної роботи українською мовою)

«Development of an orientation system and environment map construction for an autonomous robot»

(тема кваліфікаційної роботи англійською мовою)

Виконав(ла): здобувач(ка) денної/заочної форми навчання
спеціальності 123 Комп'ютерна інженерія

(код, назва спеціальності)

Освітня програма _____Комп'ютерна інженерія_____

(назва)

Лісовський Максим Миколайович

(прізвище, ім'я, по-батькові здобувача)

Керівник ст. викл. Шаріпова І. В. _____

(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент доц. Ларін Д. Г.

(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
комп'ютерних систем та технологій
№ ___ від ____.____. 20___ р.

Завідувач кафедри Юрій ГУНЧЕНКО

_____ _____

(підпис) (прізвище, ім'я)

Захищено на засіданні ЕК № __5___
протокол № __від ____.____.20___ р.

Оцінка_____/_____/_____

(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК Світлана АНТОЩУК

_____ _____

(підпис) (прізвище, ім'я)

Одеса 2025

АНОТАЦІЯ

У кваліфікаційній роботі розробляється тема «Розробка системи орієнтування та будування карти навколишнього середовища для автономного робота».

Актуальність теми. У сучасному світі, де інформаційні технології стали невід'ємною частиною життя більшості людей, важливою задачею є створення карти навколишнього середовища за допомогою робототехнічних систем.

Мета роботи – дослідження та реалізація методів побудови карт навколишнього середовища для мобільних роботів, за допомогою алгоритму G-mapping.

Розглянуто методи картографування та досліджено вимоги до інструментів для побудови віртуальних сцен. Запропоновано та реалізовано власний підхід до створення такого інструменту, який дозволяє проводити тестування алгоритму у симульованому середовищі. Отримані результати дозволяють оцінити ефективність різних методів картографування та їх практичне застосування у робототехніці.

Об’єктом дослідження є методи та алгоритми створення карт навколишнього середовища для мобільних роботів, зокрема алгоритми одночасної локалізації та картографування (SLAM) та інструменти для віртуального моделювання карт.

Розроблена система дозволяє мобільному роботу ефективно орієнтуватися в просторі та будувати карту на основі сенсорних даних. Проведені експерименти у віртуальному середовищі продемонстрували можливість застосування реалізованого підходу для автономної навігації. Отримані результати можуть бути використані для подальшого вдосконалення алгоритмів SLAM та їхнього впровадження в реальні робототехнічні системи.

Ключові слова: картографування, фільтр Рао-Блеквела, Монте-Карло, мобільні роботи, віртуальні сцени, навколишнє середовище.

ABSTRACT

In the qualification work, the topic “Development of an Orientation System and Building an Environment Map for an Autonomous Robot” is being developed. The work aims to study and implement methods for building environmental maps for mobile robots using the G-mapping algorithm.

Relevance of the topic. In the modern world, where information technology has become an integral part of most people's lives, an important task is to create a map of the environment using robotic systems.

The purpose of the work is to study and implement methods for building environmental maps for mobile robots using the G-mapping algorithm.

Mapping methods are considered, and the requirements for tools used in building virtual scenes are investigated. An own approach to creating such a tool is proposed and implemented, which allows testing the algorithm in a simulated environment. The results obtained allow us to evaluate the effectiveness of various mapping methods and their practical application in robotics.

The object of the study is methods and algorithms for creating environmental maps for mobile robots, including simultaneous localization and mapping (SLAM) algorithms and tools for virtual mapping.

The developed system allows a mobile robot to effectively navigate in space and build a map based on sensor data. Experiments in a virtual environment have demonstrated the possibility of using the implemented approach for autonomous navigation. The obtained results can be used for further improvement of SLAM algorithms and their implementation in a real robotic system.

Keywords: mapping, Rao-Blackwell filter, Monte Carlo, mobile robots, virtual scenes, environment.

ЗМІСТ

ВСТУП

Вступ до дослідження алгоритмів навігації мобільних роботів є важливою частиною сучасних досліджень у сфері робототехніки, адже автономні роботи все частіше використовуються для виконання різних завдань: логістика, сільське господарство, дослідження небезпечних середовищ та багато іншого. Ключовою проблемою, яку необхідно вирішити для успішного функціонування таких роботів, є забезпечення надійної та точної навігації в умовах динамічного і невизначеного середовища, якщо враховувати проблему переміщення роботів з точки А в точку В, що є базовою проблемою, яка стосується усіх роботів [8].

Алгоритми навігації мобільних роботів включають низку методів, що дозволяють роботу визначити своє місцезнаходження, побудувати маршрут і досягти кінцевої точки, уникаючи перешкод. Однією з важливих задач при цьому є вибір та селекція значущих точок на маршруті руху, які допомагають роботу орієнтуватися в просторі, знижують обчислювальні витрати та підвищують ефективність планування шляху. Селекція значущих точок передбачає визначення тих позицій, які мають суттєві значення для побудови траєкторії руху. Це можуть бути орієнтири, які легко розпізнати, або критичні точки, де змінюється напрямок руху [9].

Класифікація мобільних роботів базується на кількох критеріях: за типом середовища, механізмом пересування, джерелом енергії, рівнем автономності та призначенням. Така класифікація допомагає вибрати оптимальні рішення для різних завдань і середовищ використання.

Способи управління мобільними роботами класифікуються на: дистанційне керування, автономне управління, групове та напівавтономне управління.

Для вибору значущих точок (ключових орієнтирів) по маршруту руху мобільного робота можна використовувати наступні критерії: місця, де робот змінює напрямок руху; точки, де робот повинен змінити траєкторію.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Основні поняття картографування середовища

В даному розділі розглядаються основні поняття та методи, які пов’язані з розробкою системи орієнтування та створення карти навколишнього середовища для автономного робота. Зокрема, акцентується увага на ключових аспектах картографії, сенсорних технологіях та алгоритмах, які дозволяють ефективно здійснювати роботу побудови карти та навігацію в реальному часі. Ключову увагу можна виділити обробці даних та процесам збору використовуючи різні датчики, таких як ультразвукові сенсори та камери, LiDAR та метод SLAM. Отже, система орієнтування забезпечує дуже ефективне та точне створення карти навколишнього середовища для автономного робота, яке забезпечує його здатність до навігації в складних умовах.

Автономну навігацію мобільних роботів можна розподілити на три сфери: картографування, планування шляхів та локалізація. Картографування існує для того, щоб об’єднати окремі спостереження про навколишнє середовище в одну зручну модель, яка показує, як буде виглядати світ. Планування шляху знаходить кращий маршрут на карті, для того щоб переміщатися в навколишньому середовищі та дістатися до потрібного місця. Локалізація використовується для того, щоб досить точно визначити поточне місцезнаходження робота в середовищі [1].

Одним з основних завдань автономного робота є одночасне будування карти довкілля та визначення свого місцезнаходження в просторі, а також планування можливого шляху для переміщення. Одним з основних методів, який описує цей напрям є SLAM (Simultaneous Localization And Mapping). SLAM-це задача в робототехніці, яка вимагає від робота свого одночасного визначення місцезнаходження та створення карти середовища в якому він знаходиться. Завдяки цьому, інтелектуальні роботи збирають інформацію про навколишнє середовище та орієнтуються в просторі.

Метод полягає в тому, що робот застосовує різноманітні датчики(гіроскоп, акселерометр, камеру та лазерний далекомір) для збору інформації про своє місцезнаходження та навколишнього середовища. Після цього, він обробляє ці дані, щоб визначити своє розташування в просторі та сформувати уявлення про середовище. Треба зазначити, що SLAM не є окремим алгоритмом, це є набір методів та інструментів, які допомагають вирішувати завдання щодо визначення місцезнаходження робота для побудови карти середовища. Наразі існує достатньо велика кількість методів, які використовують програмні та апаратні можливості. Вибір самого методу залежить від поставленого завдання та характеристик використовуваних датчиків. Але кінцевою метою всіх цих методів-це є досягнення високої точності та швидкості обробки даних з цих датчиків, для того щоб робот міг ефективно орієнтуватися в просторі та безпечно в ньому рухатися.

Основною ідеєю більшої кількості методів SLAM та їх алгоритмів полягає в тому, що робот, перебуваючи в певній точці виявляє навколо себе інші об’єкти та вимірює і запам’ятовує до них відстань, дала він рухається до інших об’єктів. Коли всі предмети в деякому просторі виявлені і виміряні відстані, робот повертається на початкову позицію. Однак, скоріше за все, він потрапить в інше місце через помилку даних які отримуються із сенсорів руху [2].

Існує кілька методів SLAM, які використовуються залежно від типу сенсорів та задач, які ставляться перед роботами. Одними з найбільш популярних методів є Visual SLAM, LiDAR SLAM та Multi-Sensor SLAM. Виходячи з назви методу, Visual SLAM потребує використання зображення, яке надходить з камери та інших датчиків. Тут використовуються досить прості камери (сферичні, «риб’яче око» та ширококутні), стерео та багатокамерні камери, а також камери глибини та різкості. LiDAR SLAM – це метод, який використовує лазерні сканери для збору інформації та точного виміру відстані до об’єкта. Multi-Sensor SLAM це саме той метод який об’єднує вище перераховані SLAM методи. Він може збирати дані з різних

сенсорів враховуючи ультразвукові датчики, акселерометри, камери, LiDAR та гіроскопи для того щоб отримати більш точну локалізацію робота та точну карту навколишнього середовища. Але даний метод є більш складним, оскільки потрібно обробляти дані з кількох джерел, а також потрібна комплексна обробка, однак використовуючи цей метод можна забезпечити високу точність до різних середовищ [3].

SLAM має низку переваг, однією з важливих є його автономність, тобто цей метод дозволяє роботам працювати без попередньо створеної карти навколишнього середовища. Крім цього, він є досить таки гнучким, оскільки підтримує різноманітні типи датчиків. Завдяки точному визначенню місцезнаходженню та створенню карти, робот ефективно орієнтується в незнайомих та складних середовищах, що дозволяє уникати перешкоди. Також, SLAM працює в реальному часі – це дає можливість швидко адаптуватися роботу до змін у середовищі та планувати рух у реальному часі. Ще однією перевагою методу є масштабованість, він може працювати в малих приміщеннях, так і в більш відкритих просторах. Найголовнішою перевагою методу, також є те що він забезпечує високий рівень точності побудови карти.

Водночас цей метод має і низку недоліків. Одним з основних недоліків є високі обчислювальні витрати, оскільки обробляючи великий обсяг даних із різних датчиків у реальному часі потрібні значні обчислювальні потужності. Наприклад, якщо робот використовує лазерний сканер або камеру, то перш за все об’єкт потрібно розпізнати та створити 3D-карту. Для обробки даних в реальному часі, потрібен швидкий процесор щоб все проходило швидко і без затримок. Також, для деяких задач, потрібен великий обсяг оперативної пам’яті, оскільки метод постійно генерує нову інформацію про середовище і система повинна мати достатньо пам’яті для даних в реальному часі, щоб швидше працювати з нею. Крім того, точність SLAM може погіршуватись у складних умовах, наприклад, при поганій видимості для камер або лазерних сканерів.

Метод SLAM є досить потужним інструментом для автономної навігації

роботів. Він активно використовується в різних сферах, де точне картографування середовища та локалізація є важливою для успішної роботи. Наприклад метод SLAM використовується в таких галузях як: робот-пилосос для прибирання кімнати, дрони, медицина, безпілотні автомобілі. Якщо казати про робот-пилосос, то це є один з найкращих прикладів використанню методу SLAM. Якби не цей метод, то рухи пилососу були б просто хаотичними і він не зміг би виявити перешкоди, а також не зміг би запам’ятати вже очищені ділянки, що ускладнювало б досягненню мети по очищенню кімнати. Використовуючи SLAM для безпілотників, він дозволяє дронам знаходити своє місцезнаходження, змінювати маршрут в реальному часі та створювати карту середовища. В БПЛА присутні LiDAR сканери, завдяки яким дрони точно корегують свої маршрути і можуть працювати без людського втручання. В медицині використання SLAM методу є дуже важливим, оскільки це допомагає лікарям проводити точні операції з мінімальним втручанням в тіло пацієнта, тому що створюється 3D-модель органів людини. Технологія SLAM є досить важливою для безпілотних автомобілів, програмне забезпечення може визначати: розмітку дороги, кольори світлофору, параметри зустрічного транспорту та навіть людей. Оскільки технології кожного року вдосконалюються, то безпілотні авто відкриють нові можливості для безпечно та автономної їзді по дорозі [4].

Отже, в даному розділі розглянуто основні принципи роботи методу SLAM, він є ключовим для навігації мобільних роботів. Метод допомагає роботам будувати карту навколишнього середовища та визначати своє місцезнаходження, що є важливим для безпечного та ефективного переміщення на площині. Підкреслено, що метод використовує різні сенсори, такі як камери, LiDAR, ультразвукові датчики. Можна зазначити, що метод має як недоліки так і переваги, але постійне вдосконалення технологій зменшує недоліки, забезпечуючи ефективні рішення. Якщо, дивитися загалом, то метод SLAM є потужним інструментом для розвитку автономних систем.

1.2 EKF-SLAM: розширений фільтра Калмана

EKF-SLAM (Extended Kalman Filter-Simultaneous Localization and Mapping) – це метод, який дає змогу роботу одночасно визначати своє місцезнаходження в просторі та відразу будувати карту навколишнього середовища. Зазвичай метод спирається на математичну модель, яка враховує рух матеріального тіла робота та отриманих даних із сенсорів руху. Досить часто дану модель називають – модель передбачення. В цьому випадку використовується модель переміщення робота, яка спрощена до моделі одноколісного транспортного засобу. Також є і врахування помилок, які можуть виникати шляхом вимірювання його руху за допомогою даних, що отримуються з датчиків руху. Середовище повинно спостерігатися через орієнтири. Головними характеристиками орієнтирів є їхня легка упізнаваність, легкість у спостереженні та незмінність. Наприклад, найкращими орієнтирами в середовищі можуть слугувати кімнатні стіни або досить великі вертикальні площини. Дані вказівні предмети слугують для лазерного сканера як прямі лінії. Важливим етапом методу є визначення орієнтирів, які надходять з датчиків руху. Дані, які представлені набором точок, надходять у полярних координатах, тобто в системі координат, де точка на площині визначається відстанню та кутом. Після цього точки потрібно перевести в координати, що визначаються за допомогою двох чисел (x, y) і вже далі точки визначаються як орієнтири. Наступним кроком, самі орієнтири розглядаються як математичні лінії, але вони вже будуть вважатися нескінченними. Знаходячись на карті, робот має початок координат. Лінії на карті будуть позначені точкою, яка відповідає місцю, де початок координатної системи перетинає саме цю лінію. Виходячи з цього, кожен орієнтир на карті позначений лише точкою. Наступним важливим етапом SLAM є асоціація даних. На даному етапі визначається чи є орієнтир новим або відомим. Саме тут зіставляються дані поточного спостереження з відомими орієнтирами, у випадку, якщо спостереження не збігається з орієнтиром, то він вважається новим. EKF

враховує повторне спостереження, оновлюючи позицію або дода новий орієнтир до карти [5].

Як і будь-який алгоритм, EKF-SLAM має свої переваги та недоліки, які визначають його ефективність у різних умовах. Основними перевагами методу є взаємозалежність, карта яка отримується в ході картографування є точною та узгодженою. Також метод може працювати в закритих приміщеннях, він є ефективним у картографуванні обмежених середовищ, складських приміщеннях, кімнатах та коридорів. Ще метод можна використовувати у різних типах роботів, а саме: наземних, повітряних, підводних. Завдяки цьому він є універсальним. Але і є недоліки методу, сюди можна віднести: новий орієнтир, який може випадково співвіднестись з вже існуючим і це може спричинити неправильну побудову карти. Іноді робот помиляється в оцінці свого місцезнаходження і EKF-SLAM не зможе виправити карту, оскільки він не враховує декілька можливих варіантів одночасно.

Зображення «див. рис. 2.1» показує процес роботи EKF-SLAM. Робот позначений червоним еліпсом оцінює своє положення в просторі та визначає орієнтири (сині еліпси). Чорними хрестиками представлені орієнтири які надходять з сенсорних вимірювань. Чорні лінії між роботом та орієнтирами свідчать про перевірку даних, а саме чи є спостереження новим або відомим.

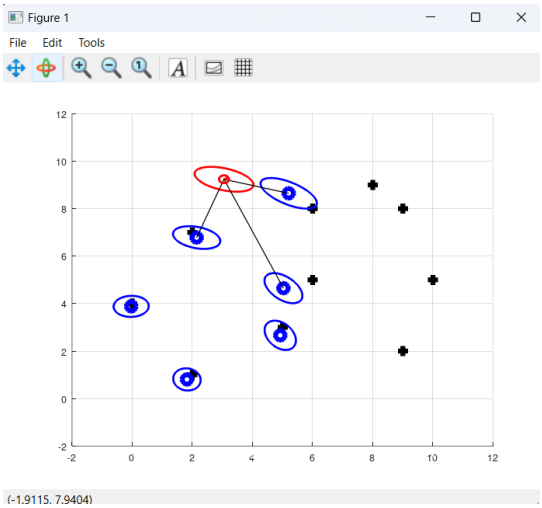

Рисунок 1.2.1 – Робот впродовж опрацювання програми

Фільтр Калмана прогнозує положення робота на основі моделі руху та коригує його відповідно до отриманих спостережень. Еліпси відображають невизначеність у позиції робота та орієнтирів. Залежно від розмірів еліпсу визначається точність положення робота, тобто якщо еліпс менший то положення визначено точно і навпаки. І якщо робот знову знаходить орієнтир, то його місцезнаходження уточнюється, а невизначеність зменшується. Якщо, орієнтир новий, то він додається до карти і алгоритм продовжує оновлення стану робота та карти.

Отже, метод EKF-SLAM є ефективним підходом до одночасного визначення місцезнаходження робота та побудови карти навколишнього середовища. Завдяки використанню фільтра Калмана, він дозволяє оцінювати положення робота та орієнтирів, поступово уточнюючи дані та зменшуючи невизначеність у позиціонуванні. Цей метод використовується у сфері автономної навігації, зокрема мобільні роботи, безпілотні автомобілі та дрони. Також, метод добре працює в місцях де немає доступу до GPS, це робить його корисним для роботи в закритих приміщеннях. Загалом, маючи переваги та недоліки, він добре підходить для закритих приміщень, є універсальним та можна використовувати для різних типів роботів. Але і є деякі обмеження, може помилково співвідносити новий орієнтир з уже існуючим, що може спричинити неправильну побудову карти. Тому, EKF-SLAM є потужним інструментом в робототехніці, який може забезпечити точну побудову карти.

1.3 DP-SLAM: побудова карти за допомогою частинок

DP-SLAM (Dual-Particle Simultaneous Localization and Mapping)-це алгоритм одночасної локалізації, тобто робот одразу визначає своє місцезнаходження в просторі і може виконувати інші завдання, наприклад він може рухатися або будувати карту. Алгоритм використовує фільтр, за допомогою якого визначається поточний стан системи, навіть якщо вхідні дані неточні або нечіткі і він має назву – частинковий фільтр. Принцип роботи фільтру наступний: генеруються частинки, де може знаходитися об’єкт, далі частинка оновлюється, на основі руху об’єкту, потім порівнюються дані з сенсору та передбаченнями частинки, після цього частинки, які більш схожі з реальністю залишаються, а менші видаляються. Завдяки цьому фільтру, метод DP-SLAM тимчасово зберігає декілька різних варіантів карт, допоки робот не отримає більше даних для уточнення. Тому помилки майже не накопичуються і з часом карта стає більш точнішою. Головною перевагою методу є те, що він аналізує дані, які збираються з сенсорів, лише один раз і відразу ж будує точну карту. А отже, роботу не знадобиться спеціально виправляти помилки, які були накопичені в побудові карти і які можуть виникати через похибку сенсору. Цей алгоритм, як раз добре підходить для роботів, які можуть працювати у незнайомих місцях [6].

DP-SLAM, як згадувалось вище, використовує частинковий фільтр, для того, щоб одночасно визначити своє місцезнаходження та побудувати карту. Але все ж таки, якщо постійно зберігати карту, то це може зайняти багато обчислювальних ресурсів та пам’яті. Щоб запобігти використанню великих ресурсів, метод використовує глобальну сітку зайнятості з деревом спостережень для кожної комірки. Глобальна сітка зайнятості, представляє з себе карту як набір комірок і кожна з цих комірок містить інформацію про наявність перешкоди. Замість того, щоб кожна частинка мала власну копію карти, всі вони працюють із загальною картою. Далі оновлення карти зберігаються у вигляді історії зміни, а не створюють нові версії всієї карти.

Але вже для цього, кожна комірка має в собі не тільки одне єдине значення, а ціле дерево спостережень у якому зберігаються всі оновлення, які були зроблені цими частинками.

Крім цього, DP-SLAM використовує дерево зв'язків між частинками «див. рис. 1.3.1», воно допомагає відстежувати зміни з часом. Це дає дозвіл брати частинкам дані від тих кроків які були до цього і не копіювати всю карту наново. Використовуючи даний підхід, алгоритм працює швидше та використовує менше пам'яті, навіть якщо багато частинок [6].

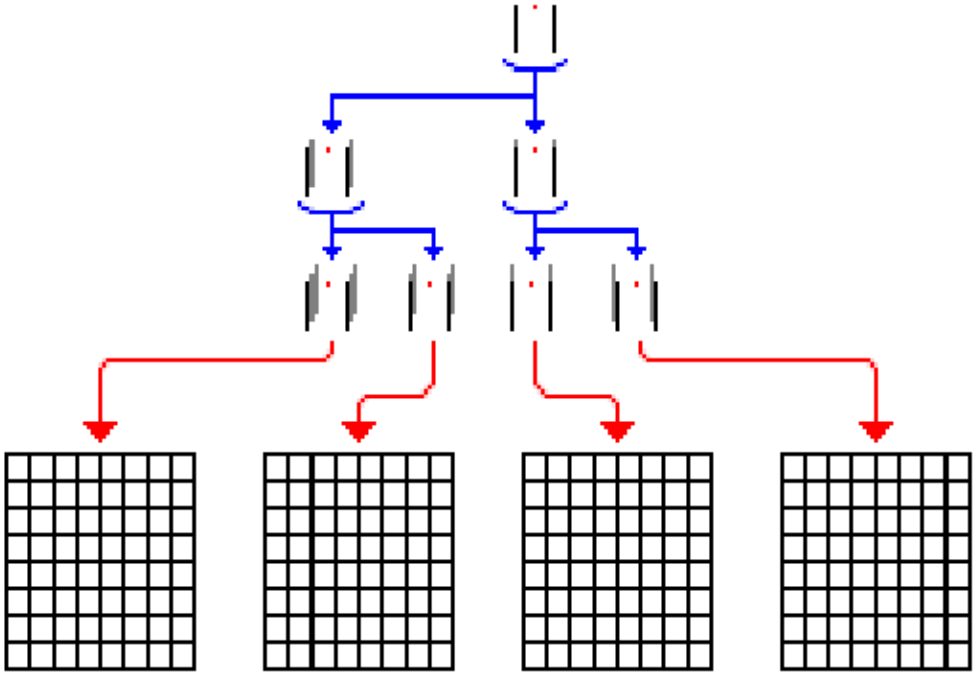

Рисунок 1.3.1-Дерево частинок та їхні відповідні карти

Рисунок «див. рис. 1.3.2» показує роботу DP-SLAM, який використовує частинковий фільтр. На рисунку можна побачити червону лінію, яка відображає траєкторію руху роботу у приміщенні. Чорні ознаки -це перешкоди, які потрібно оминути роботу. Синіми точками відображені перешкоди, а саме як бачить їх робот. Червона точка – це поточне місцезнаходження робота, а зелені – це можливе положення робота. Чорними лініями відображаються зв'язки між частинками [7].

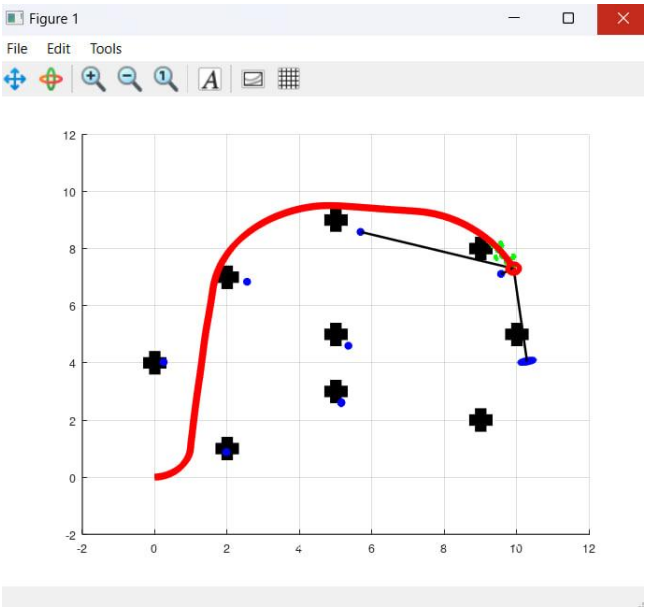

Рисунок 1.3.2-Робот впродовж опрацювання програми

Як видно з зображення, робот переміщається середовищем, вимірюючи за допомогою сенсорів відстані до перешкод, але все ж таки, деякі вимірювання мають похибку. Дану похибку можна побачити як синю точку. Сенсори визначають, які частинки краще збігаються з реальністю, і вони залишаються, а менш точні видаляються. Карта завжди оновлюється, а попередні дані постійно коригуються. Щоб не використовувати окремі карти для кожної з частинок, метод використовує загальну карту, де кожна з комірок містить дерево спостережень.

Отже, DP-SLAM є ефективним методом одночасної локалізації і побудови карти, який використовує частинковий фільтр для визначення місцезнаходження. Дерево спостережень дозволяє ефективно оновлювати карту, мінімізуючи використання пам’яті. Дерево зв’язків між частинками передає дані без повторення інформації. Завдяки цьому, алгоритм забезпечує точну побудову карти та ефективну роботу робота в незнайомому для нього середовища.

1.4 Алгоритм G-MAPPING. Робота фільтра Рао-Блеквела

1.4.1 Метод G-Mapping

На сьогоднішній день зростає впровадження автономних роботів у різні сфери, що підкреслює важливість навігаційних алгоритмів, які можуть адаптуватись до змін середовища. Одним з таких алгоритмів є G-mapping – це алгоритм одночасного свого місцезнаходження та одночасної побудови карти, який використовує фільтр частинок Рао-Блеквела. Завдяки фільтру, автономний робот будує карту невідомого середовища та одночасно визначає своє місцезнаходження, використовуючи дані з лазерних далекомірів [10].

Принцип роботи методу досить простий. Робот починає рух у невідомому середовищі і він ще не знає, які предмети або об’єкти його оточують, тому сам алгоритм створює багато варіантів карт і положень робота, їх і називають – частинками. Основним джерелом інформації, звідки надходять дані до робота – це лазерні далекоміри або дані з одометрії, це можуть бути колеса або датчики руху. Кожна з цих частинок – це той варіант, де може знаходитись робот і який вигляд має середовище. Чим більше робот буде сканувати середовище та рухатися, тим точнішими будуть дані для отримання. Фільтр частинок видаляє неточні варіанти, це ті варіанти які не відповідають реальному світу, а залишає одну найбільш точну карту і положення робота. Далі цей цикл повторюється декілька разів і робот вже знає де саме він знаходиться і починає будувати чітку карту середовища. Фільтр частинок постійно покращує точність визначення місцезнаходження, він коригує ймовірність для кожної частинки і враховує зміни в середовищі, це можуть бути рухомі об’єкти або перешкоди. Взагалі цей метод є досить потужним, але інколи його ефективність зменшується в середовищах де багато перешкод або в досить великих приміщеннях, так як там знаходиться значна кількість частинок і кожну з них треба обробити. Проте для підвищення результативності використовують метод зменшення числа частинок без втрати точності, це може бути метод перерозподілу частинок або ресемплінг.

Зображення «див. рис. 1.4.1.1» демонструє роботу алгоритму G-mapping, де на першому етапі робот збирає інформацію про середовище використовуючи сенсори. Здебільшого, дані є досить неточними і мають достатню кількість шуму. Так як робот ще не має точного уявлення про своє місцезнаходження, тому перша картинка демонструє нечіткий вигляд. На другому та третьому зображенні, робот продовжує рухатися далі, де алгоритм використовує інформацію про рух робота для подальшого оновлення карти. Карта стає точнішою і детальною, завдяки врахуванню попередніх даних про своє середовище.

Рисунок 1.4.1.1 – Початок роботи методу G-mapping

На кінцевому етапі «див. рис. 4.2» зображення показує карту приміщення. На сірому фоні можна побачити біле середовище, яке складається з різних коридорів та кімнат. Центральна частина зображення має прямокутну форму, від якої відходять декілька коридорів у різні напрямки. Деякі з цих коридорів мають нестандартну форму. Зображення демонструє як алгоритм G-mapping формує кінцеву карту середовища, використовуючи лазерний далекомір. Дана карта показує детальну структуру приміщення, яка дозволяє роботу орієнтуватися в просторі та уникати перешкоди під час навігації.

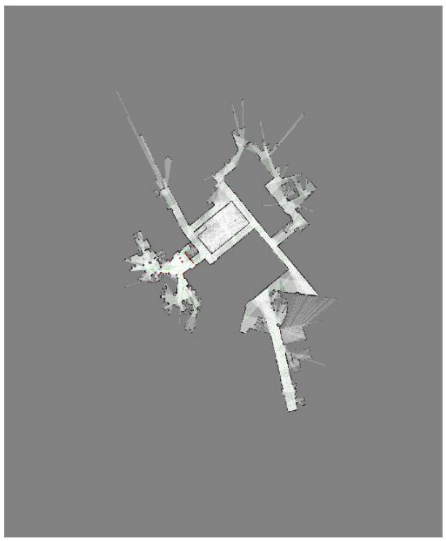

Рисунок 1.4.1.2 – Проміжна стадія роботи методу

Метод G-mapping має кілька переваг, а саме: висока точність завдяки використанню фільтра Рао-Блеквела, гнучкий у роботі з різними сенсорами, миттєве оновлення карти та масштабованість до великого приміщення. Але є і деякі недоліки, а саме: висока вибагливість до обчислювальних ресурсів, карта може погіршитися у складних умовах та залежність від точності сенсорів. А також, збільшення кількості часток для підвищення точності може призвести до зниження продуктивності методу.

Отже, автономні роботи широко застосовуються в різних сферах, що підкреслює важливість навігаційних алгоритмів, такого як G-mapping. Цей алгоритм використовує фільтр частинок Рао-Блеквела, яке дозволяє роботу одночасно знаходити своє місцезнаходження та будувати карту середовища. Робот в свою чергу збирає дані від сенсорів, створює безліч варіантів частинок і неточні видаляє, залишаючи більш реалістичну карту. Загалом, цей метод є потужним та ефективним інструментом для автономної навігації роботів у складних середовищах.

1.4.2 Робота фільтра Рао-Блеквела: основи та математичні формули

Фільтр Рао-Блеквела – це метод, який об’єднує в собі фільтр Калмана та частинковий фільтр, для того щоб вирахувати, де знаходиться робот. Даний фільтр, потрібно використовувати коли система містить в собі лінійні та нелінійні компоненти. Наприклад, лінійні частини системи оцінюються фільтром Калмана, так як він працює з рівняннями де зміни відбуваються передбачувано, а вже нелінійні частини оцінюються частинковим фільтром, оскільки потрібно перевіряти багато варіантів можливих станів. Представимо робота, який рухається по кімнаті, положення оцінюється за допомогою набору частинок, кожна з яких є можливою траєкторією, а фільтр Калмана уточнює рух. Використання даного фільтру дозволяє точніше визначити стан системи навіть якщо присутні неточні дані або зашумлені вимірювання. Метод можна використовувати в робототехніці для локалізації та побудови карт, корисний у випадках, коли звичайні фільтри справляються неефективно з великим обсягом змінних або швидкими змінами. Фільтр Рао-Блеквела розподіляє задачі, спочатку частинковий фільтр обчислює ймовірність розподілу стану, після фільтр Калмана оцінює компоненти які піддаються лінійній обробці. На рисунку «див. рис. 1.4.2.1» представлена схема роботи фільтра Рао-Блеквела, яка показує етапи генерації частинок, корекції даних, асоціювання, розподілу ваги і оновлення стану робота. Саме такий підхід використання дає високу ефективність алгоритму, тому що кожна частинка має власний фільтр Калмана, яка дозволяє одразу поєднати два методи. Основною проблемою фільтру є те, що йому потрібно мати достатню кількість частинок, щоб він працював точно, але і водночас їх не повинно бути багато, щоб не перевантажувались обчислення. Якщо частинок занадто мало, то фільтр втратить здатність оцінювати стан системи. Для того щоб цього не сталося, використовуються спеціальні методи: наприклад, об’єднуються подібні частинки або видаляються малоймовірні варіанти. Завдяки

зменшенню кількості частинок фільтр ефективно відстежує складні процеси, поєднуючи гнучкість частинкового фільтра та точність фільтра Калмана [11].

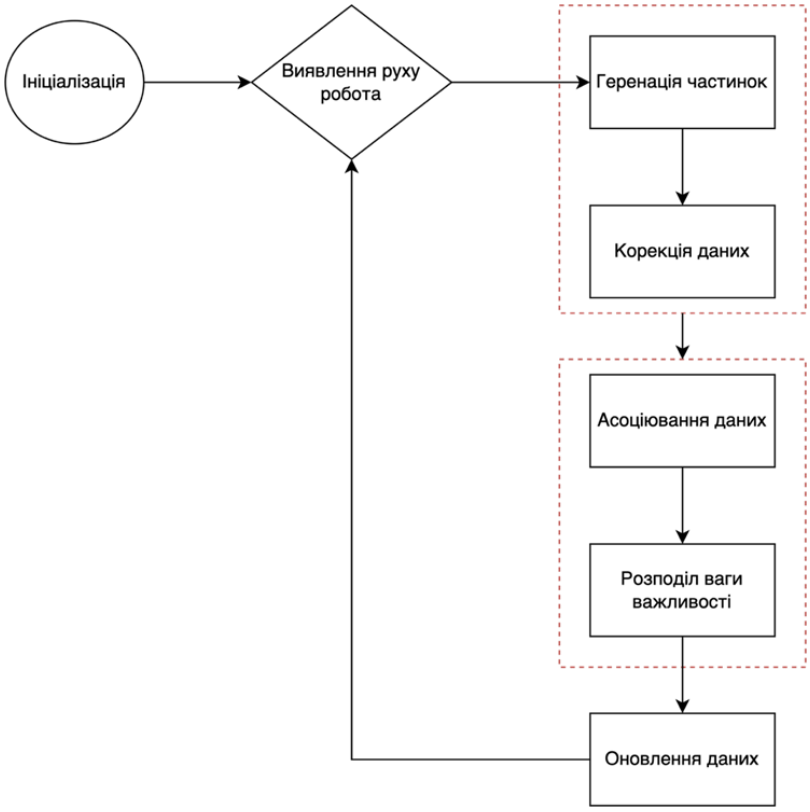

Рисунок 1.4.2.1 – Алгоритм роботи фільтру Рао–Блеквела

Фільтр Рао-Блеквела, використовує комбінацію частинкового фільтра та фільтра Калмана, для оцінки стану системи. Щоб зрозуміти, яке це відбувається математично, розглянемо рівняння, які описують процес оновлення стану та оцінки його точності у фільтрі Калмана:

$$\widehat{x_k} = F_k \widehat{x_{k-1}} + B_k u_k + w_k \qquad (1.4.1)$$

де $\widehat{x_k}$ – прогнозована векторна величина на момент часу k. Це те, де знаходиться система у часі k;

F_k – матриця переходу станів, яка описує, як змінюється стан;

$\widehat{x_{k-1}}$ – оцінка стану на попередній момент часу k-1;

B_k – матриця керування, яка визначає, як керуючі дії впливають на стан;

u_k – вектор керування на момент часу k;

w_k – величини, яка враховує невизначеність або неточності моделі;

Формула 4.1 описує прогноз стану в алгоритмі фільтра Калмана, тобто вона дозволяє передбачити, яким буде стан системи в наступний момент часу.

$$P_k = F_k P_{k-1} F_K^T + Q_k \tag{1.4.2}$$

де P_k – коваріаційна матриця похибки, яка показує, наскільки точно ми знаємо стан системи;

P_{k-1} – коваріаційна матриця похибки оцінки стану на попередньому кроці;

F_K^T – транспонована матриця F_k;

Q_k – коваріаційна матриця шуму процесу, моделює невизначеність у динамічній моделі;

Формула 4.2 перевіряє те, наскільки робот впевнений у своєму прогнозі. Вона використовується у фільтрі Калмана. Чим менше значення P_k тим точніше ми знаємо де знаходиться робот.

$$K_k = P_k H_k^T (H_k P_k H_k^T + R_k)^{-1} \tag{1.4.3}$$

де K_k – калманівський коефіцієнт, визначає вагу, з якою нове вимірювання впливає на оновлення стану;

H_k^T – транспонована матриця H_k;

H_k – матриця спостереження, яка відображає як стан пов’язаний із вимірюванням;

R_k – коваріаційна матриця шуму вимірювання, відображає невизначеність сенсорних даних;

Формула 4.3 визначає наскільки ми можемо довіряти новому вимірюванню, порівняно з попередньою оцінкою.

$$\widehat{x_k} = \widehat{x_k} + K_k(z_k - H_k\widehat{x_k}) \quad (1.4.4)$$

де z_k – виміряне значення на момент часу k, дистанція отримана з лазерного далекоміра;

Формула 4.4 коригує попередньо прогнозоване значення стану $\widehat{x_k}$ з урахуванням нових даних z_k які отримуються від лазерного далекоміра.

$$P_k = (I - K_k H_k)P_k \quad (1.4.5)$$

де I – одинична матриця того ж розміру, що й P_k. Використовується для збереження розмірності при обчисленні.

Формула 4.5 оновлює впевненість у нашій оцінці після отримання нових вимірювань.

Отже, фільтр Рао-Блеквела є ефективним методом для оцінки стану системи, що містить як лінійні так і нелінійні компоненти. Поєднуючи в собі частинковий фільтр та точність фільтра Калмана, метод може ефективно обробляти велику кількість можливих станів, зменшуючи обчислювальне навантаження. Головною перевагою такого підходу є здатність точно забезпечити точну локалізацію робота навіть за умов шумних даних.

1.5 Інші способи картографування простору

Даний матеріал взятий у інших авторів, зокрема А. А. Проценка та В. Г. Іванова, які у своїй роботі описують класичні методи картографування для мобільних автономних роботів [8]. Автори розглядають наступні методи.

1) Метод клітинної декомпозиції працює наступним чином: простів, в якому рухається робот ділиться на маленькі клітини. Кожна з цих клітин може бути вільною, де робот зможе проїхати, або зайнятою, де є перешкоди. Потім у вільних клітинах вибираються ключові точки, за якими вже будується шлях робота [12].
2) Метод штучного потенційного поля, заснований на ідеї, що робот рухається у силовому полі, де кінцева точка його шляху притягує, а перешкоди відштовхуються. Саме так, робот самостійно коригує свій рух, щоб уникати перешкоди і прямувати до цілі. Але цей метод має деякі недоліки, а саме ситуації, коли робот може застрягти в точці, де сили відштовхування та притягання врівноважуються і він не зможе продовжити рух до цілі. Щоб вирішити цю проблему, дослідники вдосконалили метод, що враховує не тільки положення робота, а ще його швидкість щодо цілі та перешкод, це дозволяє швидше реагувати на зміни в середовищі [12].
3) Вибіркові методи використовують випадковий вибір точок у просторі для пошуку маршруту. Вони не можуть гарантувати, що рішення знайдеться у всіх випадках, але якщо воно існує, то алгоритм його знайде, якщо працюватиме довго. Відомий дослідник, Жан-Клод Латомб, розробив алгоритм, який поєднує випадковий вибір з методом штучного потенційного поля, що допомагає зменшити вплив локального мінімуму. Якщо робот застрягає в деяких точках, він може пробиватися через перешкоди або використати метод Монте-Карло для вибору нового шляху. Основні підходи у вибіркових

методах включають методи на основі дорожніх карт та швидко-досліджуючих випадкових дерев [12].

4) Мережі проміжних цілей – це методи, які допомагають поділити складну задачу пошуку на декілька етапів, вони шукають не тільки шлях від початкової точки і до кінцевої, а ще й проміжні точки, через які має пройти робот. Окрім того, щоб шукати шлях від початку і до кінця, робот повинен зупинятися в проміжних точках до того поки не досягне кінцевої мети [12].

Отже, у результаті аналізу методів планування шляху для роботів, зроблено наступні висновки. Метод клітинної декомпозиції дозволяє ефективно розділити простір на вільні та зайняті ділянки, що забезпечує чітке формування шляху робота через визначення ключових точок. Метод штучного потенційного поля є досить простим у виконанні і дозволяє робота самостійно коригувати свій рух, але є і недоліки, такі як застрягання у точках локальних мінімумів. Вибіркові методи на основі випадкового вибору точок не можуть гарантувати знаходження шляху у всіх випадках, але забезпечують ймовірність його знаходження. Мережі проміжних цілей допомагають поділити складний шлях на декілька етапів, забезпечуючи пошук не тільки кінцевої точки, але й послідовних проміжних цілей, що дозволяє роботу рухатися до цілі в декілька етапів. Загалом, всі методи мають як переваги так і недоліки, але їх використання залежить від специфіки задачі та середовища в якому працює робот.

2. ПРАКТИЧНА ЧАСТИНА

2.1 Практична реалізація методу G-Mapping

У зробленій програмі реалізовано віртуальну сцену, де симулюється процес автономної навігації робота у просторі, використовуючи мову програмування Python. Це високорівнева, інтерпретована, об’єктно-орієнтована мова програмування, яка має простий синтаксис, велику кількість бібліотек та чудово підходить для розробки симуляцій, візуалізації даних та робототехнічних застосунків [14]. Мова програмування обрана Python, через його ефективність у побудові прототипів, простоту інтеграції з графічними бібліотеками та гнучкість роботи з піксельними зображеннями. Насамперед, у програмі реалізується покроковий алгоритм дослідження середовища з візуальним відображенням карти – для цього використовується бібліотека Pygame, яка поєднується з Python і дозволяє працювати з піксельними даними зображення. Ця бібліотека використовується для симуляцій та розробки двовимірних ігор, яка забезпечує роботу з графікою, створення вікон та обробка подій з клавіатури [15]. В даному проєкті Pygame використовується для завантаження растрової карти (.bmp зображення), візуалізації поточної позиції робота, маркування перешкод, вільних зон, а також для обробки клавіш керування програмою, таких як запуск, кнопкою S, вихід кнопкою Q та перемикання в повноекранний режим кнопкою F. Власне середовище побудовано у вигляді сітки пікселів, де кожен піксель має певний колір, що визначає його тип, а саме вільну зону, перешкоду, межі сцени або центр цільової області.

У програмі віртуальне середовище створюється дуже просто – замість складного 3D-світу завантажується звичайна карта у форматі .bmp. Ця картинка і є картою, де кольори пікселів показують, які зони є вільними для руху, а де розташовані перешкоди. Даний формат обраний у програмі через декілька технічних переваг, які добре підходять для симуляції навколишнього середовища. По-перше, на відміну від форматів PNG або JPEG, формат BMP

не спотворює кольори. Це важливо, адже в програмі, робот бачить світ за кольорами пікселів, і будь-яка зміна кольору може порушити логіку роботи алгоритму [16]. По-друге, бібліотека Pygame, яка використовується для візуалізації, добре підтримує BMP-файли, функція pygame.image.load() швидко та без проблем завантажує зображення, дозволяючи одразу звертатися до кольорів пікселів. По-третє, BMP не вимагає додаткового розпакування, тож обробка карти відбувається швидше. Це важливо в циклі сканування середовища, який виконується багато разів за секунду.

В цій симуляції карта виводиться на екран автоматично під час виконання програми. Робот досліджує простір навколо себе, і щойно він побачить деяку ділянку (наприклад, вільне місце або перешкоду), ця інформація відображається на екрані. Для цього використовується функція screen.set_at() з бібліотеки pygame, яка змінює колір конкретний пікселів на екрані, тим самим поступово показуючи користувачу, яку частину карти робот уже обстежив.

Карта, де проводиться дослідження – це фіксоване зображення, яке показується як вступний екран під час запуску. Вона не створюється за допомогою 3D-графіки, а є вже двовимірною мапою з чітко заданими кольорами, які позначають різні об’єкти. Такий підхід дозволяє точно й швидко відстежувати, як робот взаємодіє з оточенням.

Також, використовується механізм відстежених пікселів, звичайний словник – visited, який зберігає кількість разів, коли робот побував у конкретному місці. Це дозволяє мінімізувати повторне відвідування вже відомих ділянок та допомагає вибирати наступний напрямок руху, що сприяє ефективнішому дослідженню. Ще програма підтримує зміну швидкості руху робота та відстежує кількість вільних та досліджених пікселів, що дозволяє оцінити відсоток завершеності дослідження. Коли робот повністю відсканував карту, застосунок автоматично завершує симуляцію.

Звідси випливає, що для створення віртуальної сцени у даній програмі використовується двовимірне зображення як мапа середовища, а бібліотека

Pygame надає усі необхідні інструменти для роботи з графікою, обробкою подій та оновленням екрану. Усі обчислення, перевірка на перешкоди, розрахунок зон огляду та керування логікою дослідження, реалізовані мовою Python.

Для створення середовища використовується команда pygame.image.load, яка завантажує зображення карти. Щоб імітувати дослідження невідомої території використовується ще одна матриця – дисплейна мапа. Вона поступово оновлюється в міру того, як робот бачить нові ділянки за допомогою сканування. Програмі імітує огляд у різних напрямках до зіткнення з перешкодою або краєм карти. Це дозволяє перевіряти ідеї побудови карти та дослідження без робототехнічних систем.

Отже, застосунок реалізовано повністю на мові програмування Python з використанням бібліотеки Pygame, що виступає як головний інструмент для побудови віртуальної сцени, графічної візуалізації процесу дослідження середовища роботом, а також для забезпечення базової взаємодії з користувачем. Використання лише двох ключових технологій дозволило сконцентруватися на логіці руху, виявлення перешкод, побудови карти та ухвалення рішень. Завдяки такому підходу, застосунок легко модифікується, можна змінювати поведінку робота, колірну схему та розміри як карти так і робота. Обрана архітектура є простою, проте вона достатньо гнучка для реалізації двовимірної симуляції автономного мобільного робота. Карта середовища задається у вигляді растрового зображення, що дозволяє легко її замінити або змінити без необхідності втручання в код – це значно спрощує тестування різних сценаріїв.

2.2 Інструменти для створення віртуальних сцен

2.2.1 Призначення графічного інтерфейсу

Графічний інтерфейс користувача – це невід'ємна частина створення віртуальної сцени, оскільки він забезпечує зрозумілий та зручний спосіб взаємодії з користувачем. У контексті віртуального моделювання він дозволяє візуалізувати об'єкти, прості, траєкторію руху робота та стан карти середовища у реальному часі. Завдяки графічному інтерфейсу, користувач має змогу не тільки спостерігати за моделювання, а й впливати на нього без втручання у програмний код. У реалізованій системі, графічний інтерфейс виконує наступні функції: виведення завантаженої карти навколишнього середовища, відображення траєкторії руху робота, візуалізацію зон видимості та виявлених об'єктів, стан завершення процесу побудови карти, можливість перемикання між повноекранним та віконними режимами, а також очікування на запуск моделі. Окрім цього, інтерфейс дає змогу стежити за динамікою змін під час симуляції, миттєво реагувати на помилки або непередбачувані ситуації, зокрема такі як застрягання робота в оточенні перешкод, спроба вийти за межі карти або неправильне налаштування параметрів руху. Інтерфейс також дозволяє зберігати проміжні результати моделювання, наприклад, поточне положення робота, стан побудованої карти та виявлені об'єкти. Крім базових функцій відображення, GUI підтримує візуалізацію статистичних даних у режимі реального часу, таких як відсоток відсканованої карти. Це дозволяє оцінювати ефективність роботи алгоритму навігації та приймати рішення щодо коригування стратегії руху. Графічний інтерфейс сприяє глибшому розумінню взаємодії між роботом та середовищем, дозволяючи оцінити ефективність алгоритмів навігації. Гнучка архітектура інтерфейсу дає змогу розширювати його функціональність без суттєвого переписування коду. Отже, GUI відіграє ключову роль як у відлагодженні програми, так і у демонстрації результатів експерименту.

2.2.2 Завдання, які виконує створення інтерфейсу

Формування інтерфейсу в цьому проєкті вирішує низку важливих задач, спрямованих на забезпечення ефективної взаємодії користувача з системою. Інтерфейс дозволяє відображати початковий стан карти, яка містить невідомі ділянки та поступово оновлюється в міру руху робота. Також реалізовано графічне сканування навколишнього середовища – показуються досліджені області за допомогою сенсорного модулю, що імітується в програмному середовищі. Інтерфейс забезпечує візуалізацію руху робота з урахуванням перешкод і зони огляду, дозволяючи стежити за його траєкторією. Важливим елементом є індикація прогресу, яка надає інформацію про відсоток дослідженої карти. Окрему роль відіграє імітація завантаження карти, що демонструє процес ініціалізації, очікування початку та завершення дослідження. Крім того, інтерфейс реагує на дії користувача, зокрема забезпечує підтримку клавіш для керування режимами виведення: клавіша F перемикає повноекранний режим, Q виконує вихід з програми, а S запускає процес дослідження. Усі ці функції реалізовані відповідно до принципів простоти використання та гнучкої взаємодії з графічною частиною системи.

2.2.3 Переваги використання інтерфейсу та бібліотеки Pygame

Для реалізації інтерфейсу в проєкті було обрано бібліотеку Pygame, оскільки вона зручно поєднується з мовою Python і є простою у використанні. Вона базується на технології SDL(Simple DirectMedia Layer) і дозволяє швидко виводити графіку – малювати пікселі, прямокутники, завантажувати зображення та працювати з кольорами без зайвих ускладнень. Завдяки тому, що програма написана мовою Python, не виникає проблем із сумісністю чи необхідністю поєднувати різні мови або інструменти. Це особливо зручно для інтеграції з основною логікою програми, як-от сканування середовища, переміщення робота або обробка перешкод. Ще однією перевагою є підтримка подій від клавіатури, що дуже доречно для керування симуляцією – наприклад,

запуск або вихід з програми. Окрім цього, Pygame є кросплатформеним, тобто працює на Windows, Linux та macOS без додаткових налаштувань. Загалом, у поєднанні з Python, ця бібліотека дозволяє створювати прості, але функціональні графічні симуляції, які чудово підходять для наукових чи навчальних проєктів.

2.2.4 Обґрунтування вибору мови програмування

В проєкті для розробки було обрано мову програмування Python, а на це є декілька логічних причин. По-перше, Python має дуже простий та зрозумілий синтаксис, тому код легко читати, змінювати та підтримувати. По-друге, ця мова має велику кількість бібліотек, які особливо корисні для наукових задач, обробки графіки та зображень, що якраз потрібно в межах цього проєкту. Крім того, Python дозволяє досить швидко реалізовувати різні алгоритми моделювання, не витрачаючи багато часу на низькорівневі технічні деталі. Завдяки цьому він особливо добре підходить для експериментів у сфері робототехніки, включно з візуалізацією таких методів побудови карти, як GMapping. Усе це робить Python ідеальним вибором для створення гнучкої, ефективної та наочної симуляції SLAM-систем.

2.2.5 Недоліки бібліотеки Pygame

Попри всі переваги, бібліотека Pygame має і свої недоліки, про які варто зазначити, особливо якщо проєкт розрахований на складніші візуалізації або великі обсяги даних. Одним із основних обмежень є те, що Pygame не використовує апаратного прискорення графіки, як це роблять, наприклад OpenGL або WebGL. Через це продуктивність програми може знижуватись при обробці великої кількості графічних об’єктів або при високій частоті оновлення зображення на екрані. Також створення складних елементів користувацького інтерфейсу, таких як інтерактивні меню, кнопки, таблиці або

випадаючі списки, у Pygame непередбачене. Усе це потрібно розробляти самостійно, що збільшує обсяг роботи й ускладнює підтримку коду. Ще одним обмеженням є те, що Pygame підтримує лише базову 2D – графіку і не призначений для створення тривимірних сцен чи візуалізацій – у цьому плані він значно поступається більш потужним і гнучким рушіям. Проте для даного проєкту ці недоліки не є критичними, оскільки візуалізація зосереджена виключно на простій двовимірній карті, представленій прямокутниками, а складні елементи UI не використовуються. У результаті Pygame повністю задовольняє вимоги проєкту, не зважаючи на згадані обмеження.

2.2.6 Роль Figma у проєкті

Для проєктування зовнішнього вигляду інтерфейсу використовується Figma – зручний онлайн інструмент, який дає змогу створювати макети прямо в браузері без потреби щось встановлювати. Вона стала в пригоді ще на початковому етапі роботи, коли потрібно уявити, як виглядатиме програма. За допомогою Figma створено попередні схеми розміщення елементів – наприклад, де якого розміру буде карта, яких розмірів будуть перешкоди та де саме вони будуть розташовані. Також у Figma зручно експериментувати з кольорами – одразу визначено, яким кольором будуть перешкоди, колір карти, вільні зони й ті ділянки які ще не досліджені. Це все суттєво спростило подальшу реалізацію в коді, оскільки структура інтерфейсу продумана заздалегідь. Figma ще зручна й тим, що дозволяє легко ділитися макетами з іншими. Інтерфейс інтуїтивний, а результат можна без проблем зберегти у форматі PNG для подальшого використання. Загалом, Figma значно полегшила процес планування й допомогла зробити інтерфейс більш логічним та зручним.

2.2.7 Формат зображень: чому саме BMP?

В проєкті використовується формат BMP для збереження карти середовища, бо він зберігає кожен піксель без стиснення. Це дуже зручно, адже дозволяє точно визначити кольори, не боячись втрати якості чи спотворення відтінків. Структура цього формату проста, тому можна без зайвих складнощів зчитувати й аналізувати кольори пікселів – наприклад, визначити, де на карті чорні області, ними позначені перешкоди, а де жовті, вони відповідають вільному просто вільному простору. Інші формати, як-от PNG, можуть стискати зображення або автоматично змінювати кольори, що ускладнить точний аналіз. До того ж, завдяки прямому доступу до піксельних даних, можна швидко перетворювати карту у матрицю значень. Це значно спрощує інтеграцію віртуального середовища з інструментами аналізу та планування маршруту. Формат BMP підтримується більшістю графічних бібліотек, тож його легко обробляти у Python. Так, BMP займає більше місця, але для локального використання під час симуляцій це не створює проблем, тому цей варіант виявився найзручнішим.

2.2.8 Візуалізація інструменту конструювання віртуальних сцен

Візуалізація є ключовою складовою будь-якого інструменту, що призначений для створення або аналізу віртуального середовища. У межах реалізованої програми за допомогою бібліотеки Pygame було створено зручний візуальний інтерфейс, що дозволяє наочно спостерігати за роботою алгоритму побудови карти. Поверхня вікна програми відображає карту середовища у вигляді піксельного зображення, де кожен піксель має визначне призначення та колір, що відповідає стану конкретної області навколишнього простору.

У початковому стані вся карта має сірий фон, що символізує невідомі ділянки – ті зони, які ще не були обстежені віртуальним роботом. Як тільки скануючий робот починає рухатися та виконувати аналіз простору, на

зображенні починають з’являтися кольорові маркери. Зокрема, жовтий колір означає, що дана зона вже була просканована та є вільною для пересування. Така візуалізація дозволяє легко відрізнити відомі зони від тих, які ще не потребують аналізу.

Положення самого робота на сцені позначається синім кольором, що динамічно змінюється в процесі руху. Кожна зміна координат відображається в реальному часі, що дає змогу користувачу відслідковувати процес побудови карти. Окремо варто зазначити про перешкоди, які мають чорне забарвлення. Вони є статичними об’єктами, які не можуть бути подолані роботом. Їх наявність у сцені моделює реальні умови пересування автономного агента, де важливо враховувати наявність стін, предметів чи інших перешкод.

На рисунку 2.2.8.1 показано початковий стан програми. Вся сцена зафарбована сірим кольором – робот ще не здійснив жодного кроку, і жодна зона не була обстежена. Це початкова позиція, з якої починається сканування.

Рисунок 2.2.8.1 – Початковий стан програми до сканування

На рисунку 2.2.8.2 зображено проміжний стан карти. У верхній частині видно область, яка вже пройшла сканування. Жовті пікселі позначають вільний простір, а синя точка – поточу позицію робота.

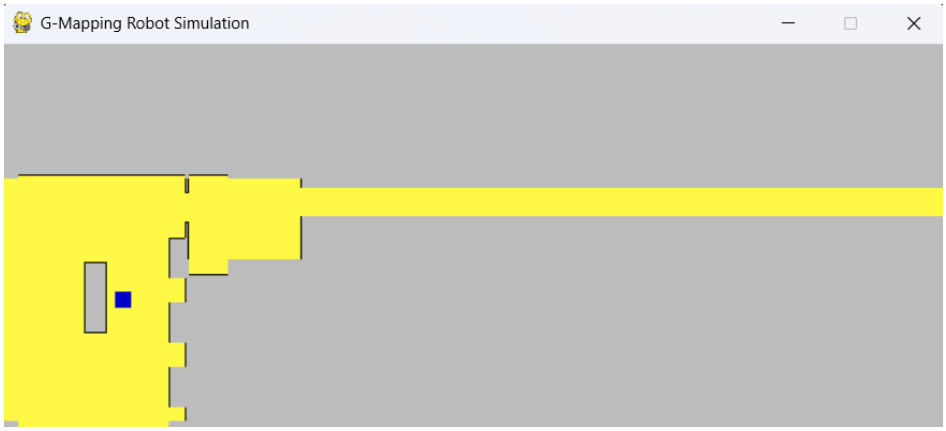

Рисунок 2.2.8.2 – Візуалізація процесу сканування віртуальним роботом

Рисунок 2.2.8.3 демонструє фінальний стан карти після завершення сканування. Уся сцена поділена на відомі (жовті) та невідомі (сірі) ділянки. Робот відсканував всю карту, а також зони які виключені з обстеження через наявність перешкод.

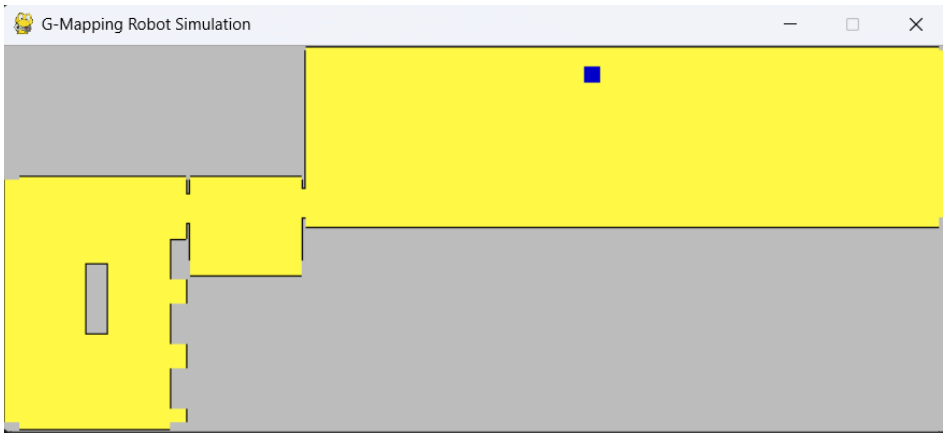

Рисунок 2.2.8.3 – Повністю збудована карта з вільними зонами та перешкодами

Крім візуалізації самої карти, було реалізоване графічне представлення інтерфейсу користувача. Для цього використовувалося середовище Figma, що дало змогу створити попередній макет програми. На рисунку 2.2.8.4 зображено розроблений прототип, який визначає загальний стиль, кольорову палітру та розташування функціональних елементів, таких як кнопки управління, заголовки та основна область відображення карти.

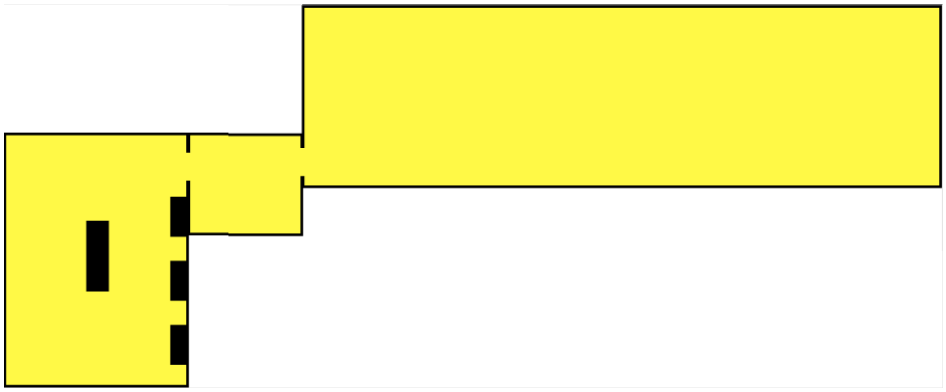

Рисунок 2.2.8.4 – Прототип графічного інтерфейсу користувача

У результаті, реалізована візуалізація інструменту повністю відповідає поставленому завданню та ефективно відображає процес побудови карти віртуального середовища. Кожен елемент на екрані – від початкової сцени до цілі, перешкод та зон, що були просканові – представлений зрозуміло та логічно. Попередній макет інтерфейсу, створений у Figma, допоміг сформувати зручне та продумане середовище взаємодії з користувачем. Усе це свідчить про те, що засоби реалізації були підібрані вірно, функціонал програми охоплює необхідна задачі, а побудована система є зручною та наочною для використання.

2.2.9 Опис основних функцій програми

У підпункті розглянуто ключові функції, які формують логіку роботи програмного засобу для побудови карти віртуального середовища з використанням бібліотеки Pygame. Наступні функції визначають основні етапи взаємодії з картою, пошуку шляху та обробки подій.

1) load_map(path) – ця функція завантажує зображення карти з BMP-файлу за заданим шляхом. Вона формує:
   - hidden_map: масив з кольорами кожного пікселя
   - display_map: початкову карту з «невідомими» значеннями
   - width і height: розмір карти
2) find_start_position(hidden_map, width, height) – ця функція повертає координати початкової позиції робота. У поточній реалізації задано статичне положення. Передбачено довільне положення робота.
3) scan_obstacles(hidden_map, width, height, x, y) – функція виконує симульоване сканування у чотирьох напрямках: північ, південь, захід, схід. Рух відбувається від меж квадрата робота до першої перешкоди, вільної зони або цілі.
4) can_move(matrix, width, height, x, y) – функція перевіряє, чи може робот переміститися у нову позицію (x, y). Перевірка відбувається для всіх пікселів квадрата, що моделює робота, на предмет виходу за межі карти або наїзду на перешкоду.
5) wait_any_key_to_start(screen, back) – відображає стартовий екран із фоновим зображенням і чекає, поки користувач натисне клавішу S для старту, F для повноекранного режиму або Q для виходу.
6) check_for_quit(size) – слідкує за натисканням клавіш або подією закриття вікна. Дозволяє вийти з програми або змінити режим відображення.
7) main(path_to_map, back) – ця функція є центральним елементом логіки всієї програми. Вона ініціалізує Pygame, завантажує карту та

визначає початкову позицію, організовує відображення процесу сканування карти, керує переміщенням робота відповідно до кількості відвідувань координат, стежить за прогресом і завершує роботу після повного сканування або в разі застрягання. Тобто, вона об’єднує всі допоміжні функції та забезпечує послідовне виконання дій для побудови карти.

Усі реалізовані функції працюють узгоджено та забезпечуються виконання поставленого завдання – поступове сканування віртуального середовища за допомогою умовного робота, що рухається по карті, уникаючи перешкоди і зчитуючи інформацію про навколишній простір. Програма враховує можливість застрягання, обробляє винятки, надає зручний візуальний супровід і дозволяє аналізувати динаміку сканування через виведення відсотка оброблених пікселів. Це підтверджує правильність та ефективність реалізації системи. Отже, описані функції демонструють структуровану організацію коду, сприяють модульності та гнучкості програмного засобу. Їх подальша оптимізація або розширення відкриває перспективи для адаптації системи для нових форматів карт, складніших алгоритмів або інтеграції з реальними сенсорними системами.

ВИСНОВКИ

Ефективне управління мобільними роботами вимагає ретельного аналізу моделей управління, вибору значущих точок на маршруті та їх оптимізації через селекцію. Кваліфікаційну роботу спрямовано на визначення проблеми SLAM за допомогою алгоритму G–Mapping і розгляду етапів в процесі симуляції поведінки робота у різних станах навколишнього середовища.

Під час виконання роботи були проаналізовані існуючі SLAM - методи, а також розглянуті особливості роботи вдосконалених алгоритмів фільтрації, які застосовуються у реалізації G–Mapping SLAM.

Було створено інструмент для побудови різних віртуальних сцен навколишнього середовища. Проведено аналіз бібліотек і технологій для розробки такого рішення, а також реалізовано інтерфейс, який дозволяє отримувати дані з цього інструменту у форматі, зручному для подальшої обробки симулятором навколишнього середовища ROS.

Побудований інструмент має перспективи подальшого розвитку: збільшення можливостей функціоналу та вдосконалення користувацького інтерфейсу. Практична складова роботи показує можливість покращення досвіду взаємодії з елементами робочого поля конструктора.

За результатами даної роботи опубліковано тези участі у XVIII Всеукраїнської науково практичної WEB конференції аспірантів, студентів та молодих вчених та XXII всеукраїнська конференція студентів і молодих науковців «Інформатика, інформаційні системи та технології».

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. https://www.researchgate.net/publication/234081012_Visual_Simultaneous_Localization_and_Mapping_A_Survey (дата звернення 19.03.2025)
2. https://openarchive.nure.ua/server/api/core/bitstreams/d64cf19b-ebcf-4413-a068-51d4b3963f5f/content (дата звернення 26.03.2025)
3. https://uk.mathworks.com/discovery/slam.html (дата звернення 01.04.2025)
4. https://www.flyability.com/blog/simultaneous-localization-and-mapping (дата звернення 07.04.2025)
5. https://www.researchgate.net/publication/274882577_A_simple_and_efficient_implementation_of_EKF_based_SLAM_relying_on_laser_scanner_in_complex_indoor_environment (дата звернення 15.04.2025)
6. https://users.cs.duke.edu/~parr/dpslam/ (дата звернення 17.04.2025)
7. https://openslam-org.github.io/dpslam.html (дата звернення 20.04.2025)
8. А.А. Проценко, В.Г. Іванов. Класичні методи планування шляху для мобільних роботів. DOI: https://doi.org/10.26906/SUNZ.2019.3.143 // Збірник наукових праць Полтавський національний технічний університет імені Юрія Кондратюка. – Полтава ,2019.- Том 3 № 55 . – С. 143 – 151
9. Ю.А.Ніцук, О.М.Семчак, Шаріпова І.В. Визначення шляхів зменшення похибок розрахунків координат бортовими ЕОМ автономного рухомого об’єкту для реалізації алгоритмів SLAM навігації // Збірник наукових праць Житомирський військовий інститут імені С. П. Корольова. – Житомир : ЖВІ, 2020.- № 27 (4). – С. 38 – 49
10. https://www.researchgate.net/publication/384216516_Autonomous_Navigation_Through_Gmapping-Based_SLAM_A_Comprehensive_Evaluation (дата звернення 01.05.2025)
11. https://www.researchgate.net/publication/220058248_The_Rao_Blackwellized_Particle_Filter_A_Filter_Bank_Implementation (дата звернення 03.05.2025)
12. https://journals.nupp.edu.ua/sunz/article/view/1569/1291 (дата звернення 05.05.2025)

13. Невлюдов І., Новосьолов С., Сухачов К. (2023). Метод одночасної локалізації та картування для побудови 2.5d карт навколишнього середовища за допомогою рос. інноваційні технології та наукові рішення для галузей , (2 (24), 145–160. https://doi.org/10.30837/ITSSI.2023.24.145
14. https://docs.python.org/uk/3/faq/general.html#what-is-python (дата звернення 15.05.2025)
15. https://pygame.readthedocs.io/en/latest/1_intro/intro.html (дата звернення 17.05.2025)
16. https://www.adobe.com/ua/creativecloud/file-types/image/raster/bmp-file.html (дата звернення 23.05.2025)
17. Лісовський М.М., Шаріпова І.В. Розробка системи орієнтування та будування карти навколишнього середовища для автономного робота // Комп'ютерні інтелектуальні системи та мережі. Матеріали XVIII Всеукраїнської науково практичної WEB конференції аспірантів, студентів та молодих вчених (25-27 березня 2025 р.). – Кривий Ріг: Криворізький національний університет, 2025. – с.281-283 – Режим доступу: https://sites.google.com/view/kicm/
18. Лісовський М.М., Шаріпова І.В. Розробка системи орієнтування та будування карти навколишнього середовища для автономного робота за допомогою G–MAPPING SLAM // Двадцять друга всеукраїнська конференція студентів і молодих науковців «Інформатика, інформаційні системи та технології» - Одеса, 2025, с. 21-23

ДОДАТОК А

Код програмного засобу для побудови карти віртуального середовища

```
import pygame
from pygame.display import toggle_fullscreen
from pygame.locals import *
import random
import sys

# TODO: change to hex
OBSTACLE_COLOR     = (0, 0, 0)
FREE_COLOR         = (255, 249, 70)
OUTSIDE_COLOR      = (255, 255, 255)
ROBOT_COLOR        = (0, 0, 200)
TARGET_CENTER_COLOR= (32, 126, 233)
TARGET_AREA_COLOR  = (0, 191, 255)
UNKNOWN_COLOR      = (0xBC, 0xBC, 0xBC)

DIRECTIONS = {
    'north': (0, -1),
    'south': (0, 1),
    'west': (-1, 0),
    'east': (1, 0)
}

ROBOT_SIZE = 12


def load_map(path):
    image = pygame.image.load(path)
    width, height = image.get_size()
    hidden_map = [[image.get_at((x, y))[:3] for x in
range(width)] for y in range(height)]

    display_map = [[UNKNOWN_COLOR for i in row] for row in
hidden_map]

    return image, hidden_map, display_map, width, height


def find_start_position(hidden_map, width, height):
    return 145, 124         # це конкретне місце
    # це рандомне місце
    # for attempt in range(5000):
    #     x = random.randint(0, width - ROBOT_SIZE)
    #     y = random.randint(0, height - ROBOT_SIZE)
    #     if all(hidden_map[y + dy][x + dx] == FREE_COLOR
    #            for dx in range(ROBOT_SIZE) for dy in
range(ROBOT_SIZE)):
    #         return x, y
```

Лістинг А.1 – Програмна реалізація сканування карти

```
    # for y in range(height - ROBOT_SIZE + 1):
    #     for x in range(width - ROBOT_SIZE + 1):
    #         if all(hidden_map[y + dy][x + dx] == FREE_COLOR
    #                for dx in range(ROBOT_SIZE) for dy in
range(ROBOT_SIZE)):
    #             return x, y
    # raise ValueError("No free start position found")


def scan_obstacles(hidden_map, width, height, x, y):
    scanned = []

    for dir_name, (dx, dy) in DIRECTIONS.items():
        for i in range(ROBOT_SIZE):
            if dir_name == 'north':
                sx, sy = x + i, y
            elif dir_name == 'south':
                sx, sy = x + i, y + ROBOT_SIZE - 1
            elif dir_name == 'west':
                sx, sy = x,     y + i
            else:  # east
                sx, sy = x + ROBOT_SIZE - 1, y + i

            cx, cy = sx, sy
            while 0 <= cx < width and 0 <= cy < height:
                scanned.append((cx, cy))
                color = hidden_map[cy][cx]
                if color == TARGET_CENTER_COLOR:
                    break
                if color == OBSTACLE_COLOR:
                    break
                cx += dx
                cy += dy
    return scanned


def can_move(matrix, width, height, x, y):

    for dy in range(ROBOT_SIZE):
        for dx in range(ROBOT_SIZE):
            nx, ny = x + dx, y + dy
            if nx < 0 or nx >= width or ny < 0 or ny >= height:
                return False
            if matrix[ny][nx] == OBSTACLE_COLOR:
                return False
    return True


def wait_any_key_to_start(screen, back):
    map_surface = pygame.image.load(back).convert()
```

Лістинг А.1, аркуш 2

```
    pygame.display.set_caption("Press any key to start")
    # Draw
    screen.blit(map_surface, (0, 0))
    pygame.display.flip()
    waiting = True
    while waiting:
        event = pygame.event.wait()
        if event.type == KEYDOWN:
            if event.key == pygame.K_s:
                waiting = False
            elif event.key == pygame.K_f:
                toggle_fullscreen()
                screen.fill(UNKNOWN_COLOR)
                pygame.display.flip()
            elif event.key == pygame.K_q:
                pygame.quit()
                sys.exit()
        elif event.type == QUIT:
            pygame.quit()
            sys.exit()


def check_for_quit(size):
    global is_fullscreen
    v = None
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        v = event
    if not v is None and (v.type == pygame.KEYDOWN or v.type ==
pygame.KEYUP):
        if v.key == pygame.K_f:
            toggle_fullscreen()
        if v.key == pygame.K_q:
            pygame.quit()
            sys.exit()



def main(path_to_map, back):
    pygame.init()
    raw_map, hidden_map, display_map, width, height =
load_map(path_to_map)
    free_pixels_count = len({
        (x, y)
        for y in range(height)
        for x in range(width)
        if hidden_map[y][x] == FREE_COLOR
    })
```

Лістинг А.1, аркуш 3

```
    screen = pygame.display.set_mode((width, height))
    screen.fill(UNKNOWN_COLOR)
    wait_any_key_to_start(screen, back)
    pygame.display.set_caption("G-Mapping Robot Simulation")

    # Place robot in a random free region
    x, y = find_start_position(hidden_map, width, height)
    visited_counts = {d: 0 for d in ('north', 'south', 'west',
'east')}
    # how many times each pixel is visited
    visited = {}

    scanned_global = set()
    clock = pygame.time.Clock()

    # map_surface = pygame.image.load(back).convert()
    # screen.blit(map_surface, (0, 0))
    move_speed = 12
    if move_speed > ROBOT_SIZE:
        print("Швидкість не може бути більша за розмір робота")
        pygame.quit()
        sys.exit(0)
    while True:
        check_for_quit((width, height))

        scanned = scan_obstacles(hidden_map, width, height, x,
y)
        prev_len = len(scanned_global)
        scanned_global.update(scanned)
        is_scanned_new_obstacles = len(scanned_global) >
prev_len

        screen.fill(UNKNOWN_COLOR)
        for px, py in scanned_global:
            # screen.set_at((px, py), SCAN_COLOR)
            screen.set_at((px, py), hidden_map[py][px])
        pygame.draw.rect(screen, ROBOT_COLOR, (x, y, ROBOT_SIZE,
ROBOT_SIZE))
        pygame.display.flip()

        free_pixels_scanned_count = len([f for f in
scanned_global if hidden_map[f[1]][f[0]] == FREE_COLOR])

        percent_scanned = free_pixels_scanned_count /
free_pixels_count * 100

        print(f"Scanned {round(percent_scanned, 3)}% of free
pixels")
        if (is_scanned_new_obstacles and
                free_pixels_count == free_pixels_scanned_count):
```

Лістинг А.1, аркуш 4

```
            print("Exploration complete! All free pixels
discovered.")
            pygame.time.wait(2000)
            break

        # Move more steps at a time
        for step in range(move_speed, 0, -1):
            candidates = []
            for d, (dx, dy) in DIRECTIONS.items():
                nx, ny = x + dx * step, y + dy * step
                if can_move(hidden_map, width, height, nx, ny):
                    candidates.append((nx, ny))

            if candidates:
                move_step = step
                break
        else:
            # не змогли зрушити навіть на 1 — застрягли
            print("Robot is stuck!")
            pygame.time.wait(2000)
            pygame.quit()
            sys.exit()

          # обираємо серед кандидатів той, що найменше
відвіданий
        exploration = min(candidates, key=lambda p:
visited.get(p, 0))
        visited[exploration] = visited.get(exploration, 0) + 1
        x, y = exploration
        # clock.tick(5000)  # limit to 10 FPS


    pygame.quit()


if __name__ == '__main__':
    main("maps/class-map.bmp", "maps/unknown.bmp")
```

Лістинг А.1, аркуш 5

ДОДАТОК Б

Код перетворення PNG-файлу карти у BMP формат

```
from PIL import Image
import os

def convert_png_to_bmp(png_path, bmp_path=None):
    """
    Convert a PNG image to BMP format.

    :param png_path: Path to the source PNG file
    :param bmp_path: Optional path for the output BMP file. If
not provided, will use same name with .bmp extension.
    """
    if not os.path.isfile(png_path):
        raise FileNotFoundError(f"No such file: '{png_path}'")

    if not png_path.lower().endswith('.png'):
        raise ValueError("Input file must be a .png file")

    if bmp_path is None:
        bmp_path = os.path.splitext(png_path)[0] + '.bmp'

    try:
        with Image.open(png_path) as img:
            img.convert('RGB').save(bmp_path, format='BMP')
        print(f"Conversion successful: {bmp_path}")
    except Exception as e:
        print(f"Failed to convert image: {e}")

# Example usage
if __name__ == "__main__":
    png_file = "../maps/test.png"  # Replace with your PNG file
path
    convert_png_to_bmp(png_file)
```

Лістинг Б.1 – Функція конвертації PNG у BMP