Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ МЕТОДІВ ПОБУДОВИ ОПТИМАЛЬНОГО ШЛЯХУ ПО ПЕРЕСІЧНІЙ МІСЦЕВОСТІ ВІДПОВІДНО ДО ЗАДАНИХ ТОЧОК
(тема)

Виконав:
студент 2 курсу, групи ІНФм-18-2

Макаренко М.О.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Машталір В.П.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Путятін Є.П.
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Макаренко Максиму Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів побудови оптимального шляху по пересічній місцевості відповідно до заданих точок.

затверджена наказом по університету від «_21__» жовтня 2019 року № 1506Ст.

2. Термін подання студентом роботи до екзаменаційної комісії 02 12 2019 р.

3. Вихідні дані до роботи *Зображення карти до змагань з рогейну з червоним кольором; Зображення карти до змагань з рогейну з кольором маджента.*

4. Перелік питань, що потрібно опрацювати в роботі

1. Огляд методів класифікації реперних точок на зображенні.
2. Огляд методів розпізнавання цифр на зображенні.
3. Побудова та оптимізація графу за отриманими даним
4. Побудова оптимального маршруту з урахуванням різної вартості вершин за графом.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) *Карта для рогейну; графічне відображення побудованого графа; класичні види кольорів які застосовують на картах; Принцип роботи алгоритму.*

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

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на атестаційну роботу	21.10.2019	
2	Аналіз завдання, підбір літератури	05.11.19-10.11.19	
3	Аналіз літератури з досліджуваної проблеми	10.11.19-12.11.19	
4	Аналіз технічних засобів	12.11.19-13.11.19	
5	Розробка методу	13.11.19-16.11.19	
6	Програмна реалізація	16.11.19-22.11.19	
7	Оформлення пояснювальної записки	22.11.19-28.11.19	
8	Перевірка на плагіат	29.11.19	
9	Рецензування	29.11.19	
10	Підготовка презентації та доповіді	30.12.19	
11	Занесення роботи в електронний архів	01.12.19	
12	Попередній захист атестаційної роботи	02.12.19	

Дата видачі завдання _____ _____20__ р.

Студент _____
(підпис)

Керівник роботи _____ ___проф. Машталір В. П._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до атестаційної роботи: 65 с., 29 рис., 38 джерел.

ГРАФ, ПОШУК ОПТИМАЛЬНОГО ШЛЯХУ, РОГЕЙН, ПЕРЕТВОРЕННЯ ХАФА, ДЕТЕКТОР ГРАНИЦЬ КЕННІ, ДЕТЕКТУВАННЯ ОКРУЖНОСТЕЙ, РОЗПІЗНАННЯ ЧИСЕЛ, OCR.

Метою роботи є розробка системи побудови оптимального шляху з обмеженням максимальної протяжності, через точки різної вартості. Необхідно набрати якомога більше балів проходячи через різні точки лише по одному разу. Інформація про положення та вартість точок отримується за допомогою розпізнавання зображення карти для змагань.

Об’єктом дослідження є карти для змагань з рогейну.

Використано методи розпізнавання градієнтним перетворенням Хафа та розпізнавання чисел за допомогою OCR. Проведено дослідження з методами обходу на графах, пошуком оптимального шляху з додатковими параметрами.

У результаті роботи здійснена програмна реалізація розпізнавання карти, для визначення координат КП та їх вартості, на підставі яких побудовано граф. Розроблено алгоритм видалення надлишкових ребер із графа. Також реалізовано алгоритм пошуку оптимального шляху по отриманим даним.

GRAPH, SEARCHING THE OPTIMAL WAY, ROGAINE, HOUGH TRANSFORM, CANNY EDGE DETECTOR, CIRCLE HOUGH TRANSFORM, NUMBER RECOGNITION, OCR.

The purpose of the work is to develop a system for the optimal path constructing with the maximum length limitation, through points with different weight. It is needed to score as many points as it is possible by going through different points only once. The location and weight of points are obtained by recognizing the image of the competition map.

The object of the research is rogaine competition cards.

Circles were recognized using Hough transform, numbers recognition were detected with OCR support. The research was conducted with graph traversing methods, the optimal pathfinding with additional parameters.

As a result, the application was implemented to recognize a map, determine control points and its weights. The graph is built based on this information. The algorithm for redundant edges removing is developed. Also, there is an optimal route finder based on received data.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 6
Вступ 7
1 Аналіз предметної області 9
1.1 Актуальність задачі 9
1.2 Опис задачі 10
1.3 Постановка задачі дослідження 11
2 Огляд методів класифікації реперних точок на зображеннях 15
2.1 Методи розпізнання об’єктів на зображеннях 15
2.1.1 Пошук примітивів на зображенні. Перетворення Хафа 15
2.1.2 Детектор границь Кенні 24
2.2 Оптичне розпізнавання символів 29
3 Огляд методів пошуку оптимального шляху 33
3.1 Неінформовані алгоритми пошуку 33
3.1.1 Пошук в глибину (Depth-first search) 33
3.1.2 Пошук в ширину (Breadth-first search) 34
3.1.3 Евристичний пошук 35
3.2 Iterative Diffuse Path Planner 36
3.3 Мурашиний алгоритм 37
4 Результати досліджень та програмна реалізація 41
4.1 Обґрунтування вибору середовища програмної реалізації 41
4.2 Програмна реалізація 43
4.2.1 Розпізнавання зображення 45
4.2.2 Формування та оптимізація графа 52
4.2.3 Пошук маршруту 53
4.3 Інструкція користувача 56
Висновки 59
Перелік джерел посилання 61

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

КП – контрольний пункт

OCR – optical character recognition

ЗК – значущий колір

HSV – Hue Saturation Value, діапазон кольорів

BGR – діапазон кольорів прийнятий в OpenCV

BFS – Breadth-first search

DFS – Depth-first search

ВСТУП

Пошук оптимального шляху дуже давня проблема, розв’язок якої досі покращують, намагаючись досягти більших швидкостей, для більшої кількості об’єктів. Рішеннями цієї задачі користуються у великій кількості різноманітних галузей. Починаючи зі звичайного GPS-навігатору, закінчуючи задачами логістики, складання розкладів руху то що.

За своєю сутністю, задача пошуку оптимального шляху зводиться до обходу на графі, яким представляють місцевість, де дугами чи ребрами слугують звичайні дороги чи тропи. Отже для знаходження оптимального шляху необхідно зробити пошук на графі. Для різних областей застосування види графів можуть відрізнятися спрямованістю, обмеженнями на кількість ребер і додатковими даними про вершини або ребра. Починаючи з однієї (стартової) точки і досліджуючи суміжні вузли до тих пір, поки не буде досягнутий вузол призначення (кінцевий вузол). Крім того, в алгоритми пошуку шляху в більшості випадків закладена також мета знайти найкоротший шлях. Деякі методи пошуку на графі, такі як пошук в ширину, можуть знайти шлях, якщо дано достатньо часу. Інші методи, які «досліджують» граф, можуть досягти точки призначення набагато швидше.

Однак, у багатьох випадках перед тим як шукати шлях, необхідно, спочатку отримати інформацію про місцевість, наприклад з супутникового знімку. Це призводить до задачі розпізнавання зображення.

У випадку цієї роботи уся інформація міститься на зображенні карти для змагань, де вони позначаються окружностями певного кольору, поруч з якими підписані номера КП, тобто бали які отримуються за відвідування КП представляються як вершини, або вузли графа, а зв'язки як ребра.

Завдяки побудованому оптимальному шляху, з’являється можливість для учасників змагань порівняти свій результат із побудованим, прийнявши його за еталон. Це дає змогу проаналізувати свої помилки, перевірити свій власний побудований шлях. Адже слід пам’ятати, що далеко не усі хто

приймає участь у подібних змаганнях взагалі розбираються у графах та побудові оптимального шляху, а роблять це інтуїтивно. Також це дозволить початківцям, які не вміють планувати свій шлях, моделювати його за допомогою програми розглядаючи різноманітну відстань яку вони могли би пройти за час змагань. За допомогою побудови оптимального шляху судді які ставлять ці змагання, можуть перевірити трасу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі

Задача пошуку оптимального шляху дуже давня і навіть зараз, коли існує велика кількість різноманітних GPS-навігаторів, вони здебільшого працюють на автодорогах і дають не точні результати для пересічної місцевості. Також зі збільшенням кількості об’єктів які розглядаються системою – зазвичай, вершин на графі обрахункові затрати та час збільшується, що потребує розробку нових більш оптимальних алгоритмів.

Рішення задач для пошуку шляху за інформацією, яку спочатку необхідно розпізнати із зображення залишається актуальною до сих пір. Пошук оптимального шляху за картою без доступу до інтернету. Побудова шляхів за супутниковими знімками такими застосунками як *GoogleMaps* викликає потребу або у трекерах якими будуть позначені дороги, або розпізнання доріг на зображенні.

Наприклад, при використанні будь якого GPS-навігатору без завантаженої до нього карти, він буде лише показувати ваше місце знаходження у координатах, але це вам майже нічим не допоможе. Саме для цього існують розроблені карти, за вдяки яким на екрані відображаються різноманітні об’єкти, дороги, то що.

В тім оптимальний шлях – це далеко не завжди найкоротший, наприклад швидкість руху на різній місцевості може бути зовсім різною і залежати не тільки від рельєфу, а навіть від пори року, та погоди. До того ж, найкоротший шлях на площині не обов’язково буде таким саме у реальному, тривимірному просторі.

Багато існуючих алгоритмів пошуку шляху, гарно працюють лише в доволі вузькому діапазоні завдань та не дають гарних результатів у інших випадках. До того ж, в епоху смартфонів та доступності інформації у будь-який момент завдяки безпровідній мережі, за для задоволення потреб

користувачів необхідно мати велику швидкість та динамічність пошуку для різноманітних алгоритмів. Аби інформація була актуальною у будь-який момент. Наприклад, при декілька-хвилинному спізнені одного потягу, при відсутності динамічного підлаштування графіків, відставання буде нарощуватися.

1.2 Опис задачі

Концепція рогейну, як виду спорту, виникла в Австралії і веде своє походження від першого 24-х годинного заходу в 1972 році, і перше змагань з рогейну в 1976 році. З тих пір рогейн поширився по всьому світу і в даний час в усьому світі щорічно проводиться декілька сотень змагань з рогейну.

Даний вид спорту має схожість з деякими іншими видами (наприклад, такими, як біг по пересіченій місцевості, трейлраннінг, спортивний туризм, орієнтування, пригодницькі перегони та ін.), Але має свій власний унікальний характер і правила [1].

Мета команд в змаганнях з рогейну полягає в наборі за встановлений час максимальної суми очок, що присуджуються за відвідування контрольних пунктів, встановлених на місцевості і позначених на мапі змагань. До старту змагань командам дається певний час для планування свого маршруту на місцевості і порядку відвідування контрольних пунктів.

Традиційна тривалість змагань з рогейну – 24 години, але проводяться і більш короткі змагання тривалістю від 3 до 12 годин.

Контрольні пункти в змаганнях з рогейну мають різну вартість, виражену в очках, в залежності від їх віддаленості від старту і навігаційної складності їх взяття. Проходити контрольні пункти можна в довільному порядку. Як правило, взяти всі контрольні пункти за встановлений час змагань неможливо [2].

Склад команд на змаганнях з рогейну від двох до п'яти учасників. Команди можуть пересуватися виключно способом визначеним певним форматом змагань (пішки (ходьба і біг), на велосипеді, на лижах та ін.)

Для навігації команди можуть використовувати тільки видані карти змагань, магнітні компаси та годинник. Використання інших навігаційних приладів, в тому числі супутникових приймачів заборонено.

На багатьох змаганнях з рогейну організовується центральний базовий табір, який забезпечує команди гарячим харчуванням. Команди можуть повернутися в будь-який час, щоб поїсти і відпочити. Команди пересуваються в своєму темпі, що робить рогейн доступним як для молодих, так і для літніх учасників, цікавим як для сильних спортсменів, так і для простих учасників.

Рогейн спрямований на підвищення у учасників поваги до навколишнього природного середовища, на розвиток навігаційних навичок, впевненості в собі, поліпшення загального фізичного стану учасників і здатності працювати в команді.

Метою даної роботи було поставлено, розробку системи пошуку оптимального шляху на змаганнях з рогейну по зображенню карти для змагань за для отримання інформації та подальшого порівняння своїх результатів із побудованим комп’ютером шляхом.

1.3 Постановка задачі дослідження

Насамперед, слід зазначити, що ціль використання програмного застосунку, який розробляється у рамках атестаційної роботи, полягає не у застосуванні його учасниками під час змагань, що суперечить правилам, а для навчання початківців або для аналізу отриманих результатів. Також можливе використання для аналізу під час формування карти організаторами змагань. Виходячи з цього, до програмного застосунку не має вимоги роботи

у «реальному часі», та важливіше отримати кращий варіант ніж швидко його побудувати.

Розроблена система має побудувати найкоротший маршрут, який не перевищує зазначений користувачем, з максимальною можливою кількістю набраних балів. Розіб’ємо задачу на дві частини. Першу, в якій ми повинні отримати дані з зображення карти (рис. 1.1) і на підставі цих даних побудувати граф. Та другу, на якій враховуючи вартість кожного КП, потрібно віднайти такий шлях з точки старту, до точки фінішу, щоб відвідуючи КП лише один раз набрати якомога більше балів за найменшу можливу відстань, при цьому не перевищивши зазначену користувачем максимальну довжину шляху.

Рисунок 1.1 – Карта для рогейну

Для вирішення першої частини, необхідно розв’язати наступні завдання:

– розробити маску для виділення значущого кольору;

– серед пікселів які залишилися віднайти кола, що позначають місце знаходження КП;

– знайти точку старту і точку фінішу, у випадку коли вони не співпадають;

– поруч з кожним колом віднайти і розпізнати номер даного КП;

– виміряти відстань між усіма КП;

– на основі отриманих даних побудувати граф (рис 1.2).

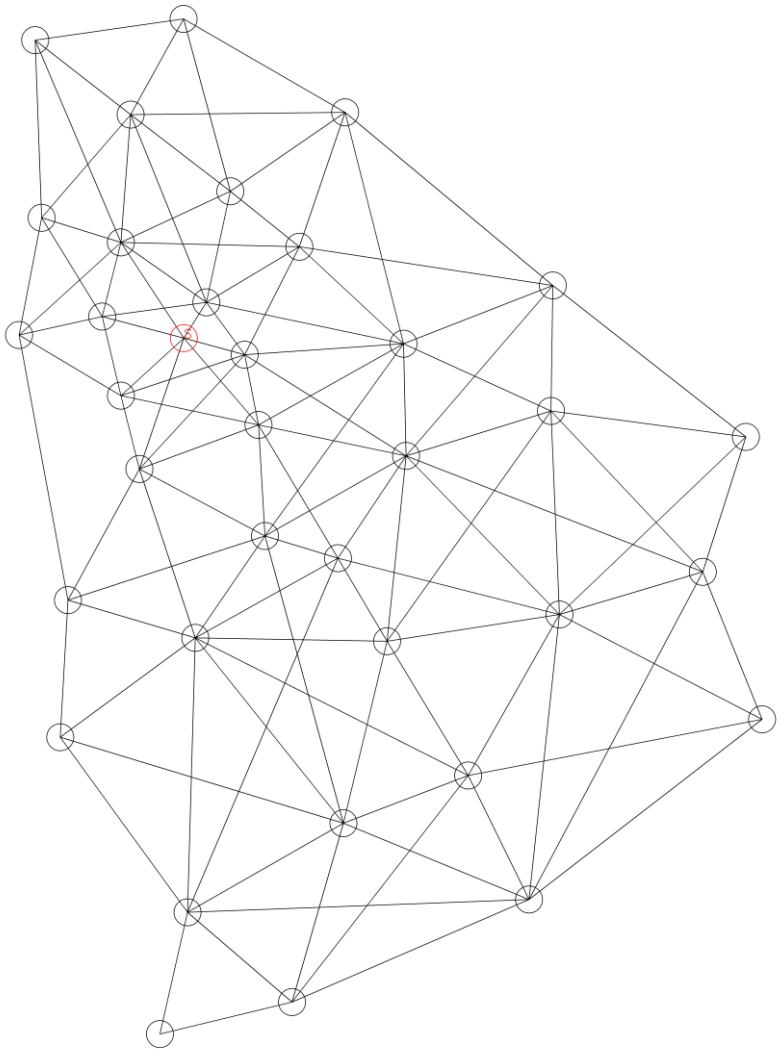


Рисунок 1.2 – Графічне відображення побудованого Графа

Після побудови графу, необхідно дослідити методи ого оптимізації, адже якщо зробити повний граф, то ми отримаємо велику кількість ребер, які жоден алгоритм оптимального шляху використовувати не буде. Усі ці ребра людина дивлячись на граф відкидає інтуїтивно, в тім необхідно розробити метод оптимізації для програми.

На підставі сформованого графу необхідно знайти усі варіанти шляху від старту до фінішу які не будуть перевищувати задану користувачем відстань. За для цього необхідно дослідити методи обходу графів і на підставі існуючих алгоритмів розробити програму враховуючи те, що у деяких випадках старт і фініш співпадають, і необхідно аби розроблений алгоритм міг повернутися до початкової точки. Це дасть змогу вилучити варіанти шляху, на початку якого узяти усі вершини навколо фінішу, а потім віддалення від нього, наприклад у вигляді спіралі (рис 1.3).

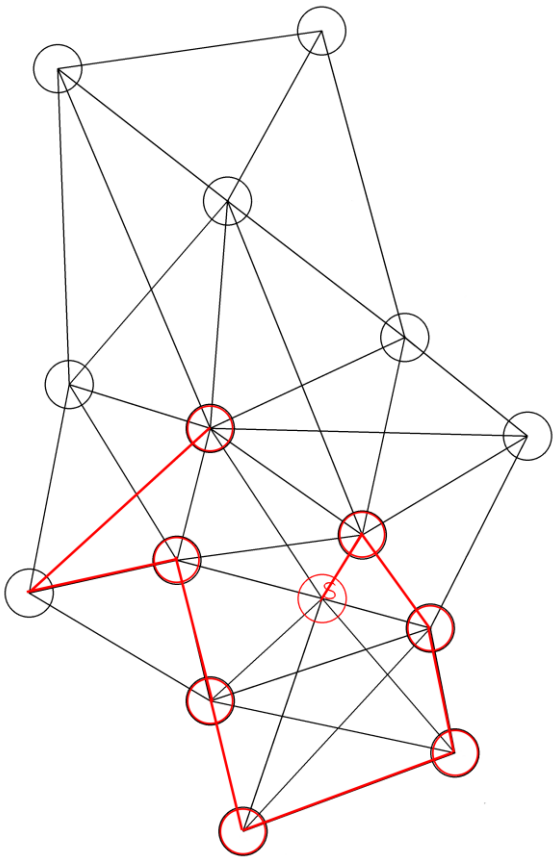


Рисунок 1.3 – Відрізана можливість повернення до фінішу

2 ОГЛЯД МЕТОДІВ КЛАСИФІКАЦІЇ РЕПЕРНИХ ТОЧОК НА ЗОБРАЖЕННЯХ

2.1 Методи розпізнання об’єктів на зображеннях

На даний момент існує багато різноманітних методів для розпізнання зображень, та виділення на них певних об’єктів. Задля цього використовуються як і класичні алгоритми комп’ютерного зору, так і останнім часом сильно розповсюджені методи засновані на нейронних мережах. Втім, в загальних випадках, для виявлення нескладних об’єктів набагато вигідніше застосовувати класичні методи, ніж складні в навчанні нейронні мережі [32].

2.1.1 Пошук примітивів на зображенні. Перетворення Хафа

Автоматичний аналіз цифрових зображень часто має проблему з ідентифікацією простих фігур, таких як прямі, кола або еліпси. У багатьох випадках алгоритм пошуку кордону використовується в якості попередньої обробки для отримання точок на кривій у зображенні. Однак, або через наявність шумів на зображенні, або через недосконалість алгоритму виявлення кордонів, можуть з'явитися «втрачені» точки на кривій, а також невеликі відхилення від ідеальної форми лінії, кола або еліпса. З цих причин часто важко приписати знайдені кордони до відповідних ліній, кругів або еліпсів на зображенні. Метою перетворення Хафа є вирішення проблеми групування граничних точок шляхом застосування певної процедури голосування до набору параметризованих об'єктів зображення [31].

У найпростішому випадку перетворення Хафа є лінійним перетворенням для виявлення прямих. Пряма лінія може бути задана рівнянням $y = mx + b$ і може бути обчислена по будь-якій парі точок (x, y) на

зображенні. Головною ідеєю перетворення Хафа – врахувати характеристики прямої не як рівняння, побудоване по парі точок зображення, а в термінах її параметрів, тобто m – кутового коефіцієнта і b – точки перетину з віссю ординат. Виходячи з цього пряма, задана рівнянням $y = mx + b$, може бути представлена у вигляді точки з координатами (b, m) в просторі параметрів.

Однак прямі лінії, паралельні осі ординат, мають нескінченні значення для параметра m. Отже, більш зручно представляти лінію, використовуючи інші параметри, відомі як r і θ. Параметр r являє собою довжину радіуса-вектора найближчої до початку координат, точки на прямій (тобто нормалі до лінії, проведеної з початку координат), тоді як θ – являє собою кут між цим вектором і віссю абсцис. При такому описі ліній нескінченні параметри не виникають.

Таким чином, рівняння прямої можна записати як

$$y = \left(-\frac{\cos\theta}{\sin\theta} \right) x + \left(\frac{r}{\sin\theta} \right), \tag{2.1}$$

або після перетворення

$$y = x\cos\theta + \sin\theta. \tag{2.2}$$

Тому, можливо зв’язати кожну пряму на вихідному зображенні (площині X–Y) з точкою у якої координати r, θ в площині параметрів, яка буде унікальною за умовою, що $\theta \in [0, \pi]$ та $r \in R$, або якщо $\theta \in [0, 2\pi]$ і $r \geq 0$. Через будь-яку точку на площині може проходити нескінчена кількість прямих. Якщо описати координати точки як (x_0, y_0), то усі прямі, які перетинають її, відповідають рівнянню:

$$r(\theta) = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta. \tag{2.3}$$

Це відповідає синусоїдальній лінії в просторі Хафа *(r, θ)*, яка, в свою чергу, є унікальною для даної точки і однозначно визначає її. Якщо ці лінії (криві), що відповідають двом точкам, перекривають один одного, то точка (в просторі Хафа), де вони перетинаються, відповідає прямим лініям (в вихідному місці зображення), які проходять через обидві точки. Як правило, ряд точок, які утворюють пряму лінію, визначають синусоїди, які перетинаються в точці параметрів для цієї лінії. Таким чином, проблема виявлення колінеарних точок може бути зведена до проблеми виявлення кривих що перетинаються.

Алгоритм перетворення Хафа використовує масив, званий акумулятором, для визначення наявності лінії $y = mx + b$. Розмірність батареї дорівнює числу невідомих параметрів простору Хафа.

Наприклад, для лінійного перетворення вам потрібно використовувати двовимірний масив, оскільки є два невідомих параметри: m і b. Два виміри акумулятора відповідають квантованим значенням параметрів m і b. Для кожної точки і її сусідів алгоритм визначає, чи достатня вага кордону в цій точці. Якщо так, то алгоритм розраховує параметри лінії і збільшує значення в елементі акумулятору, відповідне цим параметрам.

Потім, шляхом знаходження елементів акумулятора з максимальними значеннями, зазвичай шляхом пошуку локального максимуму в просторі акумулятора, можна визначити найбільш підходящі прямі лінії. Найпростіший спосіб – це порогова фільтрація. Однак в різних ситуаціях різні методи можуть давати різні результати .

Оскільки отримані прямі не містять інформації про довжину, наступним кроком є пошук частин зображення, відповідних знайденим прямим. Більш того, через помилки на етапі визначення меж фігури в просторі акумулятора також будуть містити помилки. Це робить пошук відповідних ліній нетривіальним.

2.1.1.1 Перетворення Хафа для пошуку ліній

Класичне перетворення Хафа було початково розроблено з ціллю виділення прямих ліній, на бінарному зображенні. Воно засноване на використанні простору параметрів, в якому і здійснюється пошук прямих (рис. 2.1). Найбільш розповсюджені наступні параметричні рівняння прямих:

$$Y = kX + b, \tag{2.4}$$

$$X\cos\theta + Y\sin\theta = p. \tag{2.5}$$

Однак, оскільки прямі на площині характеризуються лише двома параметрами, простір параметрів завжди буде лише з розмірністю два.

Класичне перетворення Хафа використовує параметри (p,θ) рівняння.

Припустимо, контурне зображення розглядається як множина точок (x,y) в вихідному просторі $E=(X,Y)$. Множина прямих, що перетинають точку (x,y), можуть бути розглянуті у вигляді множини точок (p,θ) у просторі $\{p,\theta\}$. Функція відображення точки у просторі Хафа називається функцією відгуку.

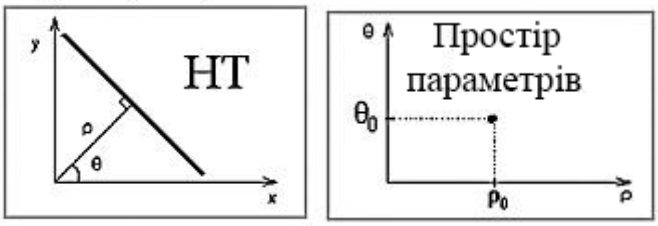


Рисунок 2.1 – Параметризація $X\cos\theta + Y\sin\theta = p$

Ідея перетворення Хафа полягає в тому, що для кожної точки простору параметрів сумується кількість голосів, поданих за неї, тобто кількість точок вихідного простору, що породжують відгуки у просторі параметрів, які перетинають дану точку (p,θ). Тут використовується той факт, що будь-які

дві синусоїди в просторі параметрів перетнуться в точці (p,θ) лише тоді, коли точки що їх породжують в вихідному просторі знаходяться на одній прямій, яку можливо описати рівнянням (2.4) з параметрами (p,θ). Задана таким чином функція $A(p,\theta)$ називається акумуляторною функцією, до того ж її абсолютне значення в точці (p,θ) дорівнює кількості точок контурного препарату, що знаходиться на відповідній прямій в вихідному просторі зображення.

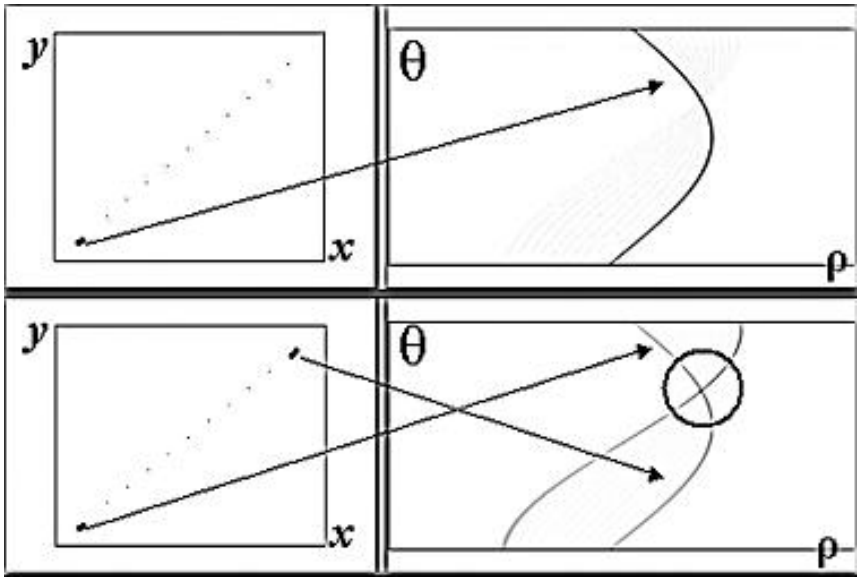


Рисунок 2.2 – Голосування точок в акумулятор

У тому випадку. Коли на зображені представлено *m* прямих, акумуляторна функція $A(p,\theta)$ буде мати рівно *m* локальних максимумів в точках, відповідних даним прямим. Таким чином, для знаходження прямих на вихідному зображені достатньо знайти усі значущі локальні максимуми акумуляторної функції. Що вельми важливо з практичної точки зору, такий алгоритм виділення прямих, на відміну від методів виділення лінеаментів, зовсім не опирається на припущення про зв'язність лінії що аналізуємо. Тому методи голосування добре працюють в умовах загороджування, або при наявності інших перешкод.

Як правило, $A(p,\theta)$ обраховується не для кожної точки простору параметрів, а лише для кожної комірки акумулятора, тобто деякої прямокутної ділянки, на які розбивається простір параметрів.

Інваріантність перетворення Хафа до здвигу, масштабування і повороту витікає безпосередньо із визначення. Більш того, оскільки прямі лінії за будь-яких проекційних перетвореннях тривимірного простору завжди переходять лише в прямі лінії (у виродженому випадку – в точки), перетворення Хафа дозволяє знайти лінеаменти інваріантні не лише до афінних перетворень площини, но и до групи проекційних перетворень у просторі. Це дає змогу використовувати перетворення Хафа для робастного детектування тривимірних об'єктів, контури яких повністю, або частково описуються лінеаментами.

Легко переконатися, що результат перетворення Хафа еквівалентний інтегруванню контурного зображення вздовж усіх можливих прямих. Це обумовлює його фільтруючі властивості та визначає велику ступінь перешкодозахищеності.

Ефективність перетворення Хафа обумовлюється наступними факторами:

– вдалий вибір параметрів. За вдяки використанню того факту, що при проекційних перетвореннях пряма завжди лишається прямою. В зв'язку з чим сформовано простір параметрів низької розмірності $(n=2)$;

– однократне використання вихідної інформації. Кожен піксель зображення опитується лише один раз. При цьому подальші обрахування виконуються лише для пікселів, несучих корисну інформацію (в даному випадку контурних). Звідси безпосередньо виходить, що обчислювальна ефективність перетворення Хафа тим вище, чим менша кількість пікселів несучих корисну інформацію, в порівнянні з площиною зображення. Це обумовлює переважне використання цього методу при аналізі контурних препаратів, а також точкових паттернів.

2.1.1.2 Перетворення Хафа для пошуку кола

Для пошуку кола с заданим радіусом R на зображенні будемо рахувати що маємо справи з двохпараметричною родиною кривих

$$(x-x_0)^2+(y-y_0)^2=R^2, \tag{2.6}$$

і проводити пошук максимуму акумуляторної функції $A(x,y)$ у просторі параметрів (x,y) [3]. Зауважимо, що простір параметрів у цьому випадку практично співпадає з вихідним (x,y). Оскільки набір центрів усіх можливих окружностей радіусу R, що перетинають задану точку, створюють окружність радіусу R навколо цієї точки, функція відгуку в перетворенні Хафа для пошуку окружностей з відомим радіусом, являє собою окружність з тим самим радіусом з центром в точці, що голосує. Максимум акумулятора відповідає положенню центра кола на зображенні, як це зображено на рисунку 2.3

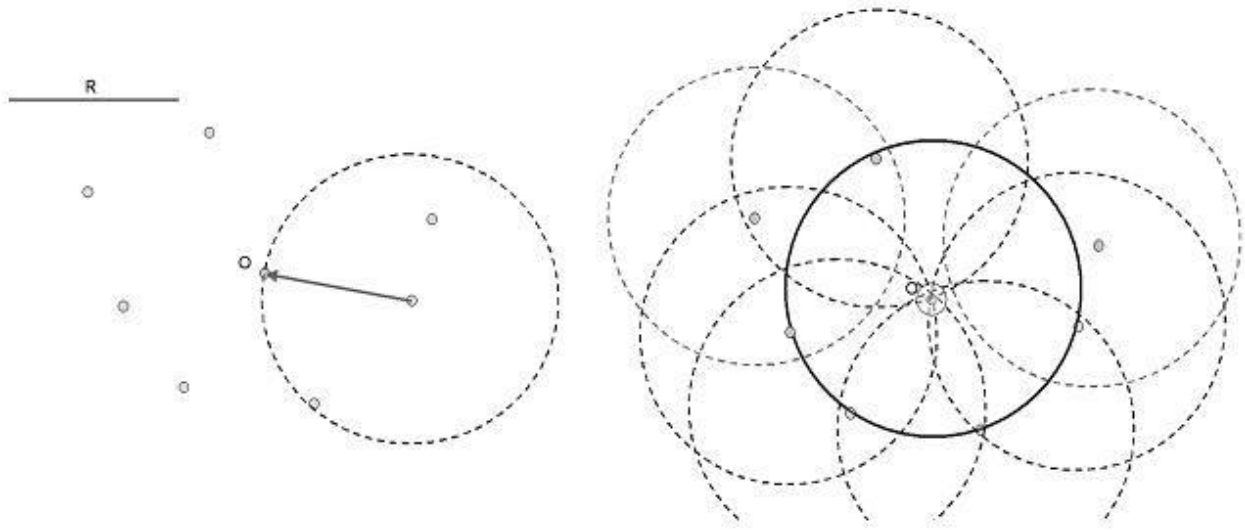


Рисунок 2.3 – Принцип пошуку кола відомого радіусу в бінарній множині точок методом голосування

Опишемо алгоритм пошуку кола с заданим радіусом на полутонових зображеннях, який використовує оцінку орієнтації нормалі в контурних точках які голосують. Першим кроком процесу буде пошук пікселів краю, що оточують периметр об’єкту.

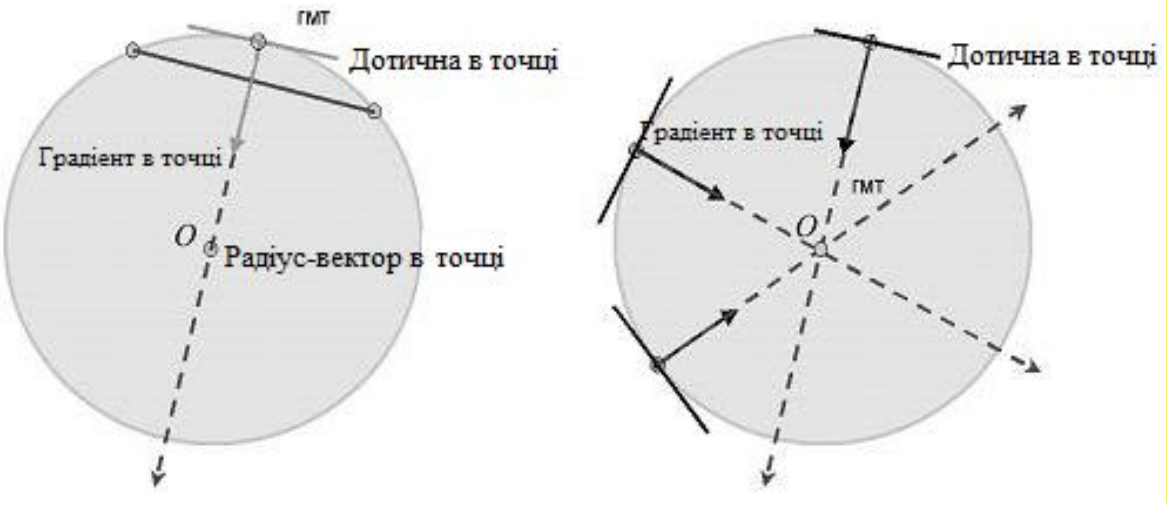
Наприклад, можна використати оператор Собела, що дає оцінку амплітуді і направлення вектору-градієнта. За точки контору що голосують, будемо рахувати точки з високим значенням модуля градієнта.

Для кожного знайденого пікселя краю використовується оцінка положення і орієнтації контуру з ціллю оцінки центру кругового об’єкту радіусом R шляхом руху на відстані R від пікселя краю в напрямку нормалі до контуру (тобто у напрямку вектору-градієнта). Якщо цю операцію повторювати для кожного об’єкту крайового пікселя, буде знайдена множина положень передбачуваних точок центра, які можуть бути усереднені для визначення точного місця центра.

Якщо радіус кола невідомий або змінний, необхідно включити R в якості додаткового параметру в параметричний простір-акумулятор: в цьому випадку процедура пошуку пику повинна визначити радіус, так само як і положення центру шляхом розгляду змін вздовж третього виміру параметричного простору [5].

Якщо розмір знайденої окружності нас не цікавить і потрібно віднайти лише її центр, можливо обійтись і без збільшення розмірності простору параметрів. Припустимо, для кожного можливого напрямку на центр контурна точка голосує не точкою на відстані R, а променем в цьому напрямку (рис. 2.4а).

Таким чином, будуть задіяні усі можливі точки центру при будь-якому масштабі об’єкта, це дозволяє шукати окружності не задаючи їм радіус (рис. 2.4б).



а) б)

Рисунок 2.4 – Принцип пошуку кола, незалежно від його радіусу на полутоновому зображенні методом голосування

На другому етапі аналізу, після виявлення потенційних центрів окружностей, можна буде повторно звернутися до зображення і уточнити радіус окружностей з центрами в знайдених точках.

Зауважимо також, що рисунок 2.4(а) демонструє цікавий зв'язок описаного сучасного алгоритму аналізу напівтонових зображень із завданням з шкільного курсу про побудову кола за трьома точками. Справді, адже напрямок градієнта в точці контуру контрастної окружності на зображенні є не що інше як межа серединного перпендикуляра до січної кола при прагненні довжини січної до нуля.

Якби у нас не було безперервного контуру кола, а була лише множина точок (як в прикладі на рис. 2.3), то ми могли б реалізувати голосування пар точок на користь відповідних серединних перпендикулярів, і, таким чином, вирішити задачу виділення кіл невідомого розміру в бінарній множині точок (рис. 2.5).

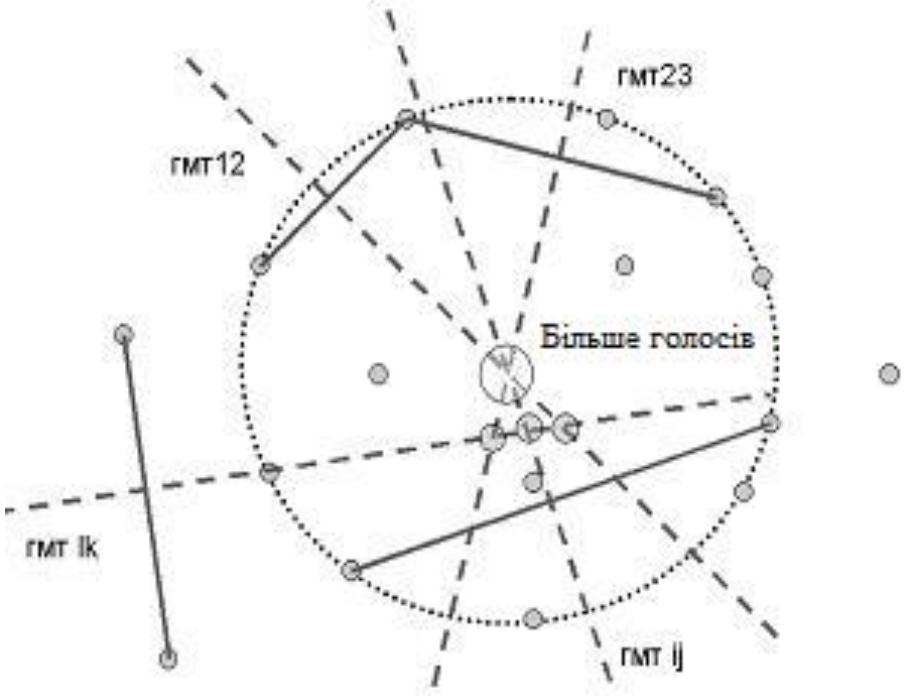


Рисунок 2.5 – Принцип пошуку кола невідомого радіусу в бінарній множині точок методом голосування

2.1.2 Детектор границь Кенні

Детектор границь Кенні (*Canny edge detector*) являє собою оператор виділення границь, який застосовує багатоступеневий алгоритм для виділення широкого спектру границь на зображеннях (рис. 2.6).

Детектор границь Кенні – це метод отримання корисної структурної інформації із різноманітних візуальних об’єктів, що дає істотне зменшення об’єму даних які підлягають обробці [4]. Широко застосовується у різноманітних системах комп’ютерного зору. Кенні зауважив, що вимоги до застосування детектора границь в системах із різноманітними об’єктами відносно схожі. Таким чином, рішення для виявлення границь яке полягає у цих вимогах може бути реалізоване у великій кількості ситуацій. Загальні критерії для виявлення границь включають:

– виявлення границь з низькою частотою помилок, що значить, що виявлення повинне точно захоплювати як можливо більше країв, показаних на зображенні;

– точка границі, що була виявлена оператором, повинна точно розміщуватися у центрі границі;

– задана границя на зображенні повинна бути відмічена лише один раз, і де це можливо шум на зображенні не має утворювати фальшиві границі.

Аби задовольнити ці критерії, Кенні використав метод варіаційного обчислення, який знаходить функцію, яка дозволяє оптимізувати даний функціонал. Оптимальна функція в детекторі Кенні описується сумою чотирьох експоненційних термінів, але її можна апроксимувати першою похідною функцією Гауса.

Серед розроблених до теперішнього часу методів виявлення границь, алгоритм виявлення границь Кенні є одним з найбільш суворо визначених методів, що забезпечує хороше і надійне виявлення. Завдяки своїй оптимальності, що відповідає трьом критеріям виявлення границь і простотою процесу реалізації, він став одним з найпопулярніших алгоритмів для визначення границь [17].

Процес алгоритму виявлення границь Кенні може бути розбитий на 5 різних кроків:

– застосування фільтру Гаусса для згладжування зображення, щоб видалити шум;

– знаходження градієнтів інтенсивності зображення;

– застосування часткового приглушення, щоб позбутися помилкової реакції на виявлення границь;

– застосування подвійного (нижній та верхній) пороги для визначення потенційних границь(рис. 2.7);

– відстеження границі за допомогою гістерезису: кінець опрацювання виявлення границь, приглушивши всі інші слабкі границі, що не пов'язані з сильними границями.

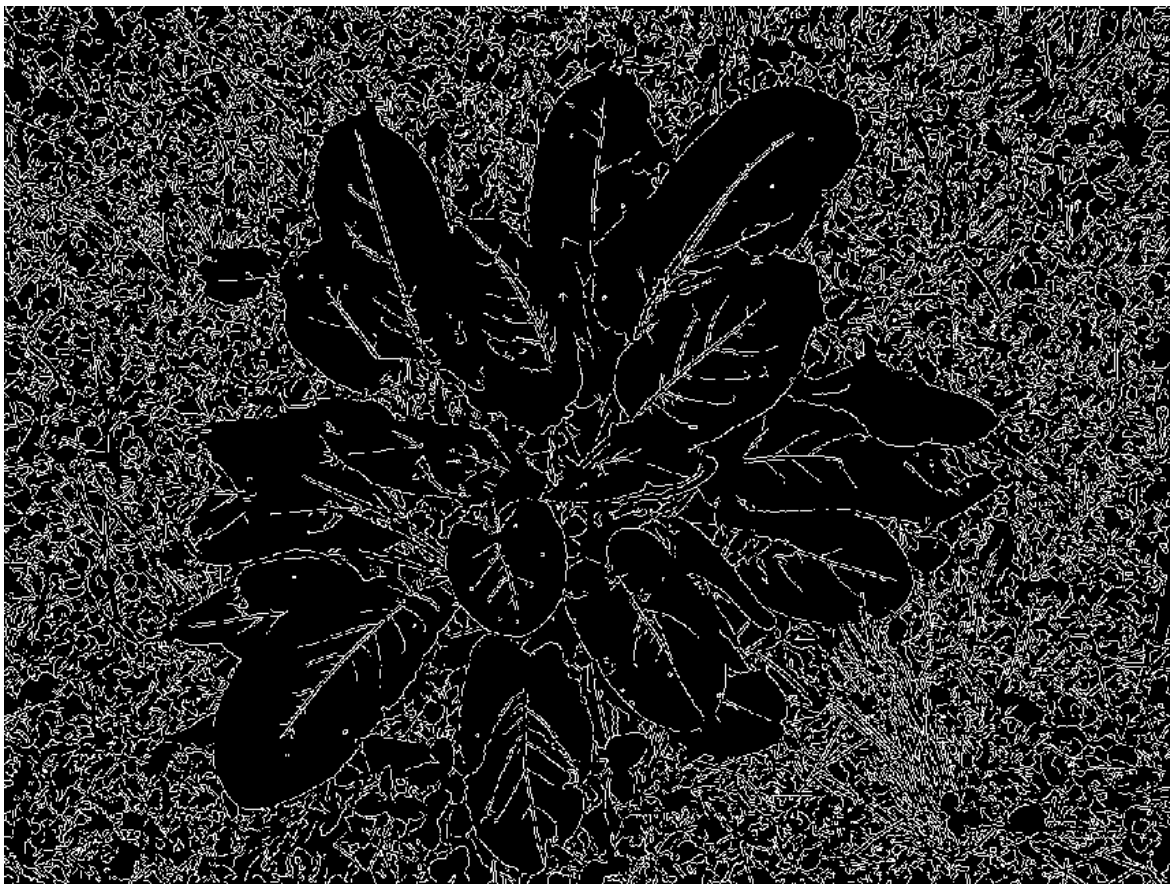


Рисунок 2.6 – Виділення контурів на зображенні рослини

Оскільки на результати виявлення границь сильно впливають шуми на зображенні, важливо відфільтрувати шум, щоб запобігти фальшивому виявленню, викликаному шумом. Щоб загладити зображення, ядро фільтру Гаусса поєднується із зображенням. Цей крок трохи згладить зображення, аби зменшити вплив явного шуму на детектор границь. Рівняння для ядра гауссовського фільтру розміром $(2k+1)\times(2k+1)$ визначається як:

$$H_{ij} = \frac{1}{2\pi\sigma^2}\exp-\left(\frac{(i-(k+1))^2+(j-(k+1))^2}{2\sigma^2}\right); \quad (2.7)$$

$$1 \le i, j \le (2k+1).$$

Важливо враховувати, що вибір розміру ядра Гаусса буде впливати на продуктивність детектора. Чим більше розмір, тим нижче чутливість детектора до шуму. Окрім цього, помилка локалізації для виявлення границь буде потроху збільшуватись зі збільшенням розміру фільтру Гусса. 5x5 –

гарний розмір для багатьох випадків, в тім він також буде змінюватись в залежності від конкретної ситуації [6].

Границі на зображенні можуть пролягати у різних напрямках, тому алгоритм Кенні використовує чотири фільтри для виявлення горизонтальних, вертикальних та діагональних границь у розмитому зображенні. Оператор виявлення границь (наприклад Собеля, Робертса або Прюітта) повертають значення для першої похідної в горизонтальному напрямі G_x та вертикальному G_y. Із цього можливо визначити градієнт і напрямок границі:

$$\begin{aligned} G &= \sqrt{G_x^{\,2} + G_y^{\,2}}, \\ \theta &= a\tan 2(G_y, G_x), \end{aligned} \tag{2.8}$$

де G може буті обрахована із використанням функції *hypot,* а *atan2* – функція арктангенса с двома аргументами. Кут направлення Границі округлений до одного із чотирьох кутів, що представляють вертикаль, горизонталь та дві діагоналі ($0^\circ, 45^\circ, 90^\circ, 135^\circ$). Напрямок границі, яка потрапляє у кожну кольорову область, буде встановлено на певні значення кута.

Не-максимальне подавлення – метод виявлення границь. Застосовується, для того щоб знайти «найбільшу» границю. Після застосування обчислення градієнту границі, вилучені зі значення градієнта, усе ще розмиті. Стосовно третього критерію, повинен бути лише один точний відгук на границі. Таким чином не-максимальне подавлення може допомогти подавити усі значення градієнту, окрім локальних максимумів, які вказують на місцеположення найбільш різкої зміни значень інтенсивності.

Пікселі з сильними границями, повинні приймати участь в кінцевому граничному зображенні. Проте, будуть виникати деякі суперечки через пікселі зі слабкими границями, так як ці границі могли бути вилучені із істинної границі, або через зміну кольору чи шуму. Щоб отримати точний результат, слабкі границі, викликані останніми причинами, повинні бути

видалені. Зазвичай, слабкі граничні пікселі, викликані істинними границями, будуть зв’язані зі сильними границями, в той час коли шумові відгуки не зв’язані. За для відстеження граничного об’єднання застосовується аналіз *BLOB*-об’єктів, обираючи слабкий граничний піксель та його вісім суміжних пікселі. Доки в *BLOB*-об’єкті присутні хоча б один піксель із сильної границі, цю точку слабкої границі, можливо ідентифікувати як точку, яку слід зберегти.

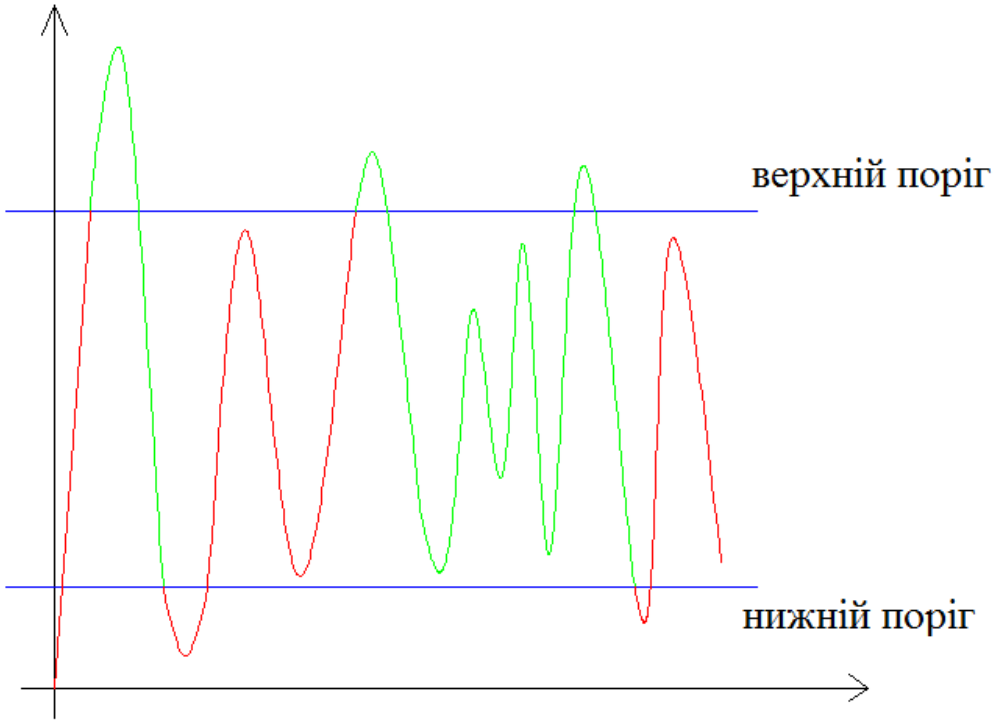


Рисунок 2.7 – Візуалізація процесу видалення «слабких» контурів за допомогою порогів

Хоча традиційний метод Кенні забезпечує відносно просту, але точну методологію для проблеми виявлення границь, з більш вимогливими умовами щодо точності та надійності виявлення, традиційний алгоритм більше не може виконувати завдання виявлення складних границь. Основні дефекти традиційного алгоритму можна звести до наступного:

– для згладжування шуму застосовується гауссовий фільтр, але він також згладжує границі, які розглядаються як функція високої частоти; це збільшить можливість відсутності слабких границь, а також виникнення ізольованих границь у результаті;

– для розрахунку амплітуди градієнта класичний алгоритм виявлення границь Кенні використовує центр в малому вікні 2 × 2 для обчислення середнього значення кінцевої різниці для представлення амплітуди градієнта; цей метод чутливий до шуму і може легко виявити помилкові границі, та, як наслідок, втратити справжні границі;

– у традиційному алгоритмі присутні два фіксованих глобальних порогових значення для фільтрації помилкових границь; однак, оскільки зображення може бути складне, різні локальні області потребують дуже різних порогових значень для точного пошуку реальних границь; крім того, глобальні порогові значення визначаються вручну за допомогою експериментів, що призводить до складності розрахунку, коли потрібно розглянути велику кількість різних зображень;

– результат традиційного виявлення не може досягти задовільної високої точності одного відгуку для кожної границі – з'являться багатоточкові відгуки.

2.2 Оптичне розпізнавання символів

Оптичне розпізнавання тексту (*optical character recognition, OCR*) – це переведення печатного або рукописного тексту у машинний код, за допомогою електронних або механічних засобів. Широко використовується за для оцифрування різноманітних текстів, будь-то документи, рахунки, паспортні дані, чи звичайні книги [15].

Перші версії реалізації необхідно було навчати окремо для кожного шрифту, в тім сучасні системи дозволяють розпізнавати більшість шрифтів, з

підтримкою різноманітних форматів вхідних даних у вигляді цифрових зображень. Деякі системи, навіть, здатні відтворювати початкове форматування тексту, зберігаючи початкові стовпці, абзаци, колонки, зображення та інші не текстові компоненти [19].

OCR поділяються на чотири типи за вхідними даними:

– оптичне розпізнавання символів – призначене для розпізнавання машинописного тексту по одному символу за раз;

– оптичне розпізнавання слів – розпізнає по одному слову за раз, призначається для язиків, що використовують пробіл у якості розділювача слів;

– інтелектуальне розпізнання символів (*Intelligent character recognition, ICR*) – використовується для розпізнавання рукописного, або печатного тексту по одному символу за раз;

– інтелектуальне розпізнавання слів (*Intelligent word recognition, IWR*) – застосовується до рукописного, або машинного тексту по одному слову за раз. Особливо корисно для язиків у яких при рукописному написані гліфи не відділяються один від одного.

OCR – це зазвичай цілковито автономний процес, який аналізує статичні документи. В тім можливо використовувати аналіз руху під час написання тексту, у якості вхідних даних для розпізнання рукописного тексту [29].

Це дозволяє зробити наскрізний процес розпізнання набагато точніше, за рахунок того, що замість використання форми гліфів та слів, ця техніка здатна фіксувати рух, такий як порядок прорисовки окремих сегментів, направлення, відпускання та підіймання ручки. Ця техніка також відома як динамічне розпізнавання символів.

Існує два основних типи алгоритму основного *OCR*, який може створити список ранжируваних символів.

Матричне узгодження включає порівняння зображення із збереженим гліфом на основі пікселя на піксель; він також відомий як «*pattern matching*»,

«*pattern recognition*» або «*image correlation*». Це залежить від того, щоб вхідний гліф був правильно ізольований від решти зображення, а збережений гліф був схожим шрифтом і в тому ж масштабі. Цей прийом найкраще працює з машинописним текстом і не працює добре, коли виникають нові шрифти. Це методика раннього фізичного використання фотоелементів на основі OCR, а не безпосередньо.

Функція виділення ознак розбиває гліфи на «особливості», такі як лінії, замкнуті петлі та перетини ліній. Вилучення особливостей, дозволяє зменшити розмірність вхідних даних та робить процес розпізнання більш ефективним з точки зору обчислювань.

Ці функції порівнюються із абстрактним векторним зображенням символу, який може зводитись до одного або некольких прототипів гліфів. Зазвичай це робиться за допомогою таких класичним методом застосовуваним у «інтелектуальному» розпізнаванні, як алгоритм кластеризації *k*-середніх.

Програмне забезпечення, наприклад *Cuneiform* і *Tesseract*, використовує двопрохідний підхід до розпізнавання [30]. Другий прохід відомий як «*adaptive recognition*» і використовує літери, розпізнавані з високою впевненістю на першому проході, щоб краще розпізнати літери, що залишилися на другому проході. Це вигідно для незвичайних шрифтів або неякісних сканувань, де шрифт спотворений (наприклад, розмитий або побляклий).

Точність OCR може бути збільшена, якщо розпізнаний тест обмежити словником – переліком слів, дозволених у документі. Це можуть бути, наприклад, усі слова англійською мовою або більш технічний лексикон для певного випадку застосування системи.

Цей метод може бути проблематичним, якщо документ містить слова, які не входять до словнику, як власні назви. *Tesseract* використовує свій словник для впливу на крок сегментації символів для підвищення точності.

Вихідний потік може представляти собою звичайний текстовий потік або файл символів, але більш досконалі системи OCR можуть зберігати оригінальний макет сторінки та створювати, наприклад, анотований PDF, що включає в себе як оригінальне зображення сторінки, так і текстове відображення, яке можна редагувати.

Знання граматики сканованої мови також може допомогти визначити, чи є слово, можливо, дієсловом чи іменником, що дозволяє підвищити точність.

Існує кілька методів вирішення проблеми розпізнавання символів іншими способами, ніж вдосконалені алгоритми OCR.

Метод формування кращого вводу, за для цього було розроблено декілька спеціальних шрифтів, наприклад *OCR-A, OCR-B,* або *MICR*. За для покращення розпізнавання для цих шрифтів було розроблено точно задані розміри, інтервали та особливі форми які дозволяють значно покращити розпізнавання. В тім за іронією багато популярних систем для розпізнавання тексту було обучено для розпізнавання популярних шрифтів таких як *Arial* або *Times New Roman* і вони не розуміють спеціальні шрифти.

Також застосовується метод «*Comb fields»* коли застосовують поля які вимушують користувача писати по одному гліфу у поле.

3 ОГЛЯД МЕТОДІВ ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ

3.1 Неінформовані алгоритми пошуку

До них відносяться класичні алгоритми пошуку на графах [18]. Дані алгоритми передбачають повний обхід вершин графів, без отримання інформації про те, наскільки кожен крок алгоритму наближує отримання результату. Застосування цих алгоритмів приводе до значних обчислювальних затрат, через що, данні алгоритми застосовуються не часто [16, 20]. До них відносяться такі алгоритми як:

3.1.1 Пошук в глибину (Depth-first search)

Сутність алгоритму полягає в обході починаючи з кореня дерева, на максимальну можливу глибину. Алгоритм пошуку описується рекурсивно: перебираємо усі ребра вихідні з вершини яку розглядаємо. Якщо ребро веде у вершину, яку ще не розглядали, запускаємо алгоритм для цієї вершини, а потім вертаємось і продовжуємо перебирати ребра. Повернення відбувається тоді, коли у вершини не залишається ребер, які ведуть до нерозглянутих вершин (рис 3.1).

Найгірша продуктивність для явного графа який було пройдено без повторів становить $O\left(|V|+|E|\right)$, для неявних графів із коефіцієнтом розгалуження b та пошуком на глибину d становить $O(b^d)$.

Найгірша складність простору дорівнює $O\left(|V|\right)$, якщо увесь граф проходити без повторів, O (найдовша шукана довжина шляху) сягає $O(bd)$ для неявних графів без усунення дублюючих вузлів [11].

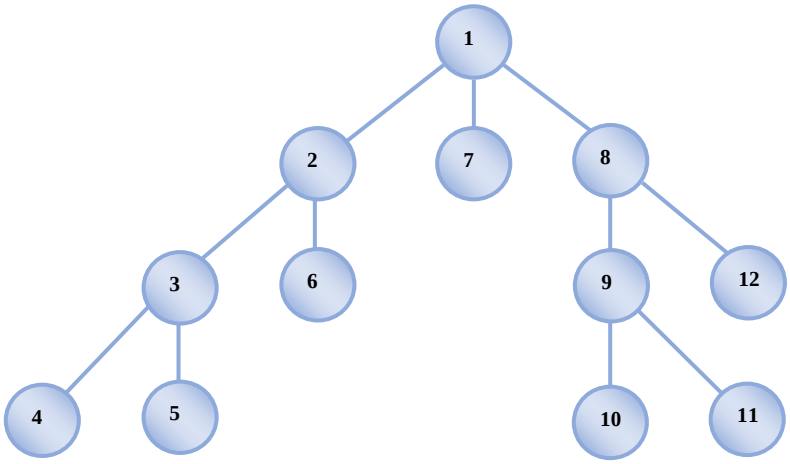


Рисунок 3.1 – Порядок обходу дерева в глибину

3.1.2 Пошук в ширину (Breadth-first search)

Пошук в ширину працює шляхом послідовного перегляду окремих рівнів графа, починаючи з вершини-джерела *u*.

Розглянемо усі ребра (*u*, *v*) вихідні з вершини *u*. Якщо наступна вершина *v* – цільова, закінчуємо пошук; інакше додаємо його в чергу. Після того, як будуть перевірені усі ребра, вихідні з вершини *u*, переходимо до наступної у черзі вершини *u*, і процес повторюється (рис 3.2).

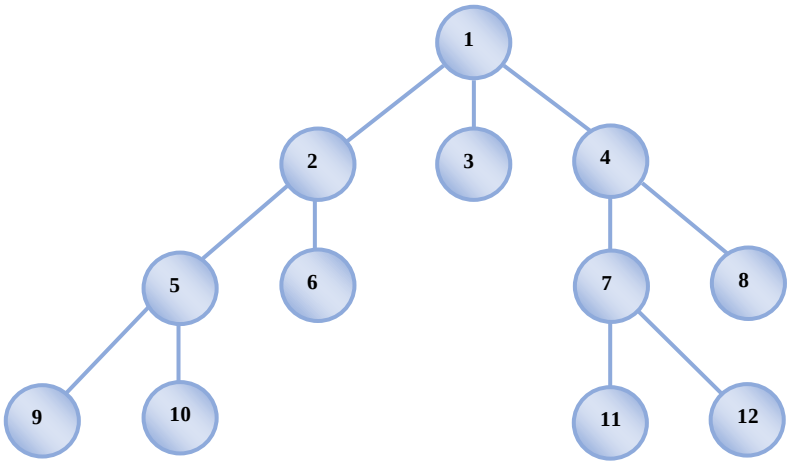


Рисунок 3.2 – Порядок обходу дерева в ширину

Не рекурсивна реалізація схожа не обхід дерева в глибину, відрізняючись застосуванням черги замість стеку, та перевіркою чи була знайдена вершина, замість того щоб відкласти цю перевірку до моменту видалення з черги [11].

Часова складність алгоритму може бути виражена як $O(|V|+|E|)$, оскільки кожна вершина та кожне ребро буде досліджено у найгіршому випадку.

3.1.3 Евристичний пошук

Евристичний пошук – стратегія пошуку рішень в просторі станів, для якої використовуються знання, які відносяться до конкретної задачі. Зазвичай забезпечують більшу швидкість в порівнянні з неінформованими методами пошуку [21].

Інформація стосовно конкретної задачі формулюється у вигляді евристичної функції. Евристична функція на кожному кроці алгоритму приймає рішення, в якому напрямку продовжувати перебір. За для цього, вона оцінює альтернативи засновані на додатковій інформації.

Прикладом евристичного пошуку слугує алгоритм A* [7]. Який знаходить маршрут найменшої вартості від початкової точки до цільової (рис 3.3).

Порядок обходу вершин визначає евристична функція відстань + вартість (зазвичай позначається як *f(x)*). Ця функція в свою чергу складається з функції вартості досягнення вершини яку розглядаємо із початкової (позначається як *g(x)* і може бути як евристичною, так і ні), та функції евристичної оцінки відстані від цільової вершини до вершини яку розглядаємо (позначається як *h(x)*).

Функція *h(x)* має бути допустимою евристичною оцінкою, тобто не повинна переоцінювати відстань до цільової вершини. Наприклад, для задачі

маршрутизації *h(x)* може представляти відстань до цілі у вигляді прямої лінії, так як це фізично найкоротша можлива відстань проміж двох точок.

За своєю сутністю, цей алгоритм є модифікацією алгоритму Дейкстри. Він досягає більшої продуктивності (за часом), за допомогою використання евристики.

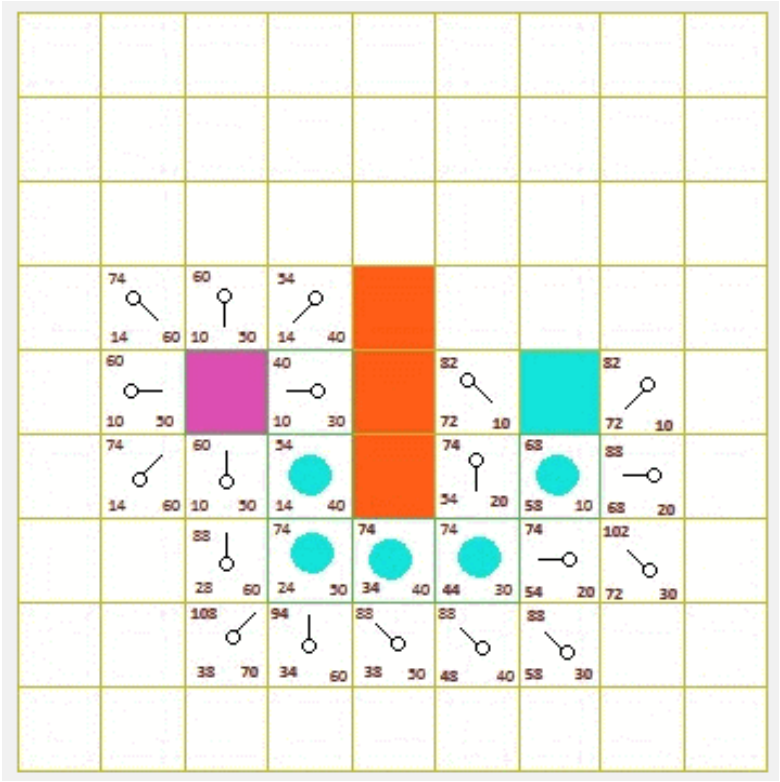


Рисунок 3.3 – Принцип дії алгоритму A*

3.2 Iterative Diffuse Path Planner

Застосування методу RRT дерев разом з лежачою в його основі «жадібної» евристикою може бути вкрай небажаним для ряду завдань планування руху. Прикладом може служити завдання автоматичної розбирання виробів, що складаються з безлічі деталей. Рух деталі поблизу вихідної конфігурації, як правило, сильно обмежено і можливе лише за рахунок малих поворотів. І навпаки, в області цільової конфігурації, що знаходиться на значній відстані, перешкоди зовсім відсутні [22].

Принцип відкладеної перевірки на безконфліктність конфігурацій в поєднанні з динамічно змінною похибкою, лежить в основі дифузійного алгоритму планування руху (*Iterative Diffuse Path Planner*). Алгоритм дозволяє значно підвищити ефективність пошуку в постановках, подібних наведеній.

В даному алгоритмі використовується ліс дерев, що ростуть. Поширення здійснюється з довільної вершини в напрямку, який визначається випадковим вектором, з кроком, який дорівнює поточному значенню похибки визначення конфліктів σ. Після включення кожної нової вершини в дерево робиться спроба його злиття. Верифікація ребер на конфлікти в процесі поширення не виконується. Значення σ ініціалізується досить великим значенням, наприклад, рівним відстані між c_{init} і c_{goal}, та ітераційно зменшується аж до заданого мінімально допустимого значення. На кожній ітерації виконується пошук шляху описаним вище способом з подальшою його верифікацією. У разі, якщо шлях виявився конфліктним, з уявлення дерев виключаються всі ребра, які не пройшли перевірку, а процедури поширення і пошуку шляху повторюються.

3.3 Мурашиний алгоритм

Мурашиний алгоритм (*ant colony optimization*, ACO) – один з ефективних поліноміальних алгоритмів для знаходження наближеного розв'язку задачі комівояжера, та подібних завдань пошуку маршрутів на графах [8]. Суть підходу полягає в аналізі та використанні моделі поведінки мурах, що шукають дороги від колонії до їжі і являє собою метаевристичну оптимізацію (рис 3.4).

Рисунок 3.4 – Поведінка мурах стала джерелом натхнення для метаевристичного методу оптимізації

У природі, мурахи деяких видів, спочатку блукають випадковим чином, у пошуках їжі. Знайшовши, вертаються до своєї колонії прокладаючи феромонові стежки. Якщо інші мурахи натрапляють на таку стежку, вони не продовжують блукати випадковим чином, а натомість йдуть слідом, вертаючись і підсилюючи його, у випадку знаходження їжі.

Проте, з часом, феромоний слід починає випаровуватися, тим самим знижуючи його привабливу міцність. Чим більше часу мурахі потрібно пройти по шляху і вернутись назад, тим більше часу феромонам доведеться випаровувати. Коротким шляхом, порівняно, користуються частіше і таким чином щільність феромонів стає вищою на коротших шляхах, ніж на довгих. Випаровування феромонів також має перевагу в уникненні конвергенції до локально оптимального рішення. Якби випаровування взагалі не було, шляхи, обрані першими мурахами, були б, як правило, надмірно привабливими для наступних. У цьому випадку дослідження простору рішення буде обмежене. Вплив випаровування феромонів в реальних мурашиних системах є незрозумілим, але він дуже важливий у штучних системах [9].

Загальний результат полягає в тому, що коли один мураха знайде хороший (тобто короткий) шлях від колонії до джерела їжі, інші мурахи скоріше будуть користуватися цим шляхом і позитивний зворотний зв'язок врешті-решт призведе до того, що багато мурах послідкують одним шляхом. Ідея алгоритму колонії мурашок полягає в тому, щоб імітувати цю поведінку за допомогою «імітованих мурашок», що гуляють навколо графа, що представляє проблему, яку потрібно вирішити.

В алгоритмі оптимізації мурашиним алгоритмом, штучний мураха – це простий обчислювальний агент, який шукає хороші рішення заданої задачі оптимізації. Щоб застосувати алгоритм колонії мурашок, проблему оптимізації потрібно перетворити на задачу пошуку найкоротшого шляху на зваженому графі. На першому етапі кожної ітерації кожен мураха стохастично будує рішення, тобто порядок, у якому слід проходити ребра на графі. На другому кроці порівнюються шляхи, знайдені різними мурахами. Останній крок складається з оновлення рівнів феромонів на кожному ребрі.

Кожен мураха повинен побудувати рішення для переміщення по графу. Щоб вибрати наступне ребро у своєму путі, мураха буде враховувати довжину кожного ребра, наявну з його поточного положення, а також відповідний рівень феромону. На кожному кроці алгоритму кожен мураха переходить із стану x у стан y, що відповідає більш повному проміжному рішенню. Таким чином, кожен мураха k обчислює набір $A_k(x)$ можливих розширень до його поточного стану в кожній ітерації та переходить до одного з них. Для мурахи k, ймовірність p_{xy}^k переходу зі стану x у стан y на поєднанні двох значень – привабливість η_{xy}, що обчислюється деякими евристичними ознаками, що вказують на апріорну бажаність цього руху та рівень сліду τ_{xy} шляху, вказуючи на те, наскільки досвідченим було в минулому зробити цей конкретний крок. Рівень стежки являє собою апостеріорну вказівку на бажаність цього ходу.

В цілому вірогідність того, що мураха k перейде зі стану x у стан y обраховується за формулою:

$$p_{xy}^{k} = \frac{(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}{\sum_{z \in allowed_x} (\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}. \quad (2.9)$$

Перший алгоритм ACO отримав назву мурашиної системи, і він мав на меті вирішити проблему комівояжера (рис 3.5), в якій мета – знайти найкоротший шлях, який би з'єднав низку міст [10, 13]. Загальний алгоритм порівняно простий і заснований на наборі мурашок, кожен з яких здійснює одну з можливих подорожей по містах. На кожному етапі мураха обирає в яке місто перейти за деякими правилами:

– він повинен відвідати кожне місто рівно один раз;

– далеке місто має менше шансів на вибір (видимість);

– чим інтенсивніше феромоновий слід прокладений на межі між двома містами, тим більша ймовірність того, що цей край буде обраний;

– завершивши свою подорож, мураха відкладає більше феромонів на всіх ребрах, якими він пройшов, якщо подорож невелика;

– після кожної ітерації сліди феромонів випаровуються.

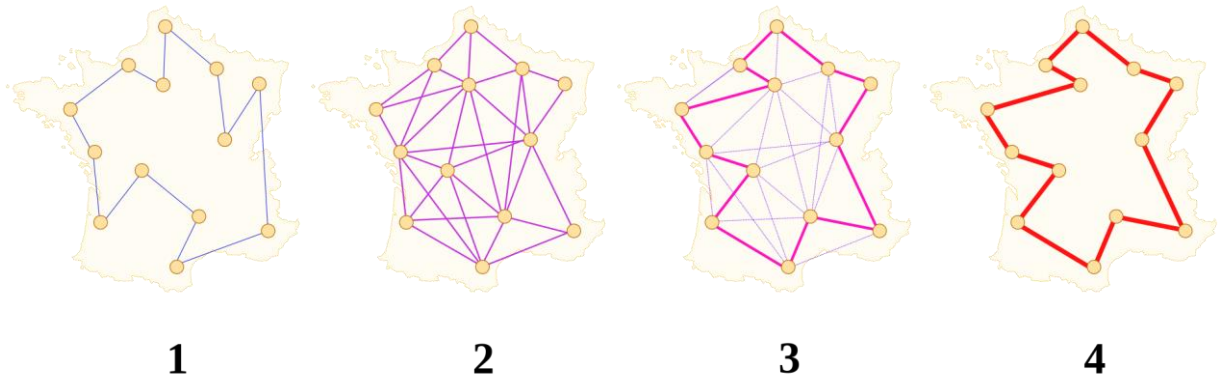


Рисунок 3.5 – Рішення задачі комівояжера мурашиним алгоритмом

4 РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Обґрунтування вибору середовища програмної реалізації

У рамках атестаційної роботи було розроблено програмний застосунок для розроблення оптимального маршруту по карті для змагання у рогейні. Для реалізації цієї програми було використано мову програмування Java, та середовище розробки Intellij IDEA. Цей вибір був обумовлений такими перевагами IDE

– вбудований відладчик коду, який дозволяє переглядати значення змінних безпосередньо у коді програми

– великий набір інструментів, таких як декомпілятор, проглядач байт-кода, *FTP*;

– інтеграція з основними системами контроля версіями, включаючи *Git, SVN, Mercurial, CVS, Perforce* та *TFS*;

– підтримка інструментів автоматичної зборки *Maven, Gradle, Ant, Gant, SBT, NPM, Webpack, Grunt, Gulp*;

– спрощує процес юніт-тестування. Представляючи інструменти запуска тестів та аналізу покриття коду і підтримує усі популярні фреймворки для тестування;

– інструмент для роботи з базами даних, включаючи редактор *SQL* та візуальний інтерфейс для перегляду даних;

Java – об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems. Програми Java зазвичай компілюються в спеціальний байт-код, тому вони можуть працювати на будь-якій віртуальній Java-машині (JVM) незалежно від комп'ютерної архітектури.

Одна з основних переваг Java – її незалежність від платформи, яка дозволяє запускати один і той-же код на різноманітних операційних системах.

Синтаксис мови Java схожий на синтаксис мов програмування *C*, та *C++*, що дозволяє використовувати цю мову програмування програмістами які знають *C*, та *C++* без значного перенавчання.

Java цілковито об'єктно орієнтована мова програмування, що полегшує розробку великих проектів великою кількістю людей.

OpenCV (*Open source computer vision*) – бібліотека комп'ютерного зору з відкритим кодом з реалізаціями алгоритмів комп'ютерного зору, обробки зображень, та чисельних алгоритмів загального призначення з відкритим кодом. Надає засоби для обробки та аналізу вмісту зображень, дозволяючи розпізнавати об'єкти на фотографіях, відстеження руху об'єктів, стандартні перетворення зображень, та застосування методів машинного навчання і виявлення загальних елементів на різних зображеннях Реалізована на мові програмування *C++*, також має підтримку таких мов програмування як *Python, Java, Ruby, Matlab, Lua*. Допускає вільне використання в академічних цілях та комерційних цілях, поширюючись на умовах ліцензії *BSD*.

OpenCV має 8 основних модулів:

– ядро функціональності – цей модуль охоплює основні структури даних, такі як *Scalar, Point, Range*. Які використовуються для розробки додатків з *OpenCV*. В доповнення до цього він також включає масив *Mat*, який використовується для зберігання зображень. В бібліотеці *Java OpenCV* цей модуль включено як пакет з ім'ям *org.opencv.imgproc*.

– відео модуль в який входять методи аналізу відео, оцінка руху, вирахування фону та відстеження руху. В бібліотеці *Java OpenCV* цей модуль включено як пакет з ім'ям *org.opencv.video*;

– модуль відео потоків вводу та виводу, містить методи захвату відео, та відеокодеки *OpenCV*. В бібліотеці *Java OpenCV* цей модуль включено як пакет з ім'ям *org.opencv.videoio*;

– модуль *calib3d* включає в себе алгоритми, стосовні основних алгоритмів геометрії деяких видів, калібрування одиночної та стерео камери, оцінки положення об'єкта, стерео відношення та елементів тримірної

реконструкції. В бібліотеці *Java OpenCV* цей модуль включено як пакет з ім’ям *org.opencv.calib3d*;

– модуль *features2d* – це модуль який включає в себе методи для виявлення та описання функцій. В бібліотеці *Java OpenCV* цей модуль включено як пакет з ім’ям *org.opencv. features2d*;

– модуль *objdetect* реалізовує алгоритми виявлення об’єктів та екземплярів попередньо виділених класів такі як обличчя, очі, люди, автомобілі та інше. В бібліотеці *Java OpenCV* цей модуль включено як пакет з ім’ям *org.opencv.objdetect*;

– модуль *highgui* – це простий в використанні інтерфейс із незабагливими можливостями інтерфейсу користувача.

Використання бібліотеки *OpenCV* дозволило швидко реалізувати програму завдяки наявності великої кількості різноманітних функцій, та зрозумілій документації з прикладами реалізації багатьох частин фунціоналу на трьох основних мовах програмування.

Tess4J оболонка для *Tesseract* реалізована у вигляді *dll* бібліотеки *Leptonica*. Представляє вже навчений словник для розпізнання англійського тексту.

4.2 Програмна реалізація

Функціональні можливості розробленої програмного застосунку дозволяють отримати інформацію з завантаженого зображення карти для рогейну за допомогою розпізнавання та побудувати граф використовуючи отриманні дані. Використовуючи *OCR Tesseract* розпізнати номера КП, на підставі яких обрахувати вартість кожного КП. Провести оптимізацію графу для видалення надлишкових ребер за для зменшення кількості варіантів шляху які переглядає програмний застосунок.

На підставі отриманого графу побудувати шлях, який задовольняє наступні критерії:

– довжина отриманого шляху не має перевищувати задану користувачем відстань;

– йдучи цим шляхом користувач отримає максимально можливу кількість балів за задану відстань;

– якщо буде знайдено декілька шляхів задовольняючих обидва попередні критерії, обрано буде шлях з меншою довжиною.

Відстань між контрольними пунктами візьмемо за пряму лінію, не враховуючи перешкоди і форми рельєфу. Для зменшення похибки введемо коефіцієнт визначений користувачем, дійсний лише для конкретної карти і користувача. У цей коефіцієнт входить: перепад рельєфу, навички орієнтування користувача (на скільки сильно він відхиляється від заданого напрямку йдучи за азимутом, вміння читати місцевість). Фізичний зміст цього коефіцієнту становить: у скільки разів більше, фактично, пройде користувач у порівнянні з абсолютною відстанню. Наприклад, за словами спортсменів – призерів змагань, зазвичай на картах з помірною формою рельєфу, вони беруть цей коефіцієнт рівним 1.3 – 1.4. Тобто, за кожен кілометр по прямій на карті, вони проходять зайві 300 – 400 метрів.

Сформулюємо це у вигляді формули:

$$L_{\max} = L_u * k \qquad (3.1)$$

де $L_{\max}$ – максимальна допустима довжина шляху, який шукатиме програма; L_u – задана користувачем відстань, яку він бажає пройти за відведений контрольний час; k – описаний вище коефіцієнт.

Існує можливість дещо підвищити фактичну точність обрахування відстані між КП за допомогою алгоритму *A**. За для цього необхідно провести розпізнання форм рельєфу використовуючи ізолінії, розпізнати зображену на карті систему доріг і використовуючи інформацію про тип

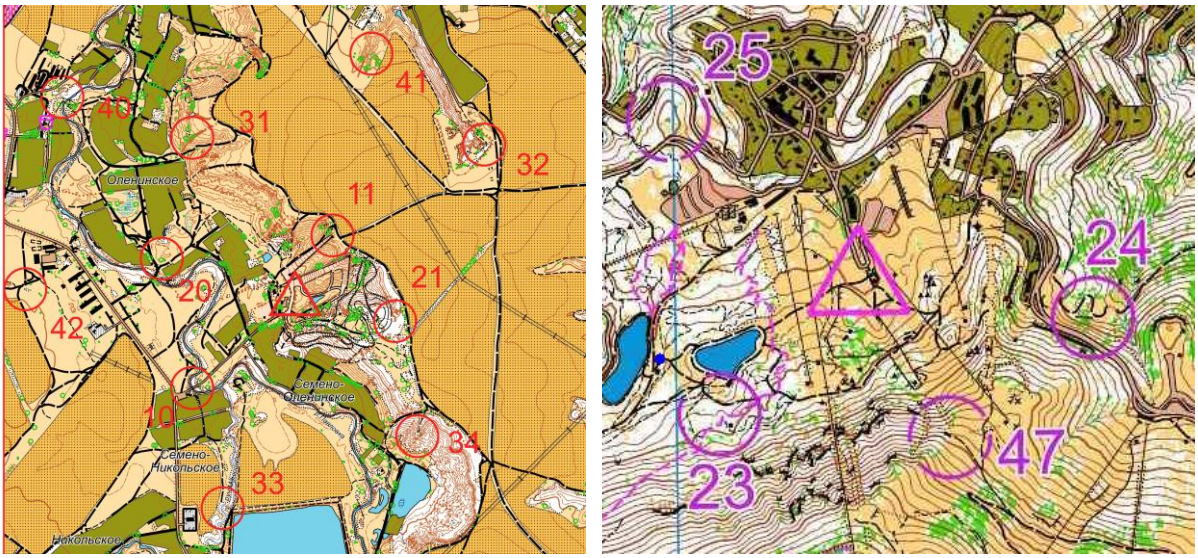
поверхні відображений кольорами пікселів, призначити до кожної умовної клітинки на карті певні бали, за допомогою яких алгоритмом *A** підрахувати відстань проміж усіх точок (можливо дещо оптимізувати і не шукати усі точки), перевагою чого методу буде урахування місць коли фактична відстань між двома сусідніми точками вагомо відрізняється від довжини прямої, якою їх можливо поєднати, наприклад, присвоївши забороненим зонам які позначено на карті штриховкою або «Х» значущим кольором, максимальні бали, щоб алгоритм не розглядав їх як можливі для шляху. В тім цей метод все рівно не дасть зовсім точних результатів – через не абсолютну точність карти і високу залежність як від сьогоденної погоди, так і від сезонної, так і від пори року. Також це істотно підвищить час роботи програми, розпізнання елементів зображення якої при даній реалізації проходить у виділеному діапазоні кольорів приведених до градацій сірого. Слід пам’ятати, що цільові зображення для роботи програми, є зображення формату А3, чи навіть більше. Через перераховані недоліки було прийнято рішення не використовувати даний метод у цій роботі.

4.2.1 Розпізнавання зображення

На етапі розпізнання зображення було розв’язано цілу низку класичних задач комп’ютерного зору, ускладнених специфікою задачі. Необхідно віднайти на зображенні кола, якими позначається місце знаходження старту позначеного на мапі символом рівностороннього трикутника, місце розташування КП позначених колами однакового діаметру, які можуть бути не суцільні, а з розривом, за для того щоб не перекривати важливу особливість місцевості як це видно на рисунку 4.1(б). Віднайти і розпізнати номера КП позначені поруч з колом у вигляді дво- або тризначних чисел.

За для цього, спочатку, було проведено попередню обробку зображення. Так як уся значуща інформація необхідна для роботі програми

позначається одним кольором є спроможність за допомогою маски видалити усю не істотну інформацію з масиву пікселів. Однак, через те, що не має чіткого стандарту оформлення карт для змагань, колір позначок на різних картах – різний, зустрічається як червоний, так і «маджента» (рис 4.1). До того ж, через нашарування декількох кольорів при перетинанні умовних позначок отримується велика кількість відтінків основного кольору, через що, при заданні неправильних порогів маски втрачається велика кількість пікселів і шукані кордони втрачають чіткість. За для вирішення цієї проблеми було розроблено два методи [37].



а) б)

Рисунок 4.1 – Класичні види кольорів які застосовують на картах:
а – червоний; б – колір маджента

Перший варіант, розробити інструмент «*eyedropper tool*», за допомогою якого користувач матиме змогу обрати колір пікселів шуканих кордонів (рис 4.4, а), але це не вирішує проблему відтінків, які у HSV діапазоні, у якому працює бібліотека OpenCV, можуть коливатися на 20 – 30% у каналах S і V, та сягають 180 у каналі H.

Другий варіант, потребує аби користувач за допомогою розробленого інструменту «rectangle marquee tool» виділив одну з шуканих окружностей.

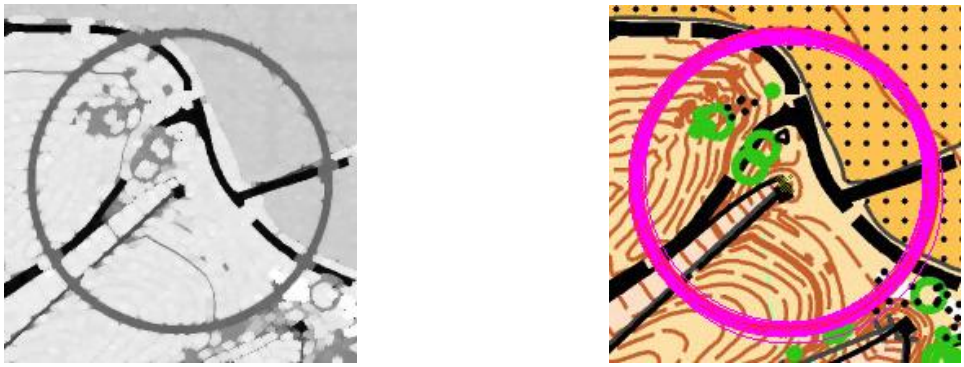
Виділений користувачем прямокутник перетворимо у зображення в градаціях сірого (рис 4.2, б) за допомогою функції бібліотеки OpenCV *Imgproc.cvtColor (Imgproc.COLOR_BGR2GRAY)*.

Для кращого виділення окружності на зображенні застосуємо до нього операцію розширення, використавши функцію *dilate* (рис 4.3, а). Після цього на виділеному прямокутнику, за допомогою функції *HoughCircles*, яка використовує метод градієнту Хафа для пошуку окружностей, віднайти усі можливі кола з радіусом який не перевищує половину розміру виділеного фрагменту і при цьому більший ніж 70% від половини його розміру [24].

Отримаємо цілу низку окружностей товщиною в один піксель майже повністю покриваючих шукане коло (рис 4.3, б). Розглянемо масив пікселів які слугують цими окружностями. На підставі цього масиву побудуємо список кольорів, відсортуємо його за кількістю надходжень, відкинемо кольори з найменшим числом входжень як випадково отримані через похибку і побудуємо маску за кольорами що залишилися (рис 4.4, б). Також, до переваг цього методу відноситься те, що ми маємо змогу визначити радіус кола, яким позначено КП і використати ці дані за для пришвидшення, та покращення результату пошуку усіх окружностей, що залишилися.

а) б)

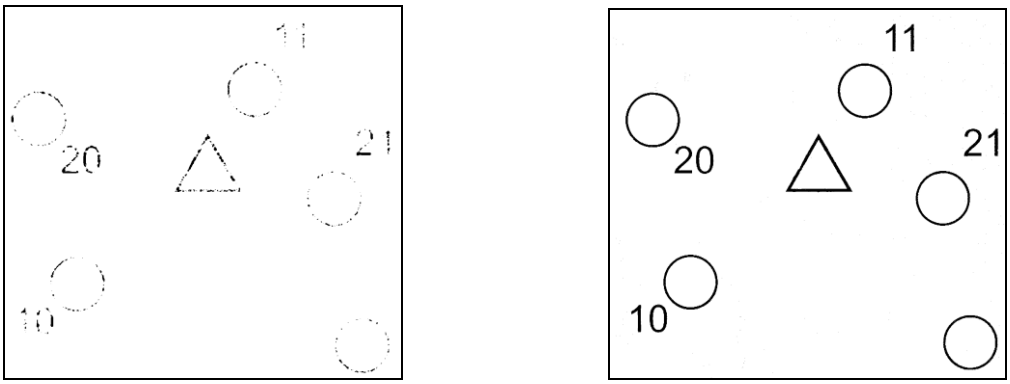
Рисунок 4.2 – Пошук кола і визначення пікселів які до нього належать:
а – задане зображення; б – зображення у градаціях сірого



а) б)

Рисунок 4.3 – Пошук кола і визначення пікселів які до нього належать: а – зображення після застосування функції *dilate*; б – знайдені окружності

Для отриманої маски, знову застосуємо метод градієнтної фільтрації – *medianBlur*. Це дозволить позбутись шумів на масці. Навколо карти на зображенні може бути рамка, того самого кольору що і значущі об’єкти, за для того щоб вона нам не заважала у подальшому необхідно її видалити з маски. Для цього використаємо функцію для пошуку прямих *HoughLines* яка використовує перетворення Хафа [24]. Задаймо в параметри функції пошуку максимальну довжину шуканої лінії як ширину, та довжину зображення, а мінімальну довжину лінії 0.5 від максимальної, Це дозволить віднайти та позбутися від усіх зайвих ліній на масці.



а) б)

Рисунок 4.4 – Інваріантне зображення фрагменту отриманої маски: а – маска з виділенням інструментом «*eyedropper tool*»; б – маска побудована за допомогою виділення списку кольорів, з пікселів кола

Після цього, приступимо до пошуку кругів якими позначаються КП. Для цього, знову використаємо функцію пошуку окружностей *HoughCircles* використовуючи вже відомий нам радіус кола [24]. В результаті виконання функції ми отримуємо координати усіх окружностей, у подальшому вони будуть вершинами графу для пошуку шляху.

За для пошуку символу фінішу, двох радіальних окружностей, знову запустимо пошук окружностей зменшивши радіус шуканого кола і перевіримо співпадіння центрів знайденого кола із уже знайденими, використавши похибку на здвиг центру рівною 0,001 від ширини і довжини зображення у пікселях відповідно.

Пошук символу старту – рівносторонній трикутник будемо проводити за допомогою функції *GeneralizedHoughBallard* яка використовує узагальнене перетворення Хафа для пошуку визначеного шаблону без його поворотів (рис. 4.5) [12, 14, 26].

Рисунок 4.5 – Використаний шаблон для пошуку позначки старту позначеної трикутником

Для пошуку та визначення значень номерів КП, видалимо з маски вже знайдені фігури. Щоб правильно розпізнати числа і зв’язати їх з відповідним КП, необхідно визначити координати центру числа. Для цього використаємо функцію розширення *dilate* із розміром ядра 80 пікселів [23]. В результаті, усі цифри в числах, зіллються в фігуру наближену до округленого чотирикутника. За допомогою функції пошуку контурів *findContours* яка використовує детектор границь Кенні, виділимо контури усіх отриманих на

зображенні фігур. Для кожного отриманого контуру, застосуємо наближення до чотирикутника з точністю ± 3 і зазначмо, що крива має бути зімкнутою. Після цього, ми знаходимо обмежуючий прямокутник і зберігаємо його центр [33, 34, 36].

Однак через те, що розмиття проводиться на зображенні у градаціях сірого, детектор границь Кенні може виділити декілька дублюючих контурів (рис. 4.6). За для того щоб позбутися усіх дублів, порівняємо координати отриманих областей, видаливши зайві, у випадку, коли їх координаті змінюються на значення не більше ніж 0,001 від ширини та довжини зображення.

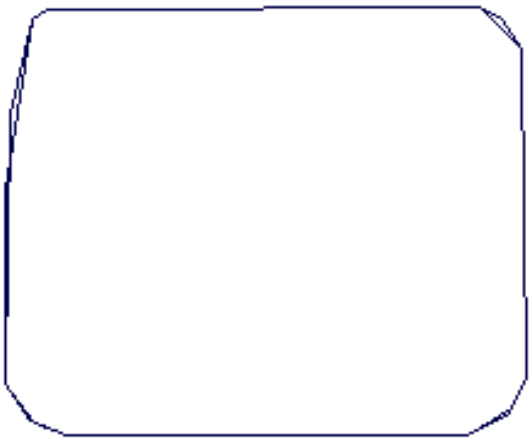


Рисунок 4.6 – Роздвоєння контурів при виділенні фігури

За допомогою отриманих прямокутників виділяємо ділянки на початковій масці і використовуючи *Tess4J*, оболонку для *Tesseract*, яка представляє, вже навчений словник для розпізнавання англійського тексту, розпізнаємо текст у кожній з виділених ділянок [27, 28].

Однак через не досконаліть системи OCR, виникає вірогідність отримання помилкового розпізнання для нулів, аби виправити це проведемо заміну усіх розпізнаних літер «О», або «о» на цифру 0 (рис. 4.7) [38].

Рисунок 4.7 – Інвертований фрагмент маски тестового зображення, OCR *Tesseract* розпізнає як 5O

Через неможливість видалити усі зайві позначки, або текст значущого кольору, серед розпізнаного тексту буде залишатися надлишковий (рис. 4.8) [35]. Використаємо той факт, що номера КП можуть бути лише дво- або тризначними числами і відкинемо увесь розпізнаний текст, що не відповідає зазначеному критерію, і не попадає до заданого діапазону. Тепер, використовуючи формулу (3.2) знайдемо відстань між усіма КП, та визначеними номера і зіставимо між собою пари з найменшою відстанню.

$$\vec{V} = \sqrt{\left(x_2 - x_1\right)^2 + \left(y_2 - y_1\right)^2}. \qquad (3.2)$$

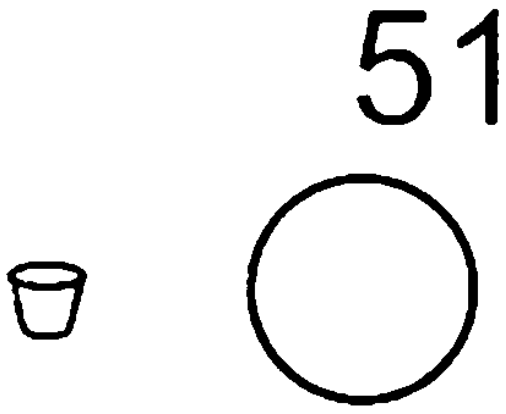


Рисунок 4.8 – Інвертований фрагмент маски; символ «*годівничка*» розпізнається як вісімка

4.2.2 Формування та оптимізація графа

Отже, ми отримали усі необхідні нам данні із зображення. На підставі цих даних побудуємо граф, за вершини графа візьмемо розпізнані нами кола. Вагою ребра будемо вважати його безпосередню довжину обраховану за допомогою формули (3.2). Втім, при поєднанні усіх вершин проміж собою ми отримуємо повний граф, який буде містити надлишкові ребра. Наприклад, у ситуації, коли ребро з однієї вершини до іншої буде проходити, безпосередньо, через ще одну вершину. Так як, з точки зору логіки, вже проходячи через вершину не відвідати її, не має сенсу – такі ребра необхідно видалити із матриці суміжності (рис. 4.9). За для цього скористаймося нерівністю трикутника:

$$\triangle ABC : |AC| = |AB| + |BC|. \tag{3.3}$$

На підставі цієї теореми виведемо формулу нерівності для обрахування подібності прямій, шляху проведеного через додаткову вершину:

$$100\left(1 - \frac{|AC|}{|AB| + |BC|}\right) < x. \tag{3.4}$$

де x – відсоток при якому шлях AC буде визнано не маючим сенсу і видаленим із матриці суміжності. Фізичний зміст x – у скільки відсотків буде довшим шлях через додаткову точку, у порівнянні зі шляхом на пряму. На тестовому зображені прийняття $x=5\%$ для 36 КП і 630 варіантів їх поєднань, дозволило зменшити цю кількість до 152, що становить більше ніж чотирикратне зменшення кількості поєднань.

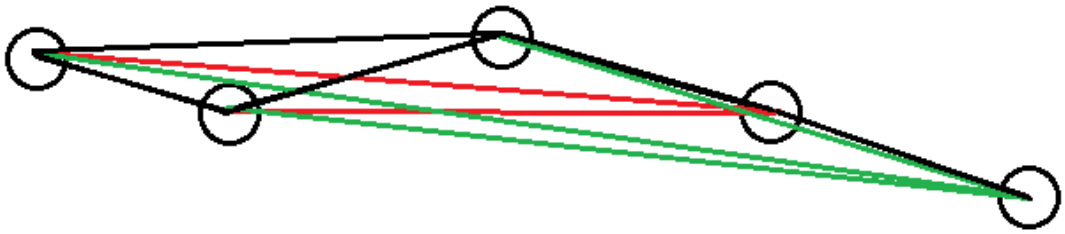


Рисунок 4.9 – Демонстрація роботи методу відкидання надлишкових зав’язків, де чорним кольором зазначені ребра, що залишилися

Проте, через те, що між двома КП може знаходитися перешкода яку неможливо подолати (прірва, надміру стрімка річка) або заборонена умовами змагань територія (орні поля, сміттєзвалища, річки), дамо користувачеві змогу видалити непотрібне з його точки зору поєднання двох вершин чи додати нове, необхідне на його думку.

За специфікою задачі у кожної вершини є додатковий параметр, що становить кількість балів, які отримуються за відвідування даної вершини. Ці бали визначаються за першою цифрою номера КП, якщо він двозначний та першими двома у випадку тризначного номеру.

4.2.3 Пошук маршруту

Так як, за умовою вірогідність виникнення задачі типу «комівояжеру» майже не можлива, віднесемо цю ситуацію до найгіршого випадку і не будемо орієнтуватися на неї при рішенні. До того ж, після того як ми відкинули надлишкові ребра, це суттєво зменшило кількість варіантів прокладання шляху. Через те, що шуканий маршрут не підпадає під категорію задач про найкоротший шлях у класичному своєму вигляді, до його розв’язку не можливо застосувати такі класичні алгоритми як Дейкстри, Беллмана-Форда, та ін.

Будуючи шлях, перш за все будемо опиратись на досягнення фінішу. Якщо з чергової вершини фініш не можливо досягти за відстань, що залишилась, то розглядати подальші вершини непотрібно і необхідно вернутись на попередню вершину.

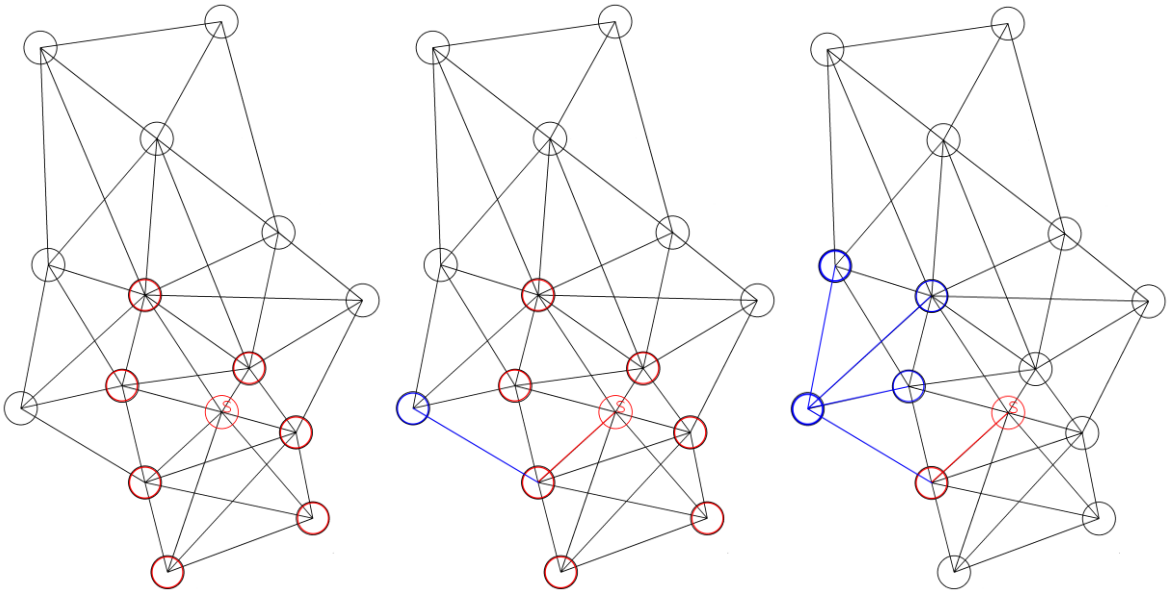
В тім, можливий варіант, коли із вершини немає зв’язку з фінішом, а усі сусідні вершини, вже виходять за граничну відстань, однак по прямій фініш можливо досягти, то такий маршрут також не будемо враховувати, так як він не може бути оптимальним, через наявність перегонів без КП. Тобто на передостанньому кроці побудованого шляху має бути вершина суміжна з фінішем.

Розробимо алгоритм для пошуку шляху на графі:

- на першому кроці, у випадку коли точки старту та фінішу не збігаються, перевіримо чи можливо взагалі досягти фінішу з точки старту за визначену користувачем відстань, для перевірки візьмемо найкоротшу можливу відстань – пряму;
- на другому кроці, за допомогою методу пошуку в ширину знайдемо усі сусідні вершини (рис. 4.10, а), перевіривши досяжність фінішу по прямій для кожної з них, у випадку, коли за відстань, що залишилася не можливо досягти фінішу, помічаємо вершину як недосяжну;
- на третьому кроці, оберемо із усіх суміжних вершин ліву і перейдемо до неї (рис. 4.10, б), після чого повторимо для неї другий крок(рис. 4.10, в);
- коли у вершини в результаті виконання другого кроку не буде знайдено жодної досяжної вершини, повернемося до попередньої вершини та виберемо наступну вершину зліва на право у таблиці суміжності;
- при досягненні фінішу запам’ятовуємо цей маршрут і вертаємось до попередньої вершини, виконуючи для неї попередній крок;
- продовжуємо до того моменту колі вернувшись до старту не знайдемо жодного можливого варіанту руху;

- сумуємо кількість балів отриманих за відвідані вершини, для кожного з побудованих маршрутів і при наявності двох чи більше рівнозначних обираємо найкоротший з них.

Цей алгоритм можливо, дещо оптимізувати з точки зору кількості збережених даних, одразу підраховуючи кількість балів отримуваних за маршрут і зберігаючи лише той який отримав найбільшу кількість балів або якщо таких маршрутів декілька то зберігати лише той у якого коротший шлях.



а б с

Рисунок 4.10 – Принцип роботи алгоритму: а – другий крок; б, с – третій крок

До переваг цього алгоритму можна віднести, те що у випадку, коли з точки не можливо досягти фінішу, алгоритм не буде розглядати інші вершини які знаходяться далі по дереву, а відкине усю гілку, що дозволяє істотно зменшити час на прохід алгоритмом усіх доступних варіантів.

4.3 Інструкція користувача

Запустіть виконання програми RogeinNavigator. У відкритому вікні натисніть Menu/OpenFile. Оберіть потрібне зображення у форматі png .

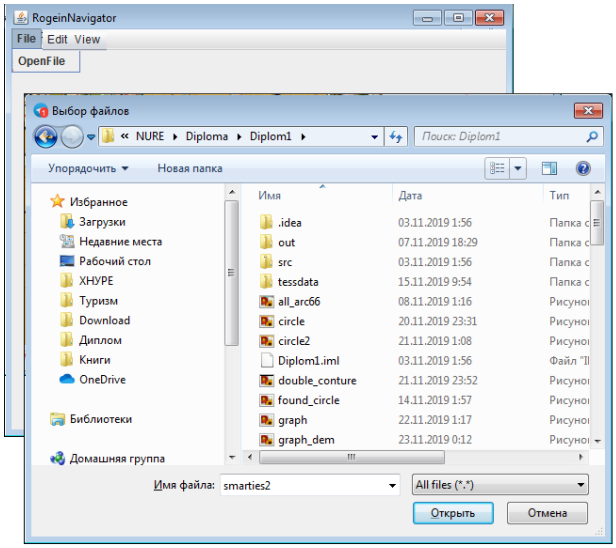


Рисунок 4.11 – OpenFile dialog

У відкритому вікні з’явиться відкрите зображення, яке можна приблизити чи віддалити, або пресувати вздовж екрану за допомогою миші або «скролу». За допомогою інструменту «rectangle marquee tool» виділити одну цілих окружностей, якими позначаються КП, як це зазначено на рисунку 4.12.

Рисунок 4.12 – Виділене КП

Після виділення кола, користувач може переглянути отриману маску кольорів, аби упевнитись у точності розпізнання зображення. За для цього можливо відкрити вікно попереднього перегляду Menu/View/Preview, де можливо подивитися виділену маску і трохи скорегувати діапазон кольорів (рис 4.13).

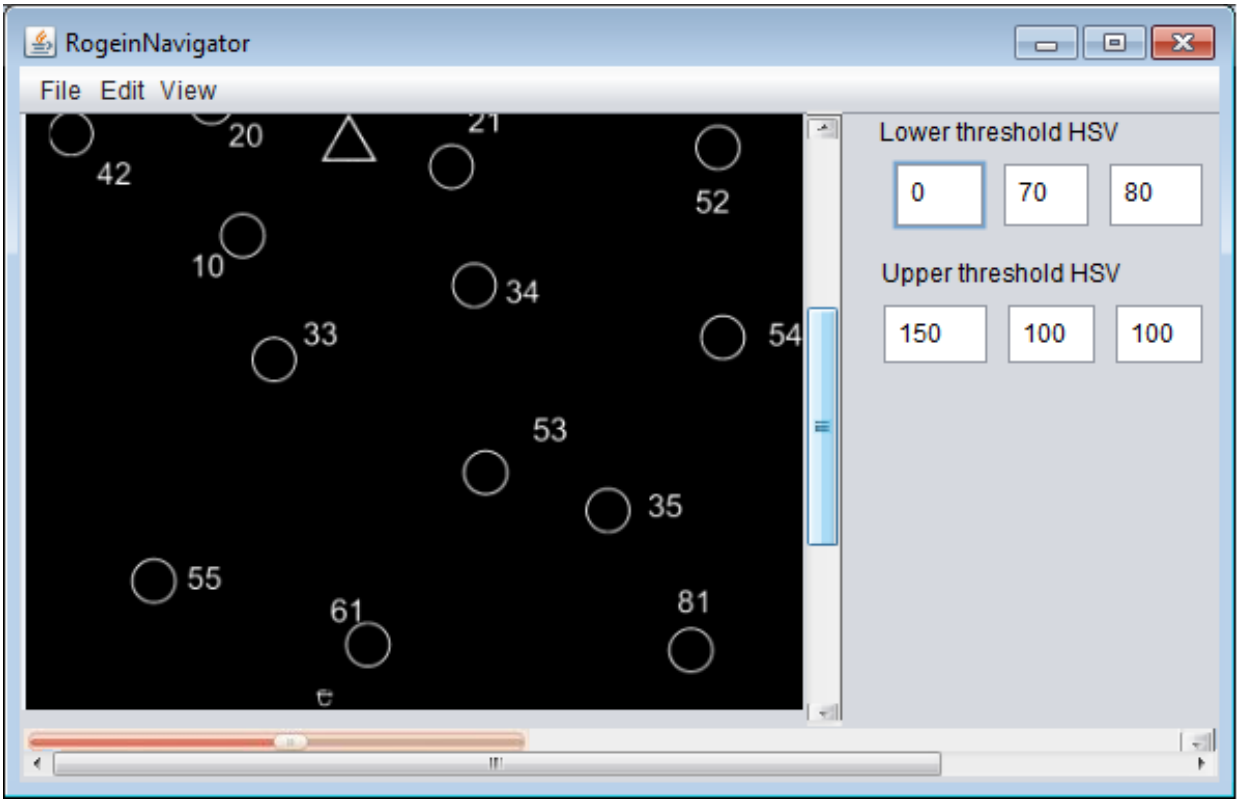


Рисунок 4.13 – Попередній перегляд виділених кольорів

Після отримання задовільного виділення кольорів, необхідно запустити процес побудови графу, який також можливо подивитись та скорегувати деякі поєднання вершин сугубо за розумінням необхідності цього користувачем, наприклад при неможливості пересуватися про між цих вершин. На випадок некоректного розпізнання тексту доступна функція корегування номерів КП.

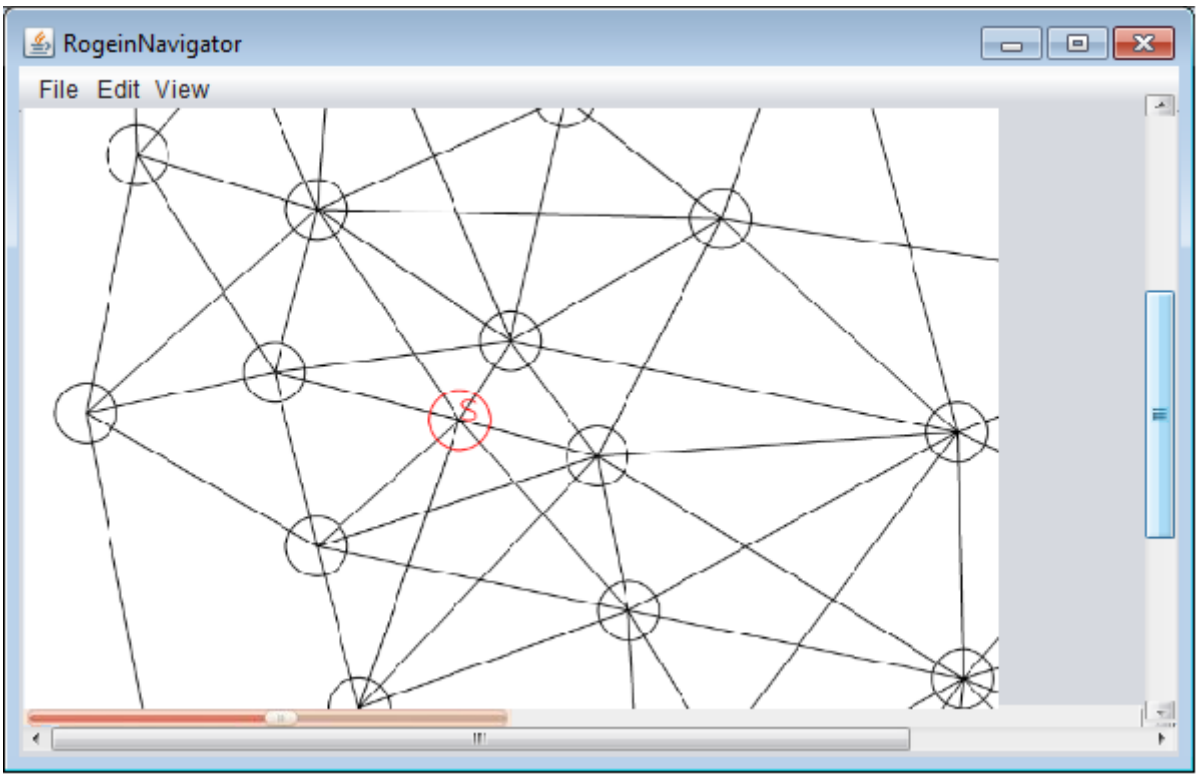


Рисунок 4.14 – Попередній перегляд Графу

За допомогою Menu/File/Settings користувачу необхідно задати максимальний допускний маршрут, та коефіцієнт для карти. Після чого натиснути кнопку RUN. У результаті виконання програми, через деякий час буде побудовано маршрут та відображено у вікні.

ВИСНОВКИ

У рамках атестаційної роботи було проведено дослідження методів побудови оптимального шляху. Було розроблено і реалізовано метод пошуку оптимального шляху по графу з різною вартістю вершин.

За для цього, спочатку було розпізнано зображення. На зображенні було відшукано такі об’єкти як кола, трикутник та лінії. За для цього було використано алгоритм градієнтного перетворення Хафа для виявлення окружностей. Перетворення Хафа для виявлення ліній та прямокутників, також застосовано метод узагальненого перетворення Хафа для пошуку трикутника з використанням шаболну. Також було виявлено текст на зображенні і розпізнано за допомогою системи *OCR Tesseract*. За для виявлення тексту, з маски було видалено усі знайдені об’єкти. Після чого було застосовано операцію розширення, в результаті чого окремі цифри в числах злилися у фігуру наближену до чотирикутника, завдяки чому з’явилася можливість виділити одразу число та вирахувати координати цього числа, для зв’язування його з відповідним КП. На підставі координат КП які були позначені колами, було підраховано відстань між КП у пікселях. За допомогою даних про роздільну здатність зображення та введених користувачем масштабу карти та формату карти відстань переводиться у метри.

На підставі отриманих даних було побудовано граф, де вершинами виступають КП, а вагами ребер буде відстань проміж КП. Кількість балів за відвідування КП, виступає додатковим параметром для вершин.

Було проведено аналіз отриманого графу і на підставі спостережень виявлено можливість його оптимізувати. За для цього було розроблено формулу засновану на нерівності трикутника. За допомогою цієї формули стало можливо вилучення надлишкових зав’язків, завдяки чому кількість ребер на графі було зменшено у близько 5 разів.

Було проведено аналіз та порівняння декількох методів розв’язання поставленої задачі. Та вирішено яким краще всього користуватися, виходячи із умов задачі, яка не потребує знаходження розв’язку у реальному часі. За для виміру відстані проміж вершин використовується абсолютна відстань – пряма лінія, через що, нехтується рельєф місцевості, однак це компенсується спеціальним коефіцієнтом заданим користувачем, до якого входять як і форми рельєфу, так і власні навички користувача. Метод обрахування відстані за допомогою алгоритму *A** було прийнято не маючим сенсу, через значне збільшення затрат на обробку зображення та незначний виграш у точності, через похибки на карті, та залежність алгоритму від точного опису місцевості балами прохідності, які в реаліях даної задачі можуть бути не постійними і залежать від мінливих зовнішніх факторів, таких як погода, та пора року.

Для побудови оптимального шляху було використано методу обходу в глибину з поверненнями, який гарантовано дасть відповідь, хоча і не за найменший час. За для цього на кожній ітерації алгоритму знаходяться усі досяжні з даної точки вершини із яких за задану відстань можливо потрапити на фініш.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Baldwin, D. (2019). Starting in rogaining. Retrieved 10 September 2019, from https://www.rogaining.com/index.php?option=com_content&view=article&id=14&Itemid=178

2. Планирование в рогейне - это высшая математика. (2019). Retrieved 10 September 2019, from https://ziurkenas.livejournal.com/38843.html

3. Leavers, V. (1992). Shape detection in computer vision using the Hough transform. London, Angleterre: Springer-Verlag.

4. Agaian, S. (2011). Shape-dependent canny edge detector. Optical Engineering, 50(8), 087008. doi: 10.1117/1.3613941

5. Ellis, T., Abbood, A., & Brillault, B. (1992). Ellipse detection and matching with uncertainty. *Image And Vision Computing*, *10*(5), 271-276. doi: 10.1016/0262-8856(92)90041-z

6. Wiley. (2009). Template Matching Techniques in Computer Vision.

7. Red Blob Games: Introduction to A*. (2019). Retrieved 26 November 2019, from https://www.redblobgames.com/pathfinding/a-star/introduction.html

8. Ant Colony Optimization Solutions for Path Planning of Logistic. (2018). *International Journal Of Technology And Engineering Studies*, *4*(3). doi: 10.20469/ijtes.4.10003-3

9. Технологический институт Федерального государственного образовательного учреждения высшего профессионального образования Южный федеральный университет в г. Таганроге. (2008). О некоторых модификациях муравьиного алгоритма. Россия, Таганрог.

10. Arora, P., Kaur, H., & Agrawal, P. (2011). Analysis and synthesis of enhanced ant colony optimization with the traditional ant colony optimization to solve travelling sales person problem. International journal of computers & technology, 1(1), 88-92. doi: 10.24297/ijct.v2i2b.2637

11. Ardiansyah, R. (2019). Implementasi Algoritma Hybrid Depth-First Search dan Breadth-First Search pada Sistem Repository Dokumen Hasil Karya

Mahasiswa (Studi Kasus Pada Jurusan Teknologi Informasi Universitas Tadulako). Scientico : Computer Science And Informatics Journal, 1(1), 65. doi: 10.22487/j26204118.2018.v1.i1.11903

12. Ballard, D. (1979). Generalizing the Hough transform to detect arbitrary shapes. Rochester, N.Y.: Department of Computer Science, University of Rochester.

13. Dorigo, M., & Gambardella, L. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions On Evolutionary Computation, 1(1), 53-66. doi: 10.1109/4235.585892

14. Ruiz-Shulcloper, J., & Sanniti di Baja, G. Progress in pattern recognition, image analysis, computer vision, and applications.

15. Newell, A., & Simon, H. (1976). Computer science as empirical inquiry: symbols and search. Communications Of The ACM, 19(3), 113-126. doi: 10.1145/360018.360022

16. Gutin, G., Yeo, A., & Zverovich, A. (2001). Traveling Salesman Should not be Greedy: Domination Analysis of Greedy-Type Heuristics for the TSP. BRICS Report Series, 8(6). doi: 10.7146/brics.v8i6.20460

17. Agaian, S. (2011). Shape-dependent canny edge detector. Optical Engineering, 50(8), 087008. doi: 10.1117/1.3613941

18. Mastorakis, N. (2011). Pathway modeling and algorithm research. Hauppauge, N.Y.: Nova Science Publishers.

19. Optical Character Recognition (OCR) - How it works. (2019). Retrieved 06 November 2019, from https://www.nicomsoft.com/optical-character-recognition-ocr-how-it-works/

20. Wilson, R., Nikitina, I., & Gavrilov, G. (1977). Введение в теорию графов. Москва: издательство «Мир.»

21. Schwinn, B. (1989). Parallelität und intelligentes Backtracking in logischen Programmen / Parallelism and Intelligent Backtracking in Logic Programs. It - Information Technology, 31(4). doi: 10.1524/itit.1989.31.4.239

22. Ahuja, R., Mehlhorn, K., Orlin, J., & Tarjan, R. (1990). Faster algorithms for the shortest path problem. Journal Of The ACM, 37(2), 213-223. doi: 10.1145/77600.77615

23. OpenCV: Eroding and Dilating. (2019). Retrieved 20 October 2019, from https://docs.opencv.org/4.1.2/db/df6/tutorial_erosion_dilatation.html

24. OpenCV: Hough Line Transform. (2019). Retrieved 20 October 2019, from https://docs.opencv.org/4.1.2/d9/db0/tutorial_hough_lines.html

25. OpenCV: Hough Circle Transform. (2019). Retrieved 20 October 2019, from https://docs.opencv.org/4.1.2/d4/d70/tutorial_hough_circle.html

26. OpenCV: cv::GeneralizedHough Class Reference. (2019). Retrieved 21 October 2019, from https://docs.opencv.org/4.1.2/d7/dd4/classcv_1_1GeneralizedHough.html

27. Tess4J. (2019). Retrieved 04 November 2019, from http://tess4j.sourceforge.net/usage.html

28. Leptonika. (2019). Retrieved 04 November 2019, from http://www.leptonica.org/

29. OCR Introduction. (2019). Retrieved 04 November 2019, from http://www.dataid.com/aboutocr.htm

30. Geetha, C. (2019). Optical Character Recognition with Tesseract. Journal of mechanics of continua and mathematical sciences, 1(2). doi: 10.26782/jmcms.spl.2019.08.00006

31. Путятин, Е. П. (2018). Нормализация и распознавание изображений. *URL: http://sumschool. sumdu. edu. ua/is-02/rus/lectures/pytyatin/pytyatin. htm*.

32. Путятін, Є. П., Гороховатський, В. О., & Матат, О. О. (2006). Методи та алгоритми комп'ютерного зору: навч. посіб. *Харків: ТОВ «Компанія СМІТ»*.

33. Bogucharskiy, S., & Mashtalir, S. (2015). Modified x-means clustering methods in image segmentation problems. Electrical and computer systems, 20(96), 106-111. doi: 10.15276/eltecs.20.96.2015.14

34. Bogucharskiy, S., & Mashtalir, V. (2015, September). Image segmentation via X-means under overlapping classes. In 2015 Xth International Scientific and Technical Conference" Computer Sciences and Information Technologies"(CSIT) (pp. 45-47). IEEE.

35. Гороховатський, В. О., & Солодченко, К. Г. (2018). Застосування апарату аналізу та оброблення бітових даних у методах класифікації зображень за множиною ключових точок. Системи управління, навігації та зв'язку, (2), 63-67.

36. Богучарский, С. И., Каграманян, А. Г., & Машталир, С. В. (2015). Модификация метода J-средних в задачах фрагментной сегментации изображений.

37. Егорова, Е. А., Киношенко, Д. К., Машталир, С. В., & Шляхов, Д. В. (2006). Метрическое сравнение результатов сегментации изображений. Радиоэлектроника и информатика, (2).

38. Луганский, А. М., & Машталир, С. В. (2007). Анализ классификации текстов с использованием весовых коэффициентов.