Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Знаходження прямих ліній на зображенні за допомогою методу Хафа

Текстова частина до курсової роботи

за спеціальністю «Інженерія програмного забезпечення»- 121

Керівник курсової роботи
Кандидат технічних наук,
Старший викладач
Бучко О. А.

_____
(підпис)
“____” _____ 2020 р.

Виконав студент ІПЗ-3:
Вєдєніков М.В.
“____” _____ 2020 р.

Київ 2020

«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

к. ф.-м. н. С. С. Гороховський

_____

(підпис)

„____”_____2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

Студента Вєдєніков Микита Володимирович факультету інформатики 3 курсу

Тема: Знаходження прямих ліній на зображенні за допомогою методу Хафа

Зміст ТЧ до курсової роботи:

Календарний план

Вступ

Частина 1: Аналіз алгоритму та вхідних даних

Частина 2: Модифікації базового методу Хафа

Частина 3: Приклади використання методу Хафа в комп’ютерному баченні

Висновки

Список літератури

Додатки (за необхідністю)

Дата видачі “____” _____ 2019 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Зміст

Календарний план виконання робот

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	09.10.2019	
2.	Пошук літератури за темою роботи	10.11.2019	
3.	Ознайомлення з науковою літературою	11.12.2019	
4.	Повторення синтаксису C++	12.01.2020	
5.	Огляд існуючих методів згладжування	30.01.2020	
6.	Огляд існуючих операторів для пошуку контурів	15.02.2020	
7.	Огляд існуючих модифікацій алгоритму Хафа	22.02.2020	
8.	Визначення структури програми	01.03.2020	
10.	Реалізація практичної частини	11.03.2020	
11.	Написання першої частини курсової роботи	10.04.2020	
12.	Написання другої частини курсової роботи	24.04.2020	
13.	Написання третьої частини курсової роботи	29.04.2020	
14.	Написання висновків курсової роботи	05.05.2020	
15.	Перегляд змісту роботи з керівником	06.05.2020	
16.	Внесення змін до роботи	07.05.2020	
17.	Створення презентації	09.05.2020	
18.	Завантаження курсової роботи	10.05.2020	

Студент Вєдєніков М.В.

Керівник Бучко О.А.

“____” _____

Вступ

На сьогоднішній день системи комп’ютерного бачення набули дуже широкого використання. Для багатьох людей ці системи значно облегшують життя, і, якщо раніше вони використовувалися лише в військових або медичних цілях, то зараз існує величезна кількість приладів, які б не могли виконувати свої функції без можливостей, що надає комп’ютерне бачення.

Майже у всіх таких системах виконується перехід від отриманого зображення до вихідних даних того типу, які необхідні на програмному рівні. Це може бути що завгодно: від класів та об’єктів цих класів, до ознак знайдених ліній на зображенні, тощо. Для більшості цих систем необхідним є етап виділення основних примітивних кривих – ліній, точок, кіл, еліпсів, прямокутників, квадратів або більш складних кривих.

Етап переходу є одним з найважливіших етапів у розпізнаванні об’єктів, оскільки найменша неточність може вплинути на правильність роботи системи та призвести до фатальних наслідків. Один з найефективніших алгоритмів для реалізації цього етапу є метод, що базується на перетворенні Хафа.

Метод Хафа є основним для пошуку кривих різних типів на зображенні. Алгоритм базового перетворення Хафа спочатку був запропонований лише для виявлення прямих. Згодом його функціонал був розширений до ідентифікації довільних форм, найчастіше еліпсів та кіл. Також слід зазначити, що на сьогоднішній день його використовують для знаходження об’єктів в 3-д вимірі.

Постановка задачі

Реалізація методу, який надасть можливість знаходження прямих ліній на зображенні.

Вимоги:

1. Проаналізувати необхідні етапи попередньої обробки зображення;

2. Визначити особливості різних фільтрів для згладжування;
3. Розглянути алгоритми виділення контурів на зображенні;
4. Реалізувати алгоритм знаходження прямих ліній;
5. Розібрати існуючі модифікації базового алгоритму Хафа;
6. Показати приклади використання методу Хафа в комп’ютерному баченні.

1. Розділ 1. Аналіз алгоритму та вхідних даних

Якість виконання алгоритму Хафа для знаходження прямих ліній дуже залежить від якості наданого зображення. Одним із найбільш важливих факторів є рівень шумів на зображенні, які ускладнюють виявлення ліній. На результат також впливає якість виділення контурів об’єкта. Тому для більш якісного виявлення ліній методом Хафа, перед цим до вхідного зображення зазвичай застосовується декілька дій:

1) Перетворення вхідного кольорового зображення у відтінки сірого;
2) Прибирання шумів на зображенні за допомогою згладжуючих фільтрів;
3) Перетворення зображення в бінарне (чорно-біле);
4) Виділення контурів предметів на зображенні за допомогою застосування операторів детекторів ребер;

Це і є приклад оптимального алгоритму обробки зображення перед використанням методу Хафа. Допускаються додавання або зміна порядку даних етапів.

1.1. Фільтрація шумів за допомогою згладжування

Згладжування є невід’ємною частиною будь-якого алгоритму для виявлення ліній або інших типів об’єктів на зображенні. Причиною для цього є велика кількість шумів, які виникають внаслідок переносу зображення з будь-якого фізичного пристрою на комп’ютер. Ці шуми являються головною перепоною для отримання контурів об’єктів. Не менш важливим є те, що при застосуванні фільтрів виникає ефект розмиття менш важливих («другорядних») частин зображення. Незважаючи на корисність, у згладжування є й недолік: при його накладанні розмиваються й потрібні нам контури. Тому його зазвичай не використовують в медицині, де контури є найважливішими[1].

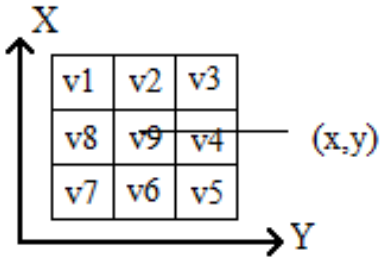

Рисунок 1.1. Маска фільтра

Основний принцип накладання масок фільтрації полягає у заміні початкових значень на зображенні усередненими значеннями за маскою фільтра. Зазвичай маска фільтра представляє собою невеликий двомірний масив (рис. 1.1) (найчастіше розмірністю 3x3), коефіцієнти в якій, залежать від мети з якою накладається фільтр. Нехай на рис.1 значення v1,v2,v3…v9 – коефіцієнти маски для пікселя (x, y) та восьми сусідніх для нього пікселів. Тоді алгоритм накладання маски можна записати формулою:

$$h[f(x,y)] = v_1 * f(x-1, y-1) + v_2 * f(x-1, y) + v_3 * f(x-1, y+1) + v_4 * f(x, y+1) + v_5 * f(x+1, y+1) + v_6 * f(x+1, y) + v_7 * f(x+1, y-1) + v_8 * f(x, y-1) + v_9 * f(x, y) \quad (1.1)$$

За алгоритмом створення коефіцієнтів відрізняють основні типи фільтрації.

Низькочастотні та високочастотні фільтри

Низькочастотні фільтри виконують функцію послаблення високочастотних та посилення низькочастотних компонентів відповідно. Прикладами високочастотних компонентів є різкі перепади яскравості та шуми на зображеннях. Ідея видалення високочастотних компонентів здебільшого направлена на більш детальне дослідження низькочастотних деталей на зображенні. [2]. Прикладами низькочастотних масок є наступні:

$$LP1 = \frac{1}{9}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix};\ LP2 = \frac{1}{10}\begin{pmatrix}1 & 1 & 1\\ 1 & 2 & 1\\ 1 & 1 & 1\end{pmatrix};LP3 = \frac{1}{16}\begin{pmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{pmatrix}$$

(1.2)

Високочастотні фільтри використовуються для посилення високочастотних компонентів та послаблення низькочастотних. Ця фільтрація здебільшого використовується для знаходження контурів та ребер об'єктів на зображенні. Інколи при накладанні маски цього типу збільшується різкість зображення за рахунок виділення шумів (які теж є високочастотним компонентом)[2]. Прикладами високочастотних масок є наступні:

$$HP1 = \begin{pmatrix}-1 & -1 & -1\\ -1 & 9 & -1\\ -1 & -1 & -1\end{pmatrix};HP2 = \begin{pmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{pmatrix};HP3 = \begin{pmatrix}1 & -2 & 1\\ -2 & 5 & -2\\ 1 & -2 & 1\end{pmatrix}$$

(1.3)

Медіанний фільтр

Медіанний фільтр використовується з цілю прибрати імпульсні шуми. Він дозволяє зберегти перепади відтінків та відкинута аномальні значення завдяки властивості зберігання монотонних послідовностей. Медіанний фільтр не є найефективнішим, оскільки при його накладанні може виникнути ефект гаусового шуму. І показують найкращий результат при низькій щільності шумів на зображенні[1].

Принцип роботи медіанного фільтру полягає в знаходженні середнього значення пікселю в порівнянні з сусідніми та присвоєння цього значення поточному пікселю. Важливим є те, що розмірність області фільтру має бути непарною. Наприклад, медіанна фільтрація для області розміром 5x5 буде мати наступний вигляд:

$$\begin{pmatrix} 12 & 17 & 22 & 23 & 11 \\ 3 & 13 & 24 & 8 & 19 \\ 4 & 5 & 14 & 25 & 21 \\ 1 & 2 & 20 & 15 & 18 \\ 6 & 7 & 9 & 10 & 16 \end{pmatrix}$$

$$\text{,де x (поточне значення)} = 14$$

(1.4)

Потім відбувається сортування значень елементів області

1, 2, 3. 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25

(1.5)

З отриманого масиву обирається середній елемент, в нашому випадку при розмірі 5x5 – 13 елемент масиву, і присвоюється поточному пікселю.

Фільтр Гауса

Фільтр Гауса є одним з найпопулярніших фільтрів для згладжування. Він широко використовується в різних графічних редакторах або алгоритмах комп'ютерного бачення для зниження рівня шумів на зображенні. Основна відмінність від інших фільтрів є те, що цей фільтр використовує нормальний розподіл (який відомий під назвою розподіл Гауса, звідки і назва фільтру) для обрахунку перетворень, які застосовуються до кожного пікселю.[3]

Розподіл Гауса для одновимірного простору має вигляд:

$$G(x) = \frac{1}{\sqrt[2]{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

, де σ- стандартне середньоквадратичне гаусове відхилення і x- координата досліджуваного пікселя

(1.6)

Розподіл для x=0 проілюстровано на рисунку 1.2

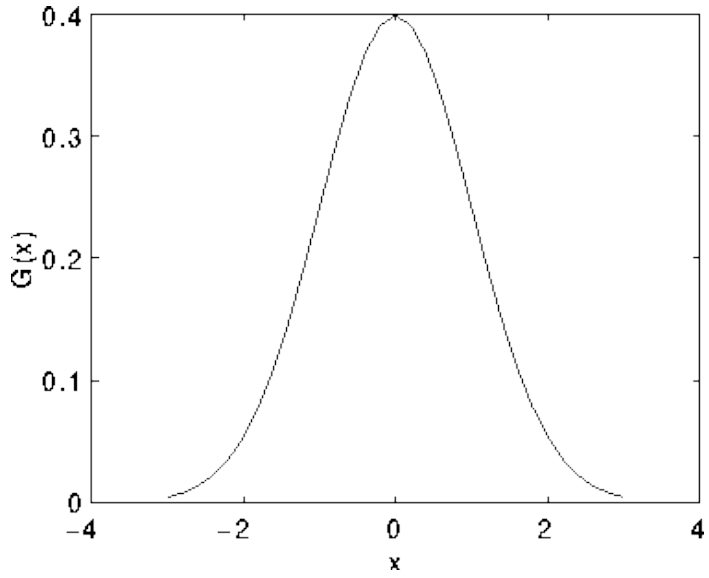

Рисунок 1.2. Графік розподілу Гауса для одновимірного простору при x=0 і $\sigma = 1$ [3]

Для двовимірного простору формула розподілу Гауса буде мати такий вигляд:

$$G(x,y) = \frac{1}{2\pi\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

, де σ- стандартне середньоквадратичне гаусове відхилення і x,y- координати досліджуваного пікселя в двовимірному просторі

(1.11) [3]

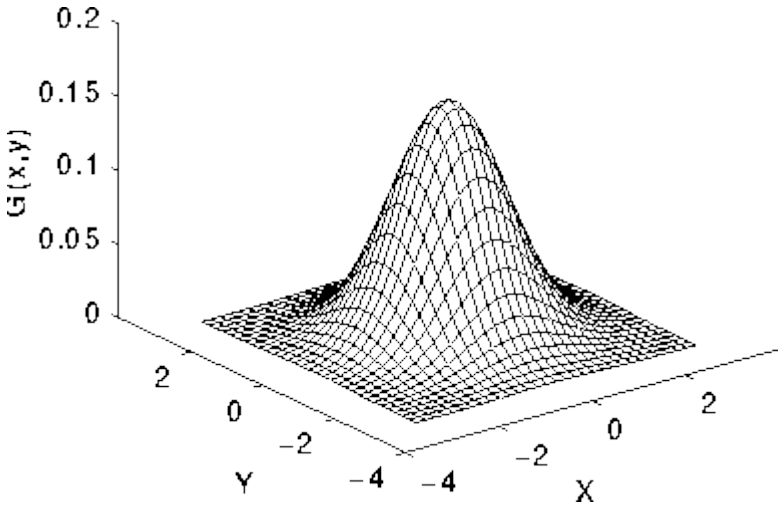

Рисунок 1.3. Графік розподілу Гауса для двовимірного простору при x,y=0 і $\sigma = 1$ [3]

Відповідно до теорії Гауса значення розподілу мають бути скрізь ненульовими і це вимагало б мати маску нескінченного розміру, але на практиці воно є настільки близьким до нуля, що ними можна знехтувати і усікти. Значення отриманні на основі формули розподілу Гауса можна використати для побудови маски, яка має наступний вигляд [3]:

$$\frac{1}{273}\begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix}$$

(1.8)

1.2. Виділення контурів на зображенні

Виділення контурів об’єктів на зображенні є важливою частиною багатьох алгоритмів, які використовуються в комп’ютерному баченні. Основний принцип полягає у знаходженні на зображенні різких перепадів яскравості або інших неоднорідностей. Зміни яскравості на зображенні можуть свідчити про зміну глибини, зміну матеріалу, зміни освітлення або зміну поверхні. Ціль накладання фільтрів для знаходження контурів полягає у досягненні виділення на зображенні набору пов’язаних кривих, які в свою чергу будуть відображати межі об’єктів, граней чи поверхонь. Не менш важливим є те, що на відфільтрованому зображенні зазвичай зменшується кількість основних об’єктів, оскільки значимість другорядних елементів знижуватися.

Оператор Робертса

Оператор Робертса – історично є одним з перших алгоритмів для знаходження контурів на зображенні. Основний принцип роботи оператора полягає у знаходженні для кожного пікселю градієнтів від двох верхніх пікселів до двох нижніх.[2] Нехай пікселі розташовані наступним чином:

(x,y)	(x+1, y)
(x, y+1)	(x+1, y+1)

Рисунок 1.4. Розташування пікселів, які беруть участь в операторі Робертса

Тоді формула для знаходження двох градієнтів G_1 та G_2 для пікселя (x,y) буде мати наступний вигляд:

$$G_1 = f(x,y) - f(x+1,y+1)$$

$$G_2 = f(x+1,y) - f(x,y+1)$$

(1.9) [4]

Тоді значення для пікселя будуть визначатися наступним чином:

$$G = \sqrt{{G_1}^2 + {G_2}^2}$$

$$G(x,y) = \sqrt{(f(x,y) - f(x+1,y+1))^2 + (f(x+1,y) - f(x,y+1))^2}$$

(1.10) [4]

Маски згортки можна відобразити наступним чином:

$$H_1 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix};\ H_2 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix};$$

(1.11) [4]

Застосування кожної маски відбувається окремо і в якості результату виступає наступна величина:

$$G = \sqrt{f(H_1)^2 + f(H_2)^2}$$

(1.12) [4]

Перевагою оператору Робертса є швидкість його накладання через прості маски. Але недоліками є його чутливість до шумів та те, що результуючі контури є тоншими ніж при використанні інших операторів.

Оператор Собеля

Оператор Собеля – один з видів операторів для виділення контурів. Його основна відмінність від інших полягає в використанні додаткових коефіцієнтів для середніх елементів. Ці коефіцієнти дозволяють зменшити ефект згладжування на зображенні за рахунок надання середнім пікселям більшої значимості.

Його основна ідея полягає у знаходженні градієнту яскравості для кожного пікселю: напрямку найбільшої зміни яскравості або її величини. Це відбувається в результаті накладання двох масок для вираховування значень похідних вертикально та горизонтально[4]. Маски мають наступний вигляд:

$$H_1 = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix};\ H_2 = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix};$$

(1.13) [4]

Після застосування кожної маски окремо, результат можна обчислити за формулою 1.12

Оператор Лапласа

На відміну від попередніх операторів, Лапласіан (оператор Лапласа) використовує другу похідну, яка визначається наступним чином:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

(1.14) [4]

При переході до дискретної маски розміром 3x3 визначають наступні формули:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

$$\nabla^2 f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

(1.15) [4]

Для наведеної вище формули існують декілька варіантів реалізації маски:

$$H_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix};\ H_2 = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix};$$

$$H_3 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix};\ H_4 = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix};$$

(1.16) [4]

Оператор Лапласа зазвичай використовують разом зі згладжуванням, оскільки він є дуже чутливим до шумів.

Оператор Кенні

Оператор Кенні вважається найоптимальнішим оператором серед усіх перерахованих. Алгоритм його виконання можна поділити на такі етапи[6]:

1) Просторова частотна фільтрація з низькою прохідністю (зазвичай використовується фільтр Гауса);
2) Застосування диференціальних масок першого порядку для знаходження інтенсивності градієнтів зображення;
3) Немаксимальне придушення, що включає субпіксельну інтерполяцію інтенсивності пікселів;
4) Застосування двойного порогу;
5) Гістерезисний поріг – доопрацювання виявлення країв, що полягає у пригніченні слабких і не пов’язаних з сильними ребер.

1.3. Метод Хафа для знаходження ліній

Основна ідея полягає у створенні акумулятивного масиву (або фазового простору), розмір якого буду відповідати кількості невідомих параметрів в заданому для знаходження кривих рівнянні. Кожна комірка цього масиву(зазвичай матриці) буде відповідати набору кривих, з близькими значеннями параметрів. Для кожної комірки ставиться у відповідність лічильник, який буде відповідати кількості точок, що проходить через криві з заданими параметрами. Після цього за допомогою аналізу вирахуваних лічильників, можна знайти параметри кривих, які проходять через найбільшу кількість точок[7].

Основний алгоритм буде мати наступний вигляд:

1) Вибір рівняння для знаходження кривих певного сімейства. Також можливий вибір ітерації для кожного з невідомого параметру кривої, що може значно вплинути на складність та ефективність алгоритму.
2) Заповнення фазового простору, що має розмірність залежну від кількості невідомих в рівнянні кривої. Заповнення зазвичай відбувається шляхом повного перебору всіх пікселів зображень та всіх варіантів параметрів кривих для цих пікселів. Через це цей етап є найдовшою частиною виконання алгоритму.
3) Пошук локальних максимумів в фазовому просторі. Можливе також додавання фільтрації кожної комірки по мінімальній кількості пікселів, що має належати до кривої.
4) Виведення параметрів кривих з максимальними кількостями точок. Можливе додавання максимальної кількості отриманих результатів. Зазвичай результат повертається у вигляді параметрів кривих для заданого рівняння, але можливі будь-які реалізації повернення результуючих значень.

Пошук прямих за допомогою методу Хафа.

Для пошуку прямої можна використати наступне рівняння:

$$y = m * x + b$$

(1.17)

,де m – кутовий коефіцієнт, а b – точки перетину

Але це рівняння не підходить для пошуку, оскільки при деяких вхідних даних воно породжує фазовий простір необмеженого розміру. Це відбувається у випадку, коли шукані прямі є паралельними до осі ординат, тоді значення m та b є нескінченними. Для вирішення цієї проблеми краще використати інші параметри: r і θ (тета), де параметр r являє собою довжину радіус-вектора найближчої до початку координат точки на прямій, а θ – кут між вектором та віссю координат. (див рисунок 1.4) [7]

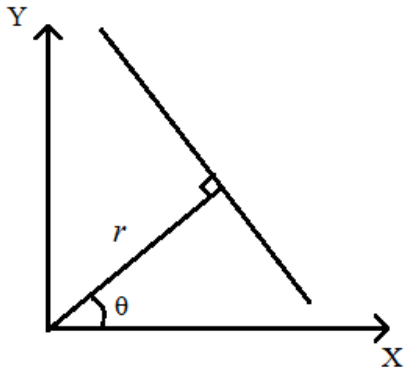

Рисунок 1.4. Представлення прямої за допомогою r і θ [7]

Тоді, рівняння прямої буде мати наступний вигляд:

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

(1.18)

Для більш зручного використання можна перетворити в:

$$r = x * cos\ \theta + y * \sin\theta$$

(1.19)

Через кожну з точок можна провести безліч прямих (див рисунок 1.5) і кожна з них буде мати відповідне рівняння прямої (див формулу 1.19), що залежить від кута та відстані радіусу. Тобто для кожної точки простору (x,y) можна поставити у відповідність набір точок фазового простору.

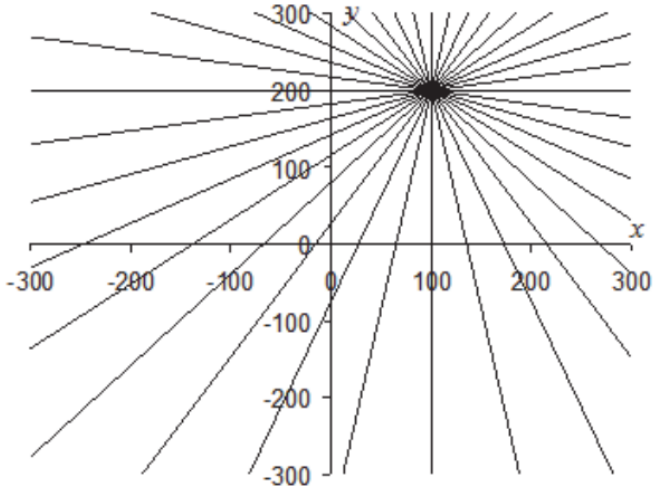

Рисунок 1.5. Прямі, що проходять через точку [7]

Для реалізації та обробки фазового простору його необхідно обмежити та виразити у вигляді матриці $(r \ ,\theta)$ (інколи називають простір Хафа). Розмірність параметру r залежить від розміру вхідного зображення, а $\theta \in [0, 2\pi]$. Для кожної клітинки з простору створюється лічильник, який з самого початку заповнюється нулями.

Алгоритм заповнення фазового простору має наступний вигляд:

- Для кожної потрібної нам точки зображення
- Для кожного кута та радіус-вектору
- Перевірити чи лежить ця точка на даній прямій
- Якщо так, то збільшити значення лічильника цієї прямої

Після отримання фазового простору, виконується пошук локальних максимумів.

Реалізація.

Для реалізації методу була використана мова програмування C++. Етапи попередньої обробки були виконані за допомогою бібліотеки функцій і алгоритмів комп'ютерного зору OpenCV:

```
// Перетворення в відтінки-сірого
cvtColor(src, rgb, COLOR_BGR2GRAY);

// Виділення контурів об'єктів та перетворення в чорно-біле зображення
// В використаному методі вже застосовується фільтрація Гауса та перетворення в бінарне зображення
Canny(gaus, bin, 50, 150, 3);
```

Визначення розміру та ініціалізація фазового простору:

```
// Визначення висоти та ширини вхыдного зображення
int height = src.rows;
int width = src.cols;

//Визначення максимальної відстані від початку  координат(діагоналі зображення)
int R = cvRound(sqrt((double)(width*width + height * height)));

// Аккумулятивний массив для зберігання фазового простору
// 0 < f < 2*PI ; 0 < r < R
phase = Mat::zeros(360, R, CV_8UC1);
```

Заповнення фазового простору відбувається наступним чином:

```
int x = 0, y = 0, r = 0, f = 0;
double theta = 0;
// Обходим всі пікселі зображення
for (y = 0; y < height; y++) {
    for (x = 0; x < width; x++) {
        if (bin.data[y*bin.step+x] != 0) { // Перевіряємо чи піксель належить контуру
            // Розглядаємо всі можливі прямі, що можуть проходитичерез цю тчоку
            for (f = 0; f < 360; f++) { //Проходим всі можливі кути нахилу
                for (r = 0; r < R; r++) { // Проходим всі можливі відстані від початку координат(радіус-веткори)
                    theta = f * CV_PI / 180.0; //Переводимо градуси в радіани
                    // Перевіряємо проходить пряма с заданими параметрами через взяту точку
                    if (abs(((y)*sin(theta) + (x)*cos(theta)) - r) < accuracy) {
                        phase.data[f*phase.step+r]++; //Збільшуємо лічільник фазового простору
                    }
                }
            }
        }
    }
}
```

Знаходження локальних максимумів відбувається за допомогою наступної функції :

```
static vector<Vec2f> findLocalMaximums(int R, int Theta, Mat phase , int minP)
{
    vector<Vec2f> result;
    for (int f = 0; f < Theta; f++)
    {
        for (int r = 0; r < R; r++) {
            int index = f * phase.step + r;
            // Перевірка чи кількість точок, що належать лыныъ быльша заданоъ мынымальноъ
            // та порівнння з сусідніми комірками простору для знаходження локального максимуму
            if (phase.data[index] > minP &&
                phase.data[index] > phase.data[index - 1] && phase.data[index] >= phase.data[index + 1] &&
                phase.data[index] > phase.data[index - R - 2] && phase.data[index] >= phase.data[index + R + 2]) {
                result.push_back(Vec2f(r, f));
            }
        }
    }
    return result;
};
```

Мінімальна кількість точок для лінії визначено числом 20.

Для відображення результату був створений наступний метод, що по отриманому вектору параметрів малює та показує результуючі прямі:

```
void showVec(vector<Vec2f> lines,int height,int width) {
    // Створюємо результуючу матрицю
    Mat res = Mat::zeros(height, width, CV_8UC1);
    double theta = 0;
    int r = 0 , x=0 , y=0;
    // Малюємо прямі для отриманих параметрів
    for (const auto &i : lines) {
        r = i[0];
        theta = i[1] * CV_PI / 180.0;
        for (y = 0; y < height; y++) {
            for (x = 0; x < width; x++) {
                if (cvRound(((y)* sin(theta) + (x)* cos(theta))) == r) {
                    res.data[y*res.step + x] = 255;
                }
            }
        }
    }
    // Відображаємо результат
    imshow("Отриманні лінії", res);
};
```

На кожній ітерації алгоритму відображається відповідне зображення. Також для контролю результату було протестоване готове рішення від OpenCV. Результати виконання можна побачити на рисунку 1.6

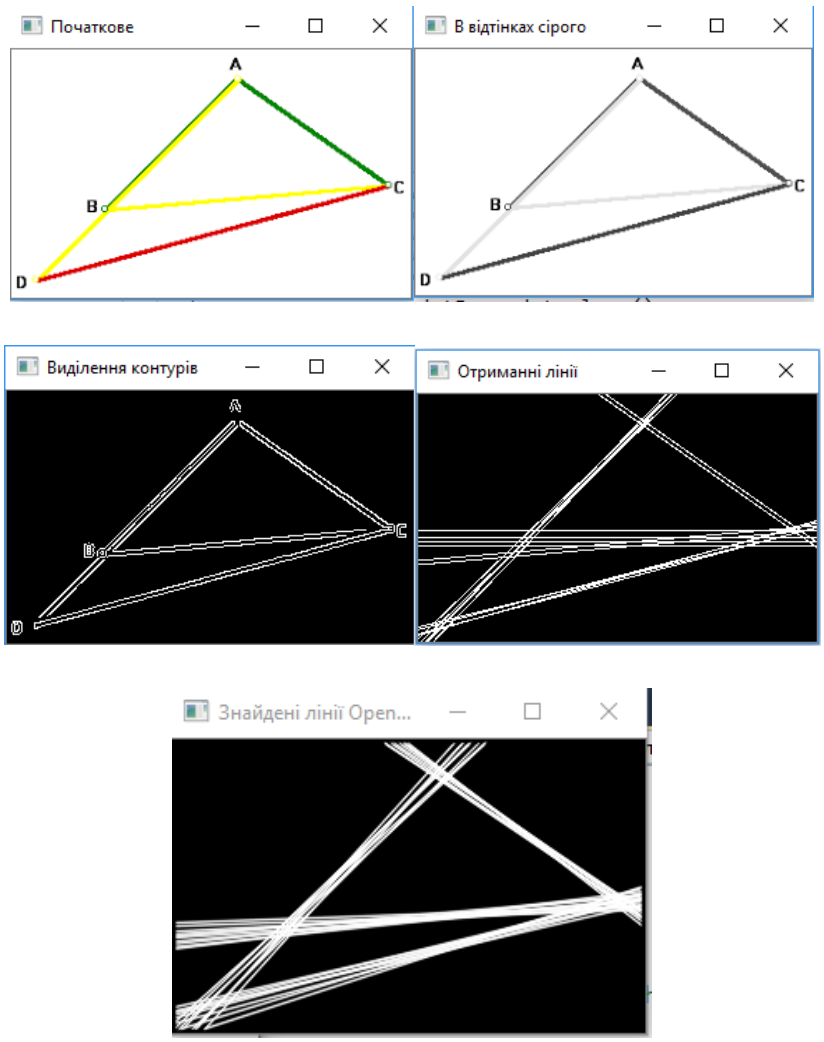

Рисунок 1.6. а) Початкове зображення б) Зображення в відтінках сірого в) Виділенні контури г) Результуючі прямі д) Знайденні лінії методом OpenCV

2. Розділ 2. Модифікації базового методу Хафа

2.1. Метод Хафа для знаходження кіл на зображенні

Для представлення точок кола використовується наступна формула:

$$(x - a)^2 + (y - b)^2 = R^2$$

(2.1)

, де (a, b) – координати центра кола, R – радіус кола, (x, y) – координати точки кола

Як можна побачити з формули 2.1, для задання кола необхідно, не 2 параметри, як в варіанті з знаходженням лінії, а вже 3 – координати центра та радіус. Це призводить до збільшення фазового простору Хафа з двох вимірного до трьох вимірного. За умови що радіус відомий завчасно, складність алгоритму значно зменшується і основна ідея буде полягати у наступному:

Якщо взяти всі можливі кола заданого радіусу для кожної точки контуру, тоді їхні центри утворюють коло цього радіусу. І тоді умовним центром можна вважати точку, де перетинається найбільша кількість цих умовних кіл (див. рисунок 2.1)[7].

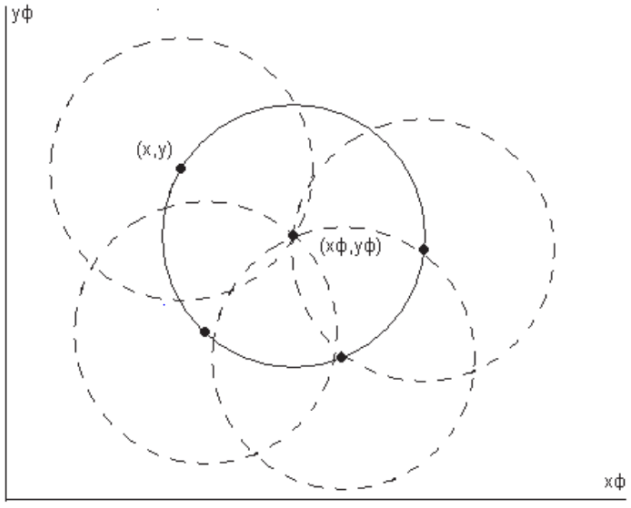

Рисунок 2.1.Виявлення центру кола заданого радіусу методом Хафа[7]

У разі відсутності завчасно відомого радіусу кола, використовується трьох вимірний фазовий простір, що призводить до ускладнення перебору.

2.2. Метод Хафа для знаходження складніших кривих

Для знаходження кривих більш складного рівня, використовується фазовий простір відповідно до кількості параметрів в рівняннях, що задають точку. Наприклад для еліпса, фазовий простір буде залежати від 4 параметрів: координат точки центру (a, b) та r з θ (тета), що відповідають за полярні координати еліпсу. [8]

Зрозуміло, що це призведе до створення та заповнення фазового простору гігантських розмірів. Тому для пошуку кривих складнішого порядку зазвичай використовують певні модифікації базового алгоритму Хафа, що наведені в наступних пунктах.

2.3. Імовірнісне перетворення Хафа.

В імовірнісній реалізації методу, розглядається лише частина пікселів зображення , шо обирається випадково з рівномірною вірогідністю. Основна ідея полягає у тому що існує такий поріг для визначення мінімальної кількості точок для лінії (який зазвичай знаходиться в діапазоні 5-15% від всієї кількості пікселів зображення), при якому кількість помилок у порівняні з стандартним методом Хафа є дуже низьким Також важливим є те, що при досягнені лічильником порогу, відбувається виділення кривої, і не потрібно виконувати повне заповнення фазового простору [9].

Алгоритм має наступний вигляд[9]:

1) Перевірка списку вхідних пікселів, якщо порожній – тоді закінчуємо;
2) Оновлення акумулятивного масиву для одного випадково взятого пікселю;

3) Видалення обраного пікселю зі списку вхідних пікселів;
4) Перевірка чи піковий елемент фазового простору, що був змінений через обраний піксель є більшим ніж визначений поріг. Якщо ні, то перейти до п.1;
5) Визначити найдовший сегмент або сегмент з зазором меншим від заданого порогу для обраного пікового елементу;
6) Переніс обраних точок з вхідного списку в сегмент;
7) Прибрати з акумулятивного масиву всі пікселі обраної лінії;
8) Якщо відрізок лінії довший від зазначеної мінімальної довжини то додавання у вихідний список.

Цей алгоритм дозволяє зменшити час втрачений при заповнені акумулятивного масиву, а також для нього не потрібне повне заповнення фазового простору, що дає можливість переривання виконання методу.

2.4. Комбінаторне перетворення Хафа

Метою цієї модифікації є швидкий пошук прямих на зображенні. За основу ми маємо бінарне зображення, яке є результатом виконання етапу виділення контурів, та фазовий простір розмірністю два, оскільки при пошуку ліній нам не відомо лише 2 параметри. Основна модифікація полягає у зміні етапу заповнення фазового простору зображення.

Основна ідея полягає у наступному: Зображення поділяється на окремі ділянки з метою зменшення кількості точок, аналіз яких відбувається. Це прискорює заповнення фазового простору. Після цього для кожної пари точок з кожної ділянки відбувається пошук прямих, до яких ця точка належить. Важливим є також обраний розмір ділянки при розбитті, оскільки від цього залежить кількість записів в масиві[10].

2.5. Адаптивне перетворення Хафа

Адаптивна модифікація дозволяє значно зменшити кількість необхідного для зберігання в пам’яті місця та пришвидшити процес знаходження кривих. Основна модифікації полягає у наступному:

Для виконання алгоритму з самого початку обирається маленький розмір області зазвичай 9x9 в випадку двомірного простору, або 3x3x3x3 у разі багатовимірного простору. Весь простір розбивається на відповідного розміру області, які заповнюються так само, як і в випадку базового методу Хафа. Оскільки розмір акумулятивного масиву дуже маленький, заповнення відбувається значно швидше. Наступним етапом після виконання заповнення є пошук максимальної комірки, яка в свою чергу стане новим фазовим простором[11].

Основною перевагою даної модифікації є значне зменшення часу та пам’яті для виконання алгоритму. Мінусом цього алгоритму є необхідність початку алгоритму з спочатку для кожної геометричної фігури. Також є додатковий шанс на знаходження помилкового максимуму для фазового простору на початку роботи алгоритму.

2.6. Випадкове перетворення Хафа

Випадкове перетворення Хафа є модифікацією, основною цілю яких є пришвидшення швидкодії алгоритму. Основна ідея цієї модифікації полягає у тому, що криву можна однозначно визначити за певним набором точок. Наприклад, коло можна задати трьома точками, пряму лінію – двома. Тоді взявши певну кількість випадкових точок, через них можна провести певну криву. Кожна з цих кривих збільшить відповідний лічильник в фазовому просторі. Результуючі параметри обираються так само як і в базовому методі Хафа[12].

Алгоритм заповнення фазового простору буде мати наступний вигляд[12].:

1) Створення набору усіх точок виділених контурів на зображенні;
2) Вибір випадкової пари точок з набору;
3) Якщо відстань між точками не задовольняє умови обмеження відстані, перейдіть до п.2, інакше до п.4;
4) Визначення параметрів кривої для взятих точок;
5) Збільшення комірки акумуляторного масиву за визначеними параметрами;
6) Повернутись до п.2;

Ця модифікація значно пришвидшує виконання пошуку кривих на зображені, але, оскільки вибір точок відбувається випадково, точність виконання є трошки нижчою ніж при використанні базового методу Хафа.

3. Розділ 3. Приклади використання методу Хафа в комп’ютерному баченні

3.1. Використання методу Хафа для розпізнавання тексту

Одним з прикладів використання перетворення Хафа в комп’ютерному баченні є його використання для розпізнавання рукописного тексту. Оскільки більшість об’єктів в рукописному тексті являють собою криві різних параметрів, метод Хафа чудово виконує свою функцію:

Після попередньої обробки методом Хафа та знаходження необхідних параметрів кривих або їх властивостей, використовують один з наступних методів реалізації для визначення тексту[14]:

1) Шаблонні класифікатори – основний принцип полягає у порівняні отриманих на етапі попередньої обробки даних за певним критерієм та вибір, з попередньо заданих шаблонів, того який найбільш підходить. Найпростіший критерій порівняння – мінімальна кількість точок, що відрізняє шаблон від вхідного зображення.

2) Класифікатори за ознаками – основний принцип в яких, полягає у аналіз не конкретно отриманого символу, а його ознак або чисел, що вираховуються з отриманих після попередньої обробки даних. Цей метод є найбільш розповсюдженим серед інших.

3) Структурні класифікатори – основним принципом яких є з отриманих даних про символ вилучити його топологічне представлення, в якому має зберігатися інформація про відносне розташування основних елементів символу.

3.2. Використання методу Хафа для розпізнавання дорожньої розмітки

У сучасному світі, де великими кроками розвивається індустрія автономних машин, дуже важливою задачею є розпізнавання дорожньої розмітки. Визначена дорожня розмітка, використовується системою керування автомобілю для контролю пересування по заданій лінії, заданому маршруту та для попередження про можливі відхилення від маршруту. Для розпізнавання заданої розмітки використовується наступний алгоритм[14]:

1) Перетворення вхідного зображення в відтінки сірого
2) Затемнення зображення з метою зменшення контрасту від певних ділянок дороги
3) Перетворення вхідного зображення у кольоровий простір HLS(Hue, Lightness, Saturation – відтінок, колір, насиченість)
4) Ізоляція жовтого кольору з отриманого кольорового для того щоб знайти жовту розмітку
5) Ізоляція білого кольору з отриманого в 2 пункті кольорового простору для знаходження білої розмітки
6) Виконання бітової операції « АБО » між жовтою та білою масками отриманими в 4 та 5 пункті відповідно, задля визначення спільної маски
7) Виконання бітової операції « І » між отриманою в попередньому пункті маскою та затемненим зображенням
8) Накладання фільтру згладжування Гауса
9) Застосування методу Кенні для виявлення контурів
10) Визначення потрібної нам частини зображення
11) Застосування методу Хафа для знаходження ліній
12) Зображення отриманих ліній на початковому зображенні

Висновки

Під час виконання даної роботи були детально проаналізовані етапи попередньої обробки зображень. Серед яких можна виділити 2 найбільш важливих: згладжування та пошук контурів.

Було визначені основні відмінності найбільш відомих фільтрів:

- Низькочастотні фільтри
- високочастотні фільтри
- медіанний фільтр
- фільтр Гауса

За отриманими даними можна зробити висновок, що фільтр Гауса є найбільш доречним для використання в випадку пошуку ліній, оскільки він найкраще виконує функцію розмиття шумів та другорядних об’єктів на зображенні, а контури більших, важливіших для нас об’єктів піддаються меншому розмиванню, порівняно з іншими фільтрами.

Також були досліджені наступні оператори для знаходження контурів:

- Собеля
- Робертса
- Лапласа
- Кенні

Після аналізу отриманих даних можна зробити висновок, що, серед перерахованих вище, оператор Кенні є найкращим майже у всіх випадках, окрім деяких поодиноких варіантів. Оскільки він майже ідеально задовольняє наступним критеріям: гарне виявлення, вірне виявлення положення контурів та єдиний відгук на кожний контур.

Був повністю досліджений та програмо реалізований базовий метод Хафа для знаходження ліній на зображенні. А також розглянуті його можливі модифікації для знаходження додаткових видів кривих. Були описанні варіанти покращення базового алгоритму з метою покращення швидкодії та оптимізації використання пам’яті.

Список літератури

1. Костенко Л. С. Методы и алгоритмы сглаживания фона изображений в системах распознавания образов / Л. С. Костенко. // Открытые информационные и компьютерные интегрированные технологии. – 2014. – №64. – С. 17.
2. Gonzalez R. Digital Image Processing. Second Edition / R. Gonzalez, R. Wood., 1992. – 793 с.
3. Gonzalez R. Digital Image Processing. Third Edition / R. Gonzalez, R. Wood., 2008. – 976 с.
4. Davies E. Machine Vision: Theory, Algorithms and Practicalities / E. R. Davies., 2005. – 855 с.
5. Edge Detection by Using Canny and Prewitt / Priyam, D. Diganta, Shreya, P. Dipanjan. // International Journal of Scientific & Engineering Research. – 2016. – №4. – С. 251–254
6. Davies E. Machine Vision: Theory, Algorithms and Practicalities / E. R. Davies., 2012. – 875 с.
7. Кудрина М. А. Использование преобразования Хафа для обнаружения прямых линий и окружностей на изображении / М. А. Кудрина. // Известия Самарского научного центра Российской академии наук. – 2014. – №4. – С. 476–478.
8. Mark S. N. Feature Extraction and Image Processing / S. N. Mark, S. A. Alberto. – Great Britain: Replika Press Pvt Ltd, 2002. – 350 с.
9. Galambosi C. Progressive Probabilistic Hough Transform for line detection / C. Galambosi, J. Matastsand, J. Kittlert. // University of Surrey. – 1999. – С. 554–560.
10. Eric W. he Combinatorics of Object Recognition in Cluttered Environments using Constrained Search / W. Eric, L. Grimson. // MASSACHUSETTS INSTITUTE OF TECHNOLOGY ARTIFICIAL INTELLIGENCE LABORATORY. – 1988. – №1019. – С. 1–40.
11. Ecabert O. Adaptive Hough transform for the detection of natural shapes under weak affine transformations / O. Ecabert, J. Thiran. // Pattern Recognition Letters. – 2003. – №25. – С. 1411–1419.
12. Randomized Hough Transform: New Extensions / P.Minkkinen, C. Jarvinen, N. Erkki, A. Ukkola. – Finland: LAPPEENRANTA UNIVERSITY OF TECHNOLOGY, 1994. – 61 с.
13. Гайдуков Н. П. ПРИМЕНЕНИЕ ПРЕОБРАЗОВАНИЯ ХАФА ДЛЯ РАСПОЗНАВАНИЯ ТЕКСТА / Н. П. Гайдуков, Е. О. Савкова. // Донецкий национальный технический университет кафедра автоматизированных систем управления. – 2012. – С. 136–140.
14. Finding Lane Lines — Simple Pipeline For Lane Detection. [Електронний ресурс]. – 2019. – Режим доступу до ресурсу:

https://towardsdatascience.com/finding-lane-lines-simple-pipeline-for-lane-detection-d02b62e7572b.