Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

С Е М И Н А Р С К А Р А Б О Т А

по предметот

ДИГИТАЛНО ПРОЦЕСИРАЊЕ НА СЛИКА

на тема

А Л Г О Р И Т М И З А Д Е Т Е К Ц И Ј А Н А Р А Б О В И

Ментор:

Проф. Д-р Ивица Димитровски

Изработил:

Бојан Ристов (211151)

01.07.2023, Скопје

С О Д Р Ж И Н А

I. Вовед

Областа на дигиталното процесирање на слики го револуционизира начинот на кој ги анализираме и интерпретираме визуелните информации, овозможувајќи ни да извлечеме вредни сознанија од сликите во различни домени. Во рамките на ова поле, детекцијата на рабови има огромно значење, служејќи како клучен чекор во бројните задачи за анализа на слики. Способноста за прецизно откривање на рабовите е од суштинско значење за задачи како што се сегментација на слики, препознавање објекти и екстракција на карактеристики. Како такво, императив е да се истражат и разберат алгоритмите кои го поткрепуваат откривањето на рабовите.

Фокусот на овој семинарски труд е да навлезе во областа на алгоритмите за детекција на рабови, со посебен акцент на нивната имплементација, анализа и евалуација. Целта е да се стекне сеопфатно разбирање за популарните техники за откривање рабови и нивната ефикасност во сценарија од реалниот свет. За да се постигне ова, ќе го користам програмскиот јазик Python и ќе користам различни библиотеки со отворен код за да ги имплементирам и анализирам овие алгоритми.

Мојата примарна цел е да истражам и имплементирам неколку широко користени алгоритми за откривање рабови, испитувајќи ги нивните основни принципи, силни страни и ограничувања.

Во текот на овој семинарски труд, ќе презентирам преглед на избраните алгоритми за детекција на рабови, нивните теоретски основи и библиотеките на Python кои се користат за нивна имплементација. Дополнително, ќе ги разгледаме перформансите и квалитетот на секој алгоритам, земајќи ги предвид факторите како што се точноста, робусноста и отпорноста на грешки. Оваа евалуација ќе обезбеди вредни сознанија за соодветноста на секој алгоритам за различни апликации и сценарија.

Понатаму, моето истражување вклучува развој на демо апликација која е пример за практичната примена на овие алгоритми за откривање рабови. Оваа апликација ќе служи како опиплива демонстрација на можностите на алгоритмите и ќе помогне во визуелизирањето на нивното влијание врз сликите од реалниот свет.

Во следните делови, детално ќе ги истражиме секој одбран алгоритам за детекција на рабови, дискутирајќи ги нивните теоретски основи, спецификите на имплементацијата и евалуацијата на перформансите. Дополнително, ќе ги претставам резултатите од мојата анализа.

II. Методологија

Во овој дел, прикажана е методологијата употребена за истражувањето и имплементацијата на алгоритми за откривање рабови. Опишани се користените збирки на податоци, применетите чекори на претходна обработка и специфичните техники и алатки кои се користат за имплементација на алгоритмите во Python.

1. Избор на збирки на податоци и претходна обработка

За да ги оцениме перформансите и квалитетот на алгоритмите за откривање рабови, користен е опсег на збирки на податоци за слики. Изборот на овие збирки на податоци беше заснован на нивното претставување на различни сценарија, вклучувајќи природни сцени, предмети направени од човекот и медицински слики. Беше внимавано збирките на податоци да опфаќаат различни типови на рабови, како што се остри рабови, текстурирани граници и транзиции со низок контраст.

Пред да бидат внесени збирките на податоци во алгоритмите за откривање рабови, извршени се чекори за претпроцесирање за да го подобриме квалитетот на сликите и да го олесниме прецизното откривање на рабовите. Овие чекори на претходна обработка вклучуваат:

- Промена на големината на сликата (опционално): Промена на големината на сликите до стандардизирана резолуција за да се обезбеди конзистентност низ збирките на податоци и да се намали комплексноста на пресметките.
- Намалување на шум: Примена на техники за отстранување на шум, како што е Гаусово замаглување или средно филтрирање, за да се ублажи влијанието на бучавата врз резултатите од откривањето на рабовите.
- Подобрување на контрастот: Користење техники како изедначување на хистограм или адаптивно истегнување на контраст за да се подобри контрастот на сликите и да се подобрат перформансите за откривање на рабовите.

2. Детали за имплементација

Имплементацијата на алгоритмите за откривање на рабовите е спроведена со користење на програмскиот јазик Python, користејќи различни библиотеки со отворен код за обработка на слики и компјутерска визија.

Секој избран алгоритам за откривање на рабови е имплементиран според неговите специфични барања и основните принципи. Користена е литература и онлајн ресурси за детални објаснувања на алгоритмите и техниките за нивна имплементација. Имплементацијата на кодот беше темелно документирана за да се обезбеди јасност и репродуктивност.

3. Методологија на евалуација

За да се оценат перформансите и квалитетот на имплементираните алгоритми за откривање рабови, користени се различни метрики и техники за евалуација. Примарните показатели што се разгледуваа беа:

- Точност: Споредување на откриените рабови со рачно означени рабови на заземјување на вистината за да се измери способноста на алгоритмот правилно да ги идентификува вистинските рабови.
- Прецизност: Пресметување на вредностите за прецизност и потсетување за да се процени способноста на алгоритмот прецизно да ги идентификува рабовите и да избегне лажни позитиви и лажни негативни.
- Компјутерска ефикасност: Анализирање на пресметковното време потребно од секој алгоритам за обработка на сликите и откривање на рабовите, земајќи ја предвид нивната соодветност за апликации во реално време.

Дополнително, визуелно е прегледан излезот од алгоритмите за откривање рабови со преклопување на откриените рабови на оригиналните слики. Оваа квалитативна евалуација овозможи да се процени способноста на алгоритмите да ги доловат релевантните рабови, да се справуваат со грешки и да ги зачуваат деталите за рабовите.

III. Познати алгоритми за детекција на рабови

1. Sobel Operator

1.1 Опис на Sobel Operator алгоритмот

Sobel Operator е широко користен алгоритам за откривање рабови базиран на градиент кој го проценува градиентот на сликата за да ги идентификува рабовите. Ги анализира промените на интензитетот во хоризонталните и вертикалните насоки на сликата за да ја одреди големината и насоката на градиентите.

1.2 Формулација

Во алгоритмот Sobel Operator, формулацијата вклучува употреба на конволуциони кернели за да се процени градиентот на сликата во хоризонтална и вертикална насока. Овие кернели се дизајнирани да ја доловат брзината на промена на интензитетот на пикселите и да обезбедат мерка за големината на градиентот на секој пиксел.

Хоризонталното јадро на Sobel е матрица 3x3 што го приближува градиентот во хоризонтална насока. Ова јадро ги нагласува промените на интензитетот помеѓу пикселите во левата и десната насока. Вертикалното јадро на Собел е матрица 3x3 што го проценува градиентот во вертикална насока. Ова јадро ги истакнува варијациите на интензитетот помеѓу пикселите во насоките нагоре и надолу.

X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1

Слика 1: Вертикално и хоризонтално јадро на Sobel Operator

Операција на конволуција: Операторот Sobel ги применува хоризонталните и вертикалните Sobel кернели на сликата користејќи операција на конволуција. На секој пиксел, соседните пиксели во прозорецот на јадрото се множат со соодветните тежини дефинирани во матриците на јадрото. Добиените вредности потоа се сумираат за да се добие приближувањето на градиентот во соодветната насока.

Пресметка на големината на градиентот: По примената на Собеловите кернели, големините на градиентот се пресметуваат со комбинирање на хоризонталните и вертикалните градиенти користејќи ја Питагоровата теорема. Магнитудата ја претставува вкупната јачина на промената на интензитетот на секој пиксел.

Со формулирање на операторот Sobel со овие конволуциони кернели и пресметување на големини на градиент, алгоритмот може ефективно да ги процени рабовите на сликата. Формулацијата овозможува откривање на рабовите и во хоризонтална и во вертикална насока, обезбедувајќи информации за ориентацијата и локацијата на рабовите.

1.3 Клучни точки за алгоритмот Sobel Operator

- Операторот Sobel е релативно лесен за имплементација и пресметковно ефикасен.
- Се фокусира на снимање на промените на интензитетот во хоризонталните и вертикалните насоки, што го прави ефикасен за откривање на рабовите во различни ориентации.
- Алгоритмот е чувствителен на грешки, така што примената на чекори за претходна обработка како Гаусовото измазнување може да ги подобри неговите перформанси.
- Операторот Sobel обезбедува информации за големината на градиентот и насоката, кои можат да бидат корисни за понатамошна анализа или екстракција на карактеристики.
- Прилагодувањето на вредноста на прагот овозможува контролирање на чувствителноста на откривањето на рабовите и количината на заробени детали.
- Операторот Sobel најчесто се користи како основна техника за откривање раб во различни апликации за обработка на слики.

1.4 Имплементација на алгоритмот Sobel Operator

Во долунаведениот код е имплементиран Sobel Operator алгоритмот.

```
import cv2
import numpy as np

# Load the images
image_filenames = ['IMG1.jpg', 'IMG2.jpg', 'IMG3.jpg', 'IMG4.jpg',
'IMG5.jpg']
for filename in image_filenames:
    # Read the image
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)

    # Apply Sobel operator
    sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)  # Sobel operator in X
direction
    sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)  # Sobel operator in Y
direction

    # Calculate gradient magnitude and direction
    gradient_magnitude = np.sqrt(sobelx ** 2 + sobely ** 2)
    gradient_direction = np.arctan2(sobely, sobelx)

    # Normalize gradient magnitude to 0-255 range
    gradient_magnitude = cv2.normalize(gradient_magnitude, None, 0, 255,
cv2.NORM_MINMAX, dtype=cv2.CV_8U)

    # Threshold the gradient magnitude to obtain the binary edge map
    threshold_value = 50  # Adjust this threshold value as needed
    _, edge_map = cv2.threshold(gradient_magnitude, threshold_value, 255,
cv2.THRESH_BINARY)

    # Display the resulting edge map
    cv2.imshow('Edge Map', edge_map)

    # Wait for a key press and then close the windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Код 1: Код за имплементација на Sobel Operator алгоритмот

2. Prewitt Operator

2.1 Опис и формулација на Prewitt Operator

Операторот Prewitt е уште еден популарен алгоритам за откривање рабови што се користи во обработката на слики. Тој е сличен на операторот Sobel и исто така работи со пресметување на градиентите на сликата.

Операторот Prewitt, како и операторот Sobel, применува неколку конволуциони кернели на сликата за да го пресмета градиентот во хоризонтална и вертикална насока.

-1	0	+1
-1	0	+1
-1	0	+1

G_x

+1	+1	+1
0	0	0
-1	-1	-1

G_y

Слика 2: Вертикално и хоризонтално јадро на Prewitt Operator

Примена на Prewitt Operator:

- Претворање на сликата во сива скала ако е во боја.
- Поврзување на сликата со кернелите Prewitt X и Prewitt Y одделно.
- Пресметка на големината на градиентот на секој пиксел
- Нанесување праг на големината на градиентот за филтрирање на послабите рабови и подобрување на посилните рабови.

Операторот Prewitt, сличен на операторот Sobel, ги детектира рабовите со истакнување на региони со значителни промени во интензитетот. Сепак, коефициентите на кернелот на операторот Prewitt се разликуваат, што резултира со малку различни пресметки на градиент. Изборот помеѓу операторот Sobel и операторот Prewitt често зависи од специфичните барања за апликација или личните преференции.

2.2 Имплементација на Prewitt Operator алгоритмот

Во долунаведениот код е прикажана имплементацијата на Prewitt Operator алгоритмот.

```
import numpy as np
from PIL import Image
import cv2

image_files = ["IMG1.jpg", "IMG2.jpg", "IMG4.jpg"]

for image_file in image_files:

    img = np.array(Image.open(image_file)).astype(np.uint8)
    gray_img = np.round(0.299 * img[:, :, 0] +
                        0.587 * img[:, :, 1] +
                        0.114 * img[:, :, 2]).astype(np.uint8)
    # Prewitt Operator
    h, w = gray_img.shape

    horizontal = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])  # s2
    vertical = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])  # s1
    newgradientImage = np.zeros((h - 2, w - 2))

    for i in range(1, h - 1):
        for j in range(1, w - 1):
            horizontalGrad = (horizontal[0, 0] * gray_img[i - 1, j - 1]) + \
                             (horizontal[0, 1] * gray_img[i - 1, j]) + \
                             (horizontal[0, 2] * gray_img[i - 1, j + 1]) + \
                             (horizontal[1, 0] * gray_img[i, j - 1]) + \
                             (horizontal[1, 1] * gray_img[i, j]) + \
                             (horizontal[1, 2] * gray_img[i, j + 1]) + \
                             (horizontal[2, 0] * gray_img[i + 1, j - 1]) + \
                             (horizontal[2, 1] * gray_img[i + 1, j]) + \
                             (horizontal[2, 2] * gray_img[i + 1, j + 1])
            verticalGrad = (vertical[0, 0] * gray_img[i - 1, j - 1]) + \
                           (vertical[0, 1] * gray_img[i - 1, j]) + \
                           (vertical[0, 2] * gray_img[i - 1, j + 1]) + \
                           (vertical[1, 0] * gray_img[i, j - 1]) + \
                           (vertical[1, 1] * gray_img[i, j]) + \
                           (vertical[1, 2] * gray_img[i, j + 1]) + \
                           (vertical[2, 0] * gray_img[i + 1, j - 1]) + \
                           (vertical[2, 1] * gray_img[i + 1, j]) + \
                           (vertical[2, 2] * gray_img[i + 1, j + 1])
            # Edge Magnitude
            mag = np.sqrt(pow(horizontalGrad, 2.0) + pow(verticalGrad, 2.0))
            newgradientImage[i - 1, j - 1] = mag
    # Convert the image to uint8 and scale the values to 0-255
    newgradientImage = (255 * (newgradientImage /
np.max(newgradientImage))).astype(np.uint8)
    cv2.imshow("Prewitt Edges", newgradientImage)
    cv2.waitKey(0)
cv2.destroyAllWindows()
```

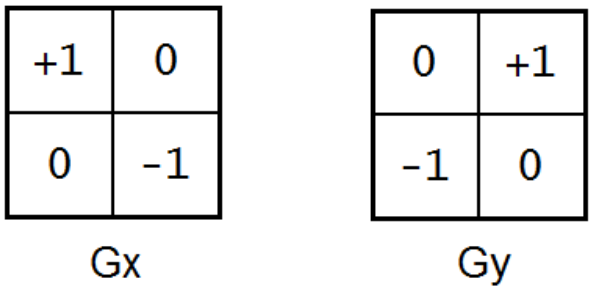
Код 2: Код за имплементација на Prewitt Operator алгоритмот

3. Roberts Cross Operator

3.1 Опис на Roberts Cross Operator алгоритмот

Roberts Cross Operator е алгоритам за откривање рабови кој ги детектира рабовите на сликата со пресметување на големината на градиентот користејќи пар од 2x2 конволуциони кернели. Овие кернели се применуваат посебно на сликата за приближување на градиентот во хоризонтална и вертикална насока.

Операторот Робертс Крос ги користи следниве конволуциони кернели:



Слика 3: Вертикални и хоризонтални кернели на Roberts Cross Operator

За да се пресметаат рабовите со помош на операторот Робертс Крос, сликата се спојува со овие две кернели одделно. Големината на градиентот потоа се пресметува како квадратен корен од збирот на квадратните градиенти во хоризонталната и вертикалната насока:

$$|G| = \sqrt{Gx^2 + Gy^2} \quad \left(|G| = |Gx| + |Gy|\right)$$

Слика 4: Формула за пресметка на рабови кај Roberts Cross Operator

Еден важен аспект што треба да се забележи за операторот Робертс Крос е дека тој е едноставен и пресметковно ефикасен алгоритам за откривање на рабовите. Сепак, може да биде чувствителен и може да произведе потенки рабови во споредба со другите алгоритми. Тој е лесен за имплементација и бара минимални пресметковни ресурси. Тој е особено погоден за апликации или сценарија во реално време каде што пресметковната ефикасност е од клучно значење.

3.2 Имплементација на Roberts Cross Operator алгоритмот

Во долунаведениот код е дадена имплементацијата на Roberts Cross Operator алгоритмот.

```
import cv2
import numpy as np
from scipy import ndimage
import os

# Path to the directory containing the images
image_dir = os.path.dirname(os.path.abspath(__file__))

# List of image filenames
image_files = ["IMG1.jpg", "IMG2.jpg", "IMG3.jpg", "IMG4.jpg", "IMG5.jpg"]

# Loop through the image files
for image_file in image_files:
    # Read the image
    image_path = os.path.join(image_dir, image_file)
    img = cv2.imread(image_path, 0).astype('float64')
    img /= 255.0

    # Apply the Roberts Cross Operator
    roberts_cross_v = np.array([[1, 0], [0, -1]])
    roberts_cross_h = np.array([[0, 1], [-1, 0]])
    vertical = ndimage.convolve(img, roberts_cross_v)
    horizontal = ndimage.convolve(img, roberts_cross_h)
    edged_img = np.sqrt(np.square(horizontal) + np.square(vertical))
    edged_img *= 255

    # Display the image

    cv2.imshow("Roberts Cross Edges", edged_img.astype(np.uint8))
    cv2.waitKey(0)

# Close all windows
cv2.destroyAllWindows()
```

Код 3: Код за имплементација на Roberts Cross Operator алгоритмот

4. Canny Operator

4.1 Опис и формулација на Canny Operator алгоритмот

Алгоритмот Canny Edge Detection е популарна и широко користена техника за детекција на рабови во компјутерската визија и обработката на слики. Тој е познат по неговата ефикасност во откривањето на рабовите со ниски стапки на грешки и прецизна локализација. Операторот Canny се состои од неколку чекори:

- Гаусово измазнување: влезната слика е споена со Гаусовиот филтер за да се намали шумот и да се елиминираат артефактите со висока фреквенција. Овој чекор помага да се добие измазнета верзија на сликата.

- Пресметка на градиент: Големината на градиентот и ориентацијата се пресметуваат за секој пиксел на измазнетата слика. Вообичаено, градиентите се пресметуваат со помош на деривати од прв ред (Sobel, Prewitt, итн.).

- Не-максимално потиснување: Овој чекор помага да се разредуваат рабовите со потиснување на сите вредности на градиентот, освен локалниот максимум, осигурувајќи дека остануваат само тенки линии што ги претставуваат рабовите.

- Двоен праг: Две вредности на праг, низок и висок праг, се користат за класифицирање на рабовите во три категории: силни рабови, слаби рабови и нерабови. Пикселите со големина на градиент над високиот праг се сметаат за силни рабови, додека пикселите помеѓу високиот и нискиот праг се сметаат за слаби рабови.

- Следење на рабовите со хистерезис: Слабите рабови се сметаат како дел од рабовите само ако се поврзани со силни рабови. Овој процес помага во намалување на бучавата и пополнување на празнините во рабовите.

Операторот Canny обезбедува одлични резултати во откривањето на рабовите и е отпорен на грешки. Широко се користи во различни задачи за компјутерска визија, како што се откривање објекти, сегментација на слики и екстракција на карактеристики.

Операторот Canny се смета за оптимална техника за откривање на рабовите кога се потребни и точност и ниски стапки на грешки. Сепак, неговата повеќестепена природа го прави пресметковно поскап во споредба со другите алгоритми за откривање рабови.

4.2 Имплементација на Canny Operator алгоритмот

Во долниот код е прикажана имплементацијата на Canny Operator алгоритмот.

```
import cv2
import os

def apply_canny_edge_detection(image):
    # Conversion to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Applying Gaussian blur
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    # Applying Canny edge detection
    edges = cv2.Canny(blurred, 50, 150)

    return edges

# Get the file paths of the images
image_folder = '.'  # Current folder
image_files = ['IMG1.jpg', 'IMG2.jpg', 'IMG3.jpg', 'IMG4.jpg', 'IMG5.jpg']

# Process each image
for image_file in image_files:
    image_path = os.path.join(image_folder, image_file)

    # Read the image
    image = cv2.imread(image_path)

    # Apply Canny edge detection
    edges = apply_canny_edge_detection(image)

    # Display the edges image
    cv2.imshow('Edges', edges)

    # Wait for key press and close the windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Код 4: Код за имплементација на Canny Operator алгоритмот

IV. Демо апликација за детекција на рабови

1. Имплементација на Демо апликацијата

Во оваа едноставна демо апликација имаме почетна страна на која е овозможен избор на фотографија од својот компјутер. Откако ќе се избере посакуваната фотографија, апликацијата не пренасочува на друга страница во која ги имаме наведено алгоритмите за детекција на рабови кои беа разработени во оваа семинарска работа. Откако ќе се селектира некој од алгоритмите истиот се применува на веќе избраната фотографија.

Во подолниот код е прикажана имплементацијата на вид едноставна демо апликација за детекција на рабови на избрана фотографија.

```
import tkinter as tk
from tkinter import filedialog
from tkinter import messagebox
import cv2
import numpy as np
from PIL import ImageTk, Image

# Function to apply Canny edge detection
def apply_canny_edge_detection(image):
    # Conversion to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Applying Gaussian blur
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    # Applying Canny edge detection
    edges = cv2.Canny(blurred, 50, 150)

    return edges

# Function to apply Sobel edge detection
def apply_sobel_edge_detection(image):
    # Conversion to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Applying Sobel edge detection
    sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
    sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
    edges = cv2.magnitude(sobelx, sobely)

    return edges
```

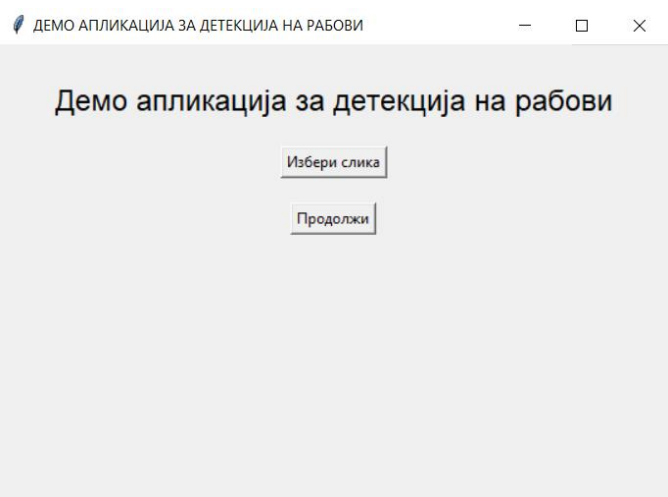
```
# Function to apply Prewitt edge detection
def apply_prewitt_edge_detection(image):
    # Conversion to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Applying Prewitt edge detection
    kernelx = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])
    kernely = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
    img_prewittx = cv2.filter2D(gray, -1, kernelx)
    img_prewitty = cv2.filter2D(gray, -1, kernely)
    edges = cv2.addWeighted(img_prewittx, 0.5, img_prewitty, 0.5, 0)

    return edges

# Function to apply Roberts Cross edge detection
def apply_roberts_cross_edge_detection(image):
    # Conversion to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Applying Roberts Cross edge detection
    roberts_cross_v = np.array([[1, 0], [0, -1]])
    roberts_cross_h = np.array([[0, 1], [-1, 0]])
    vertical = cv2.filter2D(gray, -1, roberts_cross_v)
    horizontal = cv2.filter2D(gray, -1, roberts_cross_h)
    edges = np.sqrt(np.square(horizontal) + np.square(vertical))

    return edges

# Function to handle the Next button click event
def next_button_clicked():
    # Hide the current frame
    welcome_frame.pack_forget()

    # Show the algorithm selection frame
    algorithm_frame.pack()

# Function to handle the Apply button click event
def apply_button_clicked():
    # Get the selected algorithm
    selected_algorithm = algorithm_var.get()

    # Apply the selected algorithm to the image
    if selected_algorithm == "Canny Operator":
        edges = apply_canny_edge_detection(image)
    elif selected_algorithm == "Sobel Operator":
        edges = apply_sobel_edge_detection(image)
    elif selected_algorithm == "Prewitt Operator":
        edges = apply_prewitt_edge_detection(image)
    elif selected_algorithm == "Roberts Cross Operator":
        edges = apply_roberts_cross_edge_detection(image)

    # Display the edges image
    edges_img = Image.fromarray(edges)
    edges_img_tk = ImageTk.PhotoImage(edges_img)
    output_label.configure(image=edges_img_tk)
    output_label.image = edges_img_tk
```

```
# Function to handle the Save button click event
def save_button_clicked():
    # Open a file dialog to save the processed image
    file_path = filedialog.asksaveasfilename(defaultextension=".jpg")
    if file_path:
        # Save the processed image
        cv2.imwrite(file_path, edges)
        messagebox.showinfo("Save", "Image saved successfully!")

# Create the main window
window = tk.Tk()
window.title("ДЕМО АПЛИКАЦИЈА ЗА ДЕТЕКЦИЈА НА РАБОВИ")

# Create the welcome frame
welcome_frame = tk.Frame(window)
welcome_frame.pack(pady=20)

# Create the welcome label
welcome_label = tk.Label(welcome_frame, text="Демо апликација за детекција на
рабови", font=("Arial", 18))
welcome_label.pack(pady=10)

# Create the file selection button
def select_image():
    file_path = filedialog.askopenfilename(initialdir='.', title='Select an
Image')
    if file_path:
        global image, edges
        image = cv2.imread(file_path)
        edges = None
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, (400, 300))
        image_pil = Image.fromarray(image)
        image_tk = ImageTk.PhotoImage(image_pil)
        input_label.configure(image=image_tk)
        input_label.image = image_tk

file_select_button = tk.Button(welcome_frame, text="Избери слика",
command=select_image)
file_select_button.pack(pady=10)

# Create the next button
next_button = tk.Button(welcome_frame, text="Продолжи",
command=next_button_clicked)
next_button.pack(pady=10)

# Create the algorithm selection frame
algorithm_frame = tk.Frame(window)

# Create the algorithm selection label
algorithm_label = tk.Label(algorithm_frame, text="Избери некој од наведените
алгоритми!", font=("Arial", 18))
algorithm_label.pack(pady=10)

# Create the algorithm selection buttons
algorithm_var = tk.StringVar()
algorithm_var.set("Canny Operator")
```

```
canny_button = tk.Radiobutton(algorithm_frame, text="Canny алгоритам",
variable=algorithm_var, value="Canny Operator")
canny_button.pack()

sobel_button = tk.Radiobutton(algorithm_frame, text="Sobel алгоритам",
variable=algorithm_var, value="Sobel Operator")
sobel_button.pack()

prewitt_button = tk.Radiobutton(algorithm_frame, text="Prewitt алгоритам",
variable=algorithm_var, value="Prewitt Operator")
prewitt_button.pack()

roberts_button = tk.Radiobutton(algorithm_frame, text="Roberts Cross
алгоритам", variable=algorithm_var, value="Roberts Cross Operator")
roberts_button.pack()

# Create the apply button
apply_button = tk.Button(algorithm_frame, text="Примени алгоритам",
command=apply_button_clicked)
apply_button.pack(pady=10)

# Create the input image label
input_label = tk.Label(algorithm_frame)
input_label.pack()

# Create the output image label
output_label = tk.Label(algorithm_frame)
output_label.pack()

# Create the save button
save_button = tk.Button(algorithm_frame, text="Зачувај",
command=save_button_clicked)
save_button.pack(pady=10)

# Pack the algorithm frame initially (hidden)
algorithm_frame.pack_forget()

# Start the main loop
window.mainloop()
```

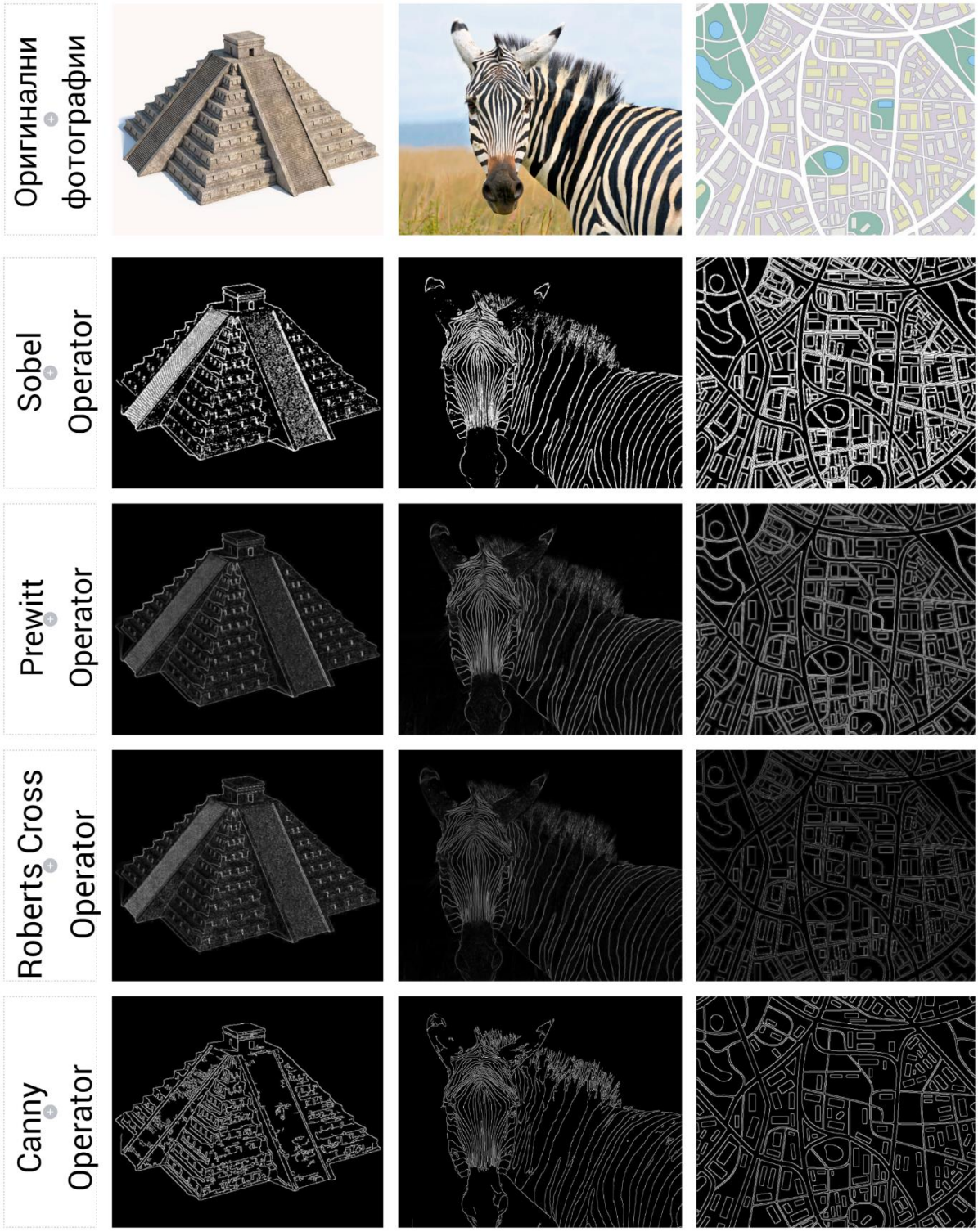
Код 5: Код од Демо апликацијата

2. Изглед на демо апликацијата



Слики 5, 6, 7 и 8: Изглед на Демо апликацијата

V. Анализа на алгоритмите за детекција на рабови



Слика 9: Преглед на резултантните слики од алгоритмите за детекција на рабови

Во продолжение е дадена анализа на алгоритмите за детекција на рабови опфатени во семинарската работа (Canny, Sobel, Prewitt и Roberts Cross), споредувајќи ги врз основа на нивните способности и можности:

1. Sobel Operator:

- Sobel е алгоритам за детекција на рабови базиран на градиент кој го приближува градиентот на сликата користејќи конволуција.

- Ја пресметува големината на градиентот за секој пиксел со конволвирање на сликата со две посебни кернели за хоризонтални и вертикални градиенти.

- Sobel произведува релативно дебели рабови поради употребата на фиксен праг за откривање на рабовите.

- Иако Sobel може прецизно да ги детектира рабовите, може да открие и слаби рабови, што доведува до лажни линии.

- Алгоритмот е релативно брз и пресметковно ефикасен во споредба со посложени методи за откривање рабови.

- Sobel е широко користен поради неговата едноставност и разумни перформанси за многу апликации.

2. Prewitt Operator:

- Алгоритмот Prewitt е сличен на алгоритмот Sobel, бидејќи исто така користи детекција на рабовите базирани на градиент користејќи конволуција.

- Prewitt ја пресметува големината на градиентот користејќи посебни кернели за хоризонтални и вертикални градиенти.

- Исто како Sobel, и Prewitt произведува релативно дебели рабови и може да биде чувствителен на грешки.

- Алгоритмот е поедноставен во споредба со понапредните техники за откривање рабови, но можеби нема да го обезбеди истото ниво на точност и робусност како методот Canny.

- Prewitt е пресметковно ефикасен и погоден за апликации во реално време со ограничени пресметковни ресурси.

3. Roberts Cross Operator:

- Roberts Cross е едноставен алгоритам за детекција на рабови кој го приближува градиентот на сликата со пресметување на квадратните разлики помеѓу дијагонално соседните пиксели.

- Користи две посебни кернели за дијагонални градиенти во две насоки.

- Робертс Крос има тенденција да произведува тенки рабови и може да биде чувствителен на грешки.

- Алгоритмот е пресметковно ефикасен, но може да не го нуди истото ниво на точност и робусност како пософистицираните техники како Canny.

- Roberts Cross може да биде корисен за апликации каде што едноставноста и брзината се приоритетни пред прецизноста за откривање на рабовите.

4. Canny Operator:

- Алгоритмот Canny е познат по својата висока точност и робусност.

- Користи повеќе чекори, вклучувајќи намалување на шумот, пресметка на градиент и не максимално потиснување.

- Алгоритмот ги одредува рабовите врз основа и на големината на градиентот и на насоката на градиентот, што резултира со тенки, добро поврзани рабови.

- Canny обезбедува флексибилност преку изборот на вредности на прагови за слаби и силни рабови, овозможувајќи контрола над чувствителноста за откривање на рабовите.

- Алгоритмот е ефикасен за откривање на широк опсег на рабови, вклучувајќи ги и оние со различен интензитет и нивоа на бучава.

- Canny е пресметковно интензивен во споредба со поедноставните техники за откривање рабови, но обезбедува супериорни резултати.

АЛГОРИТАМ	ПРЕДНОСТИ	НЕДОСТАТОЦИ
Canny Operator	Одлични перформанси во однос на останатите алгоритми и отпорен на грешки	Комплексен метод
Prewitt and Sobel Operators	Лесни за имплементација	Чувствителни на грешки
Roberts Cross Operator	Многу лесен за имплементација	Многу чувствителен на грешки

Табела 1: Краток преглед на предностите и недостатоците на алгоритмите

Забележително е дека алгоритмот за откривање на рабови Canny резултира со поголема прецизност во откривањето на рабовите и аглите, но не е најдобар за наоѓање повторливи агли, што се смета како еден од најважните критериуми за оценка на перформансите на откривањето на рабови. Во однос на ефикасноста, операторите Prewitt, Roberts и Sobel се брзи во споредба со другите за откривање на рабовите. Затоа, можеме да избереме различни детектори за рабови, наместо да го избереме детекторот за рабови Canny како идеален за секое сценарио.

Накратко, изборот на алгоритам за откривање рабови зависи од специфичните барања на апликацијата. Алгоритмот Canny обезбедува висока точност и робусност, но бара повеќе пресметковни ресурси. Операторите Sobel и Prewitt нудат едноставност и разумни перформанси, додека операторот Roberts Cross е лесна опција погодна за поедноставни задачи за откривање рабови. Разбирањето на силните и слабите страни на овие алгоритми помага во изборот на најсоодветниот врз основа на потребите на апликацијата.

VI. Заклучок

Во оваа семинарска работа се истражуваа и анализираа различни алгоритми за откривање рабови, поточно алгоритмите Sobel Operator, Canny Operator, Prewitt Operator и Roberts Cross Operator. Овие алгоритми играат клучна улога во компјутерската визија и апликациите за обработка на слики преку идентификување и извлекување на границите на објектите или регионите во сликата.

Секој алгоритам има свои силни и слаби страни, што ги прави погодни за различни сценарија и барања. Алгоритмите Sobel и Prewitt се едноставни и пресметковно ефикасни алгоритми кои добро функционираат во откривањето на рабовите со јасни насоки на градиент. Сепак, тие се почувствителни на грешки и може да создадат лажни рабови на покомплексните слики.

Canny Operator, од друга страна, е робусен и широко користен алгоритам за детекција на рабови. Вклучува повеќе чекори, како намалување на шумот, пресметка на градиент и не максимално потиснување. Алгоритмот Canny нуди подобра издржливост на грешки, контрола над праговите на јачината на рабовите и прецизна локализација на рабовите. Погоден е за различни апликации каде што е неопходно сигурно и прецизно откривање на рабовите.

Алгоритмот Roberts Cross е основен алгоритам кој ја пресметува големината на градиентот користејќи филтри 2x2. Иако е пресметковно ефикасен, му недостасуваат напредни техники присутни во Canny алгоритмот, што го прави почувствителен на грешки и склон кон лажни рабови.

Генерално, изборот на алгоритам за откривање на рабовите зависи од специфичните барања на апликацијата, карактеристиките на податоците за сликата и компромисите помеѓу точноста, сложеноста на пресметките и издржливоста на грешки. Canny алгоритмот се појавува како сеопфатно решение, нудејќи поголема флексибилност и перформанси во задачите за откривање рабови.

VII. Референци

[1] The performance analysis of edge detection algorithms for image processing in presence of noise, Munwar Ali Shaik, A. Ramesh, Y. Kumari

[2] Overview and Comparative Analysis of Edge Detection Techniques in Digital Image Processing, Chinu and Amit Chhabra

[3] Performance Analysis of Corner Detection Algorithms Based on Edge Detectors, Naurin Afrin, Wei Lai, Nabeel Mohammed

[4] Comparative analysis of common edge detection techniques in context of object extraction, S. K. Katiyar and P. V. Arun

[5] https://www.geeksforgeeks.org/implement-canny-edge-detector-in-python-using-opencv/

[6] https://www.geeksforgeeks.org/image-edge-detection-operators-in-digital-image-processing/

[7] https://nikatsanka.github.io/comparing-edge-detection-methods.html