МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет комп’ютерних наук

Кафедра програмної інженерії

ЗВІТ

з лабораторної роботи №2

з дисципліни: «ЧИСЕЛЬНІ МЕТОДИ ТА ВИСОКОПРОДУКТИВНІ ОБЧИСЛЕННЯ»

на тему: «Сегментація зображень. Знаходження кругів за допомогою перетворення Хафа»

Виконала: ст. гр. ІПЗм-19-2

Михневич Тетяна

Перевірила:

проф. Білоус Н. В.

Харків 2020

**Мета роботи**

Ознайомитись із методами сегментації зображень на основі перетворення Хафа, зробити програмну реалізацію метода.

**Хід роботи**

У лабораторній роботі буде досліджуватися метод знаходження кругів на зображенні за допомогою алгоритму Хафа. У якості датасетів було обрано дві групи об’їктів:

1. монети;
2. шари для більярду.

Для того, щоб знайти круги на зображенні, проходить декілька етапів трансформації зображення. На прикладі зображення монет розглянемо ці етапи:



Рисунок 1 – Вихідне зображення



Рисунок 2, 3 – Конвертація зображення в чорно-біле та застосування фільтра Гауса



Рисунок 4, 5 – застосування фільтра Собеля, подавлення не-максимумів

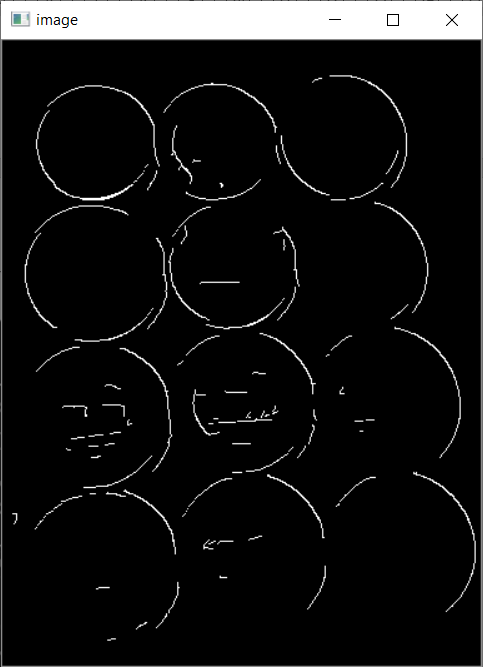


Рисунок 6, 7 -Визначення порога, бінарне зображення

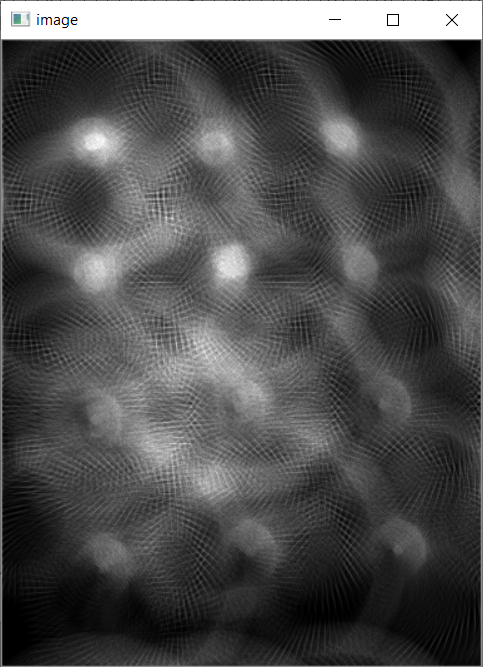


Рисунок 8, 9 – Процедура голосування, результат перетворення

Під час проведення експерименту змінювалися наступні параметри, що впливали на якість знаходження кругів:

1. мінімальний радіус пошуку (перший параметр, обов’язковий)
2. максимальний радіус пошуку (другий параметр, обов’язковий)
3. коефіцієнт фільтрації Гауса (третій параметр, за замовчуванням значення 5)
4. нижній коефіцієнт порогу (четвертий параметр, за замовчуванням значення 0.15)
5. верхній коефіцієнт порогу (п’ятий параметр, за замовчуванням значення 0.25)

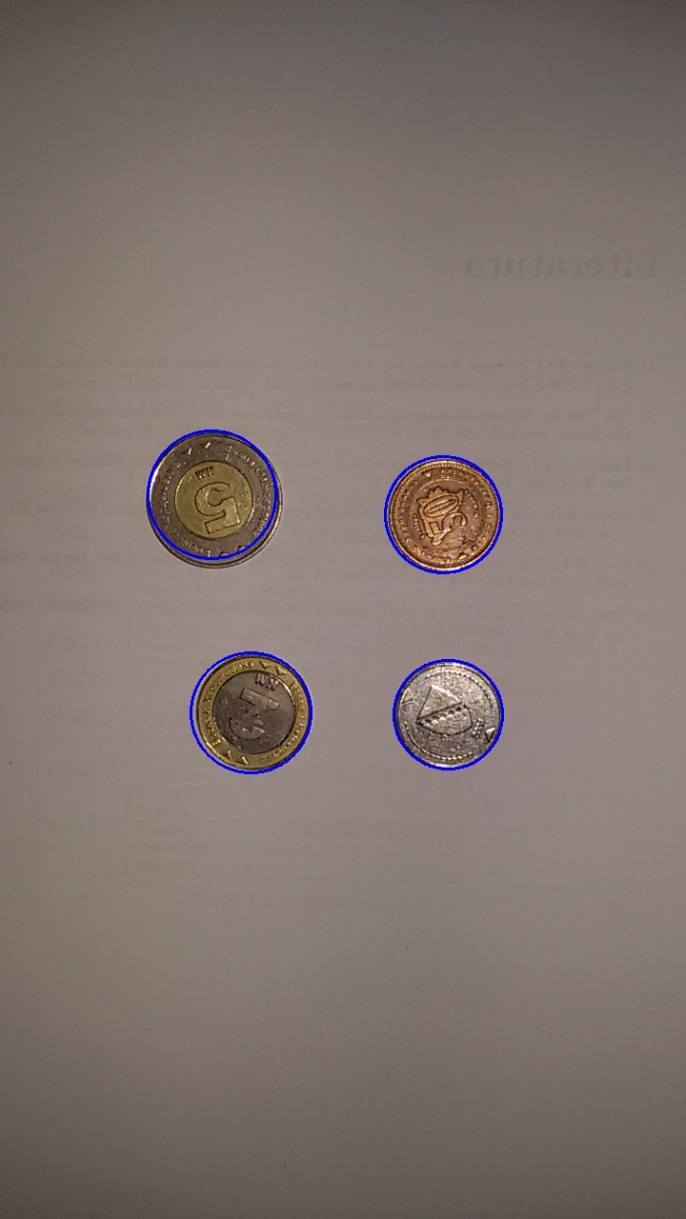


Рисунок 10 – Результат трансформації з параметрами 56, 68, 15



Рисунок 11 – Вихідне зображення



Рисунок 12 – Результат трансформації з параметрами 60, 80, 5, 0.2, 0.3



Рисунок 13 – Вихідне зображення



Рисунок 14 – Результат трансформації з параметрами 105, 120, 5, 0.15, 0.3



Рисунок 15 – Результат трансформації з параметрами 105, 120, 5, 0.3, 0.4



Рисунок 16 – Результат трансформації з параметрами 105, 120, 5, 0.2, 0.35



Рисунок 18 – Вихідне зображення



Рисунок 19 – Результат трансформації з параметрами 105, 115, 5, 0.2, 0.4



Рисунок 20 – Результат трансформації з параметрами 105, 115, 5, 0.3, 0.5



Рисунок 21 – Вихідне зображення



Рисунок 22 – Результат трансформації з параметрами 50, 75, 3



Рисунок 23 – Вихідне зображення



Рисунок 24 – Результат трансформації

**Програмний код реалізації завдання**

from collections import defaultdict

import glob

import cv2

import numpy as np

from math import cos, sin, pi

def display\_image(image):

if image.shape[0] > 1280:

scale\_percent = 25

else:

scale\_percent = 100

width = int(image.shape[1] \* scale\_percent / 100)

height = int(image.shape[0] \* scale\_percent / 100)

dim = (width, height)

image\_copy = cv2.resize(image, dim)

cv2.imshow('image', image\_copy)

cv2.waitKey()

def sobel\_filter(image):

kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)

ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float32)

dx = cv2.filter2D(image, -1, kx)

dy = cv2.filter2D(image, -1, ky)

g = np.hypot(dx, dy)

g = np.array(g / g.max() \* 255, np.uint8)

theta = np.arctan2(dy, dx)

return g, theta

def non\_max\_suppression(image, theta):

M, N = image.shape

Z = np.zeros((M, N), dtype=np.uint8)

angle = theta \* 180.0 / np.pi

angle[angle < 0] += 180

print("nonmax")

for i in range(1, M - 1):

for j in range(1, N - 1):

try:

q = 255

r = 255

# angle 0

if 0 <= angle[i, j] < 22.5 or 157.5 <= angle[i, j] <= 180:

q = image[i, j + 1]

r = image[i, j - 1]

# angle 45

elif 22.5 <= angle[i, j] < 67.5:

q = image[i + 1, j - 1]

r = image[i - 1, j + 1]

# angle 90

elif 67.5 <= angle[i, j] < 112.5:

q = image[i + 1, j]

r = image[i - 1, j]

# angle 135

elif 112.5 <= angle[i, j] < 157.5:

q = image[i - 1, j - 1]

r = image[i + 1, j + 1]

if image[i, j] >= q and image[i, j] >= r:

Z[i, j] = image[i, j]

else:

Z[i, j] = 0

except IndexError:

pass

return np.array(Z, np.uint8)

def threshold(img, low\_threshold\_ratio=0.15, high\_threshold\_ratio=0.25):

high\_threshold\_ratio = img.max() \* high\_threshold\_ratio

low\_threshold\_ratio = high\_threshold\_ratio \* low\_threshold\_ratio

M, N = img.shape

res = np.zeros((M, N), dtype=np.uint8)

weak = np.uint8(125)

strong = np.uint8(255)

strong\_i, strong\_j = np.where(img >= high\_threshold\_ratio)

weak\_i, weak\_j = np.where((img < high\_threshold\_ratio) & (img >= low\_threshold\_ratio))

res[strong\_i, strong\_j] = strong

res[weak\_i, weak\_j] = weak

return np.array(res, np.uint8)

def binary\_image(img):

weak = 125

strong = 255

M, N = img.shape

for i in range(1, M - 1):

for j in range(1, N - 1):

if img[i, j] == weak:

try:

if ((img[i + 1, j - 1] == strong) or (img[i + 1, j] == strong) or (img[i + 1, j + 1] == strong)

or (img[i, j - 1] == strong) or (img[i, j + 1] == strong)

or (img[i - 1, j - 1] == strong) or (img[i - 1, j] == strong) or (

img[i - 1, j + 1] == strong)):

img[i, j] = strong

else:

img[i, j] = 0

except IndexError:

pass

return np.array(img, np.uint8)

def voting(edges, min\_radius, max\_radius):

array = defaultdict(int)

result\_image = np.zeros((edges.shape[0], edges.shape[1]), np.uint8)

blank\_image = np.zeros((edges.shape[0], edges.shape[1]))

for r in range(min\_radius, max\_radius):

for i in range(0, edges.shape[0]):

for j in range(0, edges.shape[1]):

if edges[i][j] == 255:

for teta in range(0, 360, 5):

a = int(i - r \* cos(teta \* pi / 180))

b = int(j - r \* sin(teta \* pi / 180))

try:

blank\_image[a][b] += 1

array[(a, b, r)] += 1

except IndexError:

continue

print(r, i)

maxim = np.amax(blank\_image)

for i in range(blank\_image.shape[0]):

for j in range(blank\_image.shape[1]):

result\_image[i][j] = int(255 \* float(blank\_image[i][j] \* 1.0 / maxim \* 1.0))

return result\_image, array

def extract\_circles(array):

circles\_list = []

for k in sorted(array, key=lambda i: -i[2]):

x, y, r = k

if array[k] >= 16 and x > 0 and y > 0 and all(

(x - xc) \*\* 2 + (y - yc) \*\* 2 > rc \*\* 2 for xc, yc, rc in circles\_list):

print(array[k], x, y, r)

circles\_list.append((x, y, r))

return circles\_list

def draw\_circles(circles\_list, image):

for circle in circles\_list:

cv2.circle(image, (circle[1], circle[0]), circle[2], (255, 0, 0), 2)

#display\_image(image)

def main(image\_name, min\_radius, max\_radius, gaussian\_kernel\_dimension = 5, lowThresholdRatio = 0.15, highThresholdRatio = 0.25):

image = cv2.imread(image\_name)

img\_orig = image.copy()

image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

#display\_image(image)

image = cv2.GaussianBlur(image, (gaussian\_kernel\_dimension, gaussian\_kernel\_dimension), 0)

#display\_image(image)

image, theta = sobel\_filter(image)

#display\_image(image)

image = non\_max\_suppression(image, theta)

#display\_image(image)

image = threshold(image, lowThresholdRatio, highThresholdRatio)

#display\_image(image)

image = binary\_image(image)

#display\_image(image)

image, array = voting(image, min\_radius, max\_radius)

#display\_image(image)

circles = extract\_circles(array)

draw\_circles(circles, img\_orig)

cv2.imwrite(image\_name.replace("source", "output"), img\_orig)

for filename in glob.glob("\*"):

main(filename, 45, 65, 5, 0.2, 0.4)

**Висновки:** у результаті виконання лабораторної роботи ми ознайомились із методами сегментації зображень на основі перетворення Хафа, зробили програмну реалізацію метода.

Виходячи з результатів експерименту можна виділити позитивні та негативні сторони алгоритму.

Позитивні:

* алгоритм досить ефективно знаходить круги на зображеннях;
* алгоритм здатний виявити кілька об'єктів одного типу за одне виконання. Це відбувається через те, що за один прохід утворюється кілька локальних максимумів.

Негативні:

* алгоритм працює довго, що не дає змогу використовувати його в реальному часі. При збільшенні розмірності двоїстого простору, виростають і займані обсяги пам'яті, як наслідок цього, зростання трудомісткості обчислень;
* для того, щоб максимально ефективно виділити круги, треба задавати правильний радіус пошуку, що вимагає попереднього перегляду зображення;
* алгоритм досить чутливий до зміни параметру коефіцієнта порога, тому для максимально ефективного результату треба підібрати цей параметр для різних груп зображень вручну;
* не завжди всі круги, присутні на зображенні, знаходяться, та навпаки, знаходяться зайві;
* некоректно працює при великій кількості кругів, їх перетинанні.

Також можна розробити рекомендації щодо використання алгоритму:

1. Алгоритм Хафа можна використовувати для об'єктів, які його можна математично описати у вигляді параметричного рівняння;
2. Використання переведення Хафа на зашумлених зображеннях утруднене. Для зашумленних зображень необхідний етап попередньої обробки з метою придушення шуму;
3. Якщо зображення містить шум, треба збільшити третій параметр – коефіцієнт фільтру Гауса. Чим більше цей параметр, тим більше шуму буде знищено, але й тим менш чітким буде зображення;
4. Для максимально ефективного знайдення кругів треба як можна точніше задавати радіус кругів для пошуку;
5. Якщо на зображенні круглі об’єкти чітко виділяються, у них висока контрастність з фоном, то нижній коефіцієнт порогу треба ставити вище, щоб зайві об’єкти не були розпізнанні, як круги;
6. Якщо на зображенні круглі об’єкти виділяються не чітко, вони змішуються з фоном, то нижній коефіцієнт порогу треба ставити нижче, щоб алгоритм зміг знайти більше кругів.