**1. Đề bài**:

Bài toán tự động đọc số công tơ nước.

**2. Giới thiệu cách tiếp cận**

***2.1. Phân ngưỡng ảnh***

Thao tách phân ngưỡng ảnh thực hiện việc giảm mức xám biểu diễn trong ảnh về 2 mức, hay còn gọi là quá trình nhị phân hóa ảnh.

Gọi ảnh vào là I, ảnh kết quả sau phân ngưỡng là I’, giá trị ngưỡng là T. Khi đó, giá trị của ảnh I’ tại vị trí (i, j) được xác định theo công thức:

(1)

Với min mặc định là 0; max được chọn dựa trên cách biểu diễn, với ảnh mà mỗi điểm ảnh được biểu diễn bởi 8 bít ta chọn max=255.

***2.2. Thuật toán tách biên Canny***

Thuật toán tách biên Canny được thực hiện bằng 4 bước chính:

- Làm trơn ảnh vào bằng bộ lọc Gausse giúp giảm nhiễu. Bộ lọc Gaussian kích thước 5x5. Kích thước 5x5 thường hoạt động tốt cho giải thuật Canny.

- Tính toán ảnh biên độ Gradient và pha Gradient: ta dùng bộ lọc Sobel X và Sobel Y (3x3) để tính được ảnh đạo hàm Gx và Gy.

- Áp dụng phương pháp loại bỏ các điểm không phải cực đại cho ảnh biên độ Gradient.

- Sử dụng phân ngưỡng kép và phân tích kết nối để phát hiện và liên kết biên.

Thuật toán tách biên Canny được hỗ trợ trong python và opencv bằng câu lệnh sau đây:

blurred = cv2.GaussianBlur(image, (5, 5), 0)

edged = cv2.Canny(blurred, 60, 200, 255)

***2.3 Xử lý hình thái – Phép giãn ảnh***

- Phép giãn ảnh làm dày/ làm to đối tượng, làm cho đối tượng phát triển.

+ Điền (làm đầy, nối) những đoạn đứt gãy

+ Bổ sung những phần khuyết thiếu của đối tượng

- Phép giãn ảnh được hỗ trợ trong python và opencv bằng câu lệnh sau đây:

kernel = cv2.getStructuringElement(cv2.MORPH\_CROSS, (3,3))

edged = cv2.morphologyEx(edged, cv2.MORPH\_CLOSE, kernel)

**3. Thử nghiệm và kết quả**

***3.1.1. Thu nhận và tìm biên ảnh thu được***

Đầu tiên ta đọc dữ liệu từ ảnh:

image = cv2.imread("water\_meter.jpg")



Khi thu được ảnh màu RGB, ta tiến hành chuyển ảnh RGB sang ảnh xám rồi làm mịn ảnh và dùng thuật toán phát hiện biên để tìm ra biên bằng thuật toán phát hiện biên Canny.

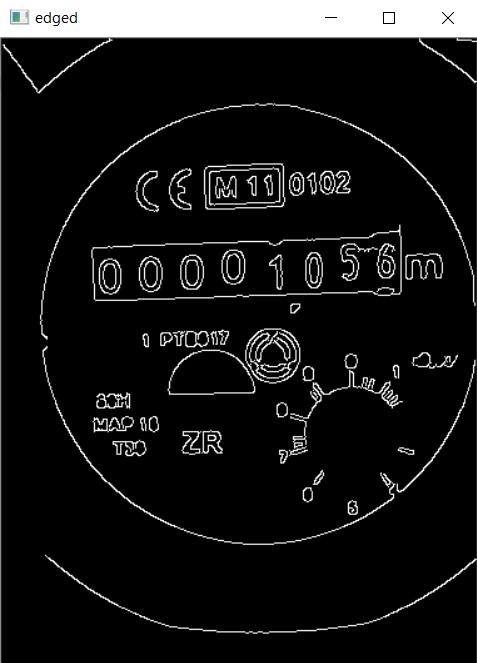
Quá trình trên có thể tóm lược bằng ngôn ngữ giải thuật (mã giả) như sau:

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

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

edged = cv2.Canny(blurred, 60, 200, 255)

Thu được kết quả như sau:



**3.1.2. Lọc bỏ biên không phù hợp**

Ta tìm các đường viền từ ảnh edged ở phần trên bằng phương thức findContours được hỗ trợ trong Python.

Khi có danh sách contours, ta lọc bỏ các contour có chu vi nhỏ - không đủ điều kiện (condition) để là biên cần tìm. Trong trường hợp này, ta bỏ các contour có chu vi nhỏ hơn 50% kích thước chiều rộng của ảnh (sử dụng phép toán thao tác với bit XOR để lọc contours).

Quá trình trên có thể tóm lược bằng ngôn ngữ giải thuật (mã giả) như sau:

# get height, width of image

height = image.shape[0]

width = image.shape[1]

condition = 0.5 \* width

cnts = cv2.findContours(edged.copy(), cv2.RETR\_CCOMP,cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

cnts = sorted(cnts, key=cv2.contourArea, reverse=True)

mask = np.zeros(image.shape, dtype="uint8")

#remove unwanted contours

for c in cnts:

if cv2.arcLength(c, False) < condition:

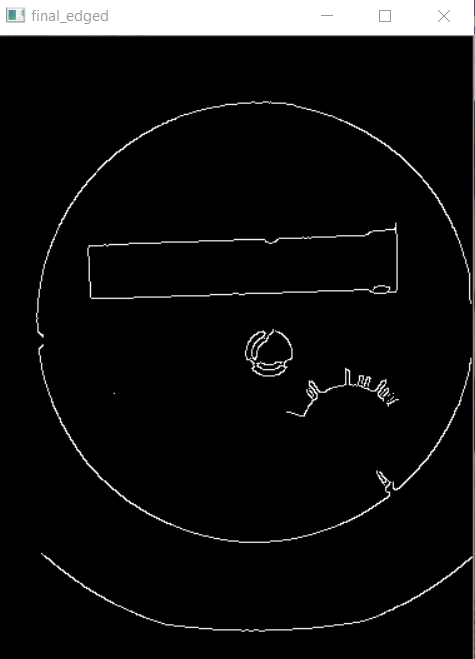
cv2.drawContours(mask, [c], 0, (255,255,255,255), 1)

#cv2.imshow('water\_meter\_canny\_copy.jpg', mask)

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

edged = cv2.bitwise\_xor(mask, edged)

Thu được kết quả như sau:



**3.1.3. Xác định biên bao ngoài vùng chỉ số công tơ**

Để tìm biên bao ngoài, ta thực hiện công việc tìm biên có dạng hình chữ nhật với diện tích lớn nhất trong ảnh. Tuy nhiên, do các đường biên là không liền, bị đứt đoạn nên ta sử dụng xử lý ảnh hình thái (cụ thể là phép giãn ảnh làm dày đối tượng) để phát hiện các hình chữ nhật chính xác hơn.

Quá trình trên có thể tóm lược bằng ngôn ngữ giải thuật (mã giả) như sau:

def is\_contour\_good(c):

#approximate the contour

peri = cv2.arcLength(c, True)

approx = cv2.approxPolyDP(c, 0.02 \* peri, True)

return len(approx) == 4

kernel = cv2.getStructuringElement(cv2.MORPH\_CROSS, (3,3))

edged = cv2.morphologyEx(edged, cv2.MORPH\_CLOSE, kernel)

cnts = cv2.findContours(edged.copy(), cv2.RETR\_CCOMP,cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

cnts = sorted(cnts, key=cv2.contourArea, reverse=True)

frame = None

max = 0.0

for c in cnts:

if is\_contour\_good(c):

peri = cv2.arcLength(c, True)

if peri > max:

max = peri

frame = c

Thu được kết quả như sau:



Khung chỉ số công tơ được trích ra từ ảnh ban đầu:



**3.1.4. Tách từng chữ số trong ảnh**

Để tiến hành tách các chữ số trong ảnh ta chỉnh lại kích cỡ của vùng chứa chỉ số công tơ 40x200 sau đó tiến hành phân ngưỡng tĩnh với giá trị ngưỡng là 110.

Tiếp tục dùng hàm findContours để tìm biên, sau đó dựa vào đặc điểm hình học các chữ số để có thể tìm ra các vùng có khả năng là chữ số. Trong trường hợp này là chiều rộng > 5 pixel và chiều cao từ 15 pixel đến 20 pixel.

Cuối cùng, sắp xếp các vùng thỏa mãn theo thứ tự từ trái sang phải để chuẩn bị cho việc nhận diện.

Quá trình trên có thể tóm lược bằng ngôn ngữ giải thuật (mã giả) như sau:

dim = (200,40)

img = cv2.resize(img, dim, interpolation = cv2.INTER\_AREA)

gray\_img = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

ret,thresh = cv2.threshold(gray\_img,110,255,cv2.THRESH\_BINARY)

cv2.imwrite("water\_meter\_cut\_bw.png",thresh)

cnts = cv2.findContours(thresh.copy(), cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

digitCnts = []

for c in cnts:

(x, y, w, h) = cv2.boundingRect(c)

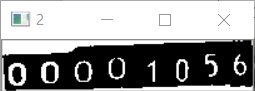
if w >= 5 and (h >= 15 and h <= 20):

digitCnts.append(c)

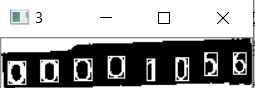
digitCnts = contours.sort\_contours(digitCnts,method="left-to-right")[0]

Kết quả thu được như sau:

- Ảnh sau khi phân ngưỡng:



- Các vùng có thể là chữ số tìm được:



**3.2 Nhận diện ảnh**

***3.2.1 Xây dựng module để nhận diện ảnh***

**# Import the modules**

import joblib

from sklearn import datasets

from skimage.feature import hog

from sklearn.svm import LinearSVC

import numpy as np

from collections import Counter

**# Load the dataset**

features, labels = datasets.fetch\_openml('mnist\_784', version=1, return\_X\_y=True)

**# Extract the hog features**

list\_hog\_fd = []

for feature in features:

fd=hog(feature.reshape((28,28)),orientations=9,pixels\_per\_cell=(14,14),cells\_per\_block=(1, 1))

list\_hog\_fd.append(fd)

hog\_features = np.array(list\_hog\_fd, 'float64')

**# Create an linear SVM object**

clf = LinearSVC()

# Perform the training

clf.fit(hog\_features, labels)

# Save the classifier

joblib.dump(clf, "digits\_cls.pkl", compress=3)

***3.2.2 Nhận diện các số trong ảnh đã trích từ ảnh gốc***

**# Import the modules**

import cv2

import joblib

from skimage.feature import hog

import numpy as np

import imutils

from imutils import contours

**# Load the classifier**

clf = joblib.load("digits\_cls.pkl")

**# Read the input image**

im = cv2.imread("water\_meter\_cut\_bw.png")

im\_gray = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY)

ctrs = cv2.findContours(im\_gray.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

**# Get rectangles contains each contour**

ctrs = imutils.grab\_contours(ctrs)

digitCnts = []

for c in ctrs:

(x, y, w, h) = cv2.boundingRect(c)

if w >= 5 and (h >= 15 and h <= 25):

digitCnts.append(c)

digitCnts = contours.sort\_contours(digitCnts,method="left-to-right")[0]

rects = [cv2.boundingRect(ctr) for ctr in digitCnts]

**# For each rectangular region, calculate HOG features and predict**

**# the digit using Linear SVM.**

for rect in digitCnts:

x, y, width, height = cv2.boundingRect(rect)

cv2.rectangle(im, (x,y), (x+width,y+height), (0, 255, 0), 1)

roi = im\_gray[y:y+height, x:x+width]

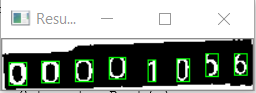
roi = cv2.resize(roi, (28, 28), interpolation=cv2.INTER\_AREA)

roi\_hog\_fd = hog(roi, orientations=9, pixels\_per\_cell=(14, 14), cells\_per\_block=(1, 1))

nbr = clf.predict(np.array([roi\_hog\_fd], 'float64'))

print(str(int(nbr[0])))

Kết quả sau khi chạy:



>>>

= RESTART: D:\silde bai giang\001\xử lý ảnh\digitRecognition\performRecognition.py

0

0

0

0

7

0

7

0

**4. Thảo luận và bài học thu được**

- Kết quả:

+ Hình ảnh đầu vào ban đầu là:



+ Kết quả thu được sau khi dùng module để nhận diện ảnh là:

0 0 0 0 7 0 7 0

- Kết quả thu được bị sai khác ở một vài số so với ảnh gốc ban đầu. Lỗi có thể xảy ra trong quá trình nhận diện do phần contours (được xác định có thể là số) không có đường nét rõ ràng sau khi phân ngưỡng, khác biệt với datasets ban đầu của module.

**5. Tài liệu tham khảo**

1. Recognizing digits with OpenCV and Python by Adrian Rosebrock

2. Phần module nhận diện chữ số được tham khảo qua đường link dưới đây:

<https://github.com/noorkhokhar99/Digit-Recognition>

**6. Phụ lục**

- Dưới đây là phần code đầy đủ để tách khung chỉ số công tơ nước. Phần mã nguồn nhận diện số đã được trình bày đầy đủ ở phần trên.

from imutils.perspective import four\_point\_transform

from imutils import contours

import imutils

import cv2

import numpy as np

from PIL import Image

def is\_contour\_good(c):

#approximate the contour

peri = cv2.arcLength(c, True)

approx = cv2.approxPolyDP(c, 0.02 \* peri, True)

return len(approx) == 4

# load the example image

image = cv2.imread("water\_meter.jpg")

cv2.imshow("image",image)

# get dimensions,height,width of image

dimensions = image.shape

height = image.shape[0]

width = image.shape[1]

condition = 0.5 \* width

# pre-process the image by resizing it, converting it to

# graycale, blurring it, and computing an edge map

image = imutils.resize(image, height=500)

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

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

edged = cv2.Canny(blurred, 60, 200,255)

cv2.imshow("edged",edged)

#cv2.imshow('water\_meter\_canny.jpg',edged)

cnts = cv2.findContours(edged.copy(), cv2.RETR\_CCOMP,cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

cnts = sorted(cnts, key=cv2.contourArea, reverse=True)

mask = np.zeros(image.shape, dtype="uint8")

#remove unwanted contours

for c in cnts:

if cv2.arcLength(c,False)<condition:

cv2.drawContours(mask, [c], 0, (255,255,255,255), 1)

#cv2.imshow('water\_meter\_canny\_copy.jpg',mask)

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

edged = cv2.bitwise\_xor(mask, edged)

cv2.imshow("final\_edged",edged)

#the second morphology

kernel = cv2.getStructuringElement(cv2.MORPH\_CROSS, (3,3))

edged = cv2.morphologyEx(edged, cv2.MORPH\_CLOSE, kernel)

cnts = cv2.findContours(edged.copy(), cv2.RETR\_CCOMP,cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

cnts = sorted(cnts, key=cv2.contourArea, reverse=True)

frame = None

max = 0.0

for c in cnts:

if is\_contour\_good(c):

peri = cv2.arcLength(c, True)

if peri > max:

max = peri

frame = c

#cv2.drawContours(image, [frame], 0, (0,255,0), 3)

x, y, width, height = cv2.boundingRect(frame)

#cv2.imshow("we\_got\_this",image)

roi = image[y:y+height, x:x+width]

cv2.imwrite("water\_meter\_cut.png", roi)

img = cv2.imread("water\_meter\_cut.png")

dim = (200,40)

img = cv2.resize(img, dim, interpolation = cv2.INTER\_AREA)

gray\_img = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

ret,thresh = cv2.threshold(gray\_img,110,255,cv2.THRESH\_BINARY)

cv2.imwrite("water\_meter\_cut\_bw.png",thresh)

cv2.imshow("2",thresh)

cnts = cv2.findContours(thresh.copy(), cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

digitCnts = []

for c in cnts:

(x, y, w, h) = cv2.boundingRect(c)

if w >= 5 and (h >= 15 and h <= 20):

digitCnts.append(c)

digitCnts = contours.sort\_contours(digitCnts,method="left-to-right")[0]

i=0

for c in digitCnts:

x, y, width, height = cv2.boundingRect(c)

cv2.rectangle(thresh, (x,y), (x+width,y+height), (255, 255, 255), 1)

i = i+1

string = "water\_meter\_cut\_"+str(i)+".png"

x, y, width, height = cv2.boundingRect(c)

roi = thresh[y:y+height, x:x+width]

cv2.imwrite(string, roi)

old\_im = Image.open(string)

old\_im = old\_im.resize((14,14))

old\_size = old\_im.size

new\_size = (20, 20)

new\_im = Image.new("RGB", new\_size)

new\_im.paste(old\_im,(int((new\_size[0]-old\_size[0])/2),int((new\_size[1]-old\_size[1])/2)))

new\_im.save(string)