Curtin University – Department of Computing

# Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

| | | | |
|---|---|---|---|
| Last name: | Yap | Student ID: | 19257442 |
| Other name(s): | Kay Men | | |
| Unit name: | Machine Perception | Unit ID: | COMP3007 |
| Lecturer / unit coordinator: | Dr. Senjian An | Tutor: | Bradley Ezard |
| Date of submission: | 27/10/200 | Which assignment? | 2 (Leave blank if the unit has only one assignment.) |

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: Kay Men Yap Date of signature: 27/10/2020

*(By submitting this form, you indicate that you agree with all the above text.)*

# MP Assignment 2

Kay Men Yap
*Student No: 19257442*
*Curtin University*
Perth, Australia
19257442@student.curtin.edu.au

CONTENTS

## I. Attempt

I've attempted every part of the assignment and successfully was able to detect the area of the building number, get the individual digits and recognise the digits using KNN.

## II. Implementation

My preprocess method in assignment.py consists of the code to preprocess, detect the bounding box of the area where the digits reside in the image and return the list of the extracted digits. I didn't refactor the code to be in separate methods in order to not waste time debugging anything that might break as i refactor the code that is already working.

My initial approach to the assignment is based on Prac 07 of MP which is simply preprocessing the image, getting the digit contours then extract the digits and finally feed the digits extracted into KNN except for the part where Prac 07 considers different orientations of the digits. Since it worked out quite well, I stuck with this approach but did tried experiementing with getting the black plate of the digts using HSV colour space and inRange function [2] which didn't work out well.

### *A. Preprocessing*

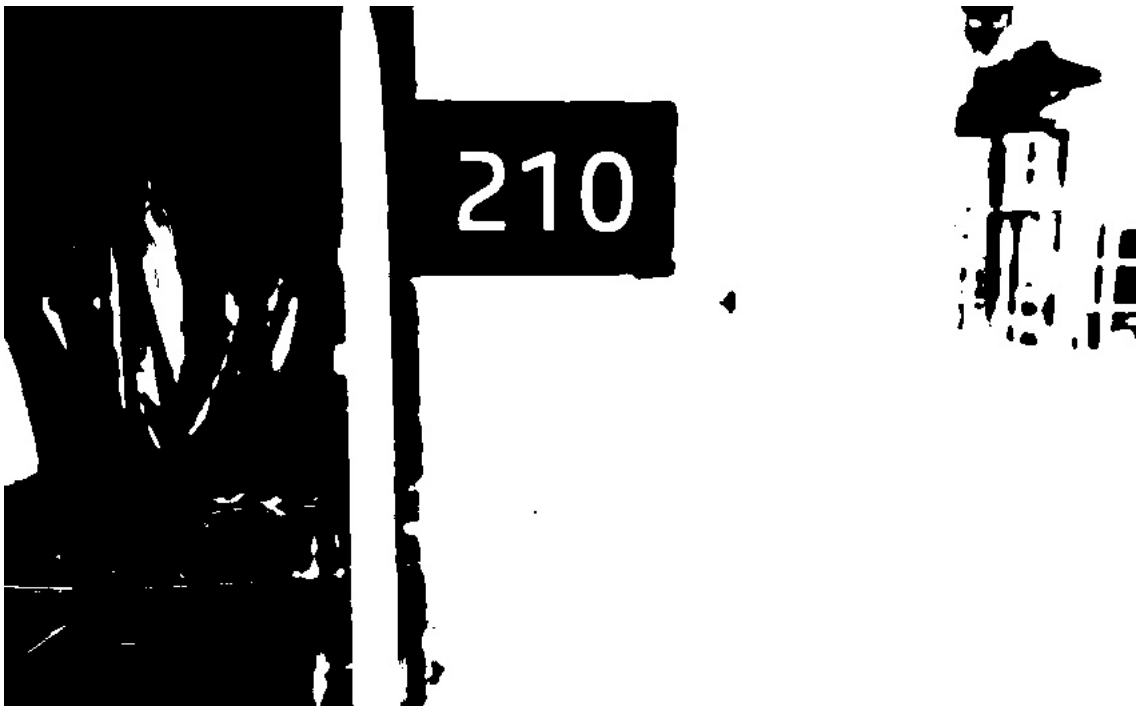



Fig. 1. tr03 binary image

In the end, I simply went with grayscaling the image, then apply bilateral filter and threshold the image using the flag, cv2.THRESH_BINARY and cv2.THRESH_OTSU.

I gray scaled the image first because I wanted to get a binary image at the end of preprocessing and a binary image only has a single colour channel and the initial image I read in will be in BGR format just in case I wanted to experimented with other colour spaces which I did try with HSV colour space

I applied bilateral filter because I was experimenting with Canny Edge Detection early on [1] and bilateral filter will blur the image while keeping the edges sharp [4] which is good to use while trying out Canny Edge Detection and I just kept using bilateral filter since it will still reduce the noisiness of the image and works with my current implementation. I went with normal thresholding and not adaptive thresholding as I wasn't able to detect the contours of all digits in the image when using adaptive thresholding so I sticked with normal thresholding. I

uses cv2.THRESH_OTSU flag with normal thresholding as each image requires a slightly different threshold value so instead of hardcoding the limit, hence the use of Otsu's binarization [3].

I also preprocess the detected area image the same way where i grayscaled the image, apply bilateral filter and threshold the image using the flag, cv2.THRESH_BINARY and cv2.THRESH_OTSU again before doing the digit extraction. My current implementation works well as long as there is no high contrast on the left and right side or top and bottom side of the black plate where the number resides. This is evident in val04 binary image as show in Figure 2 below. Hence, the need to use cv2.findContours once on the original binary image and then another time on the detected area binary image which will be mentioned in the Segmentation of Digits section.


Fig. 2. val04 binary image

I tried another method aside from my current implementation where I try to filter out the non-black-ish and non white-ish colours using HSV colour space and inRange function of OpenCV [2]. However, I was unable to get a proper mask to remove the unnecessary objects from the image using this method, hence I didn't tried to implement this method for the assignment.

*B. Detection*


Fig. 3. tr03 unfiltered contours image

After getting the binary image from preprocessing the image, I used contours to detect the digits in order to get the bounding box of the detected area. Figure 3 shows all of the contours detected initially without filtering.


Fig. 4. tr03 filter 1 contours image

After getting all contours from the binary image, I initially filtered out the contours are too big (greater than 0.25 of the image size) or too small (less than 0.0001 of the image size), any contours that don't have a bounding rectangle [5] that is a vertical rectangle by checking the height to width ratio of the bounding rectangle, and any contours where their bounding rectangle is too close to the edges of the image while keeping track the largest

contour that is valid based on the filters i mentioned just now. These filters are based on assumptions I figured out from the training and val images. The resulting contours are show in Figure 4. I initially used the contour area [6] to filter the contours before I started using the bounding rectangle area as the bounding rectangle reflects the area I will be cropping out and easier to compare to the entire image size.


Fig. 5. tr03 filter 2 contours image

After the initial filter, I filtered again on the remaining contours where I filter out any contour that doesn't have a bounding rectangle area that is at least 35% of the largest contour obtained earlier and the center y value of the bounding rectangle of the contour is near the value of the centre y value of the largest contour with an acceptable difference of 0.25 of the height of the largest contour. I performed this filter as I discovered there are still some extra contours after the initial filter and the largest contour is always one of the digit contours based on what I observed when drawing the initially filtered contours in order to come up with additional filters to remove any contours that are not the digits. The resulting contours are show in Figure 5.


Fig. 6. tr03 filter 3 contours image


Fig. 7. tr03 cropped image

After that second filter, the only remaining contours are the ones that are at where the digit is but there is still some inner contours within the contours that encompass each individual digit so I apply another filter to remove any contours that are within another contours. The resulting contours are show in Figure 6. Afterwards, I get the minimum x, y values and maximum x,y values of the remaining contours in order to get the coordinates to crop the image to get the detected area of the image and save it as a file, and write the bounding box coordinates of the detected area to the BoundingBoxX.txt file.

### C. Segmentation of digits

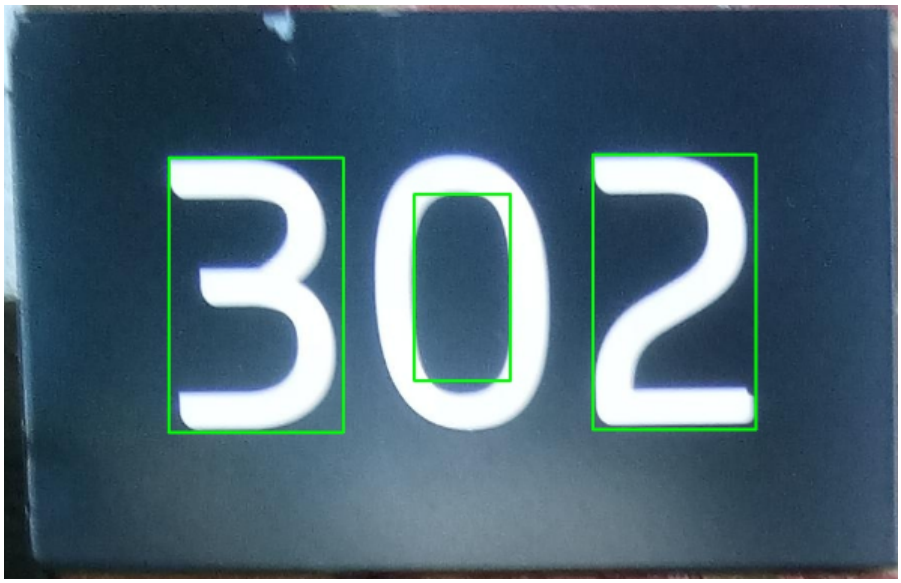



Fig. 8. Val04.jpg bad contour at digit 0

Originally I just used the digit contours from the detection phase but this approach works for all training and val images except for val04.jpg since the 0 contour doesn't cover the entire 0 digit as evident in Figure 8. Hence, I preprocess the cropped image that contains the detected area by grayscaling the image, applying bilateral filter and thresholding the same way as the original image before I tried to find the contours on the cropped binary image.

After getting all contours, I filtered out any contours that are too small and any contours are that are within any other contours in the same way I did with the original image in terms of filtering based on size and inner contours.

The resulting filtered contour list will only contain the digit contours which I will get the bounding box of the contour then cropping the image to only have the individual digit from the binary image. However, I discovered that the order of contours in the filtered list was not in order from left to right of the image so I had to sort the contours based the x coordinate of the upper left corner of the bounding rectangle of the contours [9]. Then, I added 4 pixel black border [7] to each individual cropped digit image as the cropping of the digits are too tight when compared to the training digits that have small black borders around the training digits which will affect the result I obtain from digit recognition. After getting the digit cropped out and added black borders, I resized [8] the cropped digit to be the same dimensions as the training images which is 28x40 resolution.

### D. Digit recognition

After getting the extracted digits list from segmentation of digits phase, I loaded the trained KNN model in the models folder of my assignment and convert the list into an array and reshaped each digit image into a 1d array using the numpy.reshape method and converted each reshaped digit array to an array of float32 datatype as KNN requires a 1d array of flaots for each digit image to do training and prediction. The reason I went for KNN and not SVM is because KNN works well with small data sets and the data set we have been given to work with is a small data set of images. Also, I tried KNN first and it worked out well so I didn't need to try SVM as I was getting quite accurate results overall.

I went with KNN with K=3 as that yielded the most accurate results of the digit recognition without overfitting too much when I supplied the digit array into the findNearest method of the KNN trained model. I experimented with K=1,3 and 5 and K=1 will most likely introduce overfitting and K=5 produced less accurate results of the digits when compared to using K=3 when calling findNearest method.

## III. Performance

I've tested on both training and val images and it took 1.1 seconds to run assignment.py on both folders. All of the images' detected area are correct. However, some of the images' house number isn't predicted correctly by my KNN model. The images in questions are val05, tr04, tr08, tr15 and tr16. I have included the results of all val images and all of the training images that got the digits being predicted incorrectly by my KNN model.

Below is the table of results from running assignment.py on the val images.

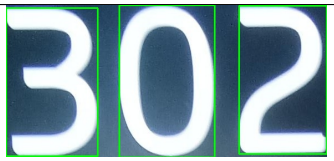
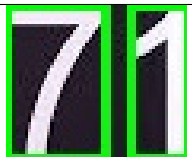
| Filename | Detected Area | Bounding Box details | House Number |
|---|---|---|---|
| val01 |  | 49,47,115,100 | Building 48 |
| val02 |  | 82,95,63,52 | Building 35 |
| val03 |  | 57,56,107,100 | Building 94 |
| val04 |  | 737,408,579,274 | Building 302 |
| val05 |  | 77,91,57,48 | Building 70 |
| val06 |  | 81,92,104,76 | Building 26 |

Table 1: Results obtained from val images

Aside from val05's house number being incorrect by 1 digit where the number "1" is detected as a "0", the rest of the val images have the correct house number. The incorrect prediction might be due to extracting the digits too tightly and not adding enough black borders around the images as I currently have hardcoded the amount of black borders to add to each digit regardless of resolution.

I had similar results with the training images and below is a table of training images that have 1 or 2 digits being predicted incorrectly by my KNN model. The rest of the training images' house numbers are detected correctly.

| Filename | Detected Area | Bounding Box details | House Number |
|---|---|---|---|
| tr04 |  | 48,96,130,69 | Building 233 |
| tr08 |  | 205,289,239,113 | Building 300 |
| tr15 |  | 77,64,97,57 | Building 323 |
| tr16 |  | 108,75,68,24 | Building 1263 |

Table 2: Results obtained from train images

# REFERENCES


[1] OpenCV. OpenCV: Canny Edge Detection. https://docs.opencv.org/3.4.2/da/d22/tutorial_py_canny.html (accessed 25/10/2020).

[2] OpenCV. OpenCV: Thresholding Operations using inRange. https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html (accessed 25/10/2020).

[3] OpenCV. OpenCV: Image Thresholding. https://docs.opencv.org/3.4.2/d7/d4d/tutorial_py_thresholding.html (accessed 25/10/2020).

[4] OpenCV. OpenCV: Smoothing Images. https://docs.opencv.org/3.4.2/d4/d13/tutorial_py_filtering.html. (accessed 25/10/2020).

[5] OpenCV. OpenCV: Contours: Getting Started. https://docs.opencv.org/3.4.2/d4/d73/tutorial_py_contours_begin.html. (accessed 25/10/2020).

[6] OpenCV. OpenCV: Contour Features. https://docs.opencv.org/3.4.2/dd/d49/tutorial_py_contour_features.html. (accessed 25/10/2020).

[7] OpenCV. OpenCV: Adding borders to your images. https://docs.opencv.org/3.4/dc/da3/tutorial_copyMakeBorder.html. (accessed 25/10/2020).

[8] OpenCV. OpenCV: Geometric Transformations of Images. https://docs.opencv.org/3.4.2/da/d6e/tutorial_py_geometric_transformations.html. (accessed 25/10/2020).

[9] Adrian Rosebrock. Sorting Contours using Python and OpenCV - PyImageSearch. https://www.pyimagesearch.com/2015/04/20/sorting-contours-using-python-and-opencv/. (accessed 26/10/2020).

## APPENDIX

### Main assignment file

```
import cv2 as cv
import numpy as np
import sys
from pathlib import Path
import os
import re

def knn(digits, folder, fileName, imageNum, fileExt):
    #modelsFolder = "models/"
    modelsFolder = "/home/student/kay_men_yap_19257442/models/"
    # KNN model loading and testing code obtained from https://docs.opencv.org/3.4.2/d8/d4b/tutorial_py_knn_opencv.html
    # Last accessed on 24/10/2020
    with np.load(modelsFolder+'knn_data.npz') as data:
        train = data['train']
        train_labels = data['train_labels']
        knn = cv.ml.KNearest_create()
        knn.train(train, cv.ml.ROW_SAMPLE, train_labels)

        # Convert the list of digit images into a numpy array and reshape it to be a 2d array containing 1d float32 array of the digits
        digit_array = np.array(digits).reshape(-1, 1120).astype(np.float32)
        ret,result,neighbours,dist = knn.findNearest(digit_array,k=3)

        #Convert the result into a list of strings
        result_list = [str(i[0]) for i in list(result.astype(np.int32))]

        # Write house number obtained to file
        with open(folder+'House'+imageNum+'.txt', "w") as houseFile:
            houseFile.write("Building "  + "".join(result_list))
        #print("Building " + "".join(result_list))

def preprocess(image, folder, fileName, imageNum, fileExt):
    original = image.copy()
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    # Bilateral filter code obtained from https://docs.opencv.org/3.4.2/d4/d13/tutorial_py_filtering.html
    # Last accessed on 24/10/2020
    gray = cv.bilateralFilter(gray,9,75,75)

    # Thresholding with Otsu's Binarsization code obtained from https://docs.opencv.org/3.4.2/d7/d4d/tutorial_py_thresholding.html
    # Last accessed on 25/10/2020
    (thresh, binary) = cv.threshold(gray, 128, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
    #cv.imwrite(folder + fileName +"-binary" +fileExt, binary)

    # Contour code obtained from https://docs.opencv.org/3.4.2/d4/d73/tutorial_py_contours_begin.html
    # Last accessed on 25/10/2020
    im2, contours, hierarchy = cv.findContours(binary, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)

    '''
    # For the purpose of displaying all contours detected initially
    for cnt in contours:
        # Bounding rectangle and contour area code obtained from https://docs.opencv.org/3.4.2/dd/d49/tutorial_py_contour_features.html
        # Last accessed on 25/10/2020
        x,y,w,h = cv.boundingRect(cnt)
        cv.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
```

```
    cv.imwrite(folder + fileName + "-contours-unfiltered" +fileExt, image)
    '''


    # Getting the image dimensions, area and max_x and max_y for filtering
    height, width = image.shape[:2]
    image_size = height * width
    max_x = width - 1
    max_y = height - 1

    image = original.copy()
    validContours = list()
    maxArea = 0
    maxIndex = 0
    index = 0
    # Filter out any contour that is too big or too small or doesn't have a height to width ratio of the bounding rectangle that is within the range specified
    # below and any contour that is too near the edge of the image
    for cnt in contours:
        # Bounding rectangle and contour area code obtained from https://docs.opencv.org/3.4.2/dd/d49/tutorial_py_contour_features.html
        # Last accessed on 25/10/2020
        x,y,w,h = cv.boundingRect(cnt)
        rectArea = h * w
        contourArea = cv.contourArea(cnt)
        areaRatio = rectArea / image_size
        # Filter the contours that are too big or too small, and the bounding rectangles are not a vertical rectangle t
        if ((areaRatio > 0.0001) and (rectArea < image_size / 4) and h / w > 1.3 and h / w < 3.5 and x > 40 and y > 20 and x + w < max_x - 40 and y + h < max_y - 20):
            validContours.append(cnt)
            #cv.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
            # Updating the biggest valid contour
            if(rectArea > maxArea):
                maxArea = rectArea
                maxIndex = index
            index += 1
    #cv.imwrite(folder + fileName + "-contours-filter1" +fileExt, image)

    # Find all contours that are at least 35% of the size of the largest contour and
    # its centre y value is near to the largest contour's centre y value with a difference
    # of less than 25% of the height of largest contour
    image =  original.copy()
    filteredList = list()
    biggestCnt = validContours[maxIndex]
    largest_x, largest_y, largest_w, largest_h = cv.boundingRect(biggestCnt)
    center_y_largest = (largest_y + largest_h) / 2
    acceptedOffset = (largest_y + largest_h - center_y_largest) / 2
    for cnt in validContours:
        x,y,w,h = cv.boundingRect(cnt)
        center_y = (y + h) / 2
        rectArea = h * w
        contourArea = cv.contourArea(cnt)
        y_centre_offset = abs(center_y_largest - center_y)
        areaRatio = rectArea / maxArea
        #print("AreaRatio: " + str(areaRatio) + ", Y_Center_Offset: " + str(y_centre_offset))
        if(areaRatio> 0.35 and y_centre_offset < acceptedOffset):
            filteredList.append(cnt)
            #cv.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
    #cv.imwrite(folder + fileName + "-contours-filter2" +fileExt, image)

    # Find all contours that are not within another contour's bounding rectangle
    digitContours = list()
```

```
    approvedIndexList = list()
    for i,cnt1 in enumerate(filteredList):
        x1,y1,w1,h1 = cv.boundingRect(cnt1)
        valid = True
        for j,cnt2 in enumerate(filteredList):
            if( i != j):
                x2,y2,w2,h2 = cv.boundingRect(cnt2)
                # Check if the cnt2's bounding rectangle is within cnt1's bounding rectangle
                if(x2 <= x1 and y2 <= y1 and x2 + w2 >= x1 + w1 and y2 + h2 >= y1 + h1):
                    valid = False
        if valid:
            approvedIndexList.append(i)

    image =  original.copy()
    for i in approvedIndexList:
        cnt = filteredList[i]
        x,y,w,h = cv.boundingRect(cnt)
        digitContours.append(filteredList[i])
        #cv.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
    #cv.imwrite(folder + fileName + "-contours-filter3" +fileExt, image)

    # Find the bounding box coordinates to crop the detected area by finding for the min and
    max values of x and y
    # from the bounding rectangles of all contours in digit contours list
    image =  original.copy()
    firstCnt = digitContours[0]
    first_x, first_y, first_w, first_h = cv.boundingRect(firstCnt)
    min_x = first_x
    max_x = first_x + first_w
    min_y = first_y
    max_y = first_y + first_h
    extractedDigits = list()
    for cnt in digitContours:
        x,y,w,h = cv.boundingRect(cnt)
        if(x < min_x):
            min_x = x
        if(x+w > max_x):
            max_x = x+w
        if(y < min_y):
            min_y = y
        if(y+h > max_y):
            max_y = y+h

    # Calculuate the width and height of the bounding box of the detected area
    crop_w = max_x - min_x
    crop_h = max_y - min_y

    # Crop the detected area from the original image
    cropped = original[min_y:max_y, min_x:max_x]

    # Write the bounding box details of the detected area to txt file as required in assignment
     spec
    with open(folder+'BoundingBox'+ imageNum+'.txt', "w") as boxFile:
        boxFile.write(str(min_x) + ',' + str(min_y) + ',' + str(crop_w) + ',' + str(crop_h))


    #cv.imwrite(folder + fileName + "-contours-final" +fileExt, image)

    # Preprocess the cropped image to perform digit extraction
    gray = cv.cvtColor(cropped, cv.COLOR_BGR2GRAY)
    gray = cv.bilateralFilter(gray,9,75,75)
    (thresh, binary) = cv.threshold(gray, 128, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
    # Find the contours in the cropped(detected area) binary image
```

```
    im2, contours, hierarchy = cv.findContours(binary, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)

    # Get the detected area image's area
    croppedArea = cropped.shape[0] * cropped.shape[1]

    # Filter out the small and inner contours
    contours = remove_small_contours(contours, croppedArea)
    contours = remove_inner_contours(contours)

    # Contour sorting using x coordinate value so that the digit contours are in order from left to right of the image
    #
    # Last accessed on 24/10/2020
    boundingBoxes = [cv.boundingRect(c) for c in contours]
    (contours, boundingBoxes) = zip(*sorted(zip(contours, boundingBoxes),key=lambda b:b[1][0], reverse=False))

    # Loop through the contours to get the individual digits and append to extractedDigits
    for cnt in contours:
        x,y,w,h = cv.boundingRect(cnt)

        #print("Contour Area: " + str(contourArea) + ", Rect Area: " + str(rectArea) + ", X: " + str(x) + ", Y: " + str(y) + ", W: " + str(w) + ", H: " + str(h) + ", Approx: " + str(len(approx)))
        #cv.drawContours(cropped, [cnt], 0, (0,255,0), 3)

        # Extract the digit from binary image
        digit = binary[y:y+h, x:x+w]

        # Adapted code to add black borders to the cropped digit images from https://docs.opencv.org/3.4/dc/da3/tutorial_copyMakeBorder.html
        # Last accessed on 24/10/2020
        padded = cv.copyMakeBorder(digit, 4,4,4,4, cv.BORDER_CONSTANT, None, [0,0,0])

        # Image resizing code obtained and adapted from https://docs.opencv.org/3.4.2/da/d6e/tutorial_py_geometric_transformations.html
        # Last accessed on 25/10/2020
        # Resize the digit to be the same size as the training digits used for training the KNN model
        digit = cv.resize(padded, (28,40), interpolation = cv.INTER_CUBIC)
        extractedDigits.append(digit)
        cv.rectangle(cropped,(x,y),(x+w,y+h),(0,255,0),2)

    cv.imwrite(folder + 'DetectedArea' + imageNum + '.jpg', cropped)
    #cv.imwrite(folder + fileName + "-cropped-contours" +fileExt, cropped)

    return extractedDigits

def remove_small_contours(contours, imageArea):
    approvedContours = list()
    for cnt in contours:
        x,y,w,h = cv.boundingRect(cnt)
        rectArea = h * w
        areaRatio = rectArea / imageArea
        if(areaRatio > 0.02):
            approvedContours.append(cnt)

    return approvedContours

def remove_inner_contours(contours):
    approvedContours = list()
    approvedIndexList = list()
    for i,cnt1 in enumerate(contours):
```

```
        valid = True
        x1,y1,w1,h1 = cv.boundingRect(cnt1)
        for j,cnt2 in enumerate(contours):
            if( i != j):
                x2,y2,w2,h2 = cv.boundingRect(cnt2)
                # Check if the cnt2's bounding rectangle is within cnt1's bounding rectangle
                if (x2 <= x1 and y2 <= y1 and x2 + w2 >= x1 + w1 and y2 + h2 >= y1 + h1):
                    valid = False
        if valid:
            approvedIndexList.append(i)

    for i in approvedIndexList:
        approvedContours.append(contours[i])
    return approvedContours

def main():
    if(len(sys.argv) == 1):
        outputFolder = "/home/student/kay_men_yap_19257442/output/"
        #outputFolder = "output/"
        if not os.path.exists(outputFolder):
            os.mkdir(outputFolder)
        folder = "/home/student/test/"
        #folder = "train_updated/"
        for filename in os.listdir(folder):
            # Sanity check if filename is a file in case I used the /home/student/train directory for
            # training images to test with
            if not os.path.isdir(folder+filename):
                path = Path(filename)
                #print(filename)
                image = cv.imread(folder+filename)
                imageNum = re.findall('[0-9]+$', path.stem)[0]
                extractedDigits = preprocess(image.copy(), outputFolder, path.stem, imageNum, path.suffix)
                knn(extractedDigits, outputFolder, path.stem, imageNum, path.suffix)
    else:
        print("Please run the program with 'python assignment.py'")

if(__name__ == '__main__'):
    main()
```

Training script

```python
import cv2 as cv
import numpy as np
import sys
from pathlib import Path
import os

# KNN training and testing code obtained and adapted from from https://docs.opencv.org/3.4.2/d8/d4b/tutorial_py_knn_opencv.html
# Last accessed on 24/10/2020
def knn(train_data, train_labels, folder):
    test_labels = train_labels.copy()

    knn = cv.ml.KNearest_create()
    knn.train(train_data, cv.ml.ROW_SAMPLE, train_labels)
    # Reuse the training data and labels as the testing data and labels to test accuracy of classification
    ret,result,neighbours,dist = knn.findNearest(train_data,k=3)
    # Check the accuracy of classification by comparing the results and the test labels
    matches = result==test_labels
    correct = np.count_nonzero(matches)
    accuracy = correct*100.0/result.size
    print("K = 3, Accuracy: " +str(accuracy) )
    np.savez(folder+'knn_data.npz',train=train_data, train_labels=train_labels)

def preprocess(image):
    gray= cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    (thresh, binary) = cv.threshold(gray, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
    return binary

def main():
    if(len(sys.argv) == 1):
        outputFolder = "/home/student/kay_men_yap_19257442/models/"
        #outputFolder = "models/"
        if not os.path.exists(outputFolder):
            os.mkdir(outputFolder)
        #folder = "Digits-2020S2/"
        #folder = "train/"
        folder = "/home/student/train/"
        digitList = list()
        labelList = list()
        count = 0
        for subfolder in os.listdir(folder):
            if os.path.isdir(folder+subfolder):
                for filename in os.listdir(folder+subfolder):
                    path = Path(filename)
                    #print(filename)
                    # Get the train label from filename
                    labelList.append(filename[5])
                    image = cv.imread(folder+subfolder+"/"+filename)
                    # Preprocess the image and reshape it into a 1d array
                    image = preprocess(image).reshape(-1,1120).squeeze()
                    digitList.append(image)

        train_array = np.array(digitList).astype(np.float32)
        train_labels = np.array([[i] for i in labelList]).astype(np.int32)
        knn(train_array, train_labels, outputFolder)
    else:
        print("Please run the program with 'python train.py'")

if(__name__ == '__main__'):
    main()
```