|  |
| --- |
|  |
| # DetectChars.py |
|  | import os |
|  |  |
|  | import cv2 |
|  | import numpy as np |
|  | import math |
|  | import random |
|  |  |
|  | import Main |
|  | import Preprocess |
|  | import PossibleChar |
|  |  |
|  | # module level variables ########################################################################## |
|  |  |
|  | kNearest = cv2.ml.KNearest\_create() |
|  |  |
|  | # constants for checkIfPossibleChar, this checks one possible char only (does not compare to another char) |
|  | MIN\_PIXEL\_WIDTH = 2 |
|  | MIN\_PIXEL\_HEIGHT = 8 |
|  |  |
|  | MIN\_ASPECT\_RATIO = 0.25 |
|  | MAX\_ASPECT\_RATIO = 1.0 |
|  |  |
|  | MIN\_PIXEL\_AREA = 80 |
|  |  |
|  | # constants for comparing two chars |
|  | MIN\_DIAG\_SIZE\_MULTIPLE\_AWAY = 0.3 |
|  | MAX\_DIAG\_SIZE\_MULTIPLE\_AWAY = 5.0 |
|  |  |
|  | MAX\_CHANGE\_IN\_AREA = 0.5 |
|  |  |
|  | MAX\_CHANGE\_IN\_WIDTH = 0.8 |
|  | MAX\_CHANGE\_IN\_HEIGHT = 0.2 |
|  |  |
|  | MAX\_ANGLE\_BETWEEN\_CHARS = 12.0 |
|  |  |
|  | # other constants |
|  | MIN\_NUMBER\_OF\_MATCHING\_CHARS = 3 |
|  |  |
|  | RESIZED\_CHAR\_IMAGE\_WIDTH = 20 |
|  | RESIZED\_CHAR\_IMAGE\_HEIGHT = 30 |
|  |  |
|  | MIN\_CONTOUR\_AREA = 100 |
|  |  |
|  | ################################################################################################### |
|  | def loadKNNDataAndTrainKNN(): |
|  | allContoursWithData = [] # declare empty lists, |
|  | validContoursWithData = [] # we will fill these shortly |
|  |  |
|  | try: |
|  | npaClassifications = np.loadtxt("classifications.txt", np.float32) # read in training classifications |
|  | except: # if file could not be opened |
|  | print("error, unable to open classifications.txt, exiting program\n") # show error message |
|  | os.system("pause") |
|  | return False # and return False |
|  | # end try |
|  |  |
|  | try: |
|  | npaFlattenedImages = np.loadtxt("flattened\_images.txt", np.float32) # read in training images |
|  | except: # if file could not be opened |
|  | print("error, unable to open flattened\_images.txt, exiting program\n") # show error message |
|  | os.system("pause") |
|  | return False # and return False |
|  | # end try |
|  |  |
|  | npaClassifications = npaClassifications.reshape((npaClassifications.size, 1)) # reshape numpy array to 1d, necessary to pass to call to train |
|  |  |
|  | kNearest.setDefaultK(1) # set default K to 1 |
|  |  |
|  | kNearest.train(npaFlattenedImages, cv2.ml.ROW\_SAMPLE, npaClassifications) # train KNN object |
|  |  |
|  | return True # if we got here training was successful so return true |
|  | # end function |
|  |  |
|  | ################################################################################################### |
|  | def detectCharsInPlates(listOfPossiblePlates): |
|  | intPlateCounter = 0 |
|  | imgContours = None |
|  | contours = [] |
|  |  |
|  | if len(listOfPossiblePlates) == 0: # if list of possible plates is empty |
|  | return listOfPossiblePlates # return |
|  | # end if |
|  |  |
|  | # at this point we can be sure the list of possible plates has at least one plate |
|  |  |
|  | for possiblePlate in listOfPossiblePlates: # for each possible plate, this is a big for loop that takes up most of the function |
|  |  |
|  | possiblePlate.imgGrayscale, possiblePlate.imgThresh = Preprocess.preprocess(possiblePlate.imgPlate) # preprocess to get grayscale and threshold images |
|  |  |
|  | if Main.showSteps == True: # show steps ################################################### |
|  | cv2.imshow("5a", possiblePlate.imgPlate) |
|  | cv2.imshow("5b", possiblePlate.imgGrayscale) |
|  | cv2.imshow("5c", possiblePlate.imgThresh) |
|  | # end if # show steps ##################################################################### |
|  |  |
|  | # increase size of plate image for easier viewing and char detection |
|  | possiblePlate.imgThresh = cv2.resize(possiblePlate.imgThresh, (0, 0), fx = 1.6, fy = 1.6) |
|  |  |
|  | # threshold again to eliminate any gray areas |
|  | thresholdValue, possiblePlate.imgThresh = cv2.threshold(possiblePlate.imgThresh, 0.0, 255.0, cv2.THRESH\_BINARY | cv2.THRESH\_OTSU) |
|  |  |
|  | if Main.showSteps == True: # show steps ################################################### |
|  | cv2.imshow("5d", possiblePlate.imgThresh) |
|  | # end if # show steps ##################################################################### |
|  |  |
|  | # find all possible chars in the plate, |
|  | # this function first finds all contours, then only includes contours that could be chars (without comparison to other chars yet) |
|  | listOfPossibleCharsInPlate = findPossibleCharsInPlate(possiblePlate.imgGrayscale, possiblePlate.imgThresh) |
|  |  |
|  | if Main.showSteps == True: # show steps ################################################### |
|  | height, width, numChannels = possiblePlate.imgPlate.shape |
|  | imgContours = np.zeros((height, width, 3), np.uint8) |
|  | del contours[:] # clear the contours list |
|  |  |
|  | for possibleChar in listOfPossibleCharsInPlate: |
|  | contours.append(possibleChar.contour) |
|  | # end for |
|  |  |
|  | cv2.drawContours(imgContours, contours, -1, Main.SCALAR\_WHITE) |
|  |  |
|  | cv2.imshow("6", imgContours) |
|  | # end if # show steps ##################################################################### |
|  |  |
|  | # given a list of all possible chars, find groups of matching chars within the plate |
|  | listOfListsOfMatchingCharsInPlate = findListOfListsOfMatchingChars(listOfPossibleCharsInPlate) |
|  |  |
|  | if Main.showSteps == True: # show steps ################################################### |
|  | imgContours = np.zeros((height, width, 3), np.uint8) |
|  | del contours[:] |
|  |  |
|  | for listOfMatchingChars in listOfListsOfMatchingCharsInPlate: |
|  | intRandomBlue = random.randint(0, 255) |
|  | intRandomGreen = random.randint(0, 255) |
|  | intRandomRed = random.randint(0, 255) |
|  |  |
|  | for matchingChar in listOfMatchingChars: |
|  | contours.append(matchingChar.contour) |
|  | # end for |
|  | cv2.drawContours(imgContours, contours, -1, (intRandomBlue, intRandomGreen, intRandomRed)) |
|  | # end for |
|  | cv2.imshow("7", imgContours) |
|  | # end if # show steps ##################################################################### |
|  |  |
|  | if (len(listOfListsOfMatchingCharsInPlate) == 0): # if no groups of matching chars were found in the plate |
|  |  |
|  | if Main.showSteps == True: # show steps ############################################### |
|  | print("chars found in plate number " + str( |
|  | intPlateCounter) + " = (none), click on any image and press a key to continue . . .") |
|  | intPlateCounter = intPlateCounter + 1 |
|  | cv2.destroyWindow("8") |
|  | cv2.destroyWindow("9") |
|  | cv2.destroyWindow("10") |
|  | cv2.waitKey(0) |
|  | # end if # show steps ################################################################# |
|  |  |
|  | possiblePlate.strChars = "" |
|  | continue # go back to top of for loop |
|  | # end if |
|  |  |
|  | for i in range(0, len(listOfListsOfMatchingCharsInPlate)): # within each list of matching chars |
|  | listOfListsOfMatchingCharsInPlate[i].sort(key = lambda matchingChar: matchingChar.intCenterX) # sort chars from left to right |
|  | listOfListsOfMatchingCharsInPlate[i] = removeInnerOverlappingChars(listOfListsOfMatchingCharsInPlate[i]) # and remove inner overlapping chars |
|  | # end for |
|  |  |
|  | if Main.showSteps == True: # show steps ################################################### |
|  | imgContours = np.zeros((height, width, 3), np.uint8) |
|  |  |
|  | for listOfMatchingChars in listOfListsOfMatchingCharsInPlate: |
|  | intRandomBlue = random.randint(0, 255) |
|  | intRandomGreen = random.randint(0, 255) |
|  | intRandomRed = random.randint(0, 255) |
|  |  |
|  | del contours[:] |
|  |  |
|  | for matchingChar in listOfMatchingChars: |
|  | contours.append(matchingChar.contour) |
|  | # end for |
|  |  |
|  | cv2.drawContours(imgContours, contours, -1, (intRandomBlue, intRandomGreen, intRandomRed)) |
|  | # end for |
|  | cv2.imshow("8", imgContours) |
|  | # end if # show steps ##################################################################### |
|  |  |
|  | # within each possible plate, suppose the longest list of potential matching chars is the actual list of chars |
|  | intLenOfLongestListOfChars = 0 |
|  | intIndexOfLongestListOfChars = 0 |
|  |  |
|  | # loop through all the vectors of matching chars, get the index of the one with the most chars |
|  | for i in range(0, len(listOfListsOfMatchingCharsInPlate)): |
|  | if len(listOfListsOfMatchingCharsInPlate[i]) > intLenOfLongestListOfChars: |
|  | intLenOfLongestListOfChars = len(listOfListsOfMatchingCharsInPlate[i]) |
|  | intIndexOfLongestListOfChars = i |
|  | # end if |
|  | # end for |
|  |  |
|  | # suppose that the longest list of matching chars within the plate is the actual list of chars |
|  | longestListOfMatchingCharsInPlate = listOfListsOfMatchingCharsInPlate[intIndexOfLongestListOfChars] |
|  |  |
|  | if Main.showSteps == True: # show steps ################################################### |
|  | imgContours = np.zeros((height, width, 3), np.uint8) |
|  | del contours[:] |
|  |  |
|  | for matchingChar in longestListOfMatchingCharsInPlate: |
|  | contours.append(matchingChar.contour) |
|  | # end for |
|  |  |
|  | cv2.drawContours(imgContours, contours, -1, Main.SCALAR\_WHITE) |
|  |  |
|  | cv2.imshow("9", imgContours) |
|  | # end if # show steps ##################################################################### |
|  |  |
|  | possiblePlate.strChars = recognizeCharsInPlate(possiblePlate.imgThresh, longestListOfMatchingCharsInPlate) |
|  |  |
|  | if Main.showSteps == True: # show steps ################################################### |
|  | print("chars found in plate number " + str( |
|  | intPlateCounter) + " = " + possiblePlate.strChars + ", click on any image and press a key to continue . . .") |
|  | intPlateCounter = intPlateCounter + 1 |
|  | cv2.waitKey(0) |
|  | # end if # show steps ##################################################################### |
|  |  |
|  | # end of big for loop that takes up most of the function |
|  |  |
|  | if Main.showSteps == True: |
|  | print("\nchar detection complete, click on any image and press a key to continue . . .\n") |
|  | cv2.waitKey(0) |
|  | # end if |
|  |  |
|  | return listOfPossiblePlates |
|  | # end function |
|  |  |
|  | ################################################################################################### |
|  | def findPossibleCharsInPlate(imgGrayscale, imgThresh): |
|  | listOfPossibleChars = [] # this will be the return value |
|  | contours = [] |
|  | imgThreshCopy = imgThresh.copy() |
|  |  |
|  | # find all contours in plate |
|  | imgContours, contours, npaHierarchy = cv2.findContours(imgThreshCopy, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE) |
|  |  |
|  | for contour in contours: # for each contour |
|  | possibleChar = PossibleChar.PossibleChar(contour) |
|  |  |
|  | if checkIfPossibleChar(possibleChar): # if contour is a possible char, note this does not compare to other chars (yet) . . . |
|  | listOfPossibleChars.append(possibleChar) # add to list of possible chars |
|  | # end if |
|  | # end if |
|  |  |
|  | return listOfPossibleChars |
|  | # end function |
|  |  |
|  | ################################################################################################### |
|  | def checkIfPossibleChar(possibleChar): |
|  | # this function is a 'first pass' that does a rough check on a contour to see if it could be a char, |
|  | # note that we are not (yet) comparing the char to other chars to look for a group |
|  | if (possibleChar.intBoundingRectArea > MIN\_PIXEL\_AREA and |
|  | possibleChar.intBoundingRectWidth > MIN\_PIXEL\_WIDTH and possibleChar.intBoundingRectHeight > MIN\_PIXEL\_HEIGHT and |
|  | MIN\_ASPECT\_RATIO < possibleChar.fltAspectRatio and possibleChar.fltAspectRatio < MAX\_ASPECT\_RATIO): |
|  | return True |
|  | else: |
|  | return False |
|  | # end if |
|  | # end function |
|  |  |
|  | ################################################################################################### |
|  | def findListOfListsOfMatchingChars(listOfPossibleChars): |
|  | # with this function, we start off with all the possible chars in one big list |
|  | # the purpose of this function is to re-arrange the one big list of chars into a list of lists of matching chars, |
|  | # note that chars that are not found to be in a group of matches do not need to be considered further |
|  | listOfListsOfMatchingChars = [] # this will be the return value |
|  |  |
|  | for possibleChar in listOfPossibleChars: # for each possible char in the one big list of chars |
|  | listOfMatchingChars = findListOfMatchingChars(possibleChar, listOfPossibleChars) # find all chars in the big list that match the current char |
|  |  |
|  | listOfMatchingChars.append(possibleChar) # also add the current char to current possible list of matching chars |
|  |  |
|  | if len(listOfMatchingChars) < MIN\_NUMBER\_OF\_MATCHING\_CHARS: # if current possible list of matching chars is not long enough to constitute a possible plate |
|  | continue # jump back to the top of the for loop and try again with next char, note that it's not necessary |
|  | # to save the list in any way since it did not have enough chars to be a possible plate |
|  | # end if |
|  |  |
|  | # if we get here, the current list passed test as a "group" or "cluster" of matching chars |
|  | listOfListsOfMatchingChars.append(listOfMatchingChars) # so add to our list of lists of matching chars |
|  |  |
|  | listOfPossibleCharsWithCurrentMatchesRemoved = [] |
|  |  |
|  | # remove the current list of matching chars from the big list so we don't use those same chars twice, |
|  | # make sure to make a new big list for this since we don't want to change the original big list |
|  | listOfPossibleCharsWithCurrentMatchesRemoved = list(set(listOfPossibleChars) - set(listOfMatchingChars)) |
|  |  |
|  | recursiveListOfListsOfMatchingChars = findListOfListsOfMatchingChars(listOfPossibleCharsWithCurrentMatchesRemoved) # recursive call |
|  |  |
|  | for recursiveListOfMatchingChars in recursiveListOfListsOfMatchingChars: # for each list of matching chars found by recursive call |
|  | listOfListsOfMatchingChars.append(recursiveListOfMatchingChars) # add to our original list of lists of matching chars |
|  | # end for |
|  |  |
|  | break # exit for |
|  |  |
|  | # end for |
|  |  |
|  | return listOfListsOfMatchingChars |
|  | # end function |
|  |  |
|  | ################################################################################################### |
|  | def findListOfMatchingChars(possibleChar, listOfChars): |
|  | # the purpose of this function is, given a possible char and a big list of possible chars, |
|  | # find all chars in the big list that are a match for the single possible char, and return those matching chars as a list |
|  | listOfMatchingChars = [] # this will be the return value |
|  |  |
|  | for possibleMatchingChar in listOfChars: # for each char in big list |
|  | if possibleMatchingChar == possibleChar: # if the char we attempting to find matches for is the exact same char as the char in the big list we are currently checking |
|  | # then we should not include it in the list of matches b/c that would end up double including the current char |
|  | continue # so do not add to list of matches and jump back to top of for loop |
|  | # end if |
|  | # compute stuff to see if chars are a match |
|  | fltDistanceBetweenChars = distanceBetweenChars(possibleChar, possibleMatchingChar) |
|  |  |
|  | fltAngleBetweenChars = angleBetweenChars(possibleChar, possibleMatchingChar) |
|  |  |
|  | fltChangeInArea = float(abs(possibleMatchingChar.intBoundingRectArea - possibleChar.intBoundingRectArea)) / float(possibleChar.intBoundingRectArea) |
|  |  |
|  | fltChangeInWidth = float(abs(possibleMatchingChar.intBoundingRectWidth - possibleChar.intBoundingRectWidth)) / float(possibleChar.intBoundingRectWidth) |
|  | fltChangeInHeight = float(abs(possibleMatchingChar.intBoundingRectHeight - possibleChar.intBoundingRectHeight)) / float(possibleChar.intBoundingRectHeight) |
|  |  |
|  | # check if chars match |
|  | if (fltDistanceBetweenChars < (possibleChar.fltDiagonalSize \* MAX\_DIAG\_SIZE\_MULTIPLE\_AWAY) and |
|  | fltAngleBetweenChars < MAX\_ANGLE\_BETWEEN\_CHARS and |
|  | fltChangeInArea < MAX\_CHANGE\_IN\_AREA and |
|  | fltChangeInWidth < MAX\_CHANGE\_IN\_WIDTH and |
|  | fltChangeInHeight < MAX\_CHANGE\_IN\_HEIGHT): |
|  |  |
|  | listOfMatchingChars.append(possibleMatchingChar) # if the chars are a match, add the current char to list of matching chars |
|  | # end if |
|  | # end for |
|  |  |
|  | return listOfMatchingChars # return result |
|  | # end function |
|  |  |
|  | ################################################################################################### |
|  | # use Pythagorean theorem to calculate distance between two chars |
|  | def distanceBetweenChars(firstChar, secondChar): |
|  | intX = abs(firstChar.intCenterX - secondChar.intCenterX) |
|  | intY = abs(firstChar.intCenterY - secondChar.intCenterY) |
|  |  |
|  | return math.sqrt((intX \*\* 2) + (intY \*\* 2)) |
|  | # end function |
|  |  |
|  | ################################################################################################### |
|  | # use basic trigonometry (SOH CAH TOA) to calculate angle between chars |
|  | def angleBetweenChars(firstChar, secondChar): |
|  | fltAdj = float(abs(firstChar.intCenterX - secondChar.intCenterX)) |
|  | fltOpp = float(abs(firstChar.intCenterY - secondChar.intCenterY)) |
|  |  |
|  | if fltAdj != 0.0: # check to make sure we do not divide by zero if the center X positions are equal, float division by zero will cause a crash in Python |
|  | fltAngleInRad = math.atan(fltOpp / fltAdj) # if adjacent is not zero, calculate angle |
|  | else: |
|  | fltAngleInRad = 1.5708 # if adjacent is zero, use this as the angle, this is to be consistent with the C++ version of this program |
|  | # end if |
|  |  |
|  | fltAngleInDeg = fltAngleInRad \* (180.0 / math.pi) # calculate angle in degrees |
|  |  |
|  | return fltAngleInDeg |
|  | # end function |
|  |  |
|  | ################################################################################################### |
|  | # if we have two chars overlapping or to close to each other to possibly be separate chars, remove the inner (smaller) char, |
|  | # this is to prevent including the same char twice if two contours are found for the same char, |
|  | # for example for the letter 'O' both the inner ring and the outer ring may be found as contours, but we should only include the char once |
|  | def removeInnerOverlappingChars(listOfMatchingChars): |
|  | listOfMatchingCharsWithInnerCharRemoved = list(listOfMatchingChars) # this will be the return value |
|  |  |
|  | for currentChar in listOfMatchingChars: |
|  | for otherChar in listOfMatchingChars: |
|  | if currentChar != otherChar: # if current char and other char are not the same char . . . |
|  | # if current char and other char have center points at almost the same location . . . |
|  | if distanceBetweenChars(currentChar, otherChar) < (currentChar.fltDiagonalSize \* MIN\_DIAG\_SIZE\_MULTIPLE\_AWAY): |
|  | # if we get in here we have found overlapping chars |
|  | # next we identify which char is smaller, then if that char was not already removed on a previous pass, remove it |
|  | if currentChar.intBoundingRectArea < otherChar.intBoundingRectArea: # if current char is smaller than other char |
|  | if currentChar in listOfMatchingCharsWithInnerCharRemoved: # if current char was not already removed on a previous pass . . . |
|  | listOfMatchingCharsWithInnerCharRemoved.remove(currentChar) # then remove current char |
|  | # end if |
|  | else: # else if other char is smaller than current char |
|  | if otherChar in listOfMatchingCharsWithInnerCharRemoved: # if other char was not already removed on a previous pass . . . |
|  | listOfMatchingCharsWithInnerCharRemoved.remove(otherChar) # then remove other char |
|  | # end if |
|  | # end if |
|  | # end if |
|  | # end if |
|  | # end for |
|  | # end for |
|  |  |
|  | return listOfMatchingCharsWithInnerCharRemoved |
|  | # end function |
|  |  |
|  | ################################################################################################### |
|  | # this is where we apply the actual char recognition |
|  | def recognizeCharsInPlate(imgThresh, listOfMatchingChars): |
|  | strChars = "" # this will be the return value, the chars in the lic plate |
|  |  |
|  | height, width = imgThresh.shape |
|  |  |
|  | imgThreshColor = np.zeros((height, width, 3), np.uint8) |
|  |  |
|  | listOfMatchingChars.sort(key = lambda matchingChar: matchingChar.intCenterX) # sort chars from left to right |
|  |  |
|  | cv2.cvtColor(imgThresh, cv2.COLOR\_GRAY2BGR, imgThreshColor) # make color version of threshold image so we can draw contours in color on it |
|  |  |
|  | for currentChar in listOfMatchingChars: # for each char in plate |
|  | pt1 = (currentChar.intBoundingRectX, currentChar.intBoundingRectY) |
|  | pt2 = ((currentChar.intBoundingRectX + currentChar.intBoundingRectWidth), (currentChar.intBoundingRectY + currentChar.intBoundingRectHeight)) |
|  |  |
|  | cv2.rectangle(imgThreshColor, pt1, pt2, Main.SCALAR\_GREEN, 2) # draw green box around the char |
|  |  |
|  | # crop char out of threshold image |
|  | imgROI = imgThresh[currentChar.intBoundingRectY : currentChar.intBoundingRectY + currentChar.intBoundingRectHeight, |
|  | currentChar.intBoundingRectX : currentChar.intBoundingRectX + currentChar.intBoundingRectWidth] |
|  |  |
|  | imgROIResized = cv2.resize(imgROI, (RESIZED\_CHAR\_IMAGE\_WIDTH, RESIZED\_CHAR\_IMAGE\_HEIGHT)) # resize image, this is necessary for char recognition |
|  |  |
|  | npaROIResized = imgROIResized.reshape((1, RESIZED\_CHAR\_IMAGE\_WIDTH \* RESIZED\_CHAR\_IMAGE\_HEIGHT)) # flatten image into 1d numpy array |
|  |  |
|  | npaROIResized = np.float32(npaROIResized) # convert from 1d numpy array of ints to 1d numpy array of floats |
|  |  |
|  | retval, npaResults, neigh\_resp, dists = kNearest.findNearest(npaROIResized, k = 1) # finally we can call findNearest !!! |
|  |  |
|  | strCurrentChar = str(chr(int(npaResults[0][0]))) # get character from results |
|  |  |
|  | strChars = strChars + strCurrentChar # append current char to full string |
|  |  |
|  | # end for |
|  |  |
|  | if Main.showSteps == True: # show steps ####################################################### |
|  | cv2.imshow("10", imgThreshColor) |
|  | # end if # show steps ######################################################################### |
|  |  |
|  | return strChars |
|  | # end function |
|  |  |