Gépi látás

GKNB_INTM038

Feleletválasztós kérdőív kiértékelése

-Fejlesztési dokumentáció-

Vincze Petra

Z8OPOU

Győr, 2020/2021/1

Tartalomjegyzék

1 Feladat leírása

Féléves projektemben egy olyan programot készítettem, mely képes leellenőrizni, hogy egy kérdőív formailag helyesen lett-e kitöltve.

A program bemenetként kap egy képet, melyen kérdések és a hozzájuk kapcsolódó válaszlehetőségek szerepelnek. Minden válasz mellett megtalálható egy kör vagy egy négyzet alakzat, ezek jelzik, hogy az adott kérdéshez egy vagy egynél több helyes megoldás tartozik-e. Ahol kör szerepel, ott csak egy lehetőség van, ahol négyzet, ott több is elfogadható. A program feladata, hogy tájékoztassa a felhasználót arról, hogy az előbb leírt feltételeknek megfelelően töltötte-e ki a kérdőívet, illetve ha nem, akkor mely kérdések esetén hibázott.

2 Projekt elkészítése

2.1 Első rész – a kép előkészítése

A programot a kerdoiv.py fájlba, Python programozási nyelven és az OpenCV használatával készítettem el. Első lépésként importáltam a szükséges Python csomagokat, mint az imutils, numpy, argparse, cv2.

Az argparse lehetővé teszi, hogy parancssorból be lehessen olvasni a kiértékelendő képet, ehhez a --image kapcsolót kell használni, mely a bemeneti kép elérési útját tartalmazza. A betöltött képet szürkeárnyalatossá kell konvertálni és csökkenteni kell a magas frekvenciájú zajokat, majd a Canny él detektáló és a cv2.findContours függvény segítségével meg kell találni a teszt körvonalait. Ez abban az esetben hasznos, ha a kitöltött papírt valamilyen eszköz segítségével fotózzuk le, azaz nem csak a lap látszódik a képen. E nélkül később nem fogjuk tudni alkalmazni a four-point transzformációt. Feltételezzük, hogy a teszt lapja helyezkedik a betöltött kép fókuszpontjában, így az rendelkezik a legnagyobb területtel. Ha a megtalált kontúrokat csökkenő sorrendbe rendezzük, akkor a lista elején kell elhelyezkednie a lapot jelölő körvonalnak is. Ha emellé még a kontúrok közelítésével megvizsgáljuk, hogy azok hány csúccsal rendelkeznek, akkor az első, 4 csúccsal rendelkező kontúr fog a tesztlaphoz tartozni.

```
38  # konturok megkeresese az elterkepen
39  #a dokumnetum konturjainak inicializalasa
40  cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
41      cv2.CHAIN_APPROX_SIMPLE)
42  cnts = imutils.grab_contours(cnts)
43  docCnt = None
44
45  #ellenorzes, hogy min egy kontur van
46  if len(cnts) > 0:
47      #konturok csokkeno sorrendbe rendezese
48      cnts = sorted(cnts, key=cv2.contourArea, reverse=True)
49
50      # vegigmegyunk a rendezett konturokon
51      for c in cnts:
52          #konturok kozelitese
53          peri = cv2.arcLength(c, True)
54          approx = cv2.approxPolyDP(c, 0.02 * peri, True)
55
56          #ha 4 pontot talalunk, akkor beazonositottuk a papirt
57          if len(approx) == 4:
58              docCnt = approx
59              break
```

Ezután a felhasználó segítségére van szükség, ugyanis a four-point transzformációt csak abban az esetben lehet végrehajtani, ha a betöltött kép nem scannelt, vagy online, azaz nincs eredetileg is felülnézetben. A felhasználónak az 1-es gombot kell megnyomnia, ha szükség van a transzformációra és a 0-s billentyűt, ha nincs.

```
#szukseg van-e four point perspective transzformaciora
kepe = input("Ha a kerdoiv valamilyen eszkozzel lett lefotozva, nyomja meg az 1-es

#ha lefotozott papir a kep
if kepe == '1':
    #alkalmazzunk four point perspective transzformaciot a felulnezeti kepert
    #az erdetei kepen
    paper = four_point_transform(image, docCnt.reshape(4, 2))
    #a szurkearnyalatos kepen
    warped = four_point_transform(gray, docCnt.reshape(4, 2))

    #Otsu metodussal alakitsuk at a kepet binaris keppe
    #elkulonul a hatter es a kerdesek, valaszok
    thresh = cv2.threshold(warped, 0, 255,
        cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]

elif kepe== '0':
    #Otsu metodussal alakitsuk at a kepet binaris keppe
    #elkulonul a hatter es a kerdesek, valaszok
    thresh = cv2.threshold(gray, 0, 255,
    cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]

else:
    print ('Hibas gomb')
```

A kép feldolgozásának utolsó lépéseként vagy a transzformált vagy a szürkeárnyalatos képet bináris képpé kell alakítani Otsu metódussal, hogy elkülönüljenek a kérdések és a válaszok a háttértől.

2.2 Második rész – a kérdőív kiértékelése

Ahhoz, hogy a teszt kitöltésének helyességét meg tudjuk vizsgálni, szükség van néhány változóra. Kell egy változó, ami azt mutatja meg, hogy hányadik kérdésnél tartunk, kezdőértéke 1 (’question’), egy, ami kérdésenként a jelölt válaszok számát tartalmazza és 0-ról indul (’answers’), illetve egy logikai változóra is szükség van, amit akkor állítunk át igazról hamisra, ha az adott kérdésnél nem megfelelő számú válasz van jelölve (’helyes’). Valamint egy ’s’ nevű változó segítségével fogjuk megvizsgálni, hogy milyen alakzat található az adott kérdés mellett.

Az ’s’ egy Shape osztályt jelöl, amit az alakzat.py fájlban hoztunk létre. Ehhez az osztályhoz tartozik egy detect függvény, melynek, ha megadunk különböző kontúrokat, a korábban is használt közelítő módszerrel meg tudja róluk állapítani, hogy hány csúcsuk van, ezáltal pedig, hogy körről vagy négyzetről van-e szó.

```python
# szukseges csomagok importalasa
import cv2


class Shape:
    def __init__(self):
        pass
    def detect(self, c):

        # initialize the shape name and approximate the contour
        shape = "unidentified"
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.04 * peri, True)

        if len(approx) == 4:
            shape = "negyzet"
        else:
            shape = "kor"

        return shape
```

A program a bináris képen újra megkeresi a kontúrokat, majd ezek közül a képarány segítségével kiválasztja azokat, amelyek a válaszlehetőségekhez tartozó alakzatokat jelölik és a ’questionCnts’ nevű listához adja őket.

A továbbhaladáshoz ismét a felhasználó segítségére van szükség, ugyanis meg kell adni, hogy hány válasz tartozik egy-egy kérdéshez. A program csak azokat a kérdőíveket képes megvizsgálni, ahol minden kérdéshez azonos számú válasz tartozik.

Ezután egy for ciklusban újra megvizsgáljuk kontúrokat. Léptéknek azt a számot állítjuk be, amit a felhasználó megadott. A cikluson belül létrehozunk egy ’negyzet’ nevű logikai, illetve egy ’hanyadik_valasz’ nevű változót, mely kezdőértéke 1 lesz. A cikluson belül elindítunk még egy ciklust, amiben először azt vizsgáljuk meg, hogy a hanyadik_valasz változó milyen értékkel rendelkezik. Ha 1-gyel, akkor meghívódik az ’s’ változó detect függvénye és eldől, hogy az adott kérdésnél négyzet vagy kör szerepel-e a válaszok mellett. Mivel egy kérdéshez csak egy fajta válaszlehetőség tartozik, ezért elég a vizsgálatot csak minden kérdés első válaszánál megtenni.

```
#vegigmegyunk a tombon, akkora leptekkel amekkorat a felhasznalo megadott
for (q, i) in enumerate(np.arange(0, len(questionCnts), answersCount)): #
    #kerdesek rendezese balrol jobbra
    cnts = contours.sort_contours(questionCnts[i:i + answersCount])[0]  #

    #logikai valtozo inicializalasa a valaszokhoz tartozo alakzat megalla
    negyzet = False
    #valtozo inicializalasa ahhoz, hogy tudjuk, a kerdeshez tartozo elso
    hanyadik_valasz = 1

    #vegigmegyunk a konturokon
    for (j, c) in enumerate(cnts):

        #megnezzuk, hogy az alakzat negyzet-e vagy sem
        #eleg csak minden kerdesnel az elso alakzatot megvizsgalni
        while hanyadik_valasz == 1:

            alakzat = s.detect(c)
            #print (alakzat)
            if alakzat == "negyzet":
                negyzet = True

            hanyadik_valasz += 1
```

Utána készítünk egy maszkot, ami az éppen aktuális válaszlehetőséget mutatja és ezt alkalmazzuk a bináris képen. Majd összeszámoljuk a nem 0 pixeleket és az értéket eltároljuk a 'total' változóban. Ha a 'total' értéke egy bizonyos értéknél nagyobb, vagy azzal egyenlő, akkor a felhasználó megjelölte a választ a képen és az 'answers' értékét növeljük eggyel.

```
#keszitsunk maszkot, ami az eppen aktualis valaszlehetoseget mutatja
mask = np.zeros(thresh.shape, dtype="uint8")
cv2.drawContours(mask, [c], -1, 255, -1)

#alkalmazzuk a maszkot a binaris kepen
#szamoljuk ossze a nem-nulla pixeleket a teruleten
mask = cv2.bitwise_and(thresh, thresh, mask=mask)
total = cv2.countNonZero(mask)

#adjunk meg egy erteket ami felett valoszinuleg jelolve van a valaszlehetoseg
#print (total)
if total >= 290:
    answers += 1
```

A ciklus végén megvizsgáljuk, hogy az alakzatnak megfelelő számú válasz volt-e jelölve és ha nem, akkor erről visszajelzést küldünk a felhasználónak. Végül a 'question' értékét eggyel növeljük, az 'answers'-nek pedig újra 0 értéket adunk.

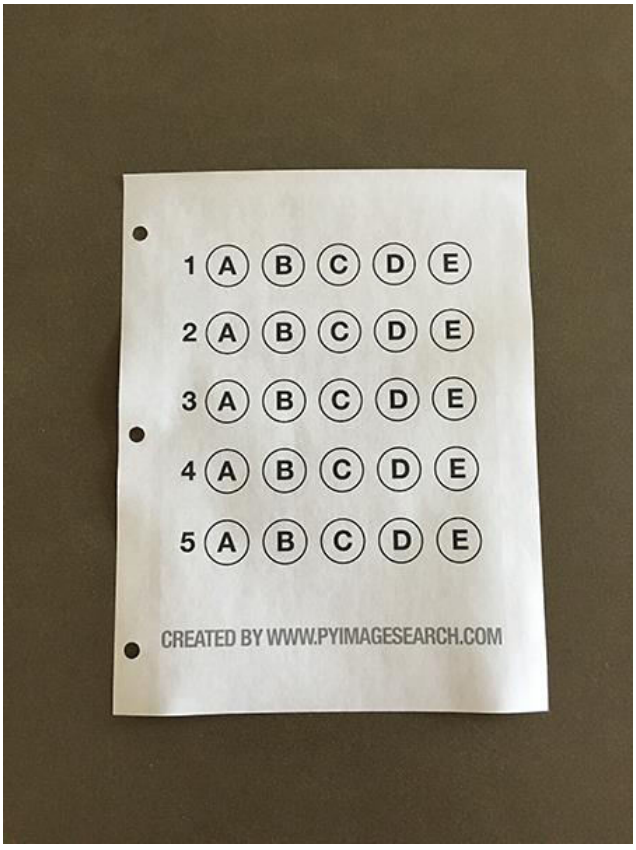
```
Ha a kerdoiv valamilyen eszkozzel lett lefotozva, nyomja meg az 1-es billentyut, ha szkennelve van, nyomja meg a 0-as bi
llentyut: 0
Egy kerdeshez tartozo valaszok szama: 5
A 2. kerdes hibasan lett kitoltve
```

```
Ha a kerdoiv valamilyen eszkozzel lett lefotozva, nyomja meg az 1-es billentyut, ha szkennelve van, nyomja meg a 0-as bi
llentyut: 0
Egy kerdeshez tartozo valaszok szama: 5
A kerdoiv helyesen lett kitoltve!
```

3 Tesztelés

A programot különböző képek segítségével teszteltem, ezek összesen 4 csoportba oszthatóak. Teszteltem számítógépen készített képekkel, illetve egy internetről letöltött fotóval is, hogy a four-point transzformáció helyes működését is meg tudjam vizsgálni. A tesztek során különböző színeket és különböző jelölési módokat is alkalmaztam, hogy megvizsgáljam, ezek hogyan befolyásolják a program működését. Ha túl halvány színnel jelöltem a válaszokat, vagy túl vékony tollal, akkor előfordultak hibák a javítás során, de összességében a program megfelelően működik. Kb. 25-30 képpel teszteltem a programot.

A teszteléshez használt eredeti képek egy része, melyeket képszerkesztővel különböző módokon kitöltöttem:



1. Kérdés

☐ válasz

☐ válasz

☐ válasz

☐ válasz

☐ válasz

2. Kérdés

○ válasz

○ válasz

○ válasz

○ válasz

○ válasz

1. Kérdés

☐ válasz

☐ válasz

☐ válasz

2. Kérdés

○ válasz

○ válasz

○ válasz

3. Kérdés

☐ válasz

☐ válasz

☐ válasz

4. Kérdés

○ válasz

○ válasz

○ válasz

1. Kérdés

☐ válasz

☐ válasz

☐ válasz

2. Kérdés

○ válasz

○ válasz

○ válasz

3. Kérdés

○ válasz

○ válasz

○ válasz

4 Források

[1] https://www.pyimagesearch.com/2016/10/03/bubble-sheet-multiple-choice-scanner-and-test-grader-using-omr-python-and-opencv/
[2] https://www.pyimagesearch.com/2016/02/08/opencv-shape-detection/