Gépi látás

Féléves Beadandó

GKNB_INTM038

Tornyossy László – SL9W96

Budapest, 2022. 12. 19.

Tornyossy László
SL9W96

Gépi látás

Diagram szkennelése

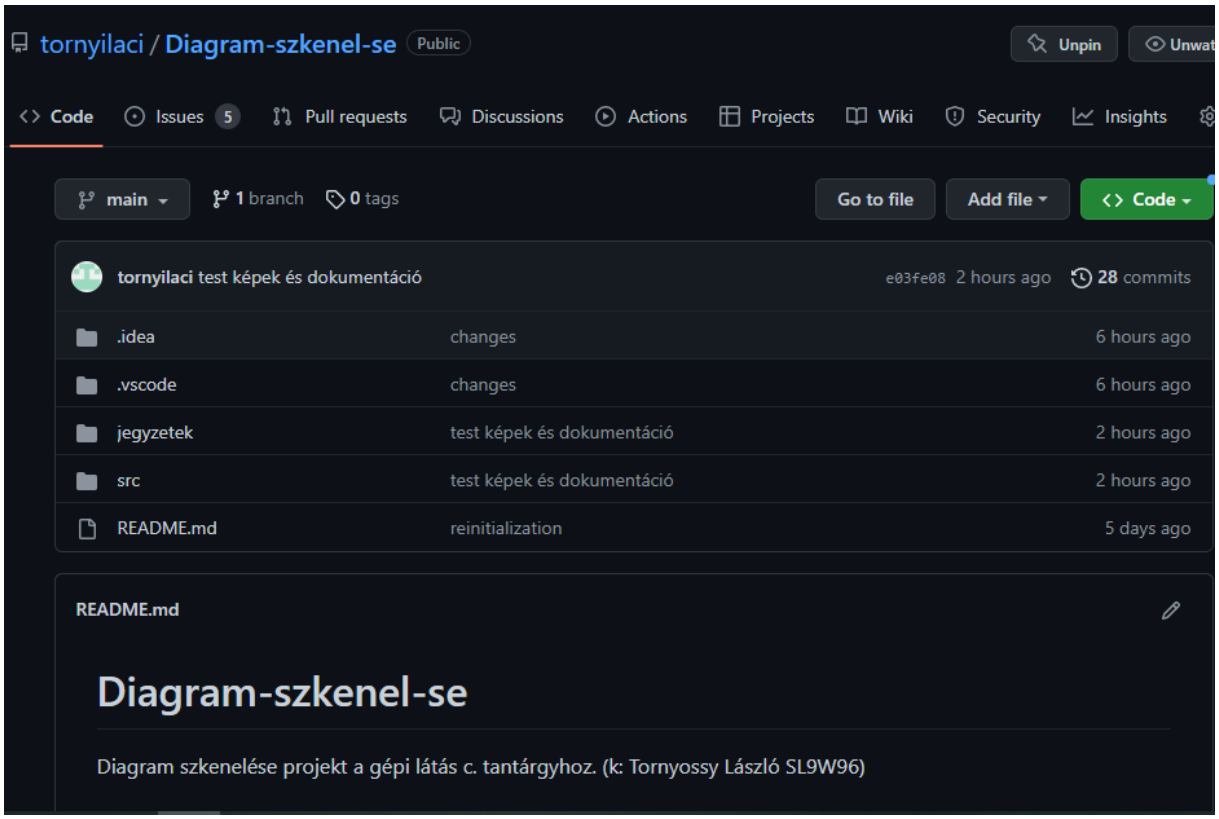
Tartalom

Projekt leírása

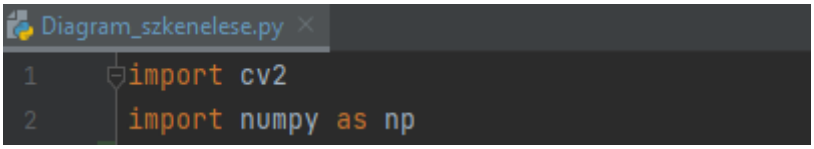
Különböző összetevőkből álló diagram (körök, nyilak, téglalapok…) felismerése, szöveges tartalom nélkül. A program lényege, hogy megállapítsa egy folyamatábráról, hogy az ténylegesen megfelel-e a feltételeknek ahhoz, hogy diagramnak nevezhessük. A projekt githubon keresztül elérhető és az alábbi programok felhasználásával jött létre: pyCharm, draw.io.

Felhasználói dokumentáció

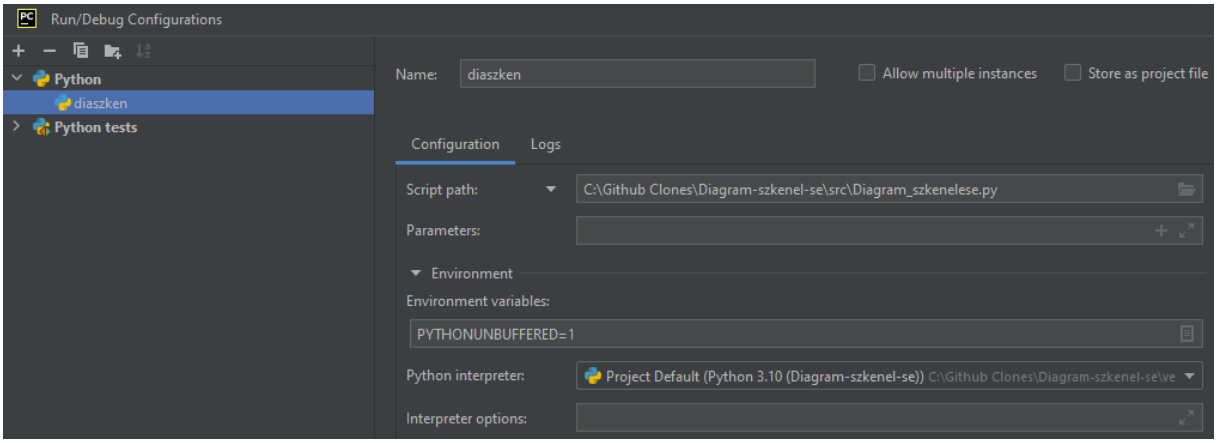
A projektet elérni a https://github.com/tornyilaci/Diagram-szkenel-se linken lehet elérni, ahol a src mappában érhető el a forrásfájl, ami a programot tartalmazza. A kódot szerkeszteni a repository klónozásával, vagy a kódrészletek kimásolásával lehetséges. A jegyzetek mappában a projekthez felhasznált feljegyzések láthatóak. A src mappában a forráskód, továbbá a teszteléshez szükséges képek találhatóak. A Documentation mappában található az itt olvasható leírás.



A szerkesztést és a futtatást a PyCharm IDE-ben javaslom, amiben előzetesen szükséges a következő packages-eket telepíteni: numpy és opencv-python.



Ahhoz, hogy a kód megfelelően fusson szükséges a python nyelv interpreterét is beállítani, melyre a konfigurációk beállításánál van lehetőség.



Ezen lépések után már futtatható is a program. Módosításokat a következő helyeken javaslom beállítani, a teljes felhasználói élmény biztosításaként.

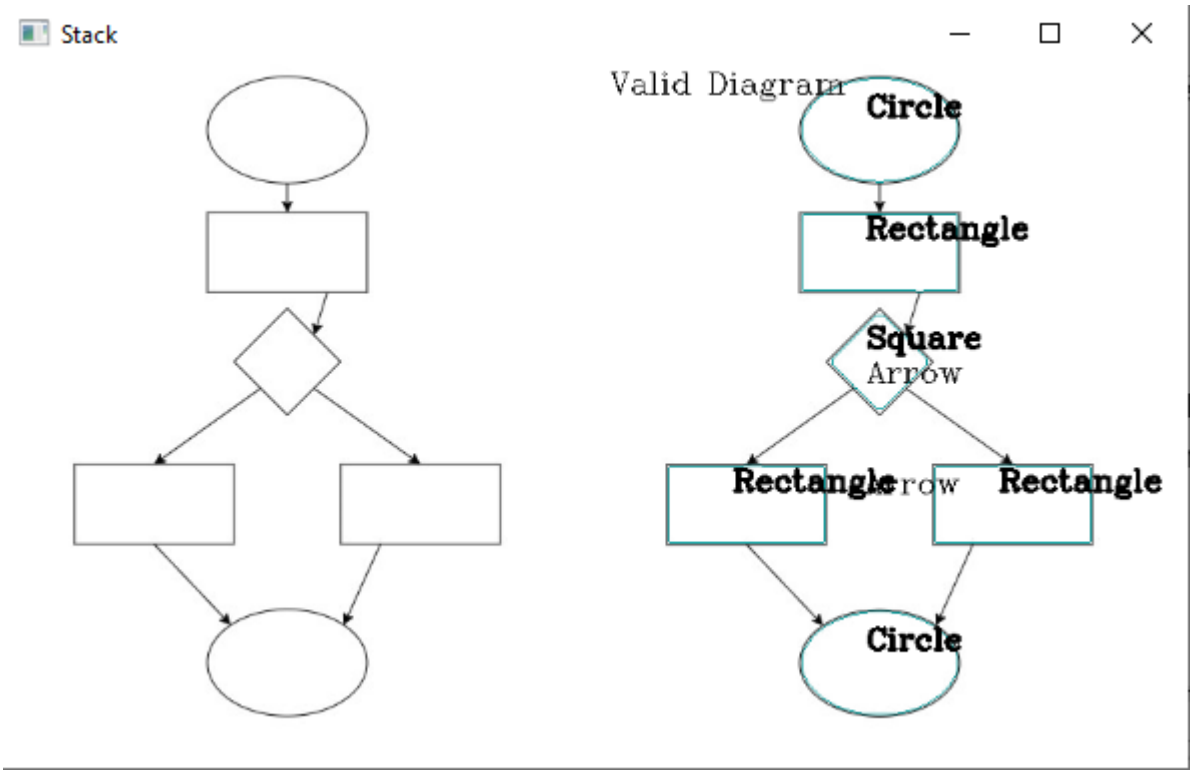
A programban a 88. sorban lehet kiválasztani, hogy melyik képet szeretnénk szerkeszteni, a példán a 6. tesztkép látható:

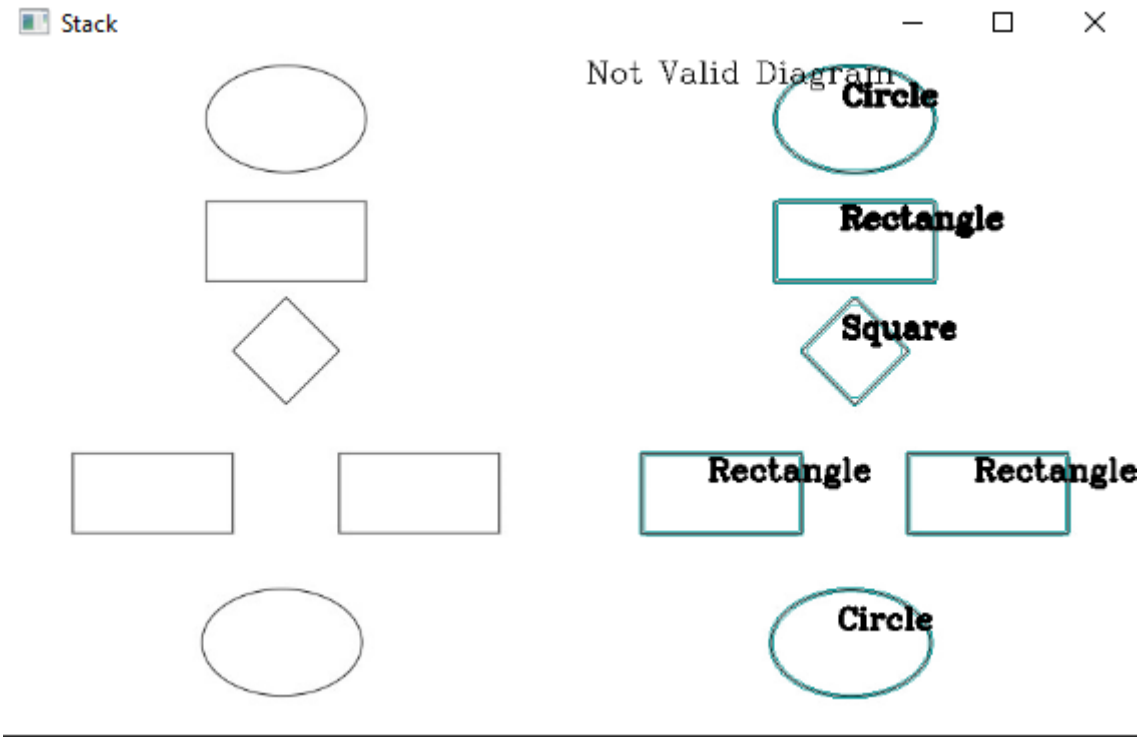
```
87
88  path = 'images/006_test.jpg'
89  img = cv2.imread(path)
90  imgContour = img.copy()
```

A felhasználói élményhez hozzátartozik a megfelelő méretű kép láthatósága. Ezt a 98. sorban változtathatjuk meg a stackImages függvény első paraméterének beállításával:

```
97  imgBlank = np.zeros_like(img)
98  imgStack = stackImages(0.7, ([img, imgContour]))
```

A program futtatása után láthatjuk az eredményt, mely egy az opencv könyvtár imshow függvény segítségével megjelenített párbeszédpanelben jeleníti meg az eredeti és a feldolgozott képünket.





Fejlesztői dokumentáció

A fejlesztés során leginkább a különböző formák felismerése volt a cél. Ehhez az opencv-python könyvtár által nyújtott beépített

függvényeket felhasználva jött létre a projekt. Ehhez a képek előzetes feldolgozására volt szükség melyben felhasználásra kerültek az alábbi algoritmusok:

```
def preprocess(img):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_blur = cv2.GaussianBlur(img_gray, (11, 11), 1)
    img_canny = cv2.Canny(img_blur, 200, 0)
    kernel = np.ones((7, 7))
    img_dilate = cv2.dilate(img_canny, kernel, iterations=1)
    img_erode = cv2.erode(img_dilate, kernel, iterations=1)
    return img_erode
```

GaussianBlur (Gauss-simítás)

A legáltalánosabb megoldási lehetőség a **konvolúció** használata, ami a képpontok egy környezetében elvégzett súlyozott átlag számítását jelenti. A súlyokat egy **k * k** méretű mátrixban adjuk meg. A mátrix rendszerint páratlan számú sorból és oszlopból áll, hogy a középső elem egyértelműen meghatározható legyen. Ez a középső elem illeszkedik az éppen vizsgált képpontra. **Nagyobb *k* értékek esetén erősebb a simítás hatása, mert nagyobb területen számolódik az átlag és a végrehajtási idő is jelentősen megnő!**

Original image

Gaussian Blur filter applied

forrás: https://help.apple.com/assets/616F5622BA

Simítás esetén elvárás, hogy kép összfényessége ne változzon. Ezt úgy érhetjük el, ha a maszkelemek összege 1. Tört számok helyett a könnyebb átláthatóság miatt sokszor egész számokkal adjuk meg a maszk elemeit, a szükséges normalizáló osztás kiemeljük a mátrix elé. Ez csak jelölésmódbeli különbség.

A konvolúciós mátrixot Numpy tömbként definiálhatjuk, például a sima átlagoló szűrő esetén:

```
kernel = np.array(
[[1, 1, 1],
[1, 1, 1],
[1, 1, 1]])

kernel = kernel / 9.0
```

Canny (éldetektálás)

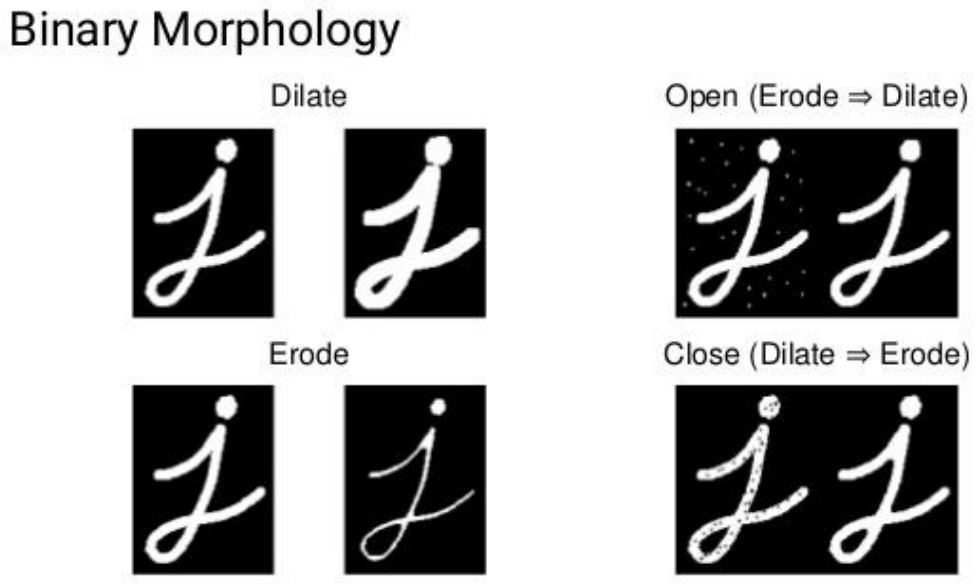
A Canny éldetektáló algoritmus folyamata öt különböző lépésre bontható:

forrás: https://miro.medium.com/max/566/

1. Gauss-szűrő használata a kép simításához a zaj eltávolítása érdekében.
2. A kép intenzitás gradienseinek megtalálása.
3. A gradiens nagyságának küszöbértékének vagy az alsó határ határérték-elnyomás használata, a hamis detektálásoktól való megkülönböztetésre
4. Potenciális élek meghatározására kettős küszöb használata.
5. Élek nyomon követése hiszterézissel: Végezze el az élek észlelését az összes többi gyenge és erős élekhez nem kapcsolódó él elnyomásával.

Dilate

Egyszerűbb képfeldolgozási metódus, amiben a képet elnyújtjuk, elhomályosítjuk. Ezen metódus során a kép világosabb pontjait kibővítjük és a sötétebbeket összeszűkítjük.



Erode

Az erózió során a sötétebb foltok kitágulnak, a világosak összeszűkülnek, így az élek az összeolvadó életek könnyebben észrevehetővé válnak.

forrás: https://files.ai-pool.com/d/a03f8fb7e828

getContours függvény leírása

A getContours függvényben történik meg annak az ellenőrzése, hogy a beolvasott képünk megfelel-e a követelményeknek, hogy diagramnak nevezhessük. Az előfeldolgozott képen lévő élek felhasználásával megkeressük, hogy ezek az élek körbezárnak-e valamilyen területet, s ha igen, akkor ezt a területet fogjuk megvizsgálni, hogy valamilyen formáról van-e szó. Az approx és hull változókban nézzük meg, hogy ezek a formák milyen körvonalakkal rendelkeznek, esetleg, ha konvex szögeik vannak, akkor ezeken kívül lévő tartományt is figyelembe kell-e vennünk. Amennyiben ezek megvannak, felderítjük, hogy ezek a körvonalak milyen egyedi vonalakból (oldalakból) állnak össze.

```
def getContours(img):
    contours, hierarchy = cv2.findContours(img, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    foundObject = set()
    circleCount = 0
    for cnt in contours:
        area = cv2.contourArea(cnt)
        if 250 < area < 10000:
            cv2.drawContours(imgContour, cnt, -1, (155, 155, 0), 1)
            peri = cv2.arcLength(cnt, True)
            approx = cv2.approxPolyDP(cnt, 0.02 * peri, True)
            hull = cv2.convexHull(approx, returnPoints=False)
            sides = len(hull)
            objCor = len(approx)
            x, y, w, h = cv2.boundingRect(approx)
            objectType = "None"
            if objCor == 4:
                aspRatio = w / float(h)
                if 0.98 < aspRatio < 1.03:
                    objectType = "Square"
                else:
                    objectType = "Rectangle"
```

Ezeket a vonalakat megvizsgálva tudjuk eldönteni, hogy az adott alakzat téglalap, négyzet [abban az esetben, hogyha a téglalap oldalainak aránya közelítenek az 1:1 arányhoz], vagy kör.

```
            else:
                objectType = "Rectangle"
        elif objCor > 7:
            objectType = "Circle"
            circleCount = circleCount + 1
        foundObject.add(objectType)
        cv2.putText(imgContour, objectType, (x + (w // 2) - 10, y + (h // 2) - 10),
                    cv2.FONT_HERSHEY_COMPLEX, 0.7, (0, 0, 0), 2)
    else:
        peri = cv2.arcLength(cnt, True)
        approx = cv2.approxPolyDP(cnt, 0.02 * peri, True)
        x, y, w, h = cv2.boundingRect(approx)
        cv2.putText(imgContour, "Arrow", (x + (w // 2) - 10, y + (h // 2) - 10), cv2.FONT_HERSHEY_COMPLEX, 0.7,
                    (0, 0, 0), 1)
        foundObject.add("Arrow")
requiredObjects = {"Arrow", "Circle", "Rectangle"}
if (requiredObjects.issubset(foundObject)) and (circleCount > 1):
    cv2.putText(imgContour, "Valid Diagram", (10, 20), cv2.FONT_HERSHEY_COMPLEX, 0.7,
                (0, 0, 0), 1)
else:
    cv2.putText(imgContour, "Not Valid Diagram", (10, 20), cv2.FONT_HERSHEY_COMPLEX, 0.7,
                (0, 0, 0), 1)
```

Mivel mindenképpen diagramhoz hasonló képeket tesztelünk, ezért feltételezzük, hogy a fennmaradó alakzat valamilyen nyíl lesz, így akként is tekintünk rá.

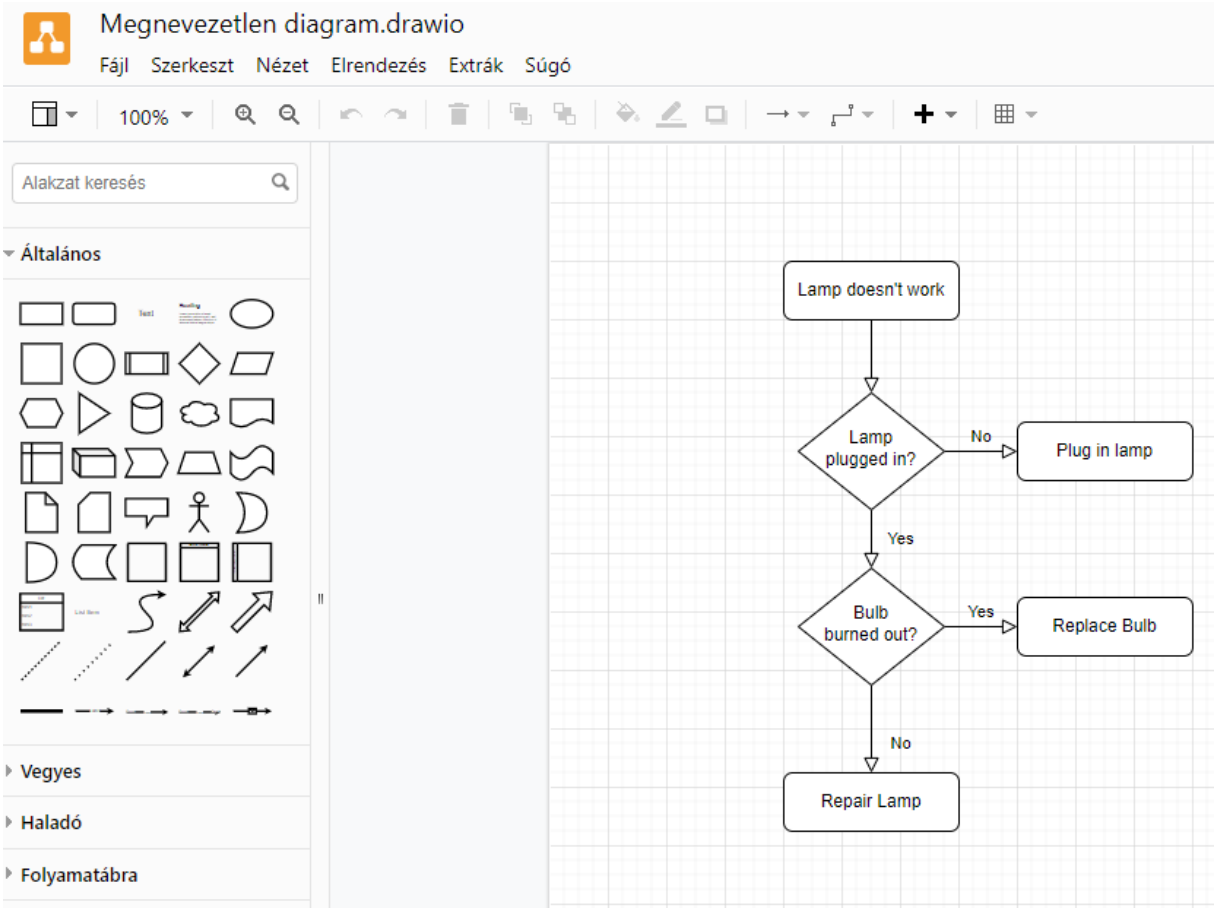
Ahhoz, hogy teljes biztonsággal megmondhassuk egy folyamatábráról, hogy az valid (érvényes, megfelelő), ahhoz először is mindenképpen kel a képen szerepelnie körnek, téglalapnak és nyílnak. Vagyis, ha ezek közül bármelyik nem teljesül, akkor máris érvénytelen diagramról beszélünk.

Még egy követelmény lett felállítva ennek ellenőrzéséhez, mégpedig, hogy egy folyamatábra kezdetét és végét egy kör vagy ovális jelzi, ezalapján mindenképpen kettő kör alakzatnak kell a diagramon szerepelnie.

Tesztek

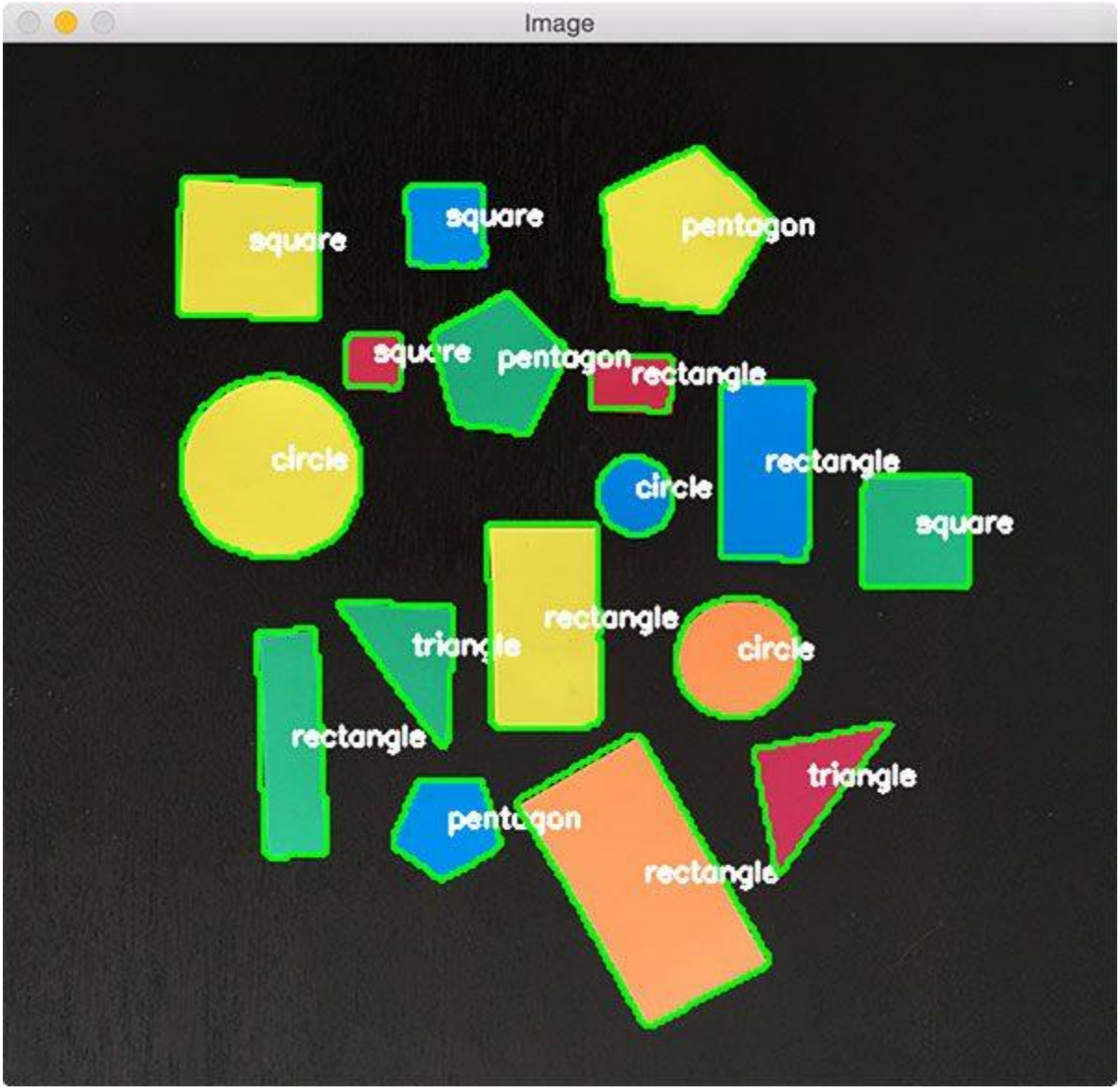
A tesztek során próbáltam a program megfelelő működését vizsgálni azáltal, hogy különböző alakzatokat olvastattam be a programmal.

Ezeket az alakzatokat draw.io segítségével hoztam létre.



A Projekt folyamata

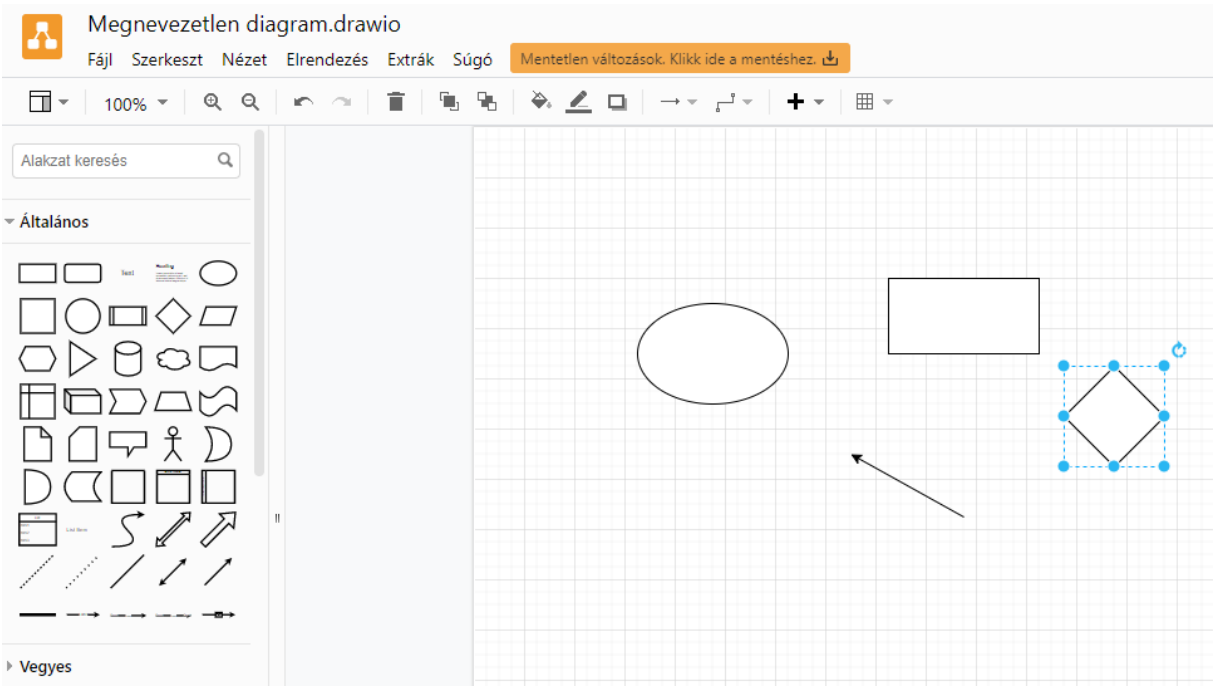
A projekt felépítéséhez szükséges volt felvázolni, hogy mire lesz szükség. Először is mindenképpen fel kellett ismerni különböző formákat, de ahhoz, hogy bármit is fel tudjon ismerni a program, ahhoz szükség volt képfeldolgozó algoritmusokra, melyeknek leírása feljebb szerepel. Miután sikerült feldolgozni a képet, hozzá lehetett fogni a kívánt formák felismeréséhez. Ehhez a folyamathoz a detektált élek számát egy alakzatban használtam fel:



forrás: https://pyimagesearch.com/2016/0

Miután sikerült így formákat felismerni, hiányoztak még a nyilak, de mivel mindenképpen csak olyan képeket tesztel a program, amikben potenciálisan egy diagramhoz szükséges összetevők szerepelnek, így ezen feltételezés szerint a fennmaradó alakzatoknak mindenképpen nyilaknak kellett lenniük, így ezek külön nem lettek ellenőrizve.

Ezután a képek elkészítése következett, ami a megfelelő komponensek felhasználásával történt a fentnevezett draw.io programmal. Itt különböző mennyiségben rajzoltam meg a program segítségével a diagrammokat, olyakor a nyilakat, a köröket vagy a téglalapokat teljesen kihagyva, így ellenőrizve, hogy a program megfelelően működik-e.



A tesztelések futtatásával és jelen dokumentáció lejegyzésével ért véget a projekt.

Bővítési lehetőségek

Bővíteni, vagy esetleg a projekten van lehetőség. Esetleg a jövőben hozzá lehetne venni a szkennelés folyamatához, a szövegfelismerést is, így akár egy teljes folyamatábrát értelmezni tudunk. Ennek a szöveggel kibővített diagramnak pedig akár a mesterséges intelligencia segítségével a leírását is megadhatjuk.

Források

https://medium.com/analytics-vidhya/contours-and-convex-hull-in-opencv-python-d7503f6651bc

https://www.aut.bme.hu/Task/22-23-osz/Kepfeldolgozas-OpenCV-alapokon

https://www.ms.sapientia.ro/~vajdat/education/imageprocessing/KepfeldolgozasJegyzet.pdf

https://www.inf.u-szeged.hu/~tanacs/pyocv/canny_ldetektor.html