“NEM FUNKCIONÁLIS RÉSZ”

Albert Ferencz

Mi is ez a „Python”?

„Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance.”

https://www.python.org/

OpenCV

„OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.”

https://opencv.org/

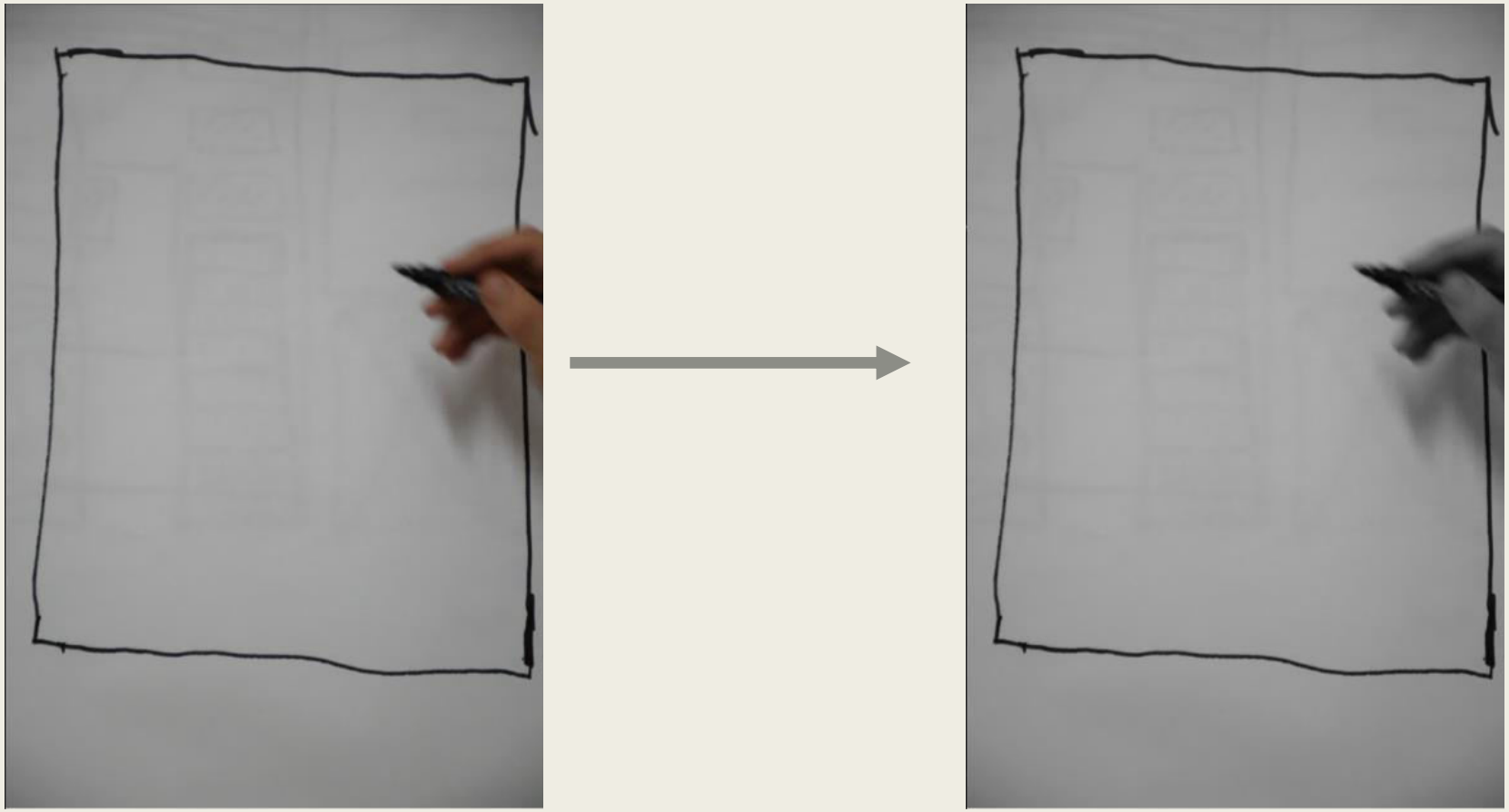
Iteráció

1. Képkocka kiolvasása a videóból / módosítások
2. A komponensek meghatározása (*Pozició, Méret)*
3. A komponensek rangsorának meghatározása
4. A komponensek tipusának meghatározása
5. Klasszifikáció

Képkocka kiolvasása a videóbol / módosítások

cv2.cvtColor

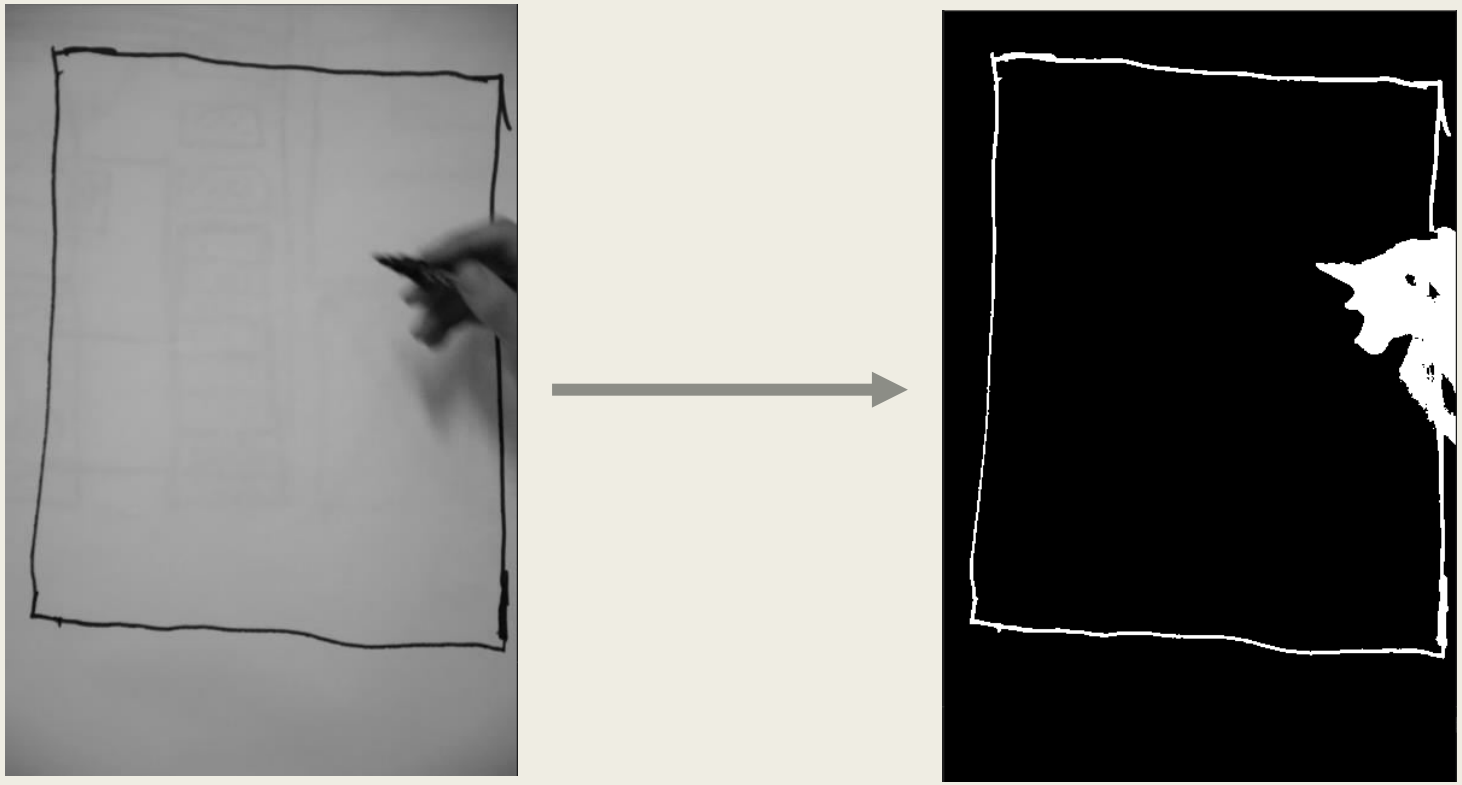
Függvény, mely egy bemeneti kép színterét átkonvertálja egy másikra



Képkocka kiolvasása a videóbol / módosítások

cv2.threshold

Függvény, mely egy adott paraméternél kisebb értékeket egy minimum értékre -; nagyobb érékeket egy maximum értékre - állít



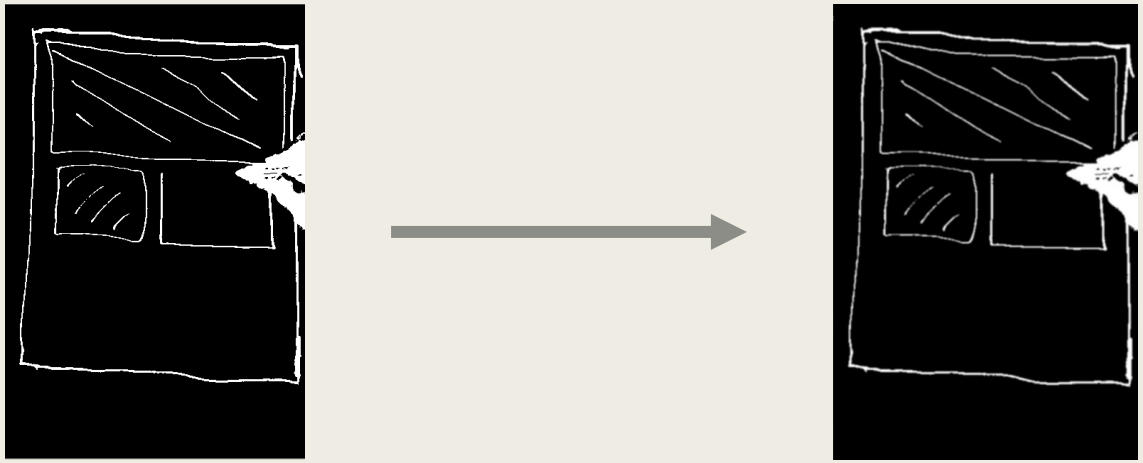
Képkocka kiolvasása a videóbol / módosítások

cv2.Canny

Edge detection algoritmus

Több szintű algoritmus.

1. Lépés *(Zajcsökkentés)*
   - *Mivel az edge detection nagyon függ attól, hogy milyen zajos a kép, átvisszük a képet 5x5 alapú Gaussian Blurron*



Képkocka kiolvasása a videóbol / módosítások

cv2.Canny

Edge detection algoritmus

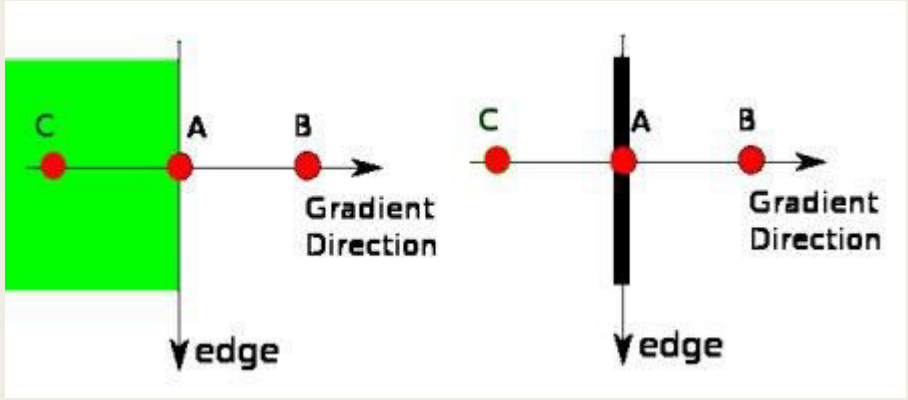
2. Intensity Gradient keresése
   - *A képet átvisszük egy Sobel kernelen, mely meghatározza az elsőrendű deriváltjait a képnek, függőleges (**Gy**) illetve vízszintes (**Gx**) irányokban*
   - *Ezek segítségével meg tudjuk határozni az él gradiensek intenzitását illetve ezek irányait.*

Képkocka kiolvasása a videóbol / módosítások

cv2.Canny

Edge detection algoritmus

3. Nem maximum kizárás
   - *Az él gradiens intenzitásának illetve irányának kifejezése után, bejárjuk a képet, és minden olyan pixelt kizárunk, amely nem lokális maximum a gradiens irányában. Röviden, egy olyan képet kapunk, mely vékony vonalakkal az edgeket ábrázolja.*

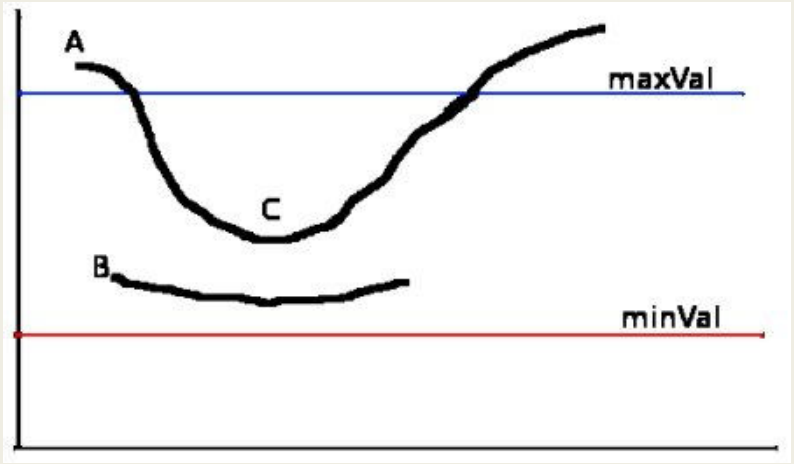


Képkocka kiolvasása a videóbol / módosítások

cv2.Canny

Edge detection algoritmus

4. Hysteresis Thresholding
   - *Ez a lépés választja szét az edgeket, a nem edgektől.*
     - Egy edge akkor biztosan edge, ha a gradiens intenzitása nagyobb mint egy paraméterben megadott érték, vagy össze van kötve egy már biztosan edge pixe-el.
     - Egy edge akkor nem edge pixel, ha a gradiens intenzitása kisebb mint egy paraméterben megadott érték. Ugyanakkor nem edge pixel, ha a gradiens intenzitása a két paraméterben megatott érték közt van, és nincs összekötve egy biztosan edge pixel-el.

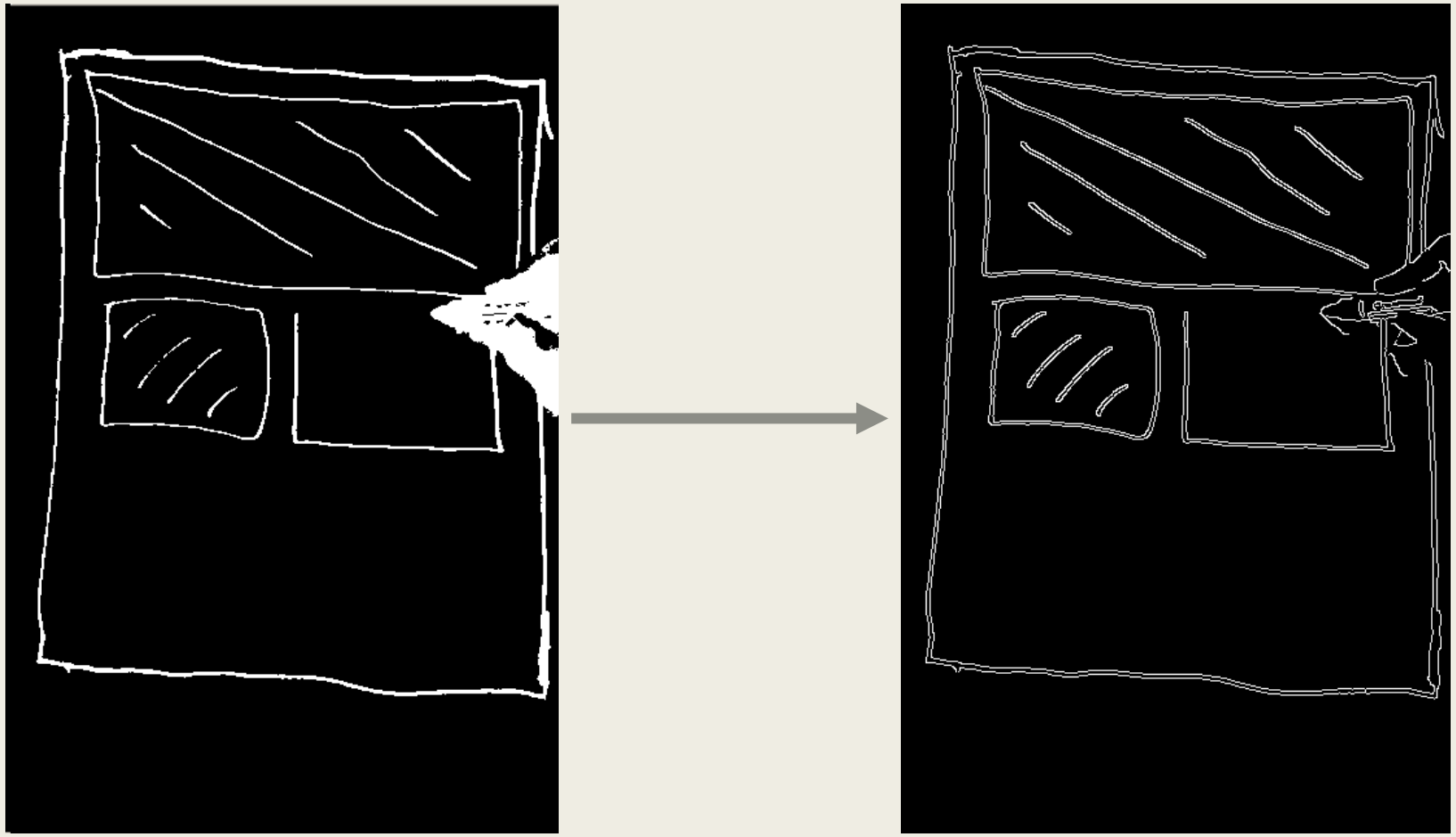


Képkocka kiolvasása a videóbol / módosítások

cv2.Canny

Edge detection algoritmus

5. Példa:



Bináris

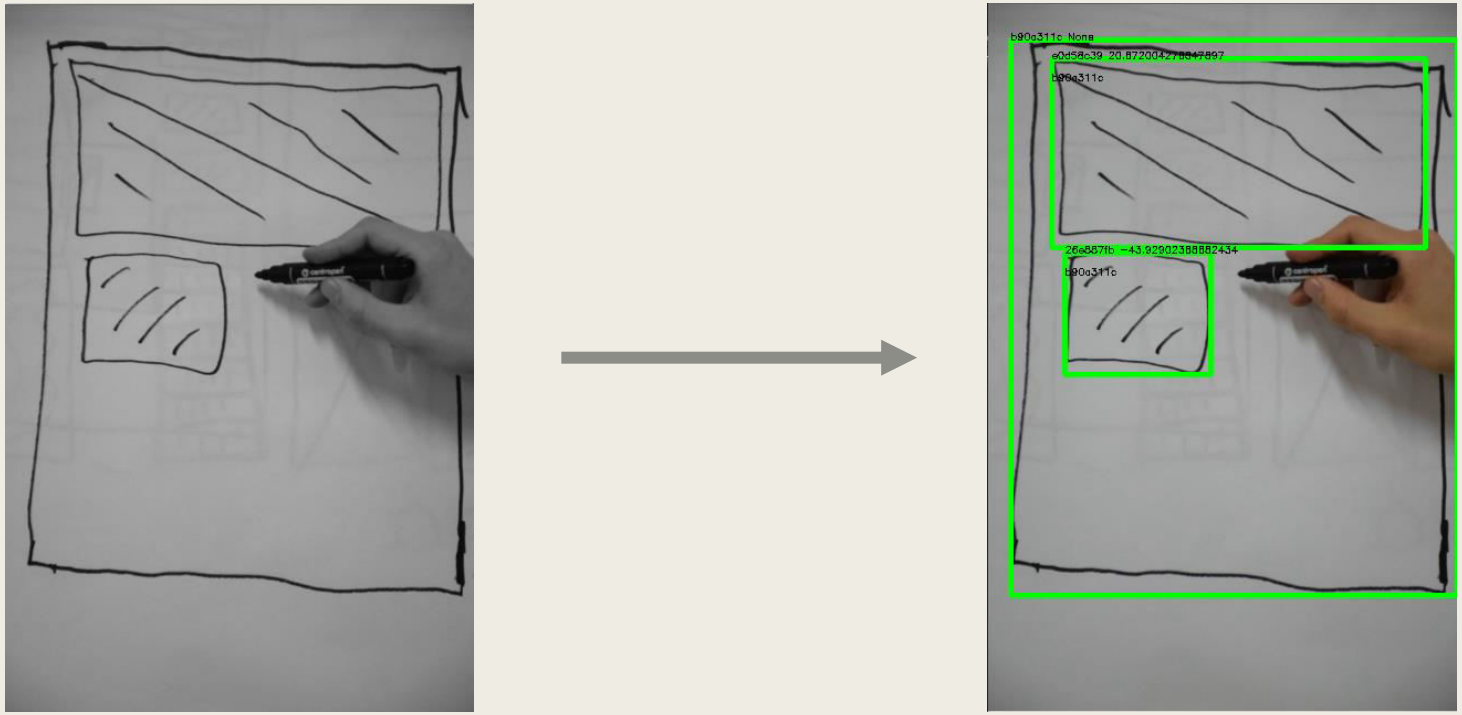
Canny után

A komponensek meghatározása (*Pozició, Méret*)

cv2.findContours

Kontűr kereső algoritmus

1. Kontűr
   - *Olyan görbék, melyek ugyan olyan intenzitású/színű pontokat kötnek össze*
   - *findContour ezeket keresi meg, a Green elmélet, a kép elsőrendű és másodrendű momentumainak segítségével*



A komponensek meghatározása (*Pozició, Méret)*

1. cv2.convexHull
   - *Megvizsgálja egy kontűrnek a konvexitását, illetve kiegészíti ha szükséges.*
2. cv2.contourArea
   - *Kiszámítja egy kontűrnek a területét*
3. cv2.arcLength
   - *Kiszámítja egz kontűrnek a kerületét*
4. Circularity
   - *Megmondja, hogy egy kontűr mennyire „kerek”*

A komponensek meghatározása (*Pozició, Méret)*

```
for contour in contours:
    hull = cv2.convexHull(contour)
    area = cv2.contourArea(hull)
    perimeter = cv2.arcLength(hull, True)
    try:
        circularity = (4 * pi * area) / (perimeter * perimeter)
        box = cv2.boundingRect(contour)
        if area > 1500 and area < width * height:
            if circularity < 0.9 and circularity > 0.65:
                B.append(Component(*box, area, None))
    except ZeroDivisionError as e:
        pass
```

A komponensek rangsorának meghatározása

```
def generate_hierarchy(A):
    for a in A:
         a.setparent(None)
    for a in A:
        for b in A:
            if not a.equals(b):
                if a.contains(b):
                    if b.parent == None:
                        b.parent = a
                    elif b.parent != None:
                        if b.parent.getarea() > a.getarea():
                            b.parent = a
```

A komponensek tipusának meghatározása

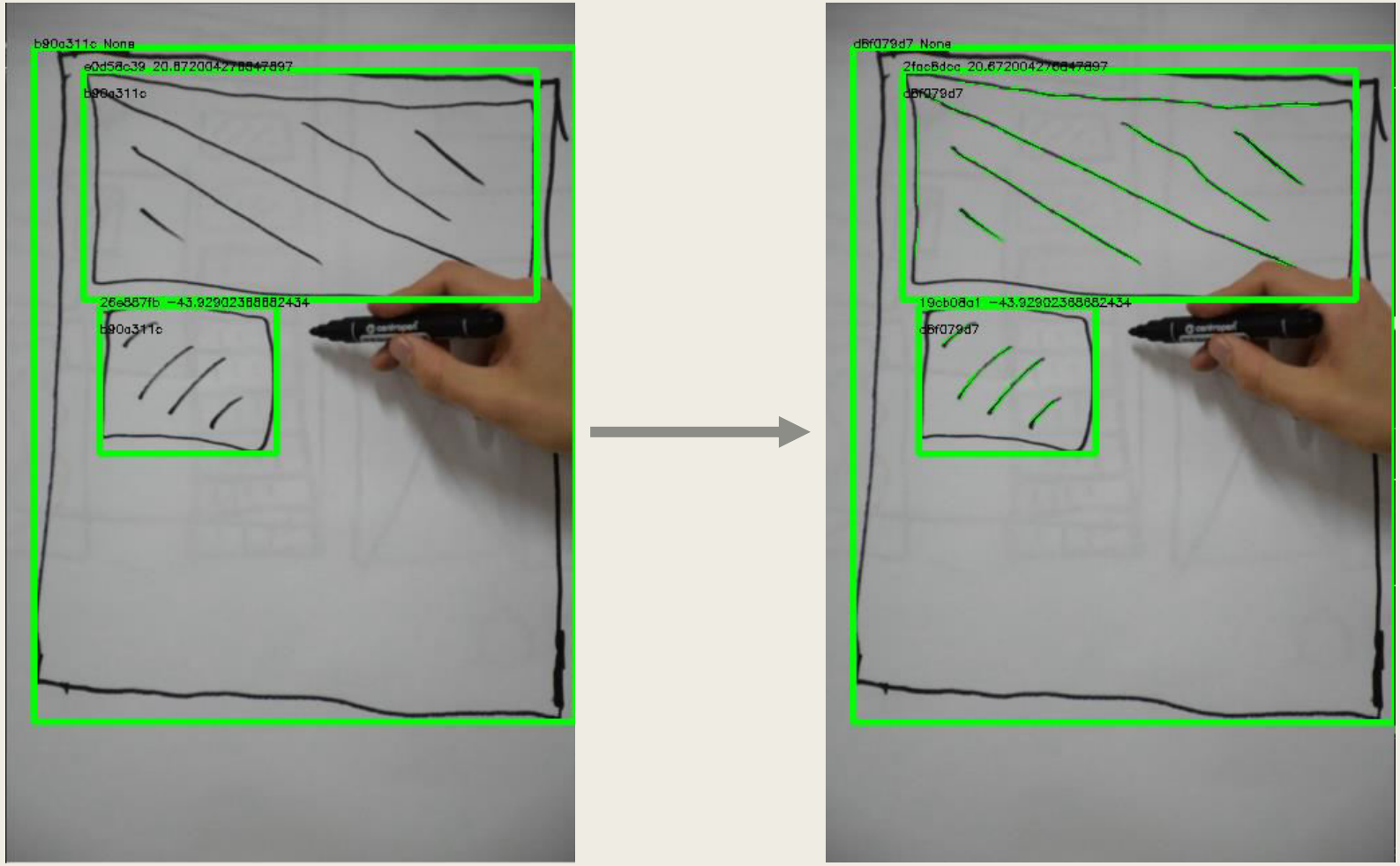
cv2.HoughLinesP
Vonal felismerő

- A hough transzformáció segítségével megkeressük azokat a pontokat, melyeknek a legnagyobb szavazata van arra, hogy vonal van köztük
- Ezeket a vonalakat a *HoughLinesP* egy listában téríti vissza mely tartalmazza a kezdőpontokat illetve a végpontokat
- Ezek segítségével meg tudjuk határozni a vonalak iránytényezőjét, majd a vonalak szögét
- Legvégül várhatóértéket számolunk a vonalak szögeiből

A komponensek tipusának meghatározása

cv2.HoughLinesP
Vonal felismerő

- Példa:

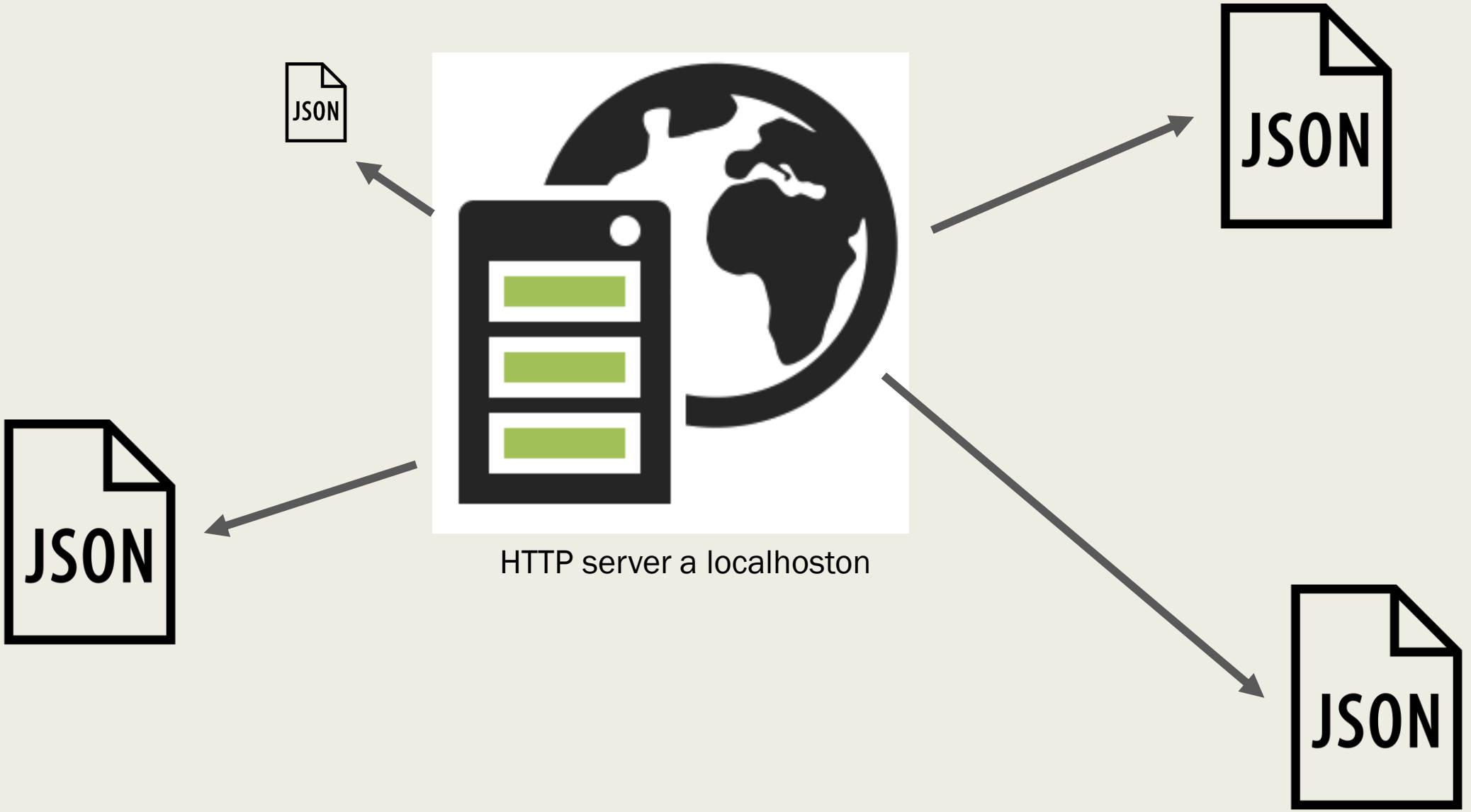


Klasszifikáció

```
# decide what's new, what's old, what to keep
while self.A and B:
    distances = [[a, b, Component.distance(a, b)] for a, b in
itertools.product(self.A, B)]
    a, b, error = min(distances, key=lambda o: o[2])
    if error > 0.02:
        break
    b.setid(a.getid())
    b.angle = a.angle

    C.append(b)
    self.A.remove(a)
    B.remove(b)
```

Komunikáció



Köszönöm a figyelmet!

- https://docs.opencv.org/4.0.1/
- https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html
- https://en.wikipedia.org/wiki/Green%27s_theorem