Gépi Látás
(GKNB_INTM038)
beadandó feladat
QR- és vonalkód olvasás

Készítette:
Doma Dániel (EN32U7)

2021/22-1

Tartalomjegyzék

1. Bevezetés, megoldandó feladat kifejtése

Beadandó feladatom témájának a QR- és Vonalkód olvasást választottam, mert mindenképpen szerettem volna valamilyen számomra új témával megismerkedni. A feladat megoldására egy olyan program elkészítését képzeltem el, amely indításakor a felhasználó megadja a kód típusát és a képe(ke)t, amelyeket fel szeretne ismertetni a programmal. Ezt a célkitűzést jó megbízhatósággal sikerült is megvalósítanom nagy örömömre.

2. Elméleti háttér

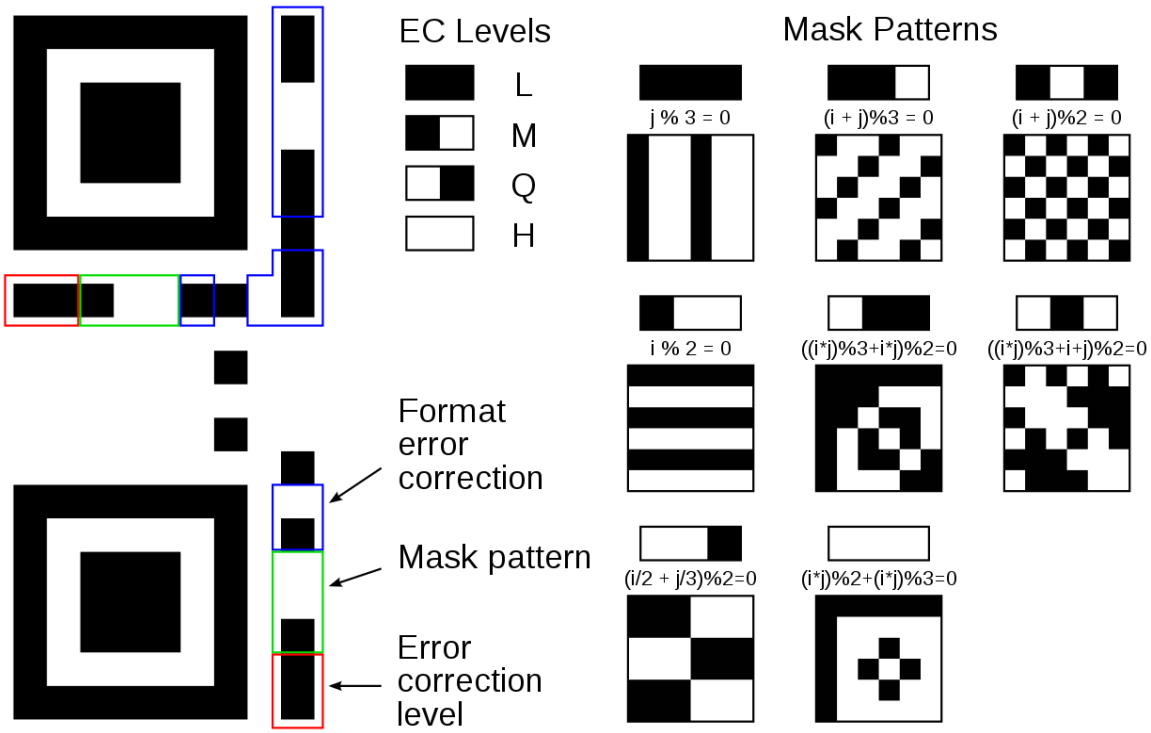
QR- és Vonalkódokkal manapság mindennap találkozunk, ennek köszönhetően mindenki ismeri őket valamilyen szinten. A detektálásukhoz és főleg az olvasásukhoz azonban mélyen ismerni kell a felépítésüket, így ezt most ki is fejtem annak érdekében, hogy a program működését könnyebben meg lehessen ismerni.

2.1 QR-kódok felépítése

A gyakorlatban sokféle méretben terjedtek el, azonban az egyszerűségre való törekvés miatt csak a legkisebb méretű és legegyszerűbb felépítésű 21x21-es méretű kódokkal foglalkoztam a program készítése közben. Ezek a kódok egy 21x21-es mátrixot alkotva tárolnak információt fekete és fehér négyzetek formájában. Az orientációjuk felismerését a 3 sarokban elhelyezett nagyobb négyzetek teszik lehetővé azáltal, hogy a job alsó sarokban fixen nincs ilyen, csak a másik háromban Az 1. képen mindez jól látható.

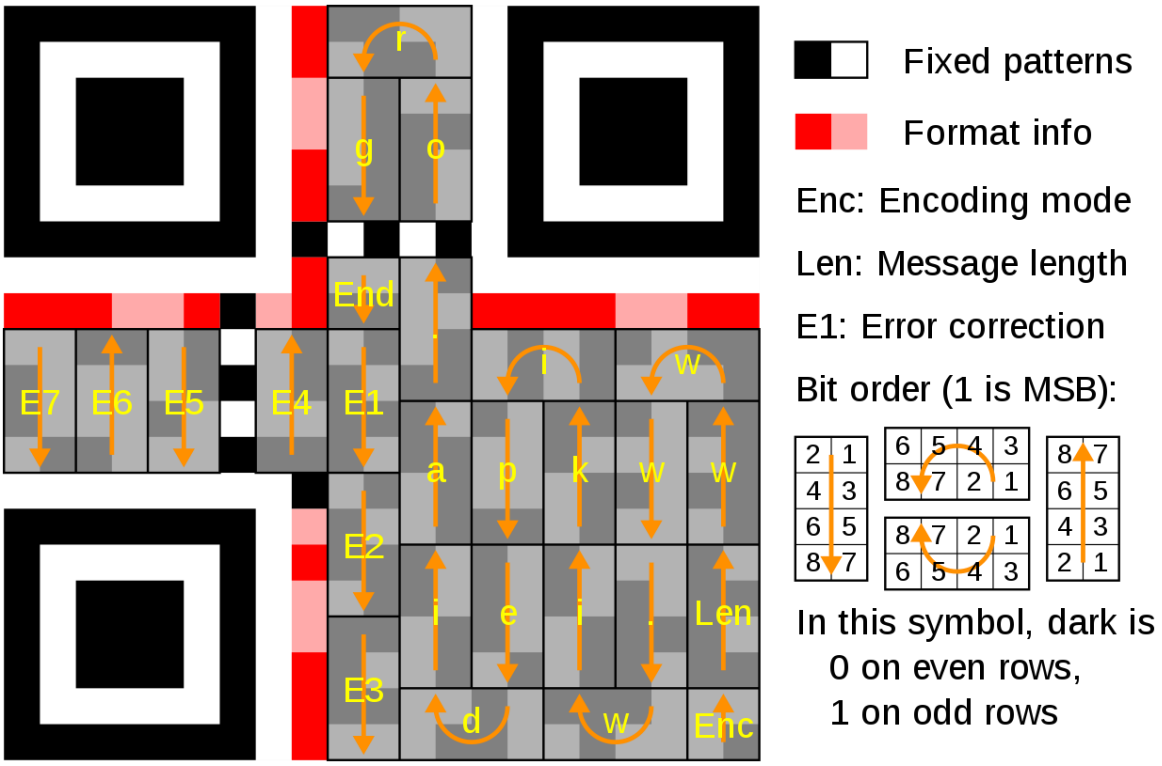
1. Kép

Annak érdekében, hogy a tárolt adatot reprezentáló bitek formája egyenletesen oszlódjon el a képen, valamint, hogy még véletlenül se alkossanak a tájoló négyzetekhez hasonló formát, a kód generálásának végén egy maszkoló eljárást alkalmaznak. Nyilván ebből többfajta létezik a kódok sokszínűsége miatt. Az alkalmazott maszkolási szabályt 3 bit jelzi a kész kódon, 2 különböző helyen dupklikáltan. Ezen 3 bit elhelyezkedése a 2. képen a zölddel jelzett helyen található, a kép jobb oldalán pedig a 8 lehetséges eljárás leírása található.



2. Kép

A megfelelő maszk alkalmazása után már olvasható is a tárolt információ a kódról. Ezt a jobb alsó sarokból kell kezdeni, az első 4 bit tárolja az alkalmazott kódolási technikát, efölött található az információ hossza 8 biten tárolva. Azonban maximum 24 karakter tárolható egy ekkora méretű kódban, így a hossz mező értéke ennél nem lehet hosszabb. A hossz mező felett már kezdődik is a hasznos információ, amely karakterenként 8-8 biten van tárolva. A karakterek olvasásánál fontos a 8-as blokkok orientációja és az azon belüli bit szignifikanciaszint helyessége. A karakterek vége után hibajavító kódok helyezkednek el, amikkel bizonyos szintig javítható a kód minősége sérült kód esetén. A 3. Képen látható a karakterek elhelyezkedése a kódon belül, és a blokkokon belüli sorrend is, ami az egyes bitek szignifikanciáját adja meg. Itt az 1 jelöli a legfontosab bitet, ami 128-at jelöl, a 8 pedig a legkevésbé fontosat, ami 1-et jelöl.



3. Kép

2.2 Vonalkódok felépítése

Szinte az összes boltban található terméken van vonalkód, ami az adott termék azonosítására szolgál. Ezekből is rengeteg fajt terjedt el, én azonban itt is csak egyre fókuszáltam, az EAN-13 típusra, a program tehát csak ezt ismeri. Az EAN-13 típusú vonalkódok egydimenziósak, 95 fekete és fehér oszlopból állnak. Az elején és a végén fixen fekete-fehér-fekete kombináció van, a közepén pedig a fehér-fekete-fehér-fekete-fehér kombináció kötelező. Ezen elemek közt helyezkedik el 6-6 számjegy 7 biten kódolva. A kódolás egyedi, még a bal és jobb oldal közt is különbözik, ezáltal tudja felismerni az olvasó azt, hogy jó irányban, vagy fejjel lefelé fordítva látható-e a kód. A 4. képen jól látható az EAN-13 típusú vonalkódok felépítése, valamint az egyes oldalakon a számjegyek kódolása is

4. Kép

2.3 A programban felhasznált módszerek elméleti háttere

- Szürkeárnyalati konverzió

  A bemeneti képből a program az egyes színcsatornákat különböző súllyal véve állítja elő a szürkeárnyalatos változatot, az OpenCV beépített fügvényével. A pontos súlyok: Piros – 0,299; Zöld – 0,587; Kék – 0,114.

```
I_gray = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)
```

- Binarizálás

  A binarizálás során egy szürkeárnyalatos képből egy kétárnyalatú, fekete-fehér kép allítódik elő. A művelet során egy megadott határértéknél kisebb intenzitások esetén fehér, az feletti intenzitások esetén pedig fehér lesz a kimeneti kép adott képppontja. A program a binarizáláshoz használt határértéket minden esetben a szürkeárnyalatos kép hisztogramja alapján állapítja meg az OTSU algoritmus segítségével.

```
_, I_bin = cv2.threshold(I_gray, 0, 255, cv2.THRESH_BINARY
    + cv2.THRESH_OTSU)
```

- Morfológia

A keletkezett bináris kép feldolgozására szolgáló műveletek. Először mindig a megfelelő méretű struktúráló kernelt kell előállítani, majd azt alkalmazni a képre. Zárással a képen található különböző alakzatok kitöltése történik, többféle kernelmérettel, így ki lehet választani a megfelelőt.

```
kernel = v2.getStructuringElement(cv2.MORPH_RECT, (k, k))
J = cv2.morphologyEx(T, cv2.MORPH_CLOSE, kernel,
    iterations = 2)
```

A nyitás művelettel a képen található apróbb zajok szűrése történik, valamint a kontúrok simítása is. Ez a művelet fix kernelmérettel hajtódik végre.

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
K = cv2.morphologyEx(J, cv2.MORPH_OPEN, kernel,
    iterations = 2)
```

- Kontúrkeresés

A morfológiával feldolgozott képen már kirajzolódik a kód alakja, ezen objektumokat a határvonalaik segítségével detektálja kontúrkereséssel. A művelet során a bináris képen a fekete háttér előtt látszódó fehér objektumok folyamatos határait keresi. Az így létrejövő bonyolult geometriai alakzatokat később egyszerűsíti.

```
contours, _ = cv2.findContours(K, cv2.RETR_TREE,
    cv2.CHAIN_APPROX_SIMPLE)
```

- Éldetektálás

Az egyenesdetektálás előfeltétele, éleket leíró képen végezhető el az egyenesdetektálás. Éldetektálás során jelen esetben a program bináris képből indul ki, és annak minden pontján közelíti a képfüggvény deriváltját. Ahol a megadott határnál nagyobb ez a közelített érték, ott él van, és a kimeneti képen ezt fehér képponttal jelzi.

```
canny = cv2.Canny(ROI, 200, 250)
```

- Egyenesdetektálás

Az éleket leíró képen a program minden képpont esetén felírja az ott áthaladó egyenesek egyenletét, és ezen görbék metszéspontjainak távolságából “válogat”, a megadott határon belülieket megtartja, a többit eldobja.

```
L = cv2.HoughLines(blurred, 1, np.pi/180, int(w / 2 * 0.8))
```

- Sarokpontdetektálás

  Sarok 2, különböző irányú él találkozásánál van. Bináris képen olyan helyeket keres a detektor, ahol minden irányú elmozdulás esetén nagy az intenzitás változása.

```
R = cv2.cornerHarris(black.astype(np.uint8), 5, 5, 0.1)
```

3. Megvalósítás

3.1 QR-kódok olvasása

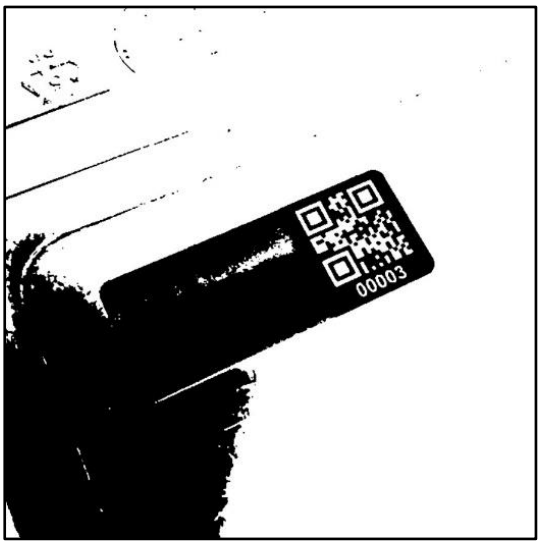
A QR-kódok felismerését a program a kép beolvasásával kezdi (5. Kép), majd azt szürkeárnyalatossá alakítja (6. Kép), és létrehozza ebből a kép binarizált változatát is (7. Kép). Ez azért szükséges, mert így gyorsabb a feldolgozás, valamint a színes változat nem taralmaz semmilyen plusz információt, mivel a kód fekete-fehér, vagyis bináris.

5. Kép

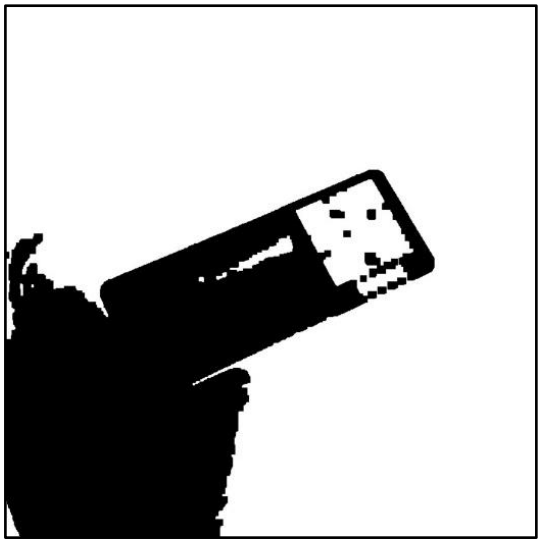
6. Kép

7. Kép

Ezután a kép bináris változatának az invertáltján (8. Kép) a program megpróbálja egy teljes négyzetté kibővíteni a kódot (9. Kép) morfológiai zárás művelettel, amit később megpróbál majd detektálni.



8. Kép



9. Kép

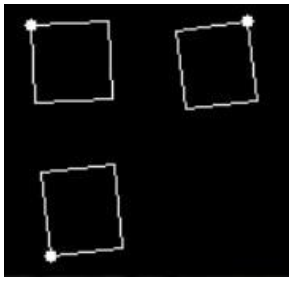
A detektálás kontúrkereséssel történik, az előző lépésben létrejött képen olyan téglalapokat keres a program, amik minél közelebb vannak a négyzethez (10. Kép). Itt még nem fontos a pontosság, csak körülbelülre állapítja meg a kód pozícióját. A detektált négyzetnek megfelelően megpróbálja egyenesbe fordítani a képet (11. Kép). A tájolás itt még szintén nem fontos, az is később less pontosítva.

10. Kép

11. Kép

Ezután a program kivágja a kép kódot tartalmazó részét (12. Kép), a többivel ezután már nem kell foglalkoznia. A kivágott részleten megkeresi a három tájoló négyzetet, valamint ezek sarkait (13. Kép), amik egyben a kód három sarkát is jelentik, és ezeknek megfelelően befordítja a pontos helyzetébe a kódot (14. Kép).

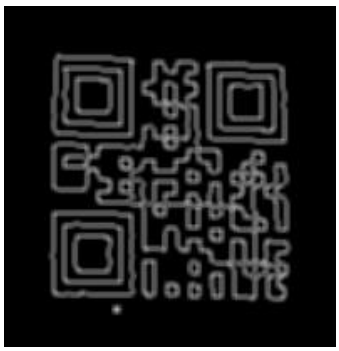
12. Kép



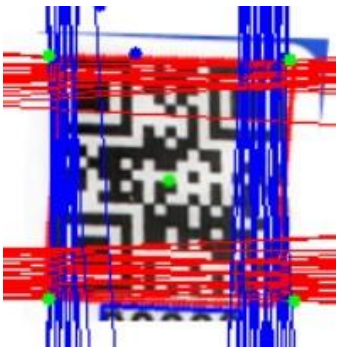
13. Kép

14. Kép

Ezen a ponton már ismert a kód 3 sarka, így tehát már csak a jobb alsó pont ismerete hiányzik. Ezt a képen található vízszintes és függőleges egyenesek metszéspontjából számolja ki a program (15, 16. Kép), emellett ezzel a módszerrel is meghatározásra kerül a másik 3 pont, és később azokat használja, amelyek pontosabbak. Mindebből már meghatározható a kód pontos helyzete is. (17. Kép)



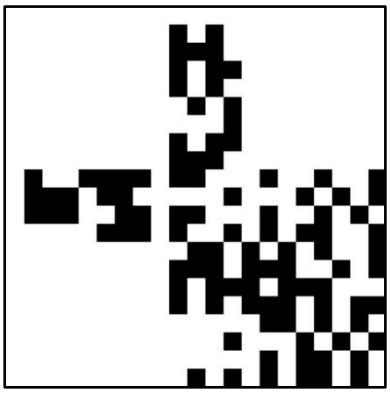
15. Kép



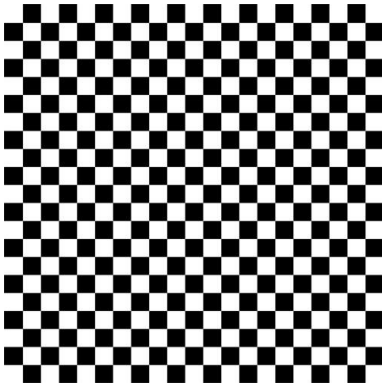
16. Kép

17. Kép

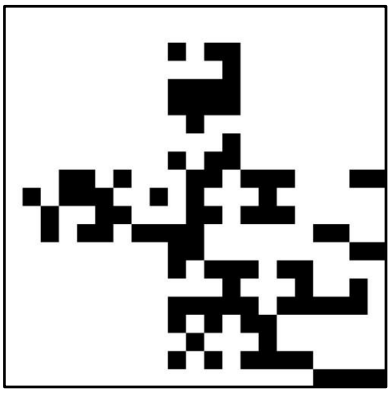
Ezt követően a program lekicsinyíti 21x21-es méretűre a kódot, és különválsztja belőle a valóban hasznos adatot tartalmazó részt (18. Kép). Emellett megállapítja a használandó maszk típusát (19. Kép) és azt alkalmazza is az adatot tartalmazó részre (20. Kép).



18. Kép



19. Kép



20. Kép

Végezetül nem maradt más hátra, mint a kapott kódból az információ értelmezése és kiírása.

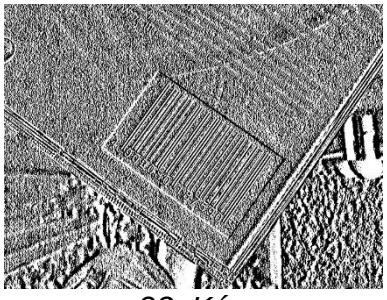
3.2 Vonalkódok olvasása

A vonalkódok olvasása is az input kép beolvasásával (21. Kép), valamint annak szürkeárnyalatossá alakításával kezdődik (22. Kép), a binarizálása viszont még csak később lesz.

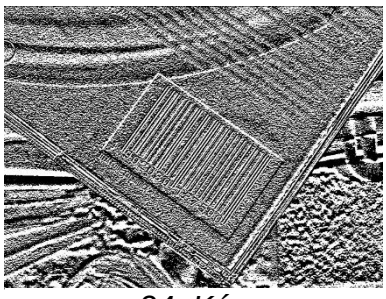
21. Kép

22. Kép

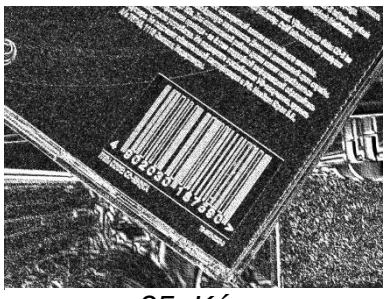
A következő lépés a szürkeárnyalatos kép X és Y irányú deriváltjának a megállapítása (23, 24. Kép) Scharr szűrővel, valamint a kettő kivonása egymásból (25. Kép).



23. Kép

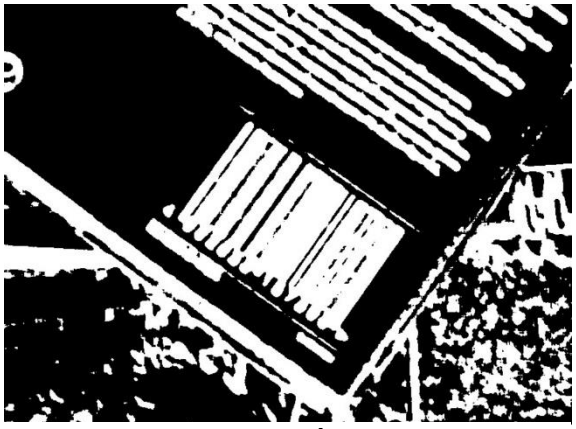


24. Kép



25. Kép

Ezután következik a kapott kép binarizálása (26. Kép) és a téglalappá bővítés (27. Kép)



26. Kép

27. Kép

Ezt követi a kapott téglalap detektálása (28. Kép), a kép beforgatása (29. Kép), valamint a kód kivágása külön képbe (30. Kép).

28. Kép

29. Kép

30. Kép

A dekóder algoritmus itt már megpróbálja értelmezni a kódot. Úgy működik, hogy a kép minden sorában megállapítja az aktuális kódot, és ha a kötelező részei megfelelőek, akkor egy ‘szavazatot’ ad le rá. A végén a legtöbb ‘szavazatot’ kapó kódot próbálja meg átalakítani számokká. Ha ezen a ponton nem sikerül az algoritmusnak a detekció, akkor a kép tovább van pontosítva egyenesdetektálással és sarokpontdetektálással is és azokon a változatokon is megpróbálja az értelmezést.

4. Tesztelés

Az elkészült programot mindkét kódfajta esetében 25-25 előnyös és 15-15 előnytelen képpel teszteltem, ez összesen tehát 80 kép. A kategóriákat úgy definiáltam, hogy előnyös képek esetén a kód szemből van fotózva, és jól elválik a környezetétől. Az előnytelen képek esetében pedig a kód nagyon oldalról van fotózva, nem válik el a környezetétől, vagy nagyon homályos. Az előnyös QR-kódok esetén a program 20 képre adott jó eredményt, az előnytelenek esetén pedig 3-ra. Vonalkódok esetén előnyösből 18-ra, előnyteleneknél szintén 3-ra kaptam jó eredményt. Az eredményeket az 1. Táblázat vizuálisan is összefoglalja.

Tesztelés Eredménye	QR-kód		Vonalkód	
	Előnyös	Előnytelen	Előnyös	Előnytelen
Összesen	25	15	25	15
Jó eredmény	20 (80%)	3 (20%)	18 (72%)	3 (20%)
Rossz eredmény	5 (20%)	12 (80%)	7 (28%)	12 (80%)

1. Táblázat

Az előnyös képek elnevezését mindkét kódtípus esetén 0-tól kezdtem, ezek közül a rossz eredményt adó képek 50-től kezdődnek. Előnytelen képek estén a nevek kezdete 100 és 150. Az alábbiakban néhány rosszul működő eset okát bemutatom.

4.1 QR

qr52.jpg

Jól megtalálja a képen a kódot, a tájolás is rendben működik, de hibás a kód a képen, nem egyeznek a maszkot jelző bitek a képen, így nem tud továbbhaladni a program.

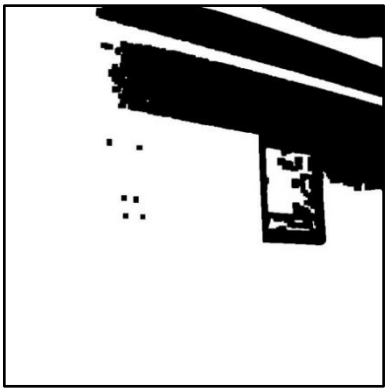
31. Kép

32. Kép

qr153.jpg

Olyan szögből van fotózva, hogy már nem tudja négyzetté kiegészíteni a képet a program.

33. Kép



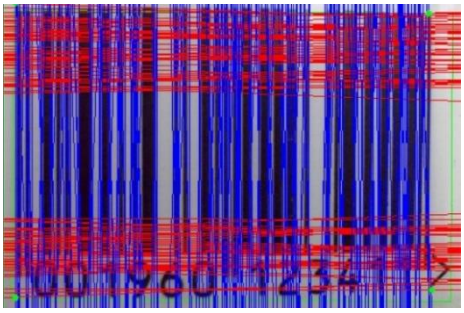
34. Kép

4.2 Vonalkód

bc50.jpg

A detektálás jól ment, azonban a binarizált képen az oszlopok kontúrjai zajosak, és már a 2-es kernelméretű nyitásnál is összefolynak. Valamint a képen a fotózási szögből adódóan a job oldali oszlopok kicsit ferdék, aminek a javítására szolgáló egyenesdetektor működésébe bezavar a jobb oldali kacsacsőr a vonalkódtól jobbra, amit nem sikerült kiszűrnöm.

35. Kép



36. Kép

bc55.jpg

A vonalkód nem válik el eléggé a környezetétől, ezért lokálásnál egy nagyobb tartományt talált, aminek a széle megzavarja a dekódoló algoritmust.

37. Kép

38. Kép

5. Felhasználói leírás

5.1 A program futtatásához szükséges csomagok:

- Python
- NumPy
- OpneCV
- sys
- collections
- copy

5.2 A program indításának menete

Parancssorban a program a neve után minimum 2 argumentumot vár. Az első helyen egy kötőjel megadása után azt kell megadni, hogy QR-kódo(ka)t vagy vonalkódo(ka)t akarunk olvasni. QR-kódot a ‘-qr’ karakterlánccal jelezhetjük, vonalkódot pedig a ‘-bc’ karakterlánccal, az angol barcode rövidítése után. A második, vagy azutáni argumentum(ok)ban a program a vizsgálandó kép(ek) nevét várja, például ‘bc0.jpg’. Itt nincs felsőkorlát, bármennyi kép megadható, annyi csak a megkötés, hogy ugyanabban a mappában legyenek a képek, mint a program.

6. Irodalomjegyzék

- Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2
- Pillazo. (2013, March 30). How to Decode a QR Code by Hand [Video]. *YouTube. https://youtu.be/KA8hDldvfv0*
- Adrian Rosebrock, Detecting Barcodes in Images with Python and OpenCV, PyImageSearch, https://www.pyimagesearch.com/2014/11/24/detecting-barcodes-images-python-opencv/, accessed on 10 January 2022
- In One Lesson. (2010, September 12). How Barcodes Work [Video]. YouTube. https://youtu.be/e6aR1k-ympo
- *Stack overflow*. (2008). Retrieved from https://stackoverflow.com/