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

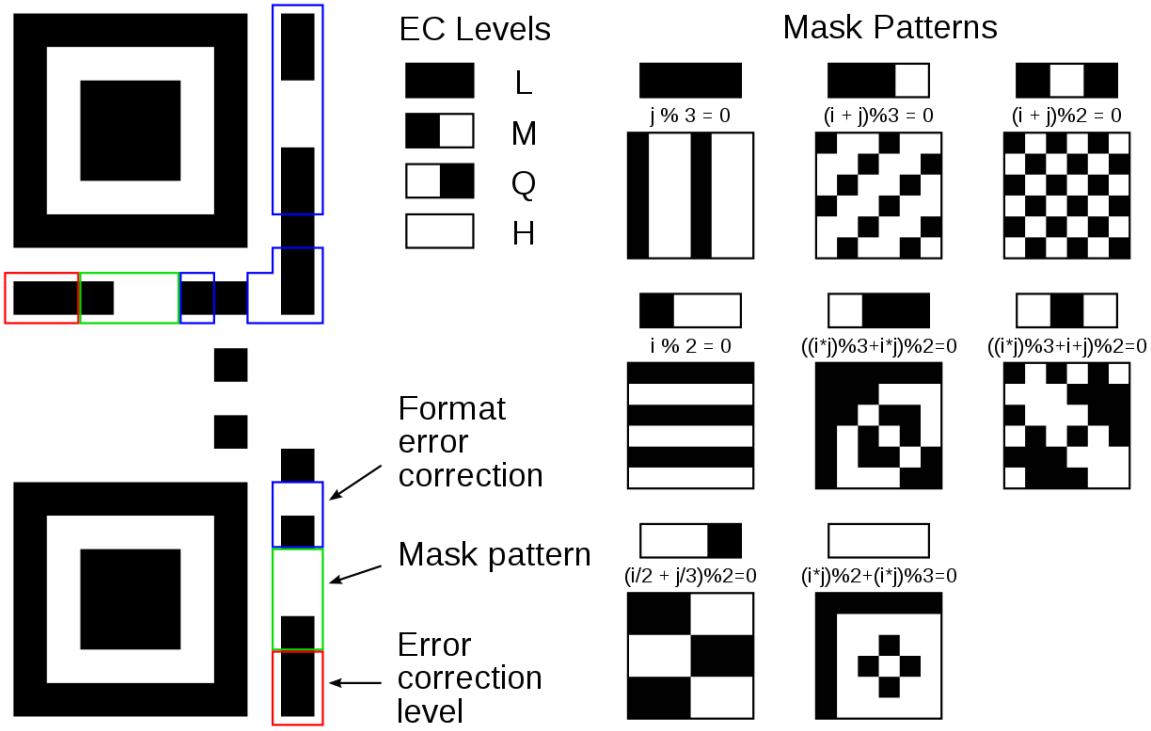
QR- és Vonalkódokkal manapság mindennap találkozunk, ennek köszönhetően mindenki ismeri őket valamilyen szinten. A detektálásukhoz és főleg az olvasásukhoz azonban mélyen ismerni kell a felépítésüket, így ezt most ki is fejtem annak érdekében, hogy a program működését könnyebben meg lehessen ismerni.

1.1. 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.

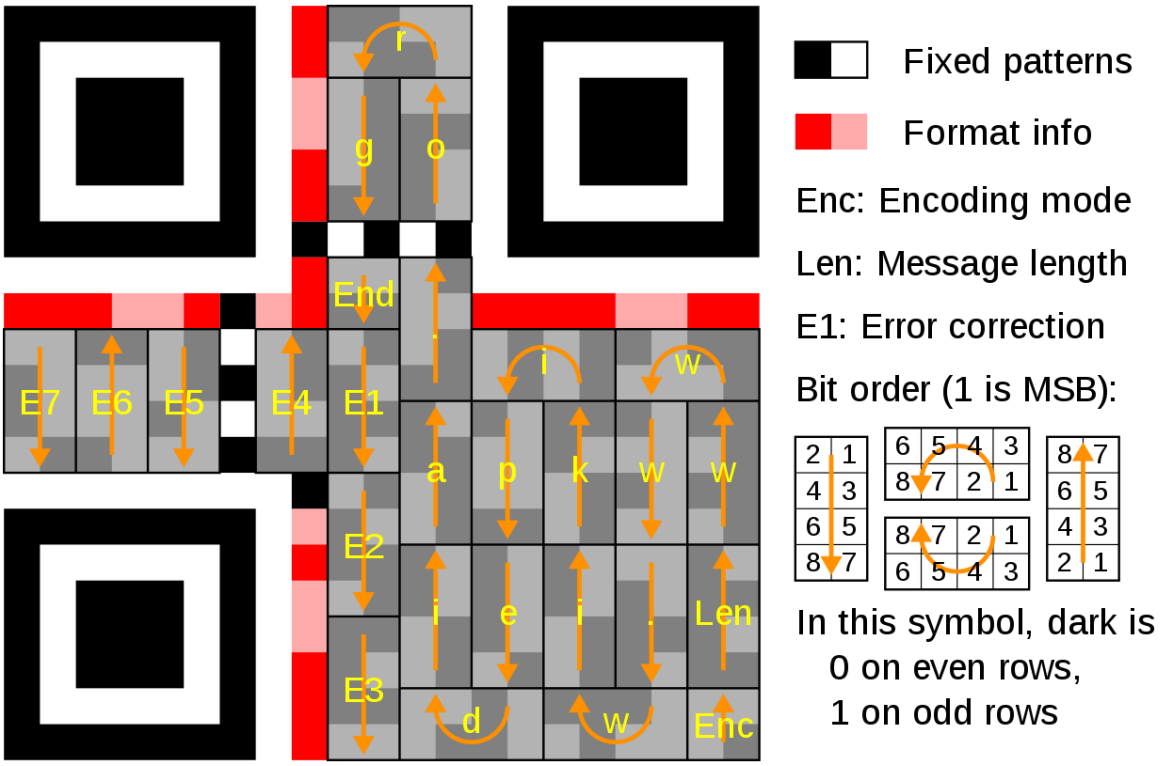
21x21-es méretű QR kód kepi ábrázolása,
jól látható rajta a 3 tájoló négyzet elhelyezkedése

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 a képen a zölddel jelzett helyeken látható az a 3 bit, amelyek tartalmazzák az alkalmazott maszkolási eljárást. A jobb oldalon látható a 8 lehetséges eljárás leírása.

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 karakterek elhelyezkedése a kódon belül, és a blokkokon belüli sorrend, 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.

1.2. 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.

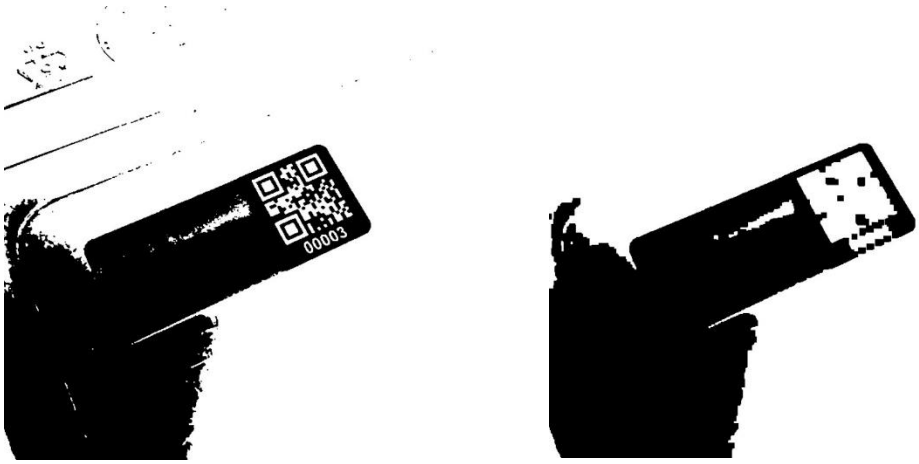
Ezen a képen jól látható az EAN-13 típusú vonalkódok felépítése, valamint az egyes oldalakon a számjegyek kódolása is

3. Megvalósítás

3.1 QR-kódok olvasása

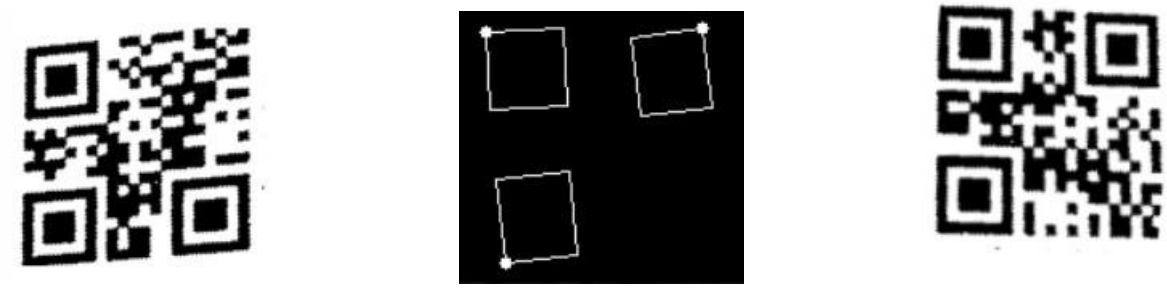
A QR-kódok felismerését a program a kép beolvasásával kezdi, majd azt szürkeárnyalatossá alakítja, és létrehozza ebből a kép binarizált változatát is. Ez azért szükséges, mert így gyorsabb a feldolgozás, valamint a színes változat nem taralmaz semmilyen plusz információt, mivel a kód fekete-fehér, vagyis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

Ezután a kép bináris változatának az invertáltján a program megpróbálja egy teljes négyzetté kibővíteni a kódot, amit később megpróbál majd detektálni.

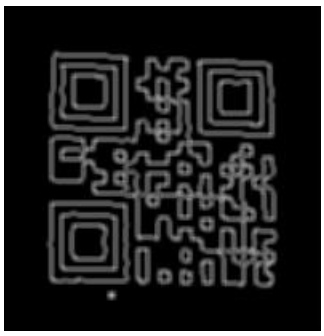
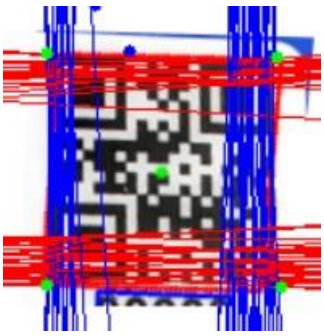


A detektálás során az előző lépésben létrejött képen olyan téglalapokat keres a program, amik minél közelebb vannak a négyzethez. Itt még nem fontos a pontosság, csak körülbelülre állapítja meg a kód pozícióját. A detektált négyzetnek megfelelően megpróbálja egyenesbe fordítani a képet. A tájolás itt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

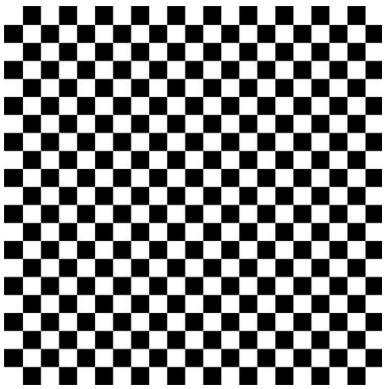
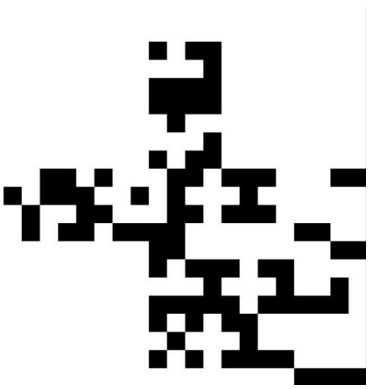
Ezután a program kivágja a kép kódot tartalmazó részét, a többivel ezután már nem kell foglalkoznia. A kivágott részleten megkeresi a három tájoló négyzetet, valamint ezek sarkait, amik egyben a kód három sarkát is jelentik, és ezeknek megfelelően befordítja a pontos helyzetébe a kódot.



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, emellett ezzel a módszerrel is meghatározásra kerül a másik 3 pont, és később azokat használja, amelyek éppen pontosabbak. Mindebből már meghatározható a kód pontos helyzete is.


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

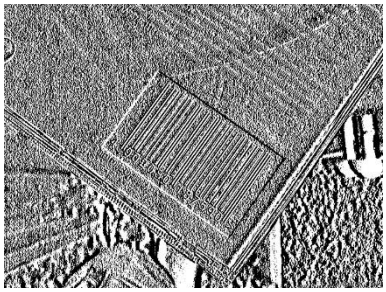
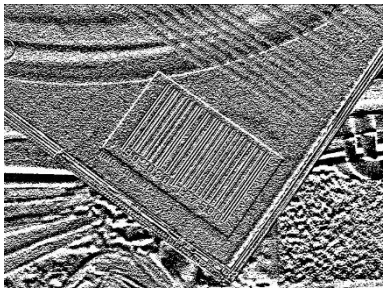
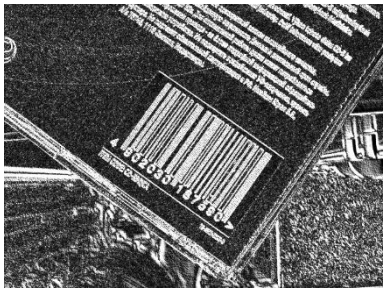




Végezetül nem maradt más hátra, mint a kapott kódból az információ értelmezése és kiírása.

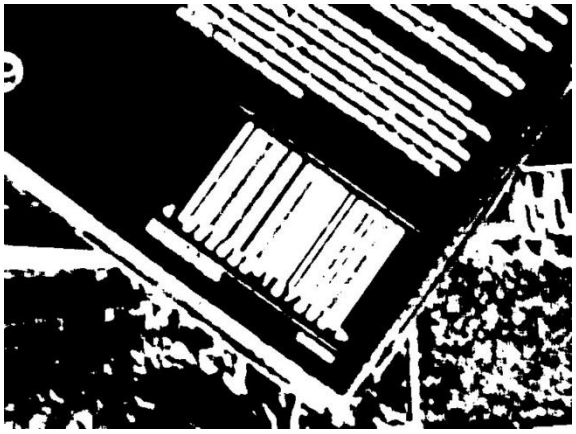
3.2 Vonalkódok olvasása

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


A következő lépés a szürkeárnyalatos kép X és Y irányú deriváltjának a megállapítása Sobel szűrővel, valamint a kettő kivonása egymásból.

Ezután következik a kapott kép binarizálása és a téglalappá bővítés




Ezt követi a kapott téglalap detektálása, a kép beforgatása, valamint a kód kivágása külön képbe.



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 25 vonalkóddal és 25 QR-kóddal teszteltem. A vonalkódok közül 18-ra adott jó eredményt, a QR-kódok közül pedig 20-ra. A jó eredményt adó képeket 0-tól kezdve számoztam be, a rossz eredményt adó képeket pedig 100-tól kezdve számoztam mindkét esetben. Az alábbiakban néhány rosszul működő eset okát bemutatom.

4.1 QR

qr102.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.

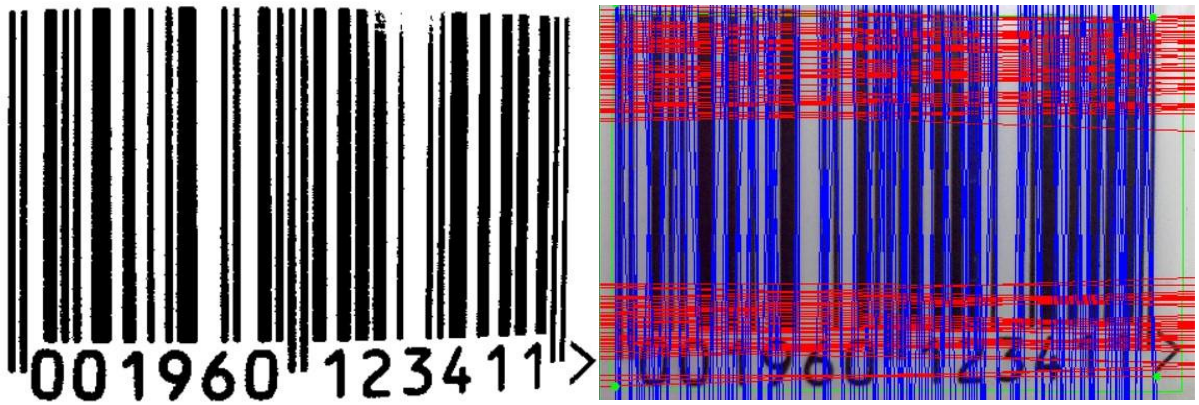
qr103.jpg

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

4.2 Vonalkód

bc100.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.



bc105.jpg

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

5. Felhasználói leírás

5.1 A program futtatásához szükséges csomagok

- 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. Felhasznált források

Tudományos cikkeket nem használtam a feladat megoldása közben, csak különböző internetes oldalakat, a fontosabbakat felsorolom:

- https://en.wikipedia.org/wiki/Barcode
- https://en.wikipedia.org/wiki/QR_code
- https://www.youtube.com/watch?v=e6aR1k-ympo&t=188s&ab_channel=InOneLesson
- https://www.youtube.com/watch?v=XPuTZMp-HE8&t=99s&ab_channel=HalfasInteresting
- https://www.youtube.com/watch?v=KA8hDldvfv0&ab_channel=Pillazo
- https://stackoverflow.com/
- https://www.pyimagesearch.com/
- https://docs.opencv.org/
- https://numpy.org/doc/stable/