Gépi látás (GKNB\_INTM038) beadandó

Pap Tünde – IU7920

2021.12.20.

**Tartalomjegyzék**

[Feladat](#_Feladat) 3

[Elméleti háttér](#_Elméleti_háttér) 3

[Megvalósítás terve és kivitelezés](#_Megvalósítás_terve_és) 5

[Tesztelés](#_Tesztelés) 9

[Felhasználói leírás](#_Felhasználói_leírás) 11

**Pénzérme számláló alkalmazás**

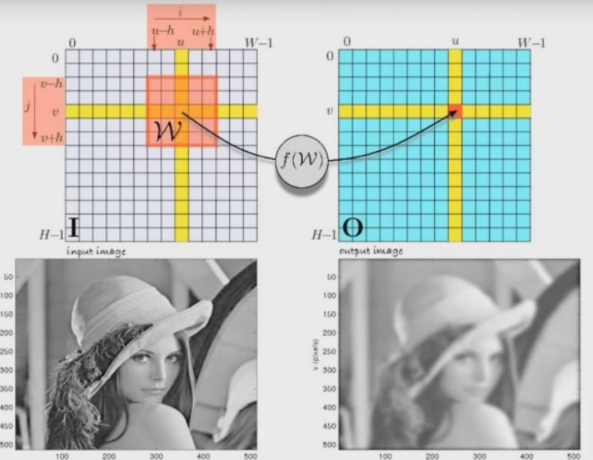
# **Feladat**

Előre definiált jellemzőkkel rendelkező pénzérméket tartalmazó fényképről megállapítani, hogy mennyi a képen látható érmék összes értéke.

# **Elméleti háttér**

Első sorban szeretném ismertetni az általam használt függvények és technikák háttér ismeretét. Lesz szó a kép homályosításáról, a kép kicsinyítéséről, a kép szín módjairól, és az él detektálásról.

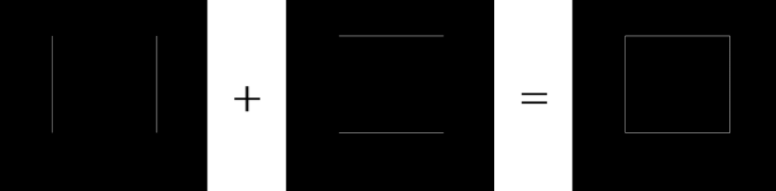
A kép homályosításánál a kép pixeleivel dolgoznak, mint a legtöbb esetben. A homályosításhoz lokális értékkészlet transzformációval dolgozik a függvény, vagyis a képek pixeleit nézi át adott távolságra. Itt az a lényeg, hogy a szomszédos pixelekhez képest milyen értékeket tárol el a képről. A homályosítást egy gauss eloszlást követő függvénnyel végeztem. A technikánál egy adott mátrix-xal végig megyünk a kép pixelein, evvel a kép homályosított mását kapjuk meg. A leggyakrabban használt mátrixot a következő ábrán látható. A technikánál a közelebbi szomszédok nagyobb súllyal kerülnek figyelembe. Igy a nagyon sötét pixelek és a nagyon világosok is megmaradnak. Ennek segítségével később kevésbé találhat hamis köröket az élkereső.



1. ábra. homályosítás

Majd a kép kicsinyítése is segíthet az előfeldolgozásban, mivel így kisebb terjedelmű képpel kell dolgoznia az algoritmusnak. A kép kicsinyítését és nagyítását összefoglalónéven interpolációs technikának nevezzük. Ebből a technikából háromfajta is létezik: a legközelebbi szomszéd elve, bilineáris interpoláció és bicubic interpoláció. A legközelebbi szomszéd elve a 4 legközelebbi szomszéd közül annak az értékét vesszük, amelyik a legközelebb van. A bilineáris interpolációnál a 4 legközelebbi szomszéd távolságát függvényében súlyozott átlagának az értékét vesszük. A bicubic interpoláció alapján pedig a 16 legközelebbi szomszéd alapján parciális deriválás felhasználásával határozzuk meg az értékeket.

Éldetektálás technikája azokon a helyeken alkalmazható, ahol nagy mértékben megváltoznak az intenzitásértékek. Nem csak az objektumok határain találhatóak ilyen területek, ezért a feldolgozást nagyban megnehezítheti; az objektumok árnyéka, törésvonal az objektumok területén és alakzatok az objektum felületén. A kép vízszintes és függőleges irányból lesz vizsgálva, majd az értékek deriváltját véve megkapjuk az eltérő intenzitásértékeket. Az éldetektálást konvoluciós szűrők segítségével lehet végezni. Az ábrán egy egyszerű példát lehet megtekinteni. Az éldetektálásnál külön nézi a függőleges és a vízszintes irányt majd összeadja a képeket.



1. ábra. éldetektálás példa

# **Megvalósítás terve és kivitelezés**

**Első sorban az algoritmusom lépéseit szeretném ismertetni:**

1. Kezdeti kép beolvasása.

2. Szürke árnyalatos kép előállítása, kicsinyítése, majd kör detektálás.

3. Kép szín módjának átalakítása.

4. Legnagyobb kör megkeresése.

5. Arány számítás a legnagyobb körhöz képest.

6. Majd előre megadott méretarányok alapján dönt.

7. Érmék összegének visszaadása a felhasználó számára.



1. ábra. eredeti kép

**Kezdetben a program tömböket hozz létre a képekhez tartozó adatok tárolásához, majd egy függvényt hív meg, amely a txt fájl tartalmából beolvassa a képek elérési útvonalát, illetve a képhez tartozó tényleges szám összeget. Ezeket az adatokat visszaadja, majd for ciklusban végig menve a képeken meghívja rá az érme detektáló függvényt.**

A detektálás elött nagyon fontos megjegyezni a kép előfeldolgozását is, a zajokat és ezeknek a javítását. Első sorban a képet szürke árnyalatosra alakítom, a cv2.cvtColor függvény segítségével.



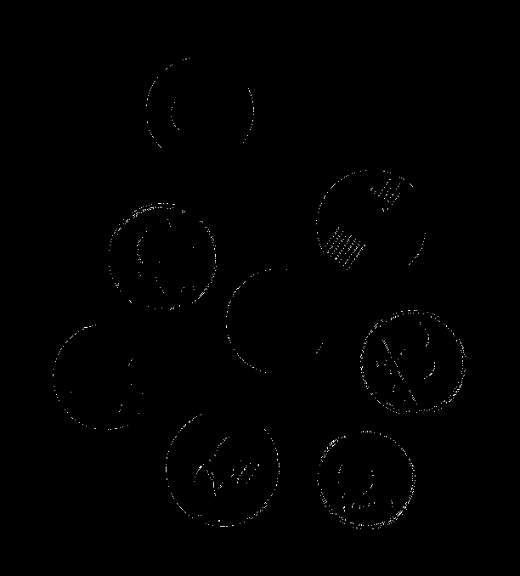
1. ábra. szürke árnyalatos kép

Ezután a képet homályosítom a zajok eltüntetése céljából. A cv2.bilateralFilter(image, szigma szín, szigma tér, szegély típusa) függvény segítségével a későbbi függvények kevesebb hamis kört fognak detektálni.



1. ábra. szürke árnyalatos és homályosított

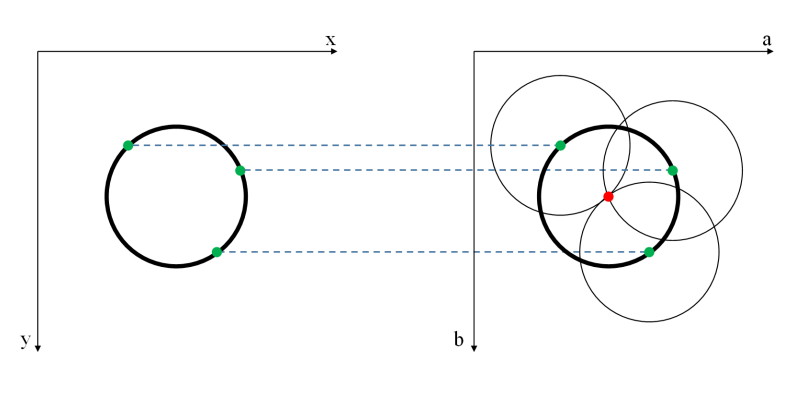
**A megvalósításom a feladattal kapcsolatban az éldetektálást veszi alapul, ezért evvel kapcsolatban szeretném az alapokat leírni.** Lelehet programozni sorról-sorra is az éldetektálót, de sokkal egyszerűbb és hatékonyabb függvényt használni, mivel van is erre egy megfelelő függvény a Canny. Az intenzitás érték nagymértékben, hirtelen megváltozik ott fog a függvény fehérvonalat hagyni, a kép többi része pedig fekete lesz. A Canny detektor minden élt felismeri, ezért érme keresésre nem a legalkalmasabb.



1. ábra. Canny detektor

Mivel érmék lesznek detektálva, ezért erre egy még egyszerűbb módszer is létezik. Olyan eljárás, amely külön köröket fog keresni. A Hough transzformáció segítségével a körök külön-külön felismerhetőek lesznek a gép számára. Körök keresésének szempontjából két fajta keresést tudunk megkülönböztetni, az egyik, ha ismerjük az érmék méreteit, a másik pedig mikor nem ismerjük az érmék méreteit.

Kör detektálás hasonlóan az éldetektáláshoz ugyan azon a megoldáson indul el. A képnek az éleit detektálja, majd azokból állapítja meg a köröket. A lényeg, hogy a Canny detektor által létrehozott kép pontjain egy vagy több változó sugarú kört helyez el. Ahol ezeknek a köröknek az íve metszi egymást ott nagy valószínűséggel a kör középpont található meg, melynek a sugara előre definiált vagy a metsző körök sugarának átlaga.



1. ábra. dia6.pdf 34 oldala

A kör detektálásában két nagy véglet van, mikor nem talál kört illetve, amikor nagyon sok kört megtalál. Ennek segítésére a cv2.HoughCircles nevű függvény alkalmas.

*cv2.HoughCircles(image, method, dp, minDist, param1, param2, minRadius, maxRadius) -> circles*

Általában a függvény jól érzékeli a körök középpontját. Előfordulhat azonban, hogy nem találja meg a megfelelő sugarakat. Viszont a sugártartományt (minRadius és maxRadius) meglehet adni, evvel segítjük a függvényt. Vagy a [HOUGH\_GRADIENT](https://docs-opencv-org.translate.goog/4.x/dd/d1a/group__imgproc__feature.html?_x_tr_sl=en&_x_tr_tl=hu&_x_tr_hl=hu&_x_tr_pto=nui,sc#gga073687a5b96ac7a3ab5802eb5510fe65ab1bf00a90864db34b2f72fa76389931d) metódus esetén belehet állítani a maxRadius-t negatív számra, hogy csak a középpontokat adja vissza sugárkeresés nélkül, és egy további eljárással megtalálja a megfelelő sugarat.

Paraméterek

|  |  |  |
| --- | --- | --- |
| image |  | 8 bites, egycsatornás, szürkeárnyalatos bemeneti kép |
| method |  | Érzékelési módszer. [HOUGH\_GRADIENT](https://docs-opencv-org.translate.goog/4.x/dd/d1a/group__imgproc__feature.html?_x_tr_sl=en&_x_tr_tl=hu&_x_tr_hl=hu&_x_tr_pto=nui,sc#gga073687a5b96ac7a3ab5802eb5510fe65ab1bf00a90864db34b2f72fa76389931d) vagy [HOUGH\_GRADIENT\_ALT](https://docs-opencv-org.translate.goog/4.x/dd/d1a/group__imgproc__feature.html?_x_tr_sl=en&_x_tr_tl=hu&_x_tr_hl=hu&_x_tr_pto=nui,sc#gga073687a5b96ac7a3ab5802eb5510fe65aad57c72131c801de427f1fdb55c8c8ad) |
| dp |  | Az akkumulátor felbontásának fordított aránya a képfelbontáshoz |
| minDist |  | Az észlelt körök középpontjai közötti minimális távolság |
| param1 |  | a magasabb küszöb átmenet a Canny éldetektor számára |
| param2 |  | a kör középpontjainak akkumulátor küszöbé az észlelési szakaszban |
| minRadius |  | Minimális körsugár |
| maxRadius |  | Maximális körsugár |

Visszatétési érték:

|  |  |  |
| --- | --- | --- |
| circles |  | Talált körök kimeneti vektora. |

****

1. ábra. Detektált kontúrok a kiinduló képre helyezve

Ezek után a cv2.HoughCircles függvényből visszatért köröket 3 érték megadásával tárolja el; a kör középpontja koordinátában megadva, illetve a kör sugara.

Egyes érmék sugarának közeli értéke miatt csak az pénzdetektálása pontatlan is lehet. Ebből kiindulva fontos volt a szín alapján való detektálás is. Ehhez a plusz megoldáshoz két másik függvényt is alkalmaztam. Az egyik lekicsinyíti a képet, majd a másik függvény a pixelek három adatát tárolja el, ami segítségével lehetett színt detektálni. Másik színmód kellett, mert az eredeti RGB szín módban nagyon tág intervallumon mozogtak az érmék színei. Így az érmék két fő színét nem lehetett megfelelően elkülöníteni egymástól.

A cv2.cvtColor(cv2.COLOR\_BGR2HSV) függvényt használtam először, amit már használtam a szürke árnyalatos képpé alakításnál. Ennek a függvénynek a segítségével a képeket sok más színmódba át lehet alakítani. A színmódók közül a HSV szín módot választottam, mert ennek segítségével megfelelően eltudtam különíteni az érmék sárgás és ezüstös színeit egymástól.

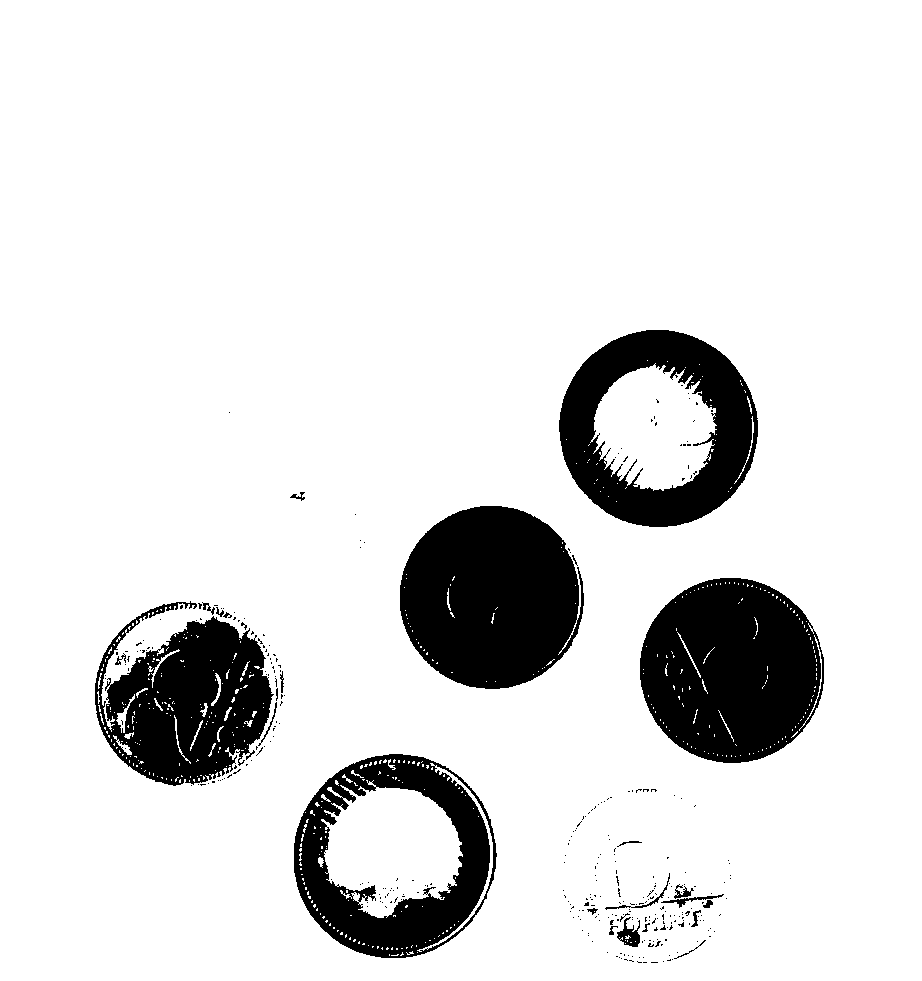
Az ezt követő függvényben a képet lekicsinyítettem, mivel a nagyfelbontású képpel való dolgozás nem cél. A kis felbontású képekkel is helyesen megoldja a képekkel való dolgozást az algoritmus, és emellett még gyorsabb lesz a program.

*cv2.resize(image, None, fx, fy) -> imgresize*

A kicsinyítés után jön egy nagyon fontos függvény, amely segítségével a képek pixeleinek adatait lekérhetjük.

*cv2.split(image) -> vectors*

A függvény visszaadott adatai a képhez kapcsolódó vektor átcsoportosított alakja. Az adatok bizonyos értékek összehasonlításával megállapítható, hogy az a bizonyos pixel sárgás vagy inkább ezüstös színeket hordoz magában. Ehhez egy egyszerű összehasonlítást végeztem el, ha megfeleltek a feltételeknek az értékek akkor meg szoroztam 255-tel. Így a képeken a megfelelő értékek beállításával vissza lehetett kapni a sárga érméket ábrázoló képet.



1. ábra: fekete-fehér kép a sárga szín megkülönböztetéséhez

A kép alapján már meg lehetett állapítani, hogy az adott definiált körök négyzetes alakjában milyen százalékban fordult elő a sárga szín. Az adatokat egy tömbben tároltam el, hogy később minden körhöz a megfelelő százalékot kitudjam olvasni. Ezt követte újra a kör sugarakkal való detektálás.

Itt ellenőriztem, hogy talált-e a Houghtcircles kört, mivel ha egy kört sem detektál, akkor egy None értéket ad vissza. Majd a körök közül egy for ciklussal végig menve megkeresem a legnagyobbat. A legnagyobb körből pedig kiszámolom a többi kör arányát egy másik for ciklust kezdve. A körök arányából feltételezni lehet az érme értékét. A körökön végig menve számolja ki az arányt. Evvel besorolja a lehetséges megfelelő érméhez. Az arány szerinti szűrés után ellenőrzi, hogy az érme színének megegyezik-e a százalékos eltárolt adat értéke. Ha megegyezik, akkor nagyobb valószínűséggel detektált jó éleket. Valamint, ha nem megfelelő a százalékosként eltárolt adat, akkor megnézi újra az arányt, hogy kisebbnek vagy nagyobbnak látta-e és eszerint átsorolja a megfelelő érméhez a detektált kört. Majd az értékét hozzáadja a végösszeghez. Ezt a végösszeget majd a függvény végén visszafogja adni. A visszaadott értéket eltárolja, majd a txt fájlból kapott tényleges értéket és az algoritmus alapján megadott értéket összehasonlítja a teszt függvény.

# **Tesztelés**

**A tesztelést egy külön függvényben végzi. A függvény két paramétert vár, egyik az igazi értékek tömbbe rendezve és az algoritmus által kiszámított értékek is külön tömbbe rendezve. Az igazi értékeket a kezdeti txt- ben vannak megadva a képek nevei mellett egy vesszővel elválasztva. Ezek az értékek a kiolvasásnál el vannak tárolva, hogy később fellehessen őket használni a teszt függvényben.**

**A függvényben végig megy az igazi értékeken és az algoritmus által kiszámított értékeken, majd összehasonlítja azokat. Ha azonosak, akkor hozzáadja a jól kiértékeltekhez, majd ebből egy statisztikát ad vissza, hogy hány százalékban adott jó értékeléseket a képekre.**

**Első próbálkozásra az eredmények nagyon rosszak voltak, de vizsgáltam a köztes képeket a köztes eredményeket és ezek alapján fejlesztettem és módosítottam a beállított értékeket, amelyeket a függvényeknek adtam.**

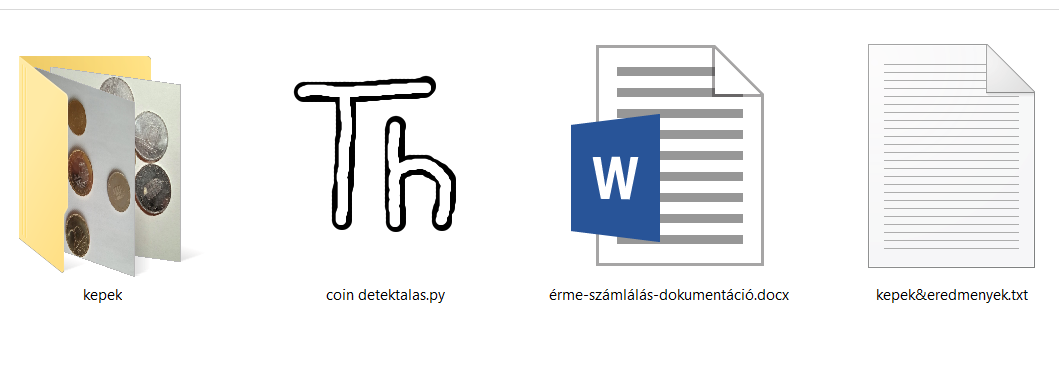
1. ábra. Diagram az érme felismerés arányokon- és színeken alapúló statisztikájáról

**A diagramon a kékkel jelölt oszlop az összes képek darabszáma, amelyen végig megy az algoritmus. A sárgával jelölt oszlop azoknak a képeknek a száma, amelyeket a legvégén teljesen jó értéket adott vissza.**

**Az arányokon alapúló megoldásnak az előnye, hogy dinamikusabb egy megoldás, a fényképező és az érmék közötti távolság lehet különböző is. Viszont hátránya, hogy ha nincs a képen a legnagyobb méretű pénzérme, akkor alapból nem számolhat jól az algoritmus. A megoldás hátránya, hogy nehezen tűri az érmék közötti kis különbségeket. Emiatt az érméket a színük alapján is bekellett sorolni. A képekről ki tudja szűrni az adott érmének a megfelelő színét, majd ezt eltárolja. Ez a plusz ellenőrzés jobb eredményeket hozott végül, mivel a rosszul detektált érméket színük alapján ellenőrizve átsorolja a megfelelő érmékhez.**

# **Felhasználói leírás**

A program futtatására a Thonny nevű programot ajánlom, de megteszi bármely Python értelmező melyet grafikus felületről indítunk. A coin-detectation.py fájlt és a hozzájuk tartozó képeket és txt fájlt kell leszedni egy mappába.



1. ábra. a mappa tartalma

A futtatás után nincs semmi dolga a felhasználónak csak a visszakapott eredményeket elolvashatja az alsó panelről. Amennyiben képet szeretne hozzáadni, a mappába helyezze be a képet. A txt fájlba be kell írni a kép adatait a következő módon: kep.jpg,1234

A képnek ne legyen ékezetes neve, illetve ügyelni kell arra, hogy megfelelő a beírt adatok módja; a kép neve, a kép formátuma és az érmék értéke vesszővel elválasztva jelenik meg.

A találat javítása érdekében célszerű olyan képet adni a programnak mely nagy felbontású és nincsenek erős árnyékok, valamint fény visszaverődés sem az érméről, sem a háttérről.