Tartalomjegyzék

Előszó 1

Folyamat leírása 3

Megvalósítás eszközei 4

A Computer Vision fogalma 5

Hol találkozhatunk ilyen technológiával? 5

Computer Vision műveletek:

Felismerés 6

Mozgás analizáció 8

3D rekonstrukció 9

Kép restauráció 9

Mi az OPEN CV és az EMGU CV? 10

Operációs rendszer támogatás 11

Könyvtárak letöltése 11

Konzolos alkalmazás vagy Form alapú? 12

Referenciák hozzáadása 13

Bemeneti jel kiolvasása és az ehhez kapcsolódó követelmények 15

Projekt előkészítés 16

Képek beolvasása 16

Példa képfájlból történő beolvasásra (színes) 17

Példa képfájlból történő beolvasásra (fekete-fehér) 19

Példa kameráról történő egyszerű beolvasásra (színes) 20

Példa kameráról történő egyszerű beolvasásra (fekete-fehér) 21

Képfeldolgozási eljárások.....22
Gauss szűrő.....22
Példa a Gauss szűrő megvalósításra EMGU CV-ben.....23
Flip (átfordítás).....24
Resize (átméretezés).....25
Threshold.....27

Szegmentációs lehetőségek.....28
Erode (hámozás).....28
Színszegmentáció.....29
HSV szétbontás.....29
HSV szegmentálási példák.....31

Detektálási módszerek.....32
Éldetektálás.....32
Sobel operátor.....33
Sarokdetektálás.....33
Foltdetektálás (blob detection).....34
Hough transzformáció.....35
SIFT (Scale Invariant Feature Transform).....36
SURF (Speeded Up Robust Feature).....40
Tesztesetek.....41
Következtetés az eredményekből.....42
Detektálási példa (SURF).....43
Homográfia és RANSAC.....44

Módszerek megválasztása.....45

Hanghatások alkalmazása.....46
Megvalósítás alapjai.....46
Objektum pozíciójának detektálása.....46
A pozíció meghatározása EMGU CV-ben.....47
Panorámázás megoldása (NAudio).....48
Az NAudio panorámázó működése.....49
A panorámázás implementálása a kódba.....50

A végleges programkód.....51
A program ismert, és javításra szoruló hibái.....54
Minimális rendszerigény:.....54
Telepítési információk.....55

Összefoglalás.....56
Summary.....57
Irodalomjegyzék.....58

Előszó

A mai informált világban egyre nagyobb teret hódít olyan eszközök fejlesztése, amely a képességeikben korlátozott személyek számára (mozgás-, látás-, hallássérültek stb.) megkönnyíti az életet. Az operációs rendszerek is rengeteg kisegítő programmal állnak rendelkezésre, a termék ezzel szélesebb köröket célozhat meg. A szakdolgozat is hasonló célkitűzésekkel született meg. Mik ezek a célok? Programozói állásinterjúkon szokott a következő kérdés előfordulni: „Miért programozunk?”. Erre a kérdésre általában kevés helyes válasz létezik, amelyek közül az egyik így hangzik: „Mert szeretnénk emberek, illetve más élőlények életét, továbbá gazdasági szereplők munkáját megkönnyíteni”. Miért olyan fontos ma, hogy egy villamosmérnök programozói tudással is rendelkezzen? A válasz a digitális technológia villámgyors fejlődésében és a gazdasági szereplők munkaerő-piaci stratégiájában, illetve az onnan történő válogatási formájában keresendő.

Nagyobb városokban tapasztaljuk, hogy egyes gyalogátkelők mellett hangjelzés figyelmezteti a látássérült személyeket a lámpa aktuális státuszáról, ami egy óriási segítségnek számít. A kérdés az, hogy tudunk-e segíteni ezeknek a személyeknek a navigációjában? Oda tudjuk-e őket navigálni az átkelőhöz? Egyes városokban (pl. Bécsben) az átkelőket olyan eszközökkel szerelték fel, amelyek folyamatos kattogó hangot bocsátanak ki magukból. Ezzel akár státuszt is jelezhetnek, illetve átkelési pozíciót. Tudunk-e segíteni abban, hogy olyan átkelőknél, ahol nincs ilyen jelzés valahogy lehetővé tegyük a látássérültek számára, hogy megtalálják az átkelési pontot? A szakdolgozatban leírtaknak célja, hogy ezen a helyzeten javítson, informálva a látássérült személyt a környezetében lévő átkelőkről. Illetve lehetőséget teremt egy második verzió autóvezetők tájékoztatásához a közelben lévő gyalogátkelőről. Egy végleges kiadható, és biztonságosnak mondható termék esetében még több éves fejlesztői, laboratóriumi és kísérleti munka szükséges, részletesebb tesztekkel, olyan felmerülő problémák kiküszöbölésével (akadályokkal való számolás, pontosabb pozicionálás), amelyre anyagi és időbeli okok miatt egy egyszemélyes fejlesztőcsapatnak jelen körülmények között nincs lehetősége.

Ettől függetlenül a dolgozat elkészítésekor igyekeztem a szemléletet és az eddigi ismereteket minél részletesebben átadni, amelyekkel ezek a tesztprojektek továbbgondolhatóak. Természetesen törekedtem nem túl informatikai szemszögből megközelíteni a problémát, hanem villamosmérnöki szemlélettel. Remélhetőleg ez a dolgozat sokak számára egy kiindulási pont lehet bonyolultabb projektek megalkotásához, a mérnöki munka, illetve a digitális képfeldolgozással kapcsolatos elképzelések mélyebb, de egyszerűbb gyakorlati megvalósításához. A működési részleteket a dolgozat további részében fejtem ki.

Jelenlegi munkám során (szoftvertesztelés) sok esetben találkozom forráskóddal, így a programozás egyre inkább az érdeklődés középpontjába került, és elkezdtem megismerni egyes programozási nyelveket. Egy videómegosztón böngészve találkoztam egy képfeldolgozói videoanyaggal, amely felkeltette az érdeklődésem, és az egyetemi tanulmányaim indulásakor már-már körvonalazódott a fejemben, hogy projekt labor keretein belül ezen ismereteket szeretném még jobban elsajátítani, és ha lehetséges a megszerzett ismeretekkel diplomatervet készíteni. Kezdetben sikerült webkamerás képet megjeleníteni, feldolgozni, hisztogramot készíteni, és végül a végleges alakfelismerő tesztprogramot előállítani. A megszerzett ismeretek alapján a szakdolgozatban kifejtem a tesztprogram megalkotásának fizikai követelményeit, a képfeldolgozási eljáráshoz nélkülözhetetlen elméleti alapokat, fejlesztéshez szükséges alapvető programozási ismereteket (ez utóbbi esetében törekedtem a minél egyszerűbb magyarázatra). A dolgozatban kitérek egyes eljárások felhasználhatóságára, illetve az azok közötti különbségekre, ugyanis több út is vezethet egyazon cél eléréséhez. Ennek tükrében össze is hasonlítok néhány ilyen módszert. Ahol lehetséges, ott igyekszem a témakörökhöz kapcsolódó példakódokat is bemutatni, a könnyebb elsajátíthatóság, az egyszerűbb megértés, és a későbbi továbbgondolás céljából.

“Semmit nem lehet megtanítani egy embernek. Csak segíteni abban, hogy rátaláljon önmagán belül.” (Galileo Galilei)

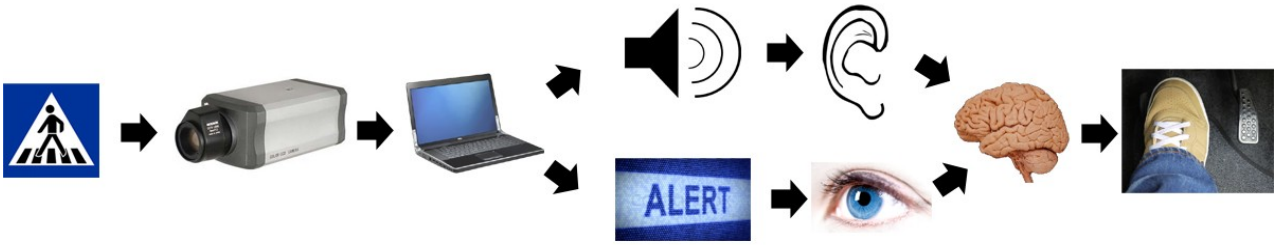
A folyamat leírása

Néhány esetben felvetődik, hogy ez inkább informatikai probléma, mint híradástechnikai. Azonban ha a problémát alaposabban körbejárjuk, akkor felismerhetjük a párhuzamot egyes képfeldolgozói eszközökkel. Ennek belátására azonban tudnunk kell, hogy a projektet milyen eszközökkel, milyen módszerrel akarjuk megvalósítani:

1.) Természetesen szükségünk lesz egy beviteli eszközre, egy kamerára (jelen esetben webkamerára), amelyen keresztül a mozgóképet beolvassuk. Fontos, hogy az eszközt össze tudjuk kötni egy számítógéppel, ugyanis nem utólagos analízist akarunk végezni tárhelyre mentett videoanyagból, hanem valósidejű beolvasást, és feldolgozást akarunk végrehajtani.
2.) A kamerát csatlakoztatnunk kell egy feldolgozó egységre (számítógép), amely valós időben beolvassa a képet, kielemzi azt, majd egy output eszközön keresztül a leszűrt/feldolgozott információt továbbítja a végfelhasználó felé.
3.) Az output eszköz lehet hangszórón keresztül kiadott jelzés, vagy képinformáció (ez felhasználási terület függvényében változik).
4.) A végfelhasználó az output eszköz jelét veszi (szem, fül), és továbbítja ingerek formájában az agy felé. Az agy a hirtelen jelzésre döntés elé kényszerül. A döntés a fejben megszületik, és a felhasználó, az átkelő felé veszi az irányt, vagy az autós verzió esetében megnyomja a pedált.

Beláthatójuk, hogy a probléma megoldása egyirányú **szimplex** kommunikációt igényel, mivel az első feldolgozó egységre nincs hatással. A részletesebb hardverkövetelményekre később térek ki. Mindenesetre a fent leírt 4 pont alapján a folyamat teljes egészében felírható, így a probléma megoldása egyszerűsödni látszik. Ilyen esetekben még fontosabb az aprólékosabb tervezés, mivel nem mindegy, hogy a bemeneten megjelenő formát miként dolgozzuk fel.

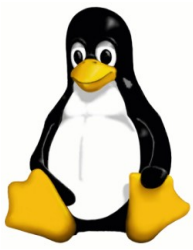
Az alábbi ábrán látható részletesebben ez a folyamat:



Tervezésnél fontos szempont, hogy a kamera és a feldolgozó egység kevés helyet foglaljon el, azonban minél inkább csökkentjük a méreteket, annál inkább fog a projektköltség növekedni. A minél kisebb eszköz előny azonban az emberre történő elhelyezésnél mutatkozik meg leginkább.

A megvalósítás eszközei

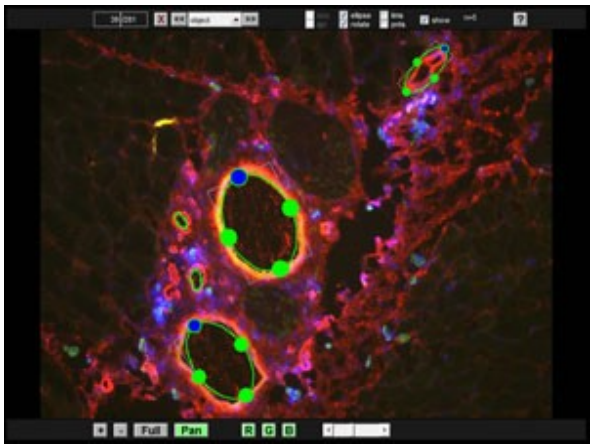
A bemeneti és kimeneti eszközökről majd külön-külön ejtek szót, itt azonban tisztázni kell, hogy milyen módszerrel lehetséges megvalósítani a belső feldolgozást. Első körben szükségünk lesz egy számítógépre, és egy fejlesztői környezetre (IDE). Fontos kiemelni, hogy olyan konfigurációt válasszunk, amely megfelel a fejlesztői környezet és a programunk futtatásához. Erről szintén később a technikai követelmények alatt ejtek szót. Szerencsére mind Windows, Linux, és OSX rendszerek alatt rendelkezésre áll számos környezet és Computer Vision csomag. Így a már meglévő felhasználóknak nem kell rendszert váltaniuk, ha fejleszteni szeretnének. Az általam írt program C# nyelven íródott.



A Computer Vision fogalma

A számítógépes látás (azaz a Computer Vision) az emberi látás azon funkcióit valósítja meg, amelyek a retinai kép elemzését végzik. Ezek elsősorban a képi tartalom értelmezését célozzák meg: a látott képből következtet az objektumok alakjára (felület rekonstrukció), az objektumok térbeli elhelyezkedésére, egymáshoz való viszonyára (mélységi információ kinyerése). Több, időben egymást követő képből a érzékelhet mozgást és követhet mozgó objektumot. Ennek segítségével vizuális, nagy adatmennyiségű információt gyűjthetünk a körülöttünk lévő megfigyelt térről. Természetesen nem minden adat minősül hasznos adatnak, így felhasználás szerint szűrhetjük a megfelelő információt.

Hol találkozunk ilyen technológiával?



Az egyik legnagyobb felhasználási terület az orvosi célú képfeldolgozás, ahol vizuális adatokat gyűjtenek a páciens állapotáról, legyen szó röntgen, vagy mikroszkopikus információról. A feldolgozás során a program kiértékel és diagnosztizál, továbbá minta alapján keres a szervezetben változásokat. Ennek segítségével pontosabban, és automatizáltan deríthetőek fel daganatok, törések, stb.

Hasonló elven működnek egyes parkolóházakban a rendszámleolvasó, és számlázó rendszerek. Az illető személy behajt egy ilyen parkolóházba, kap egy blokkot, majd órákkal később fizet, és távozik. Technikailag azonban egy kamera által beolvassák a rendszámtábla információt, feldolgozzák, továbbítják egy adatbázisba, majd kihajtásnál kiolvassák az eltelt idő lekérdezésével együtt. Hasonló formában működnek rendszámellenőrzések a rendőrségnél, az autópálya kezelőnél stb. A computer vision segítségével a részletesebb arc- és alakfelismerésre is lehetőség nyílik.

Computer Vision műveletek

Felismerés

Az egyik alapvető kérdés a gépi látás, a képfeldolgozás és a computer vision területén belül, hogy a feldolgozott kép, vagy képsorok tartalmazzák-e az általunk vizsgált elemet a pixelhalmazban, vagy sem. Ez a feladat egy ember számára nagyon egyszerű, de egy gép számára már sokkal bonyolultabb. Ha pontosabbak akarunk lenni, akkor a meghatározása a nehezebb, a végrehajtási részt már pillanatok alatt végezheti el (többszörösen hasonlíthat össze elemeket adott idő alatt, mint egy ember). A meglévő eljárások akkor működnek a leghatékonyabban, ha a vizsgálandó elemek egyszerűek. Például az alkazat egyszerű, vagy nagyon jól elkülöníthető a képen. Ehhez természetesen szükséges, hogy a háttér megfelelően különüljön el a vizsgált elemtől. Továbbá fontos a minta a kamerához viszonyított elhelyezkedése is.

<u>A felismerési folyamat :</u>

- Objektum keresése:

  A képen egy minta alapján keressük a pixelhalmazban a keresett objektumot.

- Azonosítás:

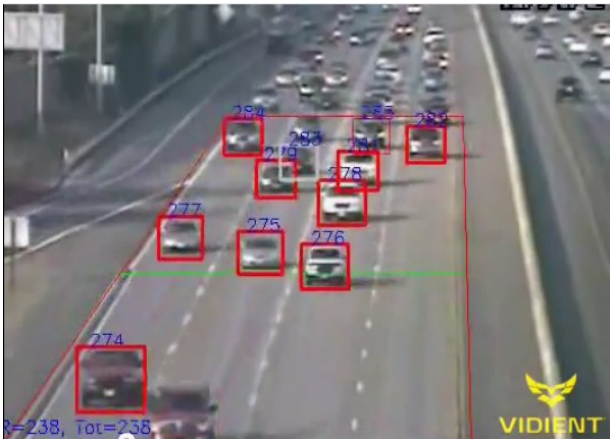
  A vizsgált mintaobjektum egy példánya felismerésre került a pixelhalmazban.

- Pozíció detektálás:

  Az elem összehasonlításra kerül, pozíció meghatározása. A detektálás többnyire egy egyszerű és gyors számítási műveleteken alapul. Sok esetben nem a teljes kép egészét, hanem egy apró részletét vizsgálják. A feldolgozott kép természetesen még tovább vizsgálható a célnak megfelelően.

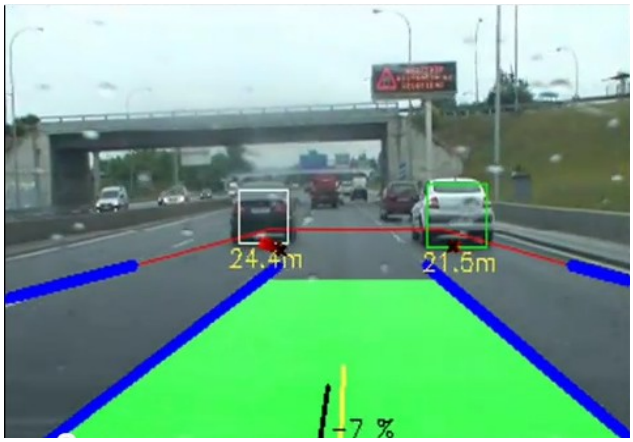
Specializált feladatkörök a felismerést alapul véve:

Tartalom alapú képvisszafejtés:



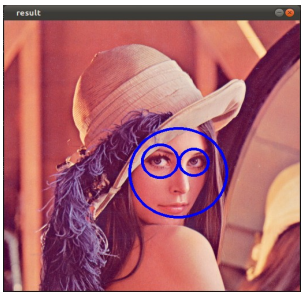
Ez a módszer arra szolgál, hogy kiszűrjük egy képen az azonos tulajdonsággal bíró elemeket (méret, szín, forma stb.) a többi fellelhető elem közül. Példaként lehet említeni akár egy computer vision alapú forgalomszámláló programot. Például kíváncsiak vagyunk, hogy egy pályaszakaszon hány autó halad át egy bizonyos idő alatt. A lekért képen ugye találhatóak mozgó, illetve fix pontok. Ebben az esetben azért van szerencsénk, mivel az aszfalt színe jóval elkülönül az autók színétől, és ha ezt pontosan be is tudjuk állítani, akkor a találati arányunk javulni fog.

Helyzetbecslés



Adott egy objektumunk a kamera előtt, aminek kíváncsiak vagyunk a pozíciójára, a kamerától való elhelyezkedésére. Fontos, hogy ebben az esetben a kamera megfelelően legyen kalibrálva, illetve a programunknak ismernie kell a mintaképként bedolgozott objektum távolságát (ebben az esetben adott a referenciaméret, és utána ehhez már egyszerűbb viszonyítani). Szintén autós példánál maradva a követési távolság kalkulációját lehet ezzel a módszerrel mérni. Minél inkább növekszik a mintaelemként definiált kép mérete, annál közelebb kerülünk az objektumhoz.

Arcfelismerés



Ha a computer vision segítségével lehetőségünk van egy képen belüli elemek felismertetésére, akkor az arcdetektálás is megvalósítható. Természetesen itt más módszert kell alkalmazni, megfelelő képi szűrőket, hogy az arcvonal kirajzolható legyen a szemekkel és a szájjal együtt. Minél pontosabb szűrést alkalmazunk, annál pontosabb eredményt kapunk. A bonyolultabb szűrés alkalmazásának egyik nagy hátránya a nagy feldolgozási idő, így célszerű egy optimális megoldást találni ennek fejlesztésekor.

Mozgás analizáció

Néhány feladat a mozgás analizálására szolgál. Ebben az esetben megfigyeljük a beolvasott képszekvencián az általunk vizsgált területet, és kockáról-kockára kielemezzük annak mozgását, aktuális pozícióját, sőt mérhetjük a sebességét is. Erre egy kiváló példa, amikor a traffipaxot computer vision módszerrel alkalmazzák.

Specializált feladatkörök a mozgás analizációt alapul véve:

Egomotion („sajátmozgás”)

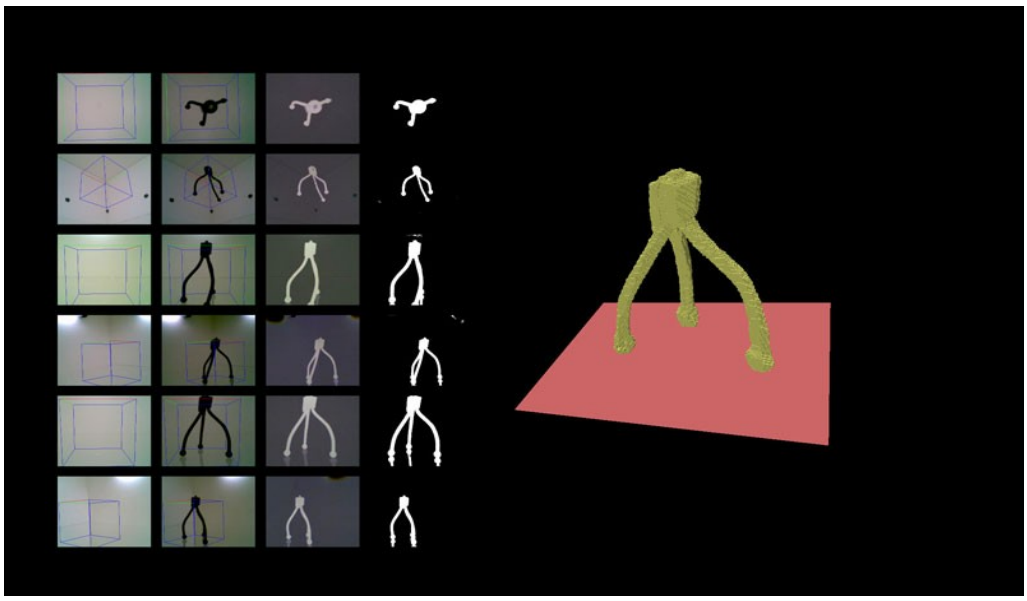
A környezetet alapul véve meg tudjuk állapítani a megfigyelő állapot (kamera) pozícióját, le tudjuk írni a mozgását. Itt nem azt vizsgáljuk, hogy egy megfigyelt elem miként mozog a térben, hanem hogy a kamera miként helyezkedik el ebben a környezetben.

Mozgáskövetés (Tracking)

Itt többnyire a vizsgált kép egy kiemelt részének a képen való elmozdulását, és a képkockánkénti pixelkülönbségeket analizáljuk. Természetesen tesszük ezt azután, hogy a képen található számunkra fontosnak vélt tartományt detektáltuk, és felismertük.

3D rekonstrukció

Egy vagy több kamerakép, vagy video alapján egy 3D-s modell állítható össze. A képeken hasonló pontok fedezhetőek fel különböző szögből, amely során lehetőség nyílik a megfigyelt objektum „körbejárására”. Minél több ilyen pont létezik, és minél több szögből, az elem annál részletesebb lesz, a feldolgozása pedig lassabb. Ez alapján pedig következtethetünk a 3D-s formára.



Kép restauráció

A képi restauráció során a képre ráült zajokat, és torzulásokat, homályos részeket szedhetjük ki különböző képi szűrők segítségével. A szűrés során figyelnünk kell a kép többi elemére is, hogy a modellünk a zajtól pontosan szétválasztható legyen. Természetesen minél részletesebb szűrést akarunk végrehajtani, annál inkább nő a feldolgozási idő.

Mi az OPEN CV és az EMGU CV?

Az OPEN CV egy az Intel által fejlesztett olyan programozási funkciókat tartalmazó könyvtár (library), amely segítségével a fent bemutatott eljárások hatékonyan valósíthatók meg. A könyvtárcsomag nyílt forráskódú, szabadon felhasználható. Az OPEN CV-t 1999-ben indították el, eleinte kutatási célokat szolgálva. Az alfa verziót 2000-ben adták ki, amelyet 2005-ig néhány béta követett. A legelső 1.0-ás verzió 2006-ban került ki, és 2008 óta folyamatos fejlesztés alatt áll. Mondhatni egy nagyon friss technológiáról van szó. Azonban ezt a verziót egy könnyebben használható változat követte, amely újabb funkciókat tartalmazott, és az implementálást megkönnyítette. Ezt a 2.0-ás verziót 2009 októberében adták ki, és napjainkig egy független orosz fejlesztőcsoport félévente ad ki rá frissítéseket. A könyvtárcsomagot eredetileg C nyelven írták, majd a 2.0-ás verzióban megjelent a C++ támogatás, ami később szinte ki is váltotta a C-t. Természetesen nem csak ezen a nyelven lehet fejleszteni computer vision témakörben, ugyanis az OPEN CV-hez fejlesztettek úgynevezett „wrapper”-eket, amelyek segítségével már C#-ban, Python, Ruby, és Java nyelveken is használható a csomag. Természetesen léteznek hasonló könyvtárcsomagok, mint például az Aforge.Net, VXL, vagy az Integrated Vision Toolkit (IVT), amelyekkel hasonló problémákat oldhatunk meg. Természetesen ezek kombinálhatóak is.

Az EMGU CV tulajdonképpen egy Open CV C# wrapper, ami annyit takar, hogy Open CV-s műveleteket C# nyelven is írhatunk. Ehhez ugyanúgy tartoznak könyvtárak, pusztán annyi különbséggel, hogy az eredeti Open CV dll fileokat is mellékelni kell a projekthez. Ehhez kapcsolódóan kevés magyar nyelvű leírás áll rendelkezésre, sőt alig lelhető fel. Néhány helyen fellelhető pár angol nyelvű útmutató, viszont a termék honlapján rengeteg leírás, és segítség található a használathoz.

Operációs rendszer támogatás

Az OPEN CV a következő rendszereken futtatható: Windows, Android, Maemo, FreeBSD, OpenBSD, iOS, Linux és Mac OS.

Az EMGU CV esetében a verziók közötti különbséget az alábbi táblázat mutatja meg:

Name	Emgu CV (Open Source)	Emgu CV (Commercial Optimized)	Emgu CV for iOS (Commercial)	Emgu CV for Android (Beta)
OS	Windows, Linux, Mac OSX	Windows	iOS (iPhone, IPad, IPod Touch)	Android
Supported CPU Architecture	i386, x64	i386, x64	armeabi, armeabi-v7, i386 (Simulator)	armeabi, armeabi-v7a, x86
GPU Processing	✓	✓	X	X
Machine Learning	✓	✓	✓	✓
Tesseract OCR	✓	✓	✓	✓
Intel TBB (multi-thread)	X	✓	X	X
Intel IPP (high performance)	X	✓	X	X
Intel C++ Compiler (fast code)	X	✓	X	X
Exception Handling	✓	✓	✓	✓
Debugger Visualizer	✓	✓	X	X
Emgu.CV.UI	✓	✓	X	X
License	GPL	Commercial License	Commercial License	Commercial License

A könyvtárak letöltése

Már számos fejlesztői környezet érhető el interneten keresztül, a felhasználóra van bízva, hogy melyik C# IDE-t használja a fejlesztésre. A könyvtárcsomag letöltéséhez és konfigurációjához a következőkre lesz szükség:

Letöltéshez:

- Nyissuk meg a http://www.emgu.com/wiki/index.php/Download_And_Installation weboldalt, ahol a telepítéssel kapcsolatos információkat találjuk.
- Nyissuk meg a http://sourceforge.net/projects/emgucv/files/emgucv/2.3.0/ weboldalt
- Válasszuk ki operációs rendszer szerint a csomagunkat, és töltsük le egy külön helyre.
- Ebben a csomagban megtalálhatóak azok a dll fileok, amelyek a CV futtatásához nélkülözhetetlenek.

Konzolos alkalmazás vagy Form alapú?

A kérdés megválaszolásához azt kell eldöntenünk, hogy szeretnénk-e felhasználói interakciókkal bővíteni a programunkat, fontos-e a vizualizáció ezen a területen, továbbá, hogy klikkelhet-e user a felületen. Ha igen, akkor mindenképp a Formos megvalósítást kell választani. Jelen program működése azonban egy képbeolvasásra, szegmentációra, detektálásra, és felismerésre lett fejlesztve, egyoldalú kommunikációval, amely hang-, illetve terminálablakban megjelenő információt jelenít meg kimenetként. Ennek tükrében konzolos alkalmazásról beszélünk.

Példa az említett alkalmazások kimenetére, és kommunikációs felületére:

```
Video sequence started
Találat - BAL - 1:27:36 - PTS:{X=52,72172, Y=426,4015}
Találat - BAL - 1:27:37 - PTS:{X=62,6114, Y=429,0466}
Találat - BAL - 1:27:37 - PTS:{X=60,98357, Y=429,3111}
Találat - BAL - 1:27:38 - PTS:{X=63,45903, Y=430,1717}
Találat - BAL - 1:27:39 - PTS:{X=66,70195, Y=425,7398}
Találat - BAL - 1:27:39 - PTS:{X=73,26549, Y=437,8641}
Találat - BAL - 1:27:40 - PTS:{X=49,81607, Y=423,5937}
Találat - BAL - 1:27:41 - PTS:{X=47,88322, Y=426,6429}
Találat - BAL - 1:27:41 - PTS:{X=48,40799, Y=427,9423}
Találat - BAL - 1:27:42 - PTS:{X=41,22762, Y=425,1699}
Találat - JOBB - 1:27:46 - PTS:{X=505,5468, Y=378,6986}
Találat - JOBB - 1:27:47 - PTS:{X=505,1952, Y=368,8297}
Találat - JOBB - 1:27:48 - PTS:{X=510,9786, Y=364,1612}
Találat - JOBB - 1:27:48 - PTS:{X=514,687, Y=364,0323}
Találat - JOBB - 1:27:49 - PTS:{X=509,9498, Y=365,6298}
Találat - JOBB - 1:27:50 - PTS:{X=507,9339, Y=362,2061}
Találat - JOBB - 1:27:51 - PTS:{X=502,3512, Y=363,4152}
Találat - JOBB - 1:27:52 - PTS:{X=490,3269, Y=360,9669}
Találat - JOBB - 1:27:52 - PTS:{X=489,0294, Y=361,9531}
Találat - BAL - 1:27:56 - PTS:{X=221,3863, Y=362,6069}
Találat - BAL - 1:27:56 - PTS:{X=222,7013, Y=363,7162}
Találat - BAL - 1:27:57 - PTS:{X=222,3982, Y=362,5516}
Találat - BAL - 1:27:58 - PTS:{X=224,4509, Y=358,3358}
Találat - BAL - 1:27:58 - PTS:{X=228,8947, Y=356,538}
```

A fent látható képen a gyalogátkelő tábla felismerő program tesztüzemének konzolos kimenete látható.

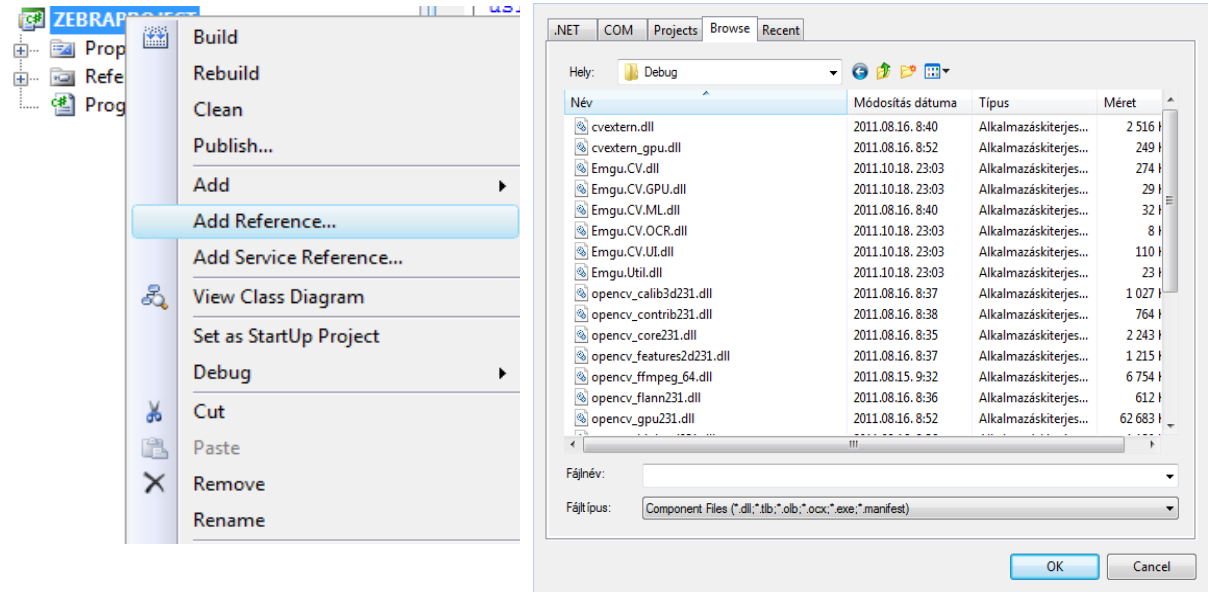
A balra látható képen pedig egy szem felismerő alkalmazás mintája látható annak interaktív felületével egy Windows Form alapú alkalmazás formájában.

A referenciák hozzáadása

Ahhoz, hogy az alapfunkciókat elérhessük, és egyáltalán használhatóvá váljon az eddig leírt információ, ahhoz a projektünkhöz referenciákat kell hozzáadni. Ezek a referenciák azok a dll fileok, amelyekből a programunk dolgozni fog. Az alapfunkciók eléréséhez a következő 3 DLL file-ra lesz szükségünk:

- Emgu.CV.dll
- Emgu.CV.UI.dll
- Emgu.Util.dll

Ezt úgy tehetjük meg, hogy a Projekt nevünkre jobb klikkelünk, majd kiválasztjuk a Referencia hozzáadása (vagy Add reference) menüpontot. Ezt követően felnyílik egy ablak, ahol a Böngészés (Browse) fül alatt kiválaszthatjuk a letöltött csomag bin/Debug könyvtárából a szükséges fájlokat



Ez a három fájl azonban még nem elegendő a sikerhez, ezzel még nem tudunk képeket manipulálni, feldolgozni, felismerni. Ezeket az aktuális feldolgozó egységnél fogom ismertetni.

A projekthez ezen felül hozzá kell adni a szükséges OPEN CV fileokat is. A Projekt nevünkre jobb klikkelünk, majd kiválasztjuk az Add existing file menüpontot, és a létező fájlt hozzáadjuk a projekthez. Ezt követően felnyílik egy ablak, ahol a letöltött könyvtárakból kiválasztjuk a következő fájlokat:

- opencv_coreXXX.dll
- opencv_imgprocXXX.dll
- opencv_highguiXXX.dll
- cvextern.dll

(Jelen esetben az XXX jelölés a letöltött csomagunk verziószámát mutatja: például a 2.3.0 esetben az opencv_core230.dll-re lesz szükségünk).

Ne felejtsük el a .NET fül alatt található System.Drawing és System.Windows.Forms referenciát!

Ezen felül be kell állítani ezeknél a fájloknál, hogy mindig másolódjanak a kimeneti könyvtárba:

- Jobb klikk a fájlon, majd kiválasztjuk a Tulajdonságok (Properties) menüpontot
- Megkeressük a Copy to Output Directory menüelemet, és a legördülő pontokból kiválasztjuk a Copy always funkciót.
- Ezt minden fájlnál be kell állítani (a hozzáadott képeknél és hangoknál is)

Győződjünk meg arról, hogy x86 vagy x64 architektúrát használunk, majd állítsuk be a platform targetet:

- Jobb klikk a projekten
- Válasszuk ki a tulajdonságok menüpontot
- Majd a felugró ablakban a Build menüpontot, és ott már látszódni fog az architektúra beállítás legördülő elem (vagy egyválasztós mező).
- Állítsuk be az architektúrának megfelelő opciót.

A bemeneti jel kiolvasása és az ehhez kapcsolódó követelmények

Egy közönséges digitális kamera felfogható egy olyan eszközként, amely a látóterében lévő 3D valós *látvány*ról egy 2D *kép*et készít. Jelen esetben olyan kamerákkal foglalkozunk, ahol ez a leképezés *középpontos vetítés* formájában valósul meg. A digitális kamera részletes működésének leírása jelen esetben elhanyagolható, mivel a feldolgozási folyamatba minimálisan szól bele.

A megfelelő bemeneti eszköz megválasztása

A projekt megtervezése során el kell döntenünk a következőket:

- A vizsgálandó elem mérete, és ennek környezete
- A vizsgálandó elemben a kérdéses információ mennyire részletes (pixelszám)
- Az elem pozíciója (állandó vagy mozog?)
- A képzaj mennyisége, illetve milyen kihatással van a vizsgált képre.
- A vizsgálandó elem előfordulása (helyszín)
- A környezet világossági tényezője

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

Az EMGU CV használatával egy kép pixelértékeit úgy olvashatjuk ki, mintha mátrixból olvasnánk. Például egy 800x600-as kép esetén egy 600 sorból és 800 oszlopból álló mátrixról beszélhetünk. Egy képszekvenciánál fontosak a képméretek, mivel annál lassabb a feldolgozási idő, így csökken a másodpercenként beolvasott képkockák száma. Egyszerű állókép esetén, ahol nincs szekvencia, ott ez elhanyagolható. A képkockák elérése példa a későbbiekben bemutatásra kerül. Célszerű a projekthez olyan kamerát választani, amely tartalmaz zoom funkciót, ha a vizsgált elem a kamerához képest távolabb helyezkedik el, és időben szeretnénk információt lekérni.

A projekt előkészítése

A projektben a hozzáadott dll fileokra hivatkozni kell (using), így érhetjük el a fejlesztéshez szükséges kódokat:

```
using System;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.UI;
//… stb.

namespace <projekt neve> //lásd: namespace tesztprojekt
{
        class MainClass
                {
                public static void Main(string[] args)
                        {
                        //(ide jön a program)
                        }
                }
}
```

A fenti példa csak egy minta. A projektben előfordulhatnak olyan kódok, amelyek más dll-eket igényelnek, így azokat is „be kell használni”.

Képek beolvasása

A beolvasásnak kétféle módja létezik:

- Kamerából, vagy videofájlból (capture)
- Fájlból megnyitva (képfájl)

A következőekben a képfájl beolvasást, és kameráról történő képkocka lekérdezést fogom prezentálni, mivel jelen projekt is ezt a 2 fajta metódust alkalmazza.

Példa a képfájlból történő beolvasásra (színes):

Először ellenőrizni kell, hogy a felhasznált referenciákat hozzáadtuk-e a projektünkhöz.

- Először szükségünk lesz egy képnézőre (ImageViewer), ahova a beolvasott képet illesztjük, és ezen a nézőkén megjelenítjük.
- Ezen felül szükségünk lesz egy tárolóelemre, amelybe a képet helyezzük, és amit majd később átadunk az ImageViewer-nek.
- Mivel színes képet olvasunk be, ezért a képtároló egységnek a következő tulajdonságokkal kell rendelkeznie:

Image<Bgr, Byte> változónév

Egy képi változó két alapvető paraméterből áll, ahol az első a színtípust adja meg, a második pedig a “mélységtípust”: **Image<TColor, Tdepth>**

Az első paraméter ez alapján a következőket tartalmazhatja:

- Gray (sima fekete-fehér egy csatornás)
- Bgr (Blue Green Red)
- Bgra (Blue Green Red Alpha)
- Hsv (Hue Saturation Value)
- Hls (Hue Lightness Saturation)
- Lab (CIE L*a*b*)
- Luv (CIE L*u*v*)
- Xyz (CIE XYZ.Rec 709 with D65 white point)
- Ycc (YCrCb JPEG)

A második paraméter lehetőségei:

- Byte
- Sbyte
- Single (float)
- Double
- Uint16
- Int16
- Int32 (int)

Példa:

Egy 480x320 felbontású Bgr kép létrehozása 8-bites előjel nélküli mélységgel:

Image<Bgr, Byte> img1 = new Image<Bgr, Byte>(480, 320);

Ugyanez, csak kék háttérrel:

Image<Bgr, Byte> img1 = new Image<Bgr, Byte>(480, 320, new Bgr(255, 0, 0));

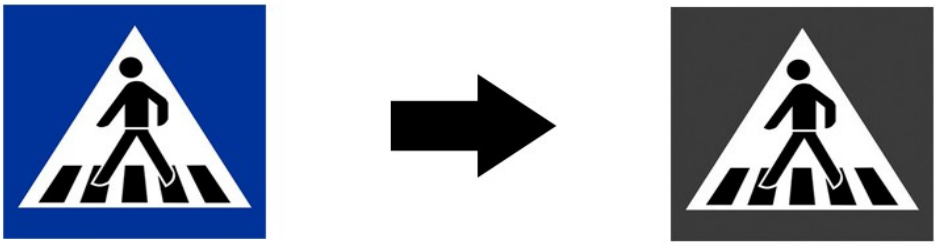
A megvalósításhoz szükséges kód:

```
using System;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.UI;

namespace tesztprojekt
{
class Program
    {
static void Main(string[] args)
        {
          ImageViewer kepnezo = new ImageViewer();    // képnéző ablak definiálása
          Image<Bgr, Byte> kep = new Image<Bgr, byte>("minta.jpg");     // kép definiálása
          kepnezo.Image = kep;  // kép hozzárendelése a képnéző ablakhoz
          kepnezo.ShowDialog(); //képnéző ablak megjelenítése a beolvasott képpel (x-re ablakzárás)
        }
    }
}
//(FONTOS! A beolvasott kép jelen esetben a projekt Debug könyvtárában kell helyet foglaljon!)
```

Példa a képfájlból történő beolvasásra (fekete-fehér):

A kód és az elv kísértetiesen hasonlít a színes verzióhoz, pusztán annyi különbséggel, hogy a képtároló paraméterét lecseréljük Bgr-ről Gray-re, és a képünket a feldolgozás előtt manuálisan átkonvertáljuk fekete-fehér képpé. Ez azonban nem tűnik korrekt megoldásnak, azonban a típusok közti átjárhatóság miatt az EMGU CV tartalmaz típuskonverziót:

```
Image<Bgr, Byte> kep = new Image<Bgr, byte>("minta.jpg");
Image<Gray, Byte> feketefeher = kep.Convert<Gray,Byte>();
```



A megvalósításhoz szükséges kód:

```
using System;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.UI;

namespace tesztprojekt
{
class Program
    {
static void Main(string[] args)
          {
           ImageViewer kepnezo = new ImageViewer(); // képnéző ablak definiálása
           Image<Bgr, Byte> kep = new Image<Bgr, byte>("minta.jpg");// színes kép definiálása
           Image<Gray, Byte> feketefeher = kep.Convert<Gray,Byte>();      // konverzió
           kepnezo.Image = feketefeher;   // a fekete-fehér kép hozzáadása a képnézőhöz
           kepnezo.ShowDialog();          //képnéző ablak megjelenítése a konvertált képpel
          }
     }
 }
//(FONTOS! A beolvasott kép jelen esetben a projekt Debug könyvtárában kell helyet foglaljon!)
```

Példa a kameráról történő egyszerű beolvasásra (színes):

Ez egy szimpla egyszerű eset, ahol nem adunk meg közbenső tárolóegységet, hanem a színinformációt a QueryFrame funkció hordozza. Jelen esetben egy Bgr paraméterű képkocka kerül a képmegjelenítőre:

A megvalósításhoz szükséges kód:

```
using System;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.UI;

namespace tesztprojekt
{

        class Program
         {
        static void Main(string[] args)
                {
                ImageViewer kepnezo = new ImageViewer();     // képnéző ablak definiálása
                Capture beolvas = new Capture(); //capture definiálás (kamerakép beolvasáshoz)

                Application.Idle += new EventHandler(delegate(object sender, EventArgs e)
                {          //addig futtassa ezt az alkalmazást, amíg a képnéző ablak be nem záródott

                kepnezo.Image = beolvas.QueryFrame(); //kamerakép beolvasása a képnéző ablakba

                });

                kepnezo.ShowDialog();  //képnéző ablak megjelenítése a beolvasott képpel
                }

         }
}
```

Abban az esetben, ha több kamerakép állna rendelkezésre, és minket konkrétan csak az egyik képinformációja érdekel, akkor a capture definiálásánál meg kell adnunk a kamerához rendelt id-t:

Capture beolvas = new Capture(x); , ahol x Ɛ N {0...n-1}, ahol n a kamerák száma

Ennek segítségével többkamerás felügyeleti rendszerek alakíthatók ki.

Példa a kameráról történő egyszerű beolvasásra (fekete-fehér):

Az előző példához hasonlóan nem adunk meg közbenső tárolóegységet, hanem a színinformációt a QueryGrayFrame funkció hordozza. Ebben az esetben egy fekete-fehér képkockát olvasunk be közvetlenül a képnézőbe.

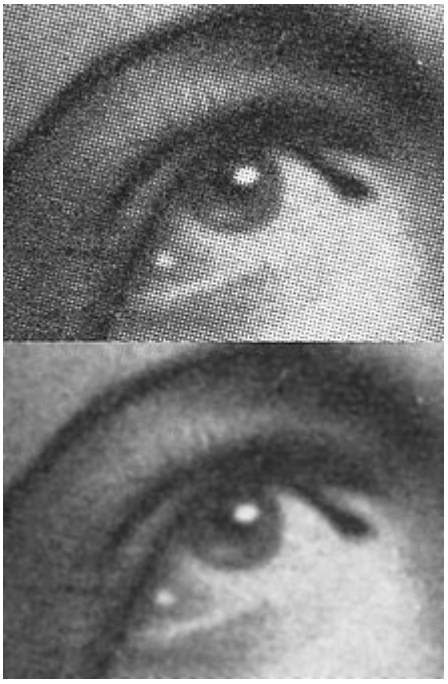
A megvalósításhoz szükséges kód:

```
using System;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.UI;

namespace tesztprojekt
{
class Program
        {
        static void Main(string[] args)
                {
                ImageViewer kepnezo = new ImageViewer();     // képnéző ablak definiálása
                Capture beolvas = new Capture(); //capture definiálás (kamerakép beolvasáshoz)

                Application.Idle += new EventHandler(delegate(object sender, EventArgs e)
                {          //addig futtassa ezt az alkalmazást, amíg a képnéző ablak be nem záródott

                kepnezo.Image = beolvas.QueryGrayFrame(); //fekete-fehér kamerakép beolvasása

                });

                kepnezo.ShowDialog();  //képnéző ablak megjelenítése a beolvasott képpel
                }

         }
}
```

Felmerül a kérdés, hogy vajon miért is jó, ha létezik külön fekete-fehér beolvasás, ha a színes képet átkonvertálhatjuk fekete-fehérré? A válasz a feldolgozási időben rejlik. Jelen esetben ez az 1 aprónak tűnő művelet komolyabban szólhat bele a másodpercenként vetített képkockák számába. Természetesen a kereskedelmi licensszel rendelkező szoftver tartalmaz erre megoldást.

Képfeldolgozási eljárások

Ebben a fejezetben magasabb szintre helyezzük az eddigi ismereteket, és itt a képeket már valóban képként kezeljük, és nem pusztán egy tömbbe zárt adathalmazként.



Az egyik alapvető ilyen eljárás a simítás (**smoothing**), vagy homályosítás (**blurring**). Legtöbb esetben a képre rakódó zajt lehet ezzel csökkenteni, amivel természetesen a részletesség is fokozatosan csökken, az élek elmosódnak, amely a később bemutatott detektáló algoritmusoknál hátrányt képez. Ezt a célt szolgálja a **Gauss szűrő** is, ami felfogható a képfüggvénynek egy Gauss-szűrő függvénnyel vett konvolúciójaként:

$$h(x,y) = f(x,y) * G(x,y)$$

ahol a „csillag” a konvolúció operátorát képezi. A súlyokat a Gauss-eloszlás adja meg. A fenti képletből G(x,y) a következő módon adható meg:

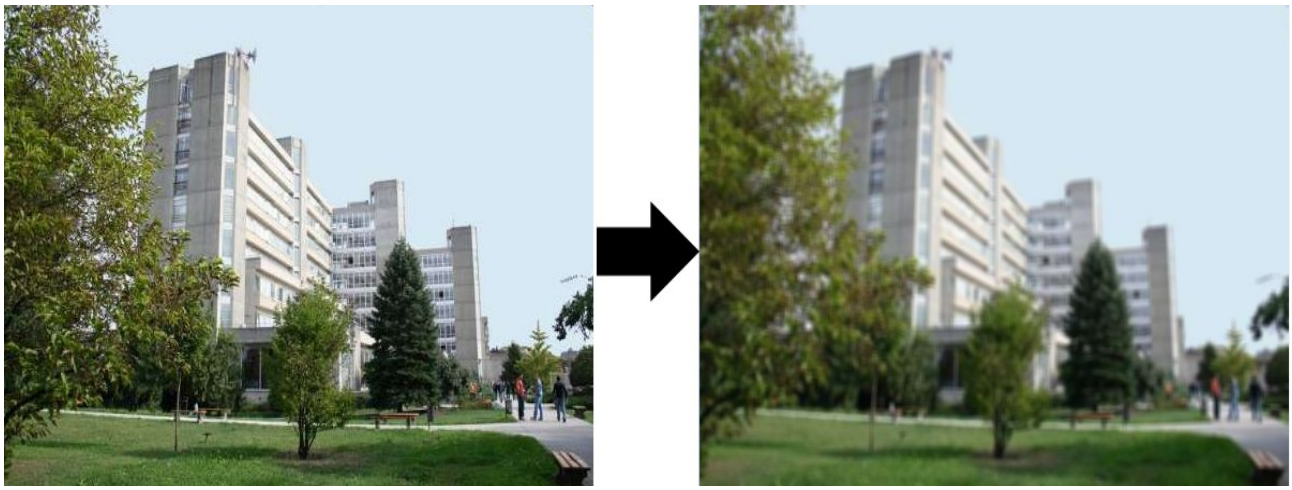
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

ahol:

- x a „forrástól” való távolság a vízszintes tengelyen,
- y a „forrástól” való távolság a függőleges tengelyen,
- σ a szűrő méretét szabályozza. Minél nagyobb ez az érték, annál nagyobb a simítás.

A szűrő maszk körszimmetrikus, és harangalakú (ez utóbbi az exponenciális tag miatt).

Nézzük meg, hogy miként használható ez a funkció EMGU CV esetén:



A fent látható jobb oldali kép EMGU CV általi Gauss-szűrővel készült a bal oldali képből. Ha élesíteni szeretnénk, akkor a megfelelő konvolúciós eljárással ezt megtehetjük.

A megvalósításhoz szükséges kód:

```
using System;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.UI;

namespace tesztprojekt
{
class Program
    {
static void Main(string[] args)
        {
          ImageViewer kepnezo = new ImageViewer();    // képnéző ablak definiálása
          Image<Bgr, Byte> kep = new Image<Bgr, byte>("egyetem.jpeg"); // kép definiálása
          kep._SmoothGaussian(5, 5, 10, 15);     // simítás hozzáadása
          kepnezo.Image = kep;  // kép hozzárendelése a képnéző ablakhoz
          kepnezo.ShowDialog(); //képnéző ablak megjelenítése a beolvasott képpel (x-re ablakzárás)
        }
    }
}
```

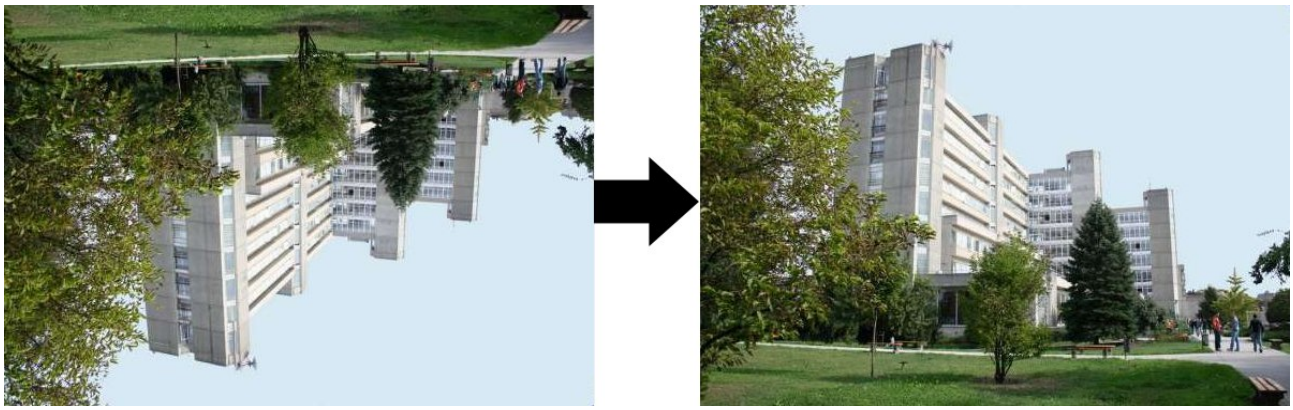
//(FONTOS! A beolvasott kép jelen esetben a projekt Debug könyvtárában kell helyet foglaljon!)

A ._SmoothGaussian funkció a következő paramétereket tartalmazza:

<kép>._SmoothGaussian(szűrőmátrix szélessége,szűrőmátrix magassága,szigma1,szigma2);

Flip (átfordítás)

Az eljárás már ismerős lehet alapvető hétköznapi képfeldolgozás szempontjából. Például a kép fejjel lefelé készült, és szeretnénk megfordítani:



A megvalósításhoz szükséges kód:

```
using System;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.UI;

namespace tesztprojekt
{
class Program
    {
static void Main(string[] args)
        {
          ImageViewer kepnezo = new ImageViewer();    // képnéző ablak definiálása
          Image<Bgr, Byte> kep = new Image<Bgr, byte>("egyetemvertical.jpeg");  // kép definiálása
          Image<Bgr, Byte> flipped = kep.Flip(Emgu.CV.CvEnum.FLIP.VERTICAL); // flip vertikálisan
          kepnezo.Image = flipped;                    // az átfordított átadása képnézőnek
          kepnezo.ShowDialog(); //képnéző ablak megjelenítése a beolvasott képpel (x-re ablakzárás)
        }
    }
}
```

//(FONTOS! A beolvasott kép jelen esetben a projekt Debug könyvtárában kell helyet foglaljon!)

Természetesen az átfordítás horizontálisan is lehetséges: **Emgu.CV.CvEnum.FLIP.HORIZONTAL** utasítással.

Resize (átméretezés)

A színkonverzió utáni másik nagyon fontos művelet az átméretezés. Egy kép mérete nagyban befolyásolja a feldolgozási időt. Ha a keresett pixelterület nem részletgazdag, akkor a képet célszerű átméretezni.

A megvalósításhoz szükséges kód:

```
using System;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.UI;

namespace tesztprojekt
{
class Program
    {
static void Main(string[] args)
        {
          ImageViewer kepnezo = new ImageViewer();    // képnéző ablak definiálása
          Image<Bgr, Byte> kep = new Image<Bgr, byte>("egyetem.jpeg");          // kép definiálása
          Image<Bgr, Byte> meretezett = kep.Resize(0.5,Emgu.CV.CvEnum.INTER.CV_INTER_AREA);
          kepnezo.Image = meretezett;   // az átméretezett verzió átadása képnézőnek
          kepnezo.ShowDialog(); //képnéző ablak megjelenítése a beolvasott képpel (x-re ablakzárás)
        }
    }
}
```

//(FONTOS! A beolvasott kép jelen esetben a projekt Debug könyvtárában kell helyet foglaljon!)

Leírás:

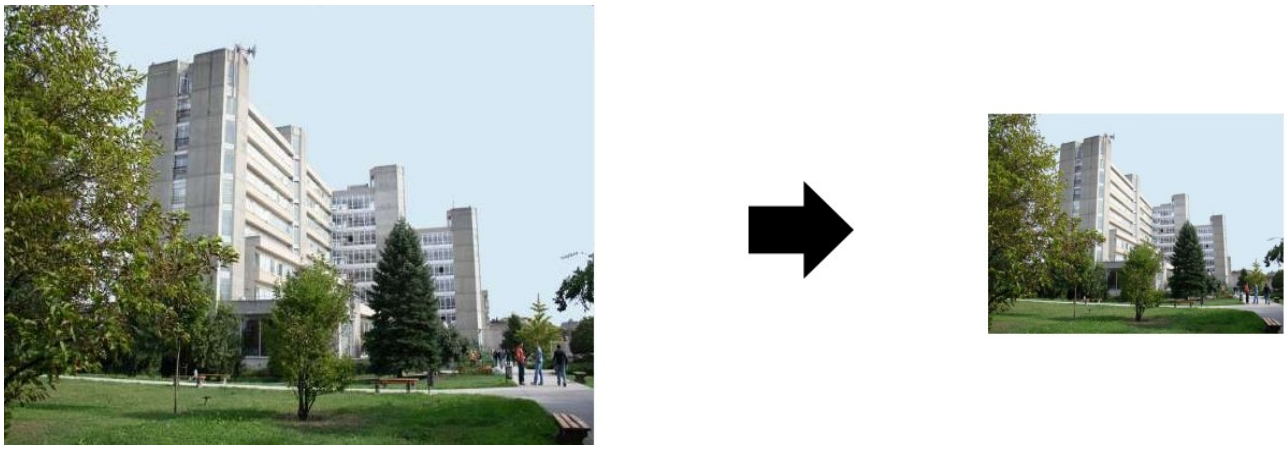
<kép>.Resize(<méretarány>, Emgu.CV.CvEnum.INTER.<interpoláció>);

Az interpolációs lehetőségek:

- CV_INTER_NN : Legközelebbi szomszéd
- CV_INTER_LINEAR: Lineáris
- CV_INTER_AREA: Pixel terület újramintavételezése
- CV_INTER_CUBIC: Köbös

A képinterpoláció lényege, hogy egy pixel színének és intenzitásának a lehető legjobb közelítését érje el, mégpedig úgy, hogy figyelembe veszi a környező cellák tulajdonságait. Minél többet tudunk ezekről a cellákról, az interpoláció annál pontosabb lesz. Ebből következik, hogy minél jobban kinyújtunk egy képet annál nagyobb mértékben romlik a minősége.

A kód által létrejött kép:

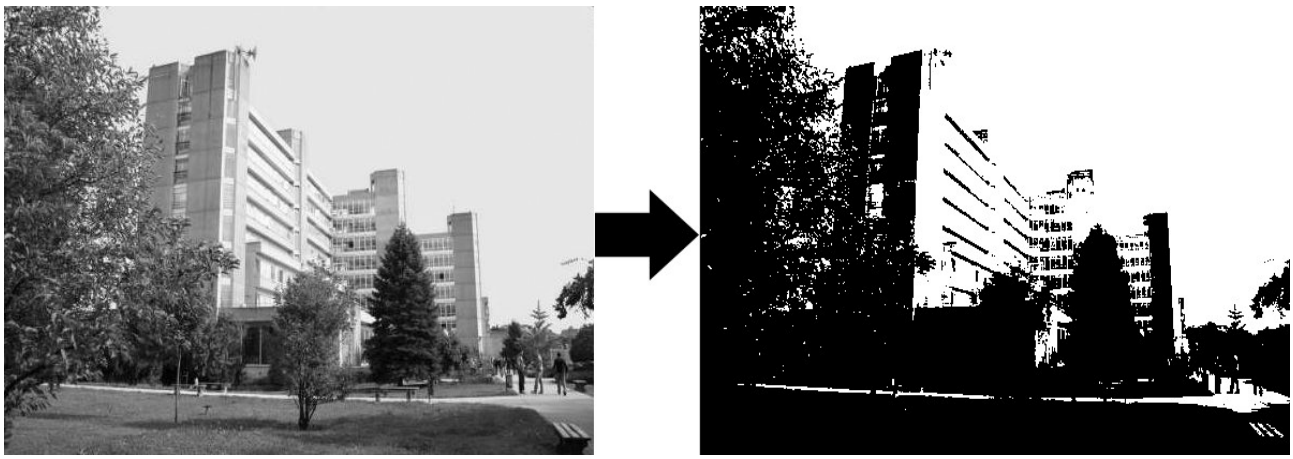


Adott esetben már a bemeneten célszerű megválasztani a megfelelő méretezést, így a kódsor mennyisége csökkenthető.

```
using....

namespace tesztprojekt
{
class Program
    {
static void Main(string[] args)
          {
            ImageViewer kepnezo = new ImageViewer();    // képnéző ablak definiálása
            Image<Bgr, Byte> kep = new Image<Bgr, byte>("egyetem.jpeg").Resize(0.5,
            Emgu.CV.CvEnum.INTER.CV_INTER_AREA);;    // kép definiálása
            kepnezo.Image = kep;  // az átméretezett verzió átadása képnézőnek
            kepnezo.ShowDialog(); //képnéző ablak megjelenítése a beolvasott képpel (x-re ablakzárás)
          }
     }
 }
//(FONTOS! A beolvasott kép jelen esetben a projekt Debug könyvtárában kell helyet foglaljon!)
```

Threshold

Ez a funkció kimondottan arra szolgál, hogy egy kétállapotú fekete-fehér verziót kapjunk a részletesebb szintén fekete-fehér képből.



Ezzel a módszerrel egy lépéssel közeledtünk a szegmentáció területéhez, azonban itt még képmanipulációs példákat mutatok be, amelyek a felhasználást tekintve hasznosak lehetnek.

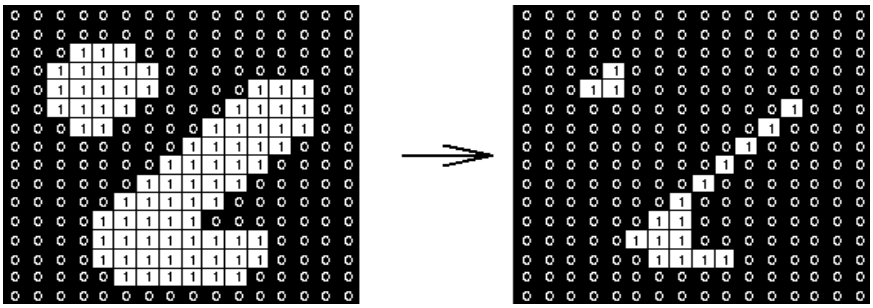
A megvalósításhoz szükséges kód:

```
using System;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.UI;

namespace tesztprojekt
{
class Program
    {
static void Main(string[] args)
        {
          ImageViewer kepnezo = new ImageViewer(); // képnéző ablak definiálása
          Image<Bgr, Byte> kep = new Image<Bgr, byte>("egyetem.jpeg");        // kép definiálása
          Image<Gray, Byte> feketefeher = kep.Convert<Gray, Byte>();          // kép konverzió
          feketefeher._ThresholdBinary(new Gray(150), new Gray(255));      // threshold beállítás
          kepnezo.Image = feketefeher;
          kepnezo.ShowDialog(); //képnéző ablak megjelenítése a módosult képpel (x-re ablakzárás)
        }
    }
}
//(FONTOS! A beolvasott kép jelen esetben a projekt Debug könyvtárában kell helyet foglaljon!)
```

Szegmentációs lehetőségek

Erode (Hámozás)

Az erózió egy nemlineáris morfológiai operátor. A módszer alapja a foltkeresés (főleg a képhibák miatt), ám sok esetben ezek a foltok nem egyeznek, illetve nem közelítenek azokhoz az alakzatokhoz, vagy mintákhoz, amelyeket keresünk. Sok esetben az egymást takaró objektumok foltjai részben elfedik egymást. Ezeket a nagyobb foltokat ezzel az eljárással bonthatjuk szét. A módszer neve onnan ered, hogy lépésenként leválasztjuk a külső rétegeket, ameddig már a további csökkentés nem érthető el.



A funkció leírása:

képi_változó.Erode(x);

ahol az Erózió egy 3x3-as térstrukturáló elemmel történik meg, és az iterációk száma = x.

A fenti képeken az erózió folyamata látható:

1: Beolvasott fekete-fehér kép

2: Bináris kép (threshold használatával)

3: Erodált kép

Színszegmentáció:

Az imént említett eljárás önmagában nem használható, ha a számunkra vizsgált objektumoknál nem szegmentálunk valamilyen tulajdonság alapján. Az egyik ilyen lehetőség a **szín** szerinti szűrés. (Sok esetben a kamera egy fix helyzetben rögzíti a képeket. Ez alapján az előtér, és a háttér is kiválóan szegmentálható képkockakülönbség szerint, ha az előtérben aktív cselekmény zajlik.)

HSV szétbontás

Adott egy Bgr kép, amelyen egy bizonyos színt szeretnénk szegmentálni. Ehhez szét kell bontani a képünket HSV csatornákra (HUE-SATURATION-VALUE).

A megvalósításhoz szükséges kód:

```
using System;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.UI;

namespace tesztprojekt
{
class Program
    {
static void Main(string[] args)
        {
          ImageViewer kepnezo = new ImageViewer(); // képnéző ablak definiálása
          Image<Bgr, Byte> kep = new Image<Bgr, byte>("egyetem.jpeg");        // kép definiálása

          Image<Hsv, Byte> hsvimg = kep.Convert<Hsv,Byte>();   // HSV konverzió

          Image<Gray, Byte>[] channels = hsvimg.Split();           // HSV Split
          Image<Gray, Byte> imghue = channels[0];       // HUE csatorna (szín)
          Image<Gray, Byte> imgsat = channels[1];       // SATURATION csatorna (telítettség)
          Image<Gray, Byte> imgval = channels[2];       // VALUE csatorna (világosság)

          kepnezo.Image = imghue;
          kepnezo.ShowDialog(); //képnéző ablak megjelenítése a módosult képpel (x-re ablakzárás)
        }
    }
}
//(FONTOS! A beolvasott kép jelen esetben a projekt Debug könyvtárában kell helyet foglaljon!)
```

Ez azonban így még kevés. Szükséges megadni az egyes csatornáknál azt a tartományt, amit szeretnénk a szűrőn átengedni. Ezt a következő módon tehetjük meg:

A megvalósításhoz szükséges kód:

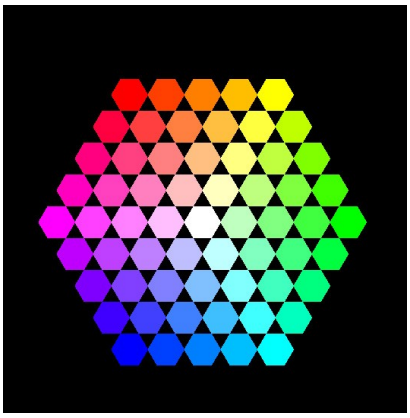
```
using System;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.UI;

namespace tesztprojekt
{
class Program
    {
static void Main(string[] args)
        {
          ImageViewer kepnezo = new ImageViewer(); // képnéző ablak definiálása
          Image<Bgr, Byte> kep = new Image<Bgr, byte>("egyetem.jpeg");          // kép definiálása
          Image<Hsv, Byte> hsvimg = kep.Convert<Hsv,Byte>();   // HSV konverzió
          Image<Gray, Byte>[] channels = hsvimg.Split();             // HSV Split
          Image<Gray, Byte> imghue = channels[0];        // HUE csatorna (szín)
          Image<Gray, Byte> imgsat = channels[1];        // SATURATION csatorna (telítettség)
          Image<Gray, Byte> imgval = channels[2];        // VALUE csatorna (világosság)

           Image<Gray, byte> huefilter = imghue.InRange(new Gray(Xa), new Gray(Xf));
                 // HUE tartomány
           Image<Gray, byte> satfilter = imgsat.InRange(new Gray(Ya), new Gray(Yf));
                 // SAT tartomány
           Image<Gray, byte> valfilter = imgval.InRange(new Gray(Za), new Gray(Zf));
                 // VALUE tartomány

           Image<Gray, byte> szumma = huefilter.And(satfilter).And(valfilter);  //összefűzés
           kepnezo.Image = szumma;
           kepnezo.ShowDialog(); //képnéző ablak megjelenítése a módosult képpel (x-re ablakzárás)
        }
    }
}
//(FONTOS! A beolvasott kép jelen esetben a projekt Debug könyvtárában kell helyet foglaljon!)
```

A tartomány jelen esetben egy 0-tól 255-ig terjedő skálán adható meg (kivéve a HUE), ahol:

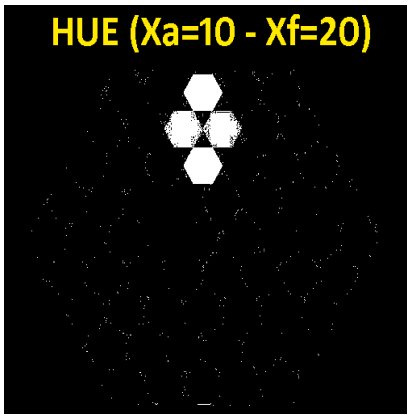
- SZÍNSKÁLA (0-179) esetében: vörös → sárga → zöld → cián → kék → lila → vörös
- TELÍTETTSÉG esetében: telítetlen (0) → telített (255)
- VILÁGOSSÁG esetében: sötét (0) → világos (255)

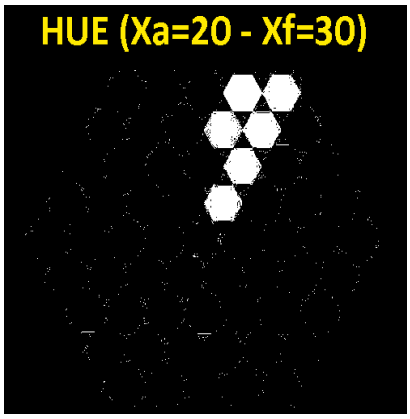
HSV szegmentálás példák:

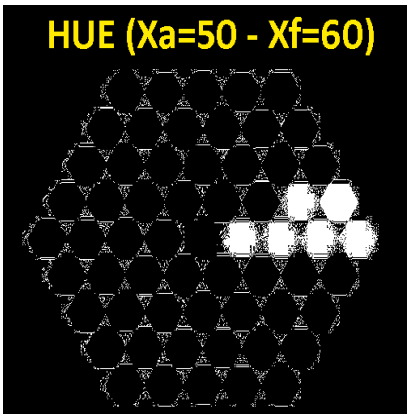
Vizsgáljuk meg alaposabban a szín szegmentációs működést EMGU CV-ben. A felhasznált kód az előző oldalon lelhető fel, azonban csak a szűrési paramétereket változtatjuk. A telítettség és világossági tényezőhöz egyelőre nem nyúlunk, azok kezdeti paraméterei (Ya,Za) 0 értékűek, a maximumok (Yf,Zf) pedig 255 értéket vesznek fel. Amit változtatni fogunk az a HUE paraméter. Definiáljunk, és töltsünk be egy színskála képet az eddigiekhez hasonlóan, majd a paraméterek változtatása után futtassuk le az imént említett kódot.

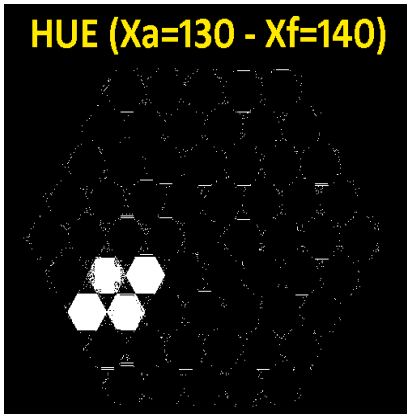


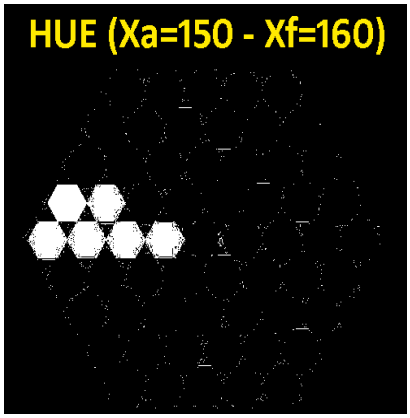
Az alábbi képeken remekül kivehető a paraméter változtatással járó eltolódás. Természetesen a szűrésen pontosíthatunk szűkebb intervallum megadással, illetve telítettségi tartomány megadásával. Azonban ezt akár meg is fordíthatjuk: Például szeretnénk a kék szín kivételével mindent átengedni. Ebben az esetben rászűrünk a kékre, majd az eredményként kapott bináris képet invertáljuk.

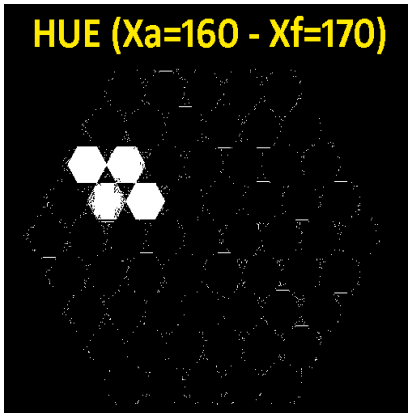












Detektálási módszerek

Abban az esetben ha sikerült megfelelően szegmentálni a az elemet, akkor ezt követően nyilván kíváncsiak vagyunk arra, hogy ez az elem a képen hol helyezkedik el, illetve egyáltalán a képen fellelhető-e, illetve az általunk vizsgált elem tulajdonságát is leírhatjuk. Ezeket a módszereket nem fogom részletes kódszinten bemutatni, pusztán annyit, amennyi a projekt szempontjából lényeges.

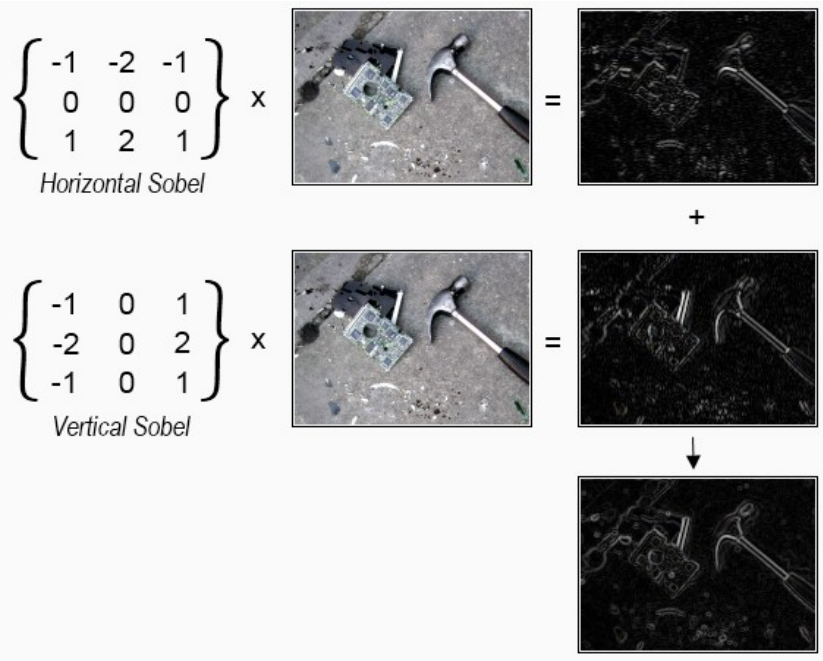
Detektálási módszerek közül néhányat említve:

- éldetektálás
- sarokdetektálás
- foltdetektálás
- Hough transzformáció
- SIFT
- SURF

Éldetektálás

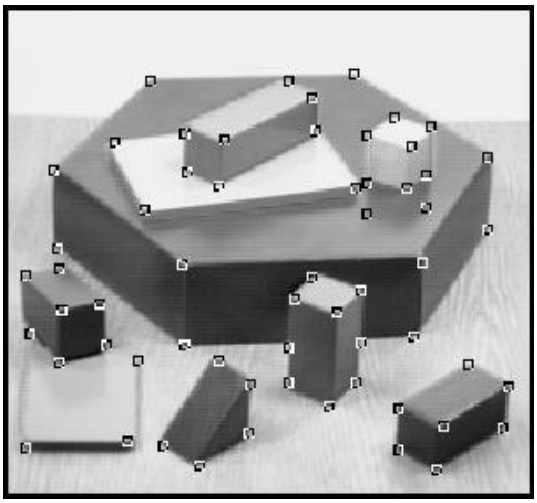
Az éldetektálás célja, hogy a képen megtaláljuk azokat a pontokat, ahol a világossági értékekben éles változás történik, diszkontinuitások optikai felfedezésében. Ennél a detektáló műveletnél fontos, hogy a kép lehetőleg zajtalan legyen, mivel a képre rakodó zaj ebben az esetben téves eredményt hozhat. Az OPEN CV és EMGU CV által alkalmazott leggyakoribb éldetektáló eljárás a **Canny**. Természetesen léteznek ezen kívül más hasonló éldetektáló algoritmusok is, de ezeket a terjedelem miatt nem részletezem.

Sobel operátor



A Sobel operátor egy diszkrét differenciáló operátor, amely az éldetektálási folyamat során van hasznunkra. Egy 3x3-as gradiens maszk, amely kevésbé zajérzékeny (a nagyobb maszkméret eltünteti a pontszerű zajt). Hátránya, hogy a meredek élek vastagabban jelentkeznek, így élvékonyítás szükséges.

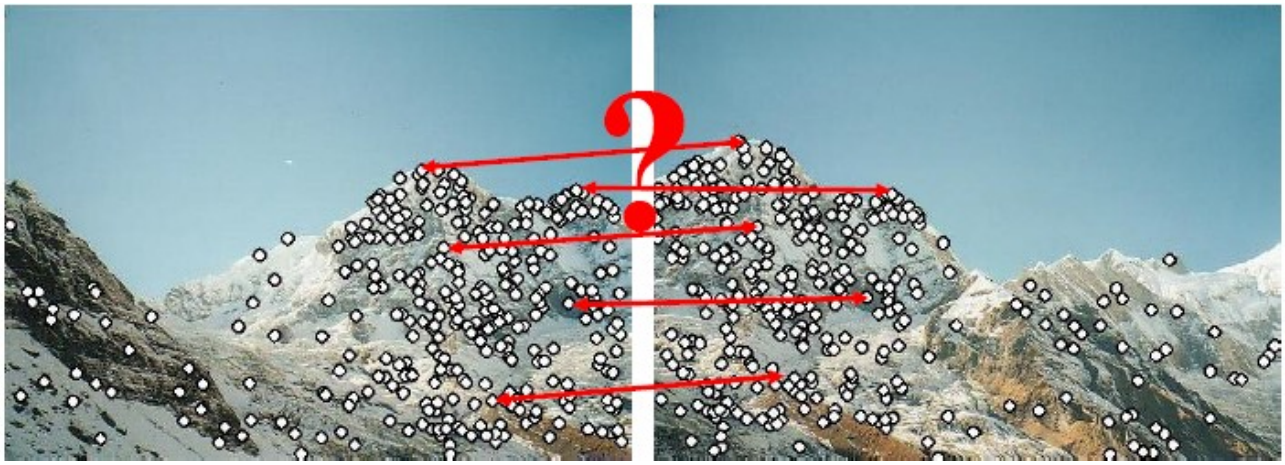
Sarokdetektálás



Egy sarokpont definiálható, mint 2 él találkozási pontja. A felismerés során ezen sarokpontok pixelkoordinátái eltárolhatók, amely alapján következtethetünk a sarkok képen történő elhelyezkedésére. Sőt ezek a pontok akár össze is köthetőek, így egy 3D kép kiválóan restaurálható ennek az algoritmusnak a segítségével. Példaként említhető a Harris-operátor, vagy a FAST eljárás (Features from Accelerated Segment Test).

Foltdetektálás (Blob detection)

Az eljárás célja, hogy összefüggő régiókat, és ezekhez tartozó pontokat keressen a szegmentált képen belül. Felhasználásának egyik legszembetűnőbb példája a panorámaképek összeillesztése. Az illesztésre szánt képpárokon megkeressük a közös pontokat, amelyek lehetnek például sarokdetektálásból adódó kulcspontok, ezeket felismertetjük, és a két képet összeillesztjük.



A kinyert pontok önmagukban nem jellemezhetők jól, ezért mindkét képet normalizálnunk kell. Erre azért van szükség, mert azonos beállítások esetén is lehet a fényviszonyok stb. miatt eltérő a két kép intenzitása, és így az ablakokból képzett vektorok egységvektorok lesznek, így eltérő képeket vizsgálva is összehasonlíthatóak.

<u>Hasonlósági mértékek:</u>

– SSD: (Normalizált) Négyzetkülönbségek összege

$$C_{\text{SSD}} = \sum_{(u,v)\in W_m} [\hat{I}_1(u,v) - \hat{I}_2(u,v)]^2 = \|w_1 - w_2\|^2$$

Minél kisebb az értéke, annál jobban illeszkedik a két vektor

– NC: Normalizált korreláció

$$C_{\text{NC}} = \sum_{(u,v)\in W_m} \hat{I}_1(u,v)\hat{I}_2(u,v) = w_1 \cdot w_2 = \cos\theta$$

Minél nagyobb az értéke, annál jobban illeszkedik a két vektor

Hough-transzformáció

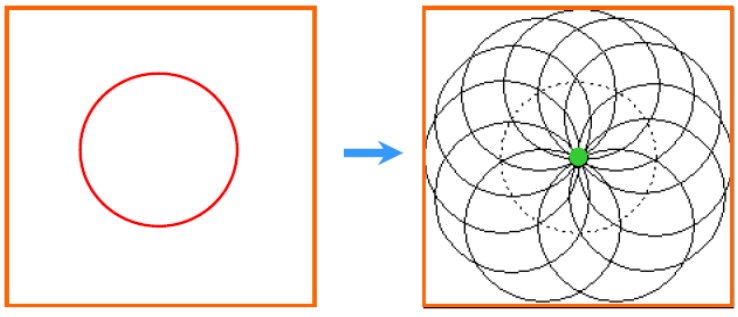
A Hough-transzformáció a digitális képfeldolgozás során egy adott kép pontjainak olyan részhalmazát határozza meg, amelyekre közös egyenes illeszthető. A művelet végrehajtása után, ha az (a;b) koordinátájú cella tartalma k (egyenesek száma), akkor az (a;b) paraméterű egyenesre az F halmaz k pontja illeszkedik (ahol az F halmaz az adott pillanatban rögzített bináris kép pontjainak halmaza). Ez a transzformációs forma kiválóan alkalmazható egyszerű alakzatok, egyenesek, körök kereséséhez, például szemmozgás figyelése, vagy autóvezetésnél a sávelhagyás ellenőrzése, vagy ideális esetben egy gyalogátkelő felismerése.

Körvonal detektálása

Általános körök esetén az (*a,b,r*) Hough-tér 3-dimenziós lesz:

$$ax2+by2=r2 \rightarrow f\,(x,y,a,b,r)=ax2+by2-r2=0$$

Amennyiben pl. adott (konstans) *r*-sugarú kört keresünk, akkor a paraméter-tér kétdimenziósra csökken – a gyakorlatban ez használatos. A kétdimenziós Hough-térben minden élpontnak egy potenciális középpontokat tartalmazó kör felel meg.



SIFT – Scale invariant feature transform

A SIFT algoritmus térbeli Gauss szűrők (aluláteresztő, a képet simító szűrő) sorozatával, és a szegmentált képek különbségének kalkulációjával a bementeti kép olyan transzformáltjait hozza létre, melyeken a foltszerű (blob) képrészletek hatékonyan megtalálhatók. Hátránya, hogy nagyon erős előfeldolgozó, és ablakozó processzortömböket igényel. Ez a módszer további gyorsabb módszerek alapjául szolgál, mint például a SURF. Futási ideje lassabb, de példaprogramokhoz és prototípusokhoz megfelelő. A szűrő σ=2 paraméter melletti futtatásával az effektív képfelbontás felére csökken. Ahhoz, hogy optimális detekciót érjünk el, szükség van a kép dupla méretű reprezentációjára, és oktávonként három további finom felosztásra. A kétdimenziós processzortömbön hatékonyan kiszámítható a Gauss szűrőknek megfelelő diffúzió, de a tömb méretének a felbontás kétszeresére történő emelése legalább négyszeres hardver komplexitást igényelne, míg az oktávonkénti finom felbontások lineáris növekedést jelentenek.

A feldolgozás folyamata

Első lépés:

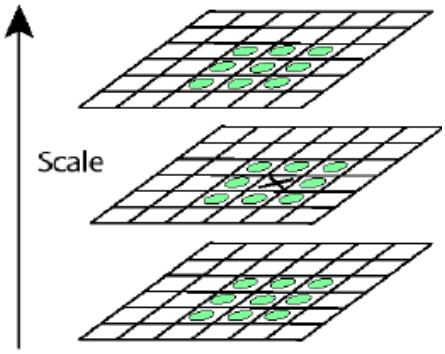
A Gauss-féle konvolúciós szűrővel hajtunk végre **simítást**. Ezt többször is elvégezzük, a kép méretét mindig felére csökkentve. Így egy képpiramis keletkezik, mely a feldolgozási sebességet növeli. Majd a szomszédos konvolúciós szintek különbségét képezzük, így egy különbség skála szintjei jönnek létre. Ennek leírására használjuk a D(x, y, б) függvényt :

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$
$$= L(x, y, k\sigma) - L(x, y, \sigma).$$

ahol $*$ a konvolúciós operátor, D(x, y, σ) a Gauss-féle konvolúciós maszk, I(x, y) a bemeneti kép, L a skálatér egy szintje, k pedig egy pozitív egész, mely a skálaszintet jelöli. σ az alkalmazott Gauss függvény szórását jelöli. Leegyszerűsítve D olyan különbség szinteket képez, melyben egyik szint éppen k-szor feljebb található a skála-térben, mint a kivonandó másik.

A második lépés:

Szélsőértékeket kell keresni a D által létrehozott Gauss-különbség térben. A kulcspontok a lokális maximum, illetve minimum értékek lesznek, melyek az aktuális pixel saját szintjén elhelyezkedő nyolcas környezetével, illetve a szomszédos szinteken lévő 18 másik szomszédjával való összehasonlításból számítandóak ki. Ez így összesen huszonhat



összehasonlítás. Ha ez a pont a többivel való összehasonlítások után minimális vagy maximális, akkor szélsőérték. Az így kapott pontokon további műveleteket kell végezni. Mivel többször is elmostuk a képeket és a méretüket is csökkentettük, ezért interpoláció segítségével a környező adatok alapján vissza kell keresnünk a kijelölt pontok eredeti helyét az eredeti képen.

Harmadik lépés: Kulcspontok számának csökkentése

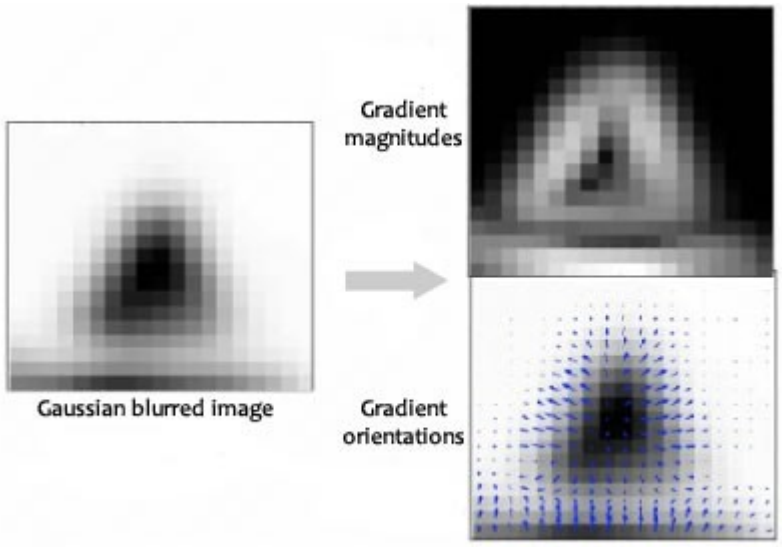
A hatékonyság növelése érdekében redukálni kell a kulcspontok számát, mivel ezek közül még nem mind hordoz fontosnak mondható információt. Erre azért van szükség, hogy csak a stabil pontok maradjanak meg a további szükséges kalkulációkhoz. Az alacsony kontrasztú pontokat és a gyenge élpontokat el kell távolítani. Ehhez ki kell számolnunk a Laplace operátorral számított értékét az adott pontnak. Ha a kontraszt érték egy megadott szint alatt van, akkor kivesszük a pontot a kulcspont listából. A stabilitáshoz azonban nem elégséges az alacsony kontrasztú pontok kiszűrése. A Gauss-különbség függvény megfelelő információkkal szolgál az élekről, de ha egy él elég gyenge, akkor a zajérzékenysége megnő. A további gyenge pontok szűréséhez meg kell vizsgálnunk, hogy van-e fő görbület az éllel párhuzamosan, vagy gyenge görbület a merőleges irányban, D-ben az adott helyen. A pontot el kell vetnünk, ha ez a különbség a legnagyobb és legkisebb sajátvektor aránya alatt van, amit 2×2-es Hesse-féle mátrixból számíthatunk ki az adott helyre illetve skálaszintre nézve.

Irányok meghatározása

A feldolgozás következő lépéseként a pontokhoz irányokat – akár többet is – kell rendelni, amelyeket a lokális gradiens jellemzők alapján határozunk meg. A gradiens m nagyságát és Φ irányát az alábbi módon határozhatjuk meg:

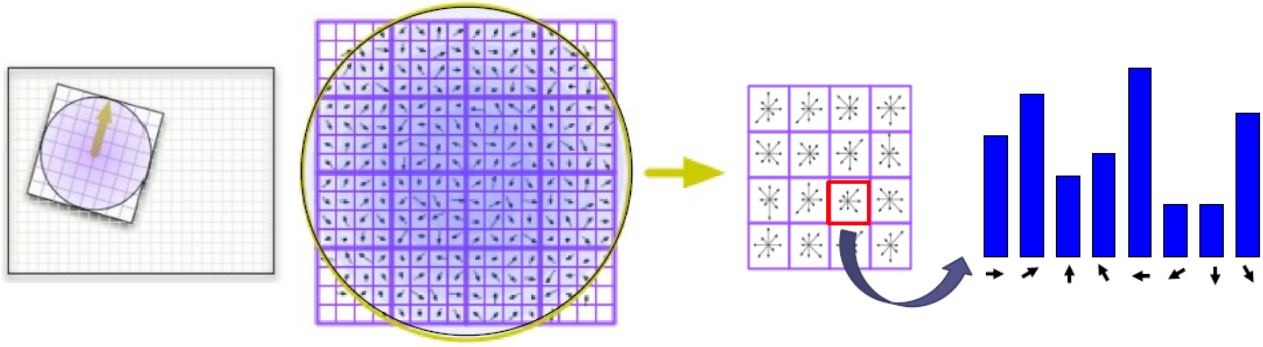
$$
\begin{aligned}
m(x,y) &= \sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2} \\
\theta(x,y) &= \operatorname{arctg}\frac{L(x,y+1)-L(x,y-1)}{L(x+1,y)-L(x-1,y)}
\end{aligned}
$$

Ezt elvégezzük az adott kulcspont adott sugarú (négy vagy nyolc) környezetére is, majd ezeket az értékeket egy olyan σ paraméterű Gauss-maszk szerint súlyozzuk, amely rendszerint másfélszerese a kulcspont léptékének.



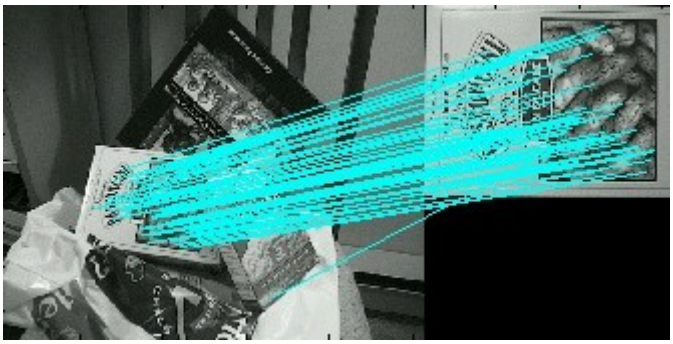
Ezt követően készítünk egy szöghisztogramot 36 csoporttal (10 fokonként 1 db). Az így létrejött régiókat összegezzük 4×4-es felbontású blokkokra úgy, hogy közben csökkentjük a csoportok számát nyolcra. Nagyon fontos, hogy az egyes kulcspontokhoz tartozó hisztogramok a ponthoz előzőleg kiszámolt orientációhoz vannak igazítva.

A domináns irányok a hisztogram kiemelkedő értékei lesznek. Alapvetően a legnagyobb ilyen csúcsot tekintjük fő iránynak, de ha előfordul még olyan érték, mely a maximális érték 80%-án belül van, akkor létre kell hozni vele egy új kulcspontot ugyanazon a helyen.



Általánosságban a pontok 15%-a rendelkezik többirányú hozzárendeléssel, ezzel csak növelik az ilyen kulcspontok stabilitását. A leíró végül úgy jön létre, hogy minden kulcsponthoz és környezetéhez tartozik 4×4 darab hisztogram egyenként 8-8 értékkel. Ez összesen egy 128 elemű vektort eredményez. Ha a megvilágításból adódó változásokat szeretnénk kiküszöbölni, akkor normalizálnunk kell a vektort, így elérhetünk bizonyos fokú invarianciát e változásokkal szemben.

Gyakorlati alkalmazását tekintve (összegzésképp) egy olyan műveletről van szó, amely segítségével egy mintát, és annak pozícióját a kameraképen detektálni lehet. Az alábbi kép ezt prezentálja:



SURF (Speeded Up Robust Feature)

A SURF (2006 – Herbert Bay) detektor megalkotását részben az imént említett SIFT leíró inspirálta. Ez a módszer sokkal gyorsabb és hatékonyabb az elődjénél, miközben számítási igénye jóval alacsonyabb.

A SURF funkció leírása:

A képek integráltját használja fel a konvolúciós lépés során, építve az eddigi módszerekre és egyszerűsíti azokat. Invariáns az objektum forgatására és átméretezésére. Kulcspont keresésre a DoH (Hesse-determináns) megoldást használja. Az kulcspontok környezetének leírására egy lokális képrészlet wavelet együtthatóit számítja ki a Haar waveletet alkalmazva. A hasonló leírók keresésére egy kontrasztot jellemző indexet használ, amellyel nagyban gyorsítható a keresés.

EMGU CV megvalósítás és teszt:

Az EMGU CV tesztünknél egy gyalogátkelő mintaképet fogok alkalmazni, és ezt fogom megkeresni a beolvasott képen a SURF detektor alkalmazásával. Találat esetén a konzolon a „Találat” felirat, ha nincs, akkor a „Nincs találat” felirat jelenik meg.

Mivel ez már egy bonyolultabb funkció több kimeneti lehetőséggel, így az olvasó számára a lefejlesztés, és kódsor bemutatása előtt néhány tesztesetet mutatnék be, amely a SURF detektálás működését prezentálja. A tesztesetekben feltüntetett fájlokat természetesen az eddig megszokott módon a projekt /bin/Debug könyvtárában kell elhelyezni, illetve ezek a fájlok a szakdolgozathoz csatolandó adathordozóra is felkerülnek.

A tesztesetekben feltüntetett képhivatkozásokat a kódban a megfelelő helyre kell beilleszteni!

Tesztesetek:

Case001: **TALÁLAT**

- A minta 150x150-es gyalogátkelő tábla
- A beolvasott kép tartalmaz táblát hasonló méretben

Case002: **NINCS TALÁLAT**

- A minta 150x150-es gyalogátkelő tábla
- A beolvasott kép NEM tartalmaz gyalogátkelő táblát

Case003: **NINCS TALÁLAT**

- A minta 150x150-es gyalogátkelő tábla
- A beolvasott kép tartalmaz táblát jóval kisebb méretben (~50x50)

Case004: **TALÁLAT**

- A minta 150x150-es gyalogátkelő tábla
- A beolvasott kép tartalmaz táblát jóval nagyobb méretben (~400x400)

Case005: **TALÁLAT**

- A minta 75x75-ös gyalogátkelő tábla
- A beolvasott kép tartalmaz táblát hasonló méretben

Case006: **NINCS TALÁLAT**

- A minta 75x75-ös gyalogátkelő tábla
- A beolvasott kép tartalmaz táblát jóval nagyobb méretben (~400x400)

Case007: **TALÁLAT**

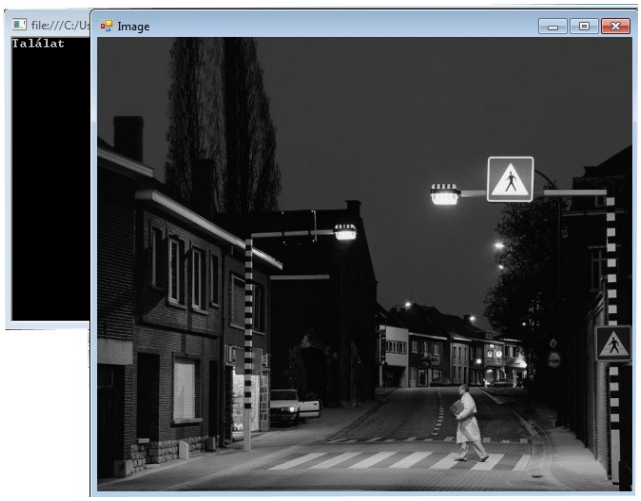
- A minta 100x100-as gyalogátkelő tábla
- A beolvasott kép több táblát tartalmaz hasonló méretben

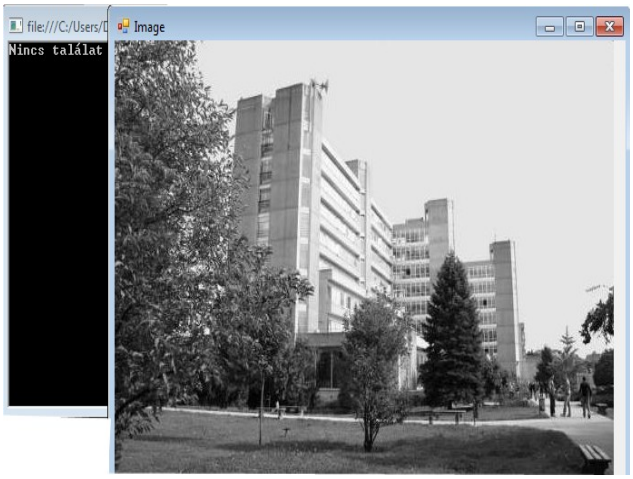
Case008: **NINCS TALÁLAT**

- A minta NEM tartalmaz gyalogátkelő táblát
- A beolvasott kép tartalmaz táblát ~ 100x100 pixel méretben

Következtetés a tesztesetek eredményeiből:

A tesztesetek eredményéből következik, hogy **az alapműködés rendben van**, azonban a mintaelem méreteit pontosan kell meghatározni. A tesztekből kiderül, hogy nagyméretű minta kisebb méretű társát nem találja meg (Case003). Ugyanúgy igaz a túl kicsi minta nagyméretű társára is (Case006). Ezen a ponton meg kell jegyezni, hogy egy élő környezetben történő felvételnél a kamerát kalibrálni kell, vagy olyan mintaelemet kell választani, amely a szoftver működési követelményének eleget tesz.





A fenti tesztképekből az egyes tesztesetek eredményei láthatóak, továbbá az éjszakai működés pontossága is megfigyelhető.

A megvalósításhoz szükséges kód:

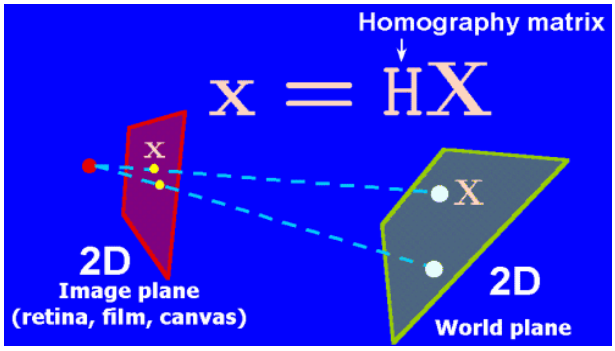
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.UI;
using Emgu.CV.Features2D;

namespace teszt001
{
  class MainClass
  {
    public static void Main(string[] args)
    {
      Emgu.CV.UI.ImageViewer viewer1 = new Emgu.CV.UI.ImageViewer();      //képnéző definiálás
      MCvSURFParams surfParam = new MCvSURFParams(500, false);            //SURF paraméterek definiálása
      SURFDetector surfDetector = new SURFDetector(surfParam);            //SURF detektor definiálása
      Image<Bgr, Byte> minta = new Image<Bgr, byte>("MINTAKÉP.jpg");      //mintakép definiálása
      Image<Gray, Byte> modelImage = minta.Convert<Gray, Byte>();         //mintakép konverzió
      //a mintakép jellemzőinek a "kibontása"
      ImageFeature[] modelFeatures = surfDetector.DetectFeatures(modelImage, null); //jellemzők detektálása
      Features2DTracker tracker = new Features2DTracker(modelFeatures); //SURF tracker létrehozása
      Image<Bgr, Byte> frame = new Image<Bgr, byte>("VIZSGÁLANDÓ.jpg"); // vizsgálandó kép definiálása
      Image<Gray, Byte> grayimg2 = frame.Convert<Gray, Byte>();           // vizsgálandó kép konverzió
      ///a vizsgálandó kép jellemzőinek a "kibontása"
      ImageFeature[] imageFeatures = surfDetector.DetectFeatures(grayimg2, null);     //jellemzők detektálása
      Features2DTracker.MatchedImageFeature[] matchedFeatures = tracker.MatchFeature(imageFeatures, 2);
       //jellemzők összevetése
      matchedFeatures = Features2DTracker.VoteForUniqueness(matchedFeatures, 0.8); //Egyediségvizsgálat
      matchedFeatures = Features2DTracker.VoteForSizeAndOrientation(matchedFeatures, 1.5, 20);
     //Filterezés – skála és rotáció megfeleltetés
      HomographyMatrix homography =
        Features2DTracker.GetHomographyMatrixFromMatchedFeatures(matchedFeatures);
      //Homográfia – Egyezés?
      if (homography != null)
      {
        Console.WriteLine("Találat");  //Homográfia – van visszatérő értéke = TALÁLAT
      }
      else
      {
        Console.WriteLine("Nincs találat"); //Homográfia – nincs visszatérő értéke = NINCS TALÁLAT
      }
      viewer1.Image = grayimg2; //kép betöltése képnézőbe
      viewer1.ShowDialog(); //kép megjelenítése
    }
  }
}
```

Homográfia és a RANSAC algoritmus

A 2D homográfia matematikailag megfogalmazva:

Adott xi ponthalmaz, és egy hozzátartozó xi’ ponthalmaz P2 térben. Határozzuk meg azt a projektív transzformációt, amely minden xi pontot a neki megfelelő xi’ -be visz át! Adott xi, xi’ pontok esetén keressük azt a H mátrixot, melyre xi’ = Hxi.

A sík egy pontja és képe között egy 3x3-as H (homography) mátrixban definiálva a kapcsolatot, lineáris egyenletrendszer megoldásával közelíthető a belső paraméterek mátrixa, majd az eredmény és H segítségével a külsőké. A síkról készített több kép segítségével finomul a megoldást.



A RANSAC egy általános metódus arra, hogy megbecsüljük egy matematikai modell paramétereit egy olyan adathalmazból, amely oda nem illő adatokat is tartalmaz.

Lényege, hogy az adatcsoportból véletlenszerűen kiragadva annak egy kis részét, ezeket a modellhez tartozóknak tételezi fel, és segítségükkel megbecsüli a modell paramétereit. Ezután az összes többi pontról egyenként megállapítja, mennyire illik ebbe a becsült modellbe. Ha e folyamat végére elég sok beleillő pontot talált, akkor a becslést jó becslésnek tekinti, azt finomítja immár az összes beleillőnek vett pont alapján, majd kiszámolja ezek hibáját a finomított modellhez képest, és tárolja a finomított verzió paramétereit. Ezt a procedúrát párszor megismétli.

A módszerek megválasztása

Ezen a ponton minden olyan lehetséges ismeretünk megvan, ami alapján meg tudjuk tervezni a programunkat:

Bemeneti oldalon lesz egy *kamerakép*, amellyel beolvassuk a képkockákat. A kamerának nem fix háttere van, ezért a szegmentálás esetében fel kell tennünk a kérdést, hogy a való élet környezetében célszerű-e szín szerint szegmentálni? A gyalogátkelő (zebra) ami nem szabványos elem (különböző országokat tekintve), kophat, így ennek detektálása túlságosan bonyolult kódot generálna, lassabb lenne a feldolgozás, és további kérdés lenne a korrekt működés. Továbbá egy fekete-fehér csíkos ruházatú emberre is bejelezne a program, amely szintén megzavarhatja a látássérült felhasználót. Ami szabványos és minden átkelő „tartozéka" az a tábla. Ha a gyalogos az úttal párhuzamosan közelíti meg, akkor a tábla látható, és a jelzés pontos lesz, sőt a minta nagysága/kamera kalibrálás alapján meg is adhatjuk a táblától való pontos távolságot. Ezeket a táblákat meg általában az átkelő előtt közvetlenül helyezik el. A színszegmentáció+foltdetektálás ebben az esetben sem egy működőképes lehetőség az ég és adott esetben a környezet más színezete miatt, ami szintén hibás működést generál. A *SURF detektor* azonban robusztus, és mivel a mintánk ugyanaz az alkalmazás során, így ez biztosan alkalmazható. A színszegmentációt ebben az esetben is elhagyhatjuk, mivel a SURF rendelkezik a célnak megfelelő tulajdonságokkal, és nem romlik az fps érték, és nem nő a feldolgozási idő. Homályosításra, intenzív zajszűrésre, erózióra jelen esetben nincs szükség.

Összefoglalva szükségünk lesz egy fekete-fehér kamerakép inputra, egy SURF detektorra egy zebra mintaképpel (ahogy azt a SURF prezentációnál láthattuk). Egyedül a felhasználó megfelelő tájékoztatása hiányzik a műveletből, amely lehet terminál kimenet, vagy valamilyen tájékoztató hanghatás. Erről a következő fejezetekben fog szó esni.

Hanghatások alkalmazása

Megvalósítás alapjai

A felhasználó tájékoztatása vizuálisan a terminálon megjelenő üzenet formájában is történhet, azonban ez a való életben nem figyelemfelkeltő, továbbá a látássérült személy esetében nem is használható. Így valamilyen formában egy hangfájl lejátszására lesz szükség.

Az alap MSDN library rendelkezik hangfájl lejátszó művelettel, amely alapvetően használható is lenne egy egyszerű hangfájl lejátszására, azonban kérdésessé válik, hogy ez mennyire teljesítené a projekt szempontjából támasztott követelményeket. A program jelenlegi fázisában úgy működik, hogy ha megtalálja a minta objektumot a vizsgálandó képen, akkor jelez. Egy látássérült személy esetén ez nem elegendő. Ő ezen a ponton annyit tud, hogy előtte valahol 10 méterre zebra található. Merre forduljon 10 méter után? Ezen a ponton vissza kell kanyarodnunk a SURF detektor témaköréhez. A program segítségével ugyanis nem csak azt lehet lekérdezni, hogy a minta elem fellelhető-e a vizsgálandó képen, hanem, hogy ez az objektum hol található a képen. Nyilván azt szeretnénk elérni, hogy ha a kamerakép közepétől számítva a talált objektum balra helyezkedik el, akkor a felhasználó 10 méter után balra, ha jobbra helyezkedik el, akkor jobbra forduljon, és mindezt természetesen hangjelzés alapján.

Objektum pozíciójának detektálása – ROI (region of interest)

A minta elemünknek van egy formája. Ennek vizsgálva a környezetét (ROI) meghatározhatjuk a sarokpontjait, ha köré egy téglalapot rajzolunk. A homográfia során ugye eddig csak azt vizsgáltuk, hogy visszatér-e valamilyen (nem nulla) eredménnyel, azonban ezeket a téglalap pontokat össze is vethetjük a homográfia adatokból adódó sarokpontokkal. Ha ismerjük ezeknek a pontoknak a koordinátáit, akkor le tudjuk írni, hogy a képen hol helyezkedik el az elemünk.

A leíró kód (A SURF detektor kód megfelelő helyére)

:

```
if (homography != null)
 {
    System.Drawing.Rectangle rect = modelImage.ROI;   //a mintakép környezetére téglalap rajz
    System.Drawing.PointF[] pts = new PointF[] {          //a téglalap sarokpontjainak a lekérdezése
                    new System.Drawing.PointF(rect.Left, rect.Bottom),      // bal alsó koordináta
                    new System.Drawing.PointF(rect.Right, rect.Bottom),     // jobb alsó koordináta
                    new System.Drawing.PointF(rect.Right, rect.Top),        // jobb felső koordináta
                    new System.Drawing.PointF(rect.Left, rect.Top)};        // bal felső koordináta
    homography.ProjectPoints(pts);
                                          //pontok megfeleltetése a homográfia eredményével
    for (int i = 0; i < pts.Length; i++)
       pts[i].Y += 0;

    grayimg2.DrawPolyline(Array.ConvertAll<System.Drawing.PointF, System.Drawing.Point>(pts,
            System.Drawing.Point.Round), true, new Gray(255.0), 5);
                                          // pontok összekötése a kimenetre szánt képen
```

:

Jelenleg a pts[] tömb tartalmazza az eredmény koordinátapontjait. Ezeket egy feltételes elágazással célszerű lekérdezni, és terminálon megjeleníteni (kód folytatása):

```
float kepkozep = (frame.Width) / 2;              // a kép közepének meghatározása
float mintakozep = (pts[0].X + pts[1].X) / 2;    // a minta horizontális közepének meghatározása
DateTime CurrTime = DateTime.Now;                //pontos idő lekérdezése

      if (mintakozep> kepkozep)  //ha a minta horizontális közepe a képközépen túl helyezkedik el
      {
         Console.WriteLine("Találat -J- {0:T} - PTS:{1}", CurrTime, pts[0]); //TALÁLAT (jobbra) + adatok
      }
      else
      {
         Console.WriteLine("Találat -B- {0:T} - PTS:{1} ", CurrTime,pts[0]); //TALÁLAT (balra) + adatok
      }
// ezt követően jön az else ág, illetve a képmegjelenítés
```

A programot lefuttatva a terminálon megjelennek a koordináták. Most már csak annyi a teendő, hogy meg kell találni a panorámázáshoz megfelelő eszközkészletet, majd a mintaközép változó értékével szinkronizálni.

Panorámázás megoldása (Naudio)

Miért is kell panorámázni? Természetesen a program lefejleszthető 2 beolvasott hanghatással („Zebra jobbra x méterre”, illetve „Zebra balra x méterre”), ahol nem szükséges a panorámázás, de ez a beolvasási ciklusok rövidsége miatt egy további ciklus befejlesztése lenne lényeges, hogy amíg a hangjelzés folyamatban van, addig ne kezdődjön el újabb. Továbbá túl zajos környezetben nehezen értelmezhető. Ezzel szemben pontosabb információt adhat egy panorámázott „pittyegés”, amelyből a tábla pozíciója lépésről-lépésre értelmezhető. Ha a hanghatás eltolódott az egyik irányba, majd megszűnt, az annyit jelent, hogy a tábla már kiúszott a kameraképből, és a felhasználó 5-10 lépés közelségbe került a gyalogátkelőhöz, és a hang-eltolódás irányába kell ezt követően fordulnia. Természetesen a felhasználáshoz pontos minta, és ha lehet, kamerakalibráció szükséges.

A panorámázás lefejlesztéséhez egy Naudio könyvtárcsomagot fogunk felhasználni, mivel az MSDN könyvtárcsomag csak a hang lejátszását támogatja, a panorámázást nem. Illetve létezik egy Microsoft XNA Framework csomag (XNA Game Studio keretén belül), amely jóval több lehetőséget nyújt az audio panorámázáson felül (hiába nem audio szoftverfejlesztés a könyvtárcsomag fő profilja), azonban ennek külön projekttípusai miatt bonyolódna a fejlesztés. Az Naudio ennek a funkciónak bőven megfelel, továbbá funkcióban gazdag csomagja széles kínálattal áll rendelkezésre audio szoftverek fejlesztése terén.

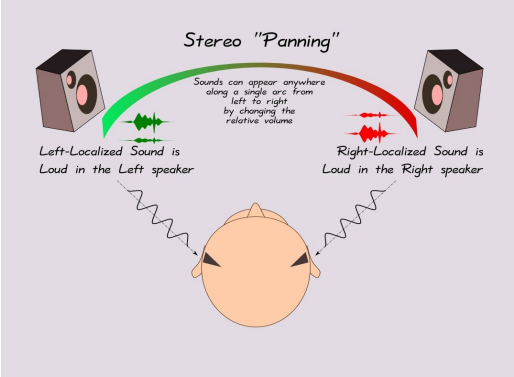
<u>Az Naudio csomaggal megvalósítható műveletek:</u>

- Audio lejátszása a következő kimeneteken: WaveOut, DirectSound, ASIO, WASAPI
- Audio kibontás a következő formátumokból: MP3, AIFF, G.711, G.722, SF2 stb.
- Hangrögzítés WaveIn, WASAPI, ASIO bemeneten
- WAV fileok írása, és olvasása
- Audio mixing, Full MIDI eseménymodell, MIDI eszköz fejlesztői támogatás
- Alap audio effektek fejlesztése, kompresszorok, stb.

Honnan tölthető le?

A könyvtárcsomag (release verzió) az http://naudio.codeplex.com/releases/ címen található egy zip fájlban, amiben Naudio XML és DLL fájlokat találunk. Ezeket mentsük ki, majd a DLL fájlokat az EMGU-s DLL-ekhez hasonlóan adjuk a projektünkhöz referenciaként.

A panorámázó működése



Először definiálnunk kell egy lejátszót, ami a hangfájlunkat fogja lejátszani (IWavePlayer), majd ezt követően szükség lesz egy WAV fájl beolvasóra (WaveFileReader). Tömörített formátumok esetén belső konverziót kell végrehajtani, ami feldolgozási időt ad a programnak, de jelen projekt esetében elegendő a WAV formátum (Nem zenei lejátszó, szerkesztő szoftvert írunk, csak egy panorámázót). Továbbá szükségünk lesz egy kimeneti csatornára (WaveChannel32), majd a hanglejátszóval megcélozzuk a DirectSound-ot (A DirectX programozási csatolófelület-készlet hanghatások generálását és a megjelenítéshez szinkronizált visszajátszását támogató része.), és inicializálunk. Definiálunk egy lebegőpontos pan változót (-1.0f [bal] – 0.0f [center] – 1.0f [jobb]), amelyet később a csatorna (WaveChannel32) Pan paramétere fog átvenni. Az inicializálást követően a betöltött hangfájlt az IwavePlayer lejátszó Play() funkciójával játszhatjuk le DirectSoundon keresztül, és a csatornán beállított panorámaérték függvényében változik a hangzás „pozíciója”. Ha a lejátszást ciklikusan akarjuk elkövetni, de nem loop formában, akkor a csatorna Position paraméterét a lejátszást követően 0-ba kell állítani. Ez annyit jelent, mintha a hang lejátszásnál egy csúszkával visszatekernénk a felvételt az elejére. Természetesen közben a paraméterértékek bármikor változtathatóak. Fontos, hogy a program végeztével a fájlbeolvasót, a lejátszót, és a létrehozott csatornát töröljük a memóriából. A hangfájlt adjuk hozzá a projekthez és az Output directoryhoz!

A panorámázó példa kódszinten:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using NAudio.Wave;
using NAudio.CoreAudioApi;

namespace teszt001
{
   class MainClass
   {
      public static void Main(string[] args)
      {
         Console.WriteLine("Program elindult");
         float pan = 0.0f;                                          //panorámázó lebegőpontos változó
         IWavePlayer waveOutDevice;                                 //hanglejátszó definiálása
         WaveFileReader hang = new WaveFileReader("sound.wav");     //hangfájl olvasó definiálása
         WaveChannel32 csatorna = new WaveChannel32(hang);          //hangcsatorna definiálása
         waveOutDevice = new DirectSoundOut();                      //DirectSound kimenet beállítás
         waveOutDevice.Init(csatorna);                              //hanglejátszó inicializálása
         pan = 0.0f;                                                //panorámázás beállítása középre
         csatorna.Pan = pan;                                        //panoráma érték beállítása csatornán
         Console.WriteLine("Most középről fogjuk hallani. Nyomj egy gombot a folytatáshoz!");     //konzol kiíratás
         Console.ReadKey();                                         //gombnyomásra várakozás
         waveOutDevice.Play();                                      //hanglejátszás
         hang.Position = 0;                                         //visszatekerés
         Console.WriteLine("A hang középen szólalt meg. Nyomj egy gombot a folytatáshoz!");      //konzol kiíratás
         Console.ReadKey();                                         //gombnyomásra várakozás
         pan = 1.0f;                                                //panorámázás beállítása jobbra
         csatorna.Pan = pan;                                        //panoráma érték beállítása csatornán
         Console.WriteLine("Most jobbról fogjuk hallani. Nyomj egy gombot a folytatáshoz!");       //konzol kiíratás
         Console.ReadKey();                                         //gombnyomásra várakozás
         waveOutDevice.Play();                                      //hanglejátszás
         hang.Position = 0;                                         //visszatekerés
         Console.WriteLine("A hang jobbról szólalt meg. Nyomj egy gombot a folytatáshoz!");       //...
         Console.ReadKey();
         pan = -1.0f;                                               //panorámázás beállítása jobbra
         csatorna.Pan = pan;                                        //panoráma érték beállítása csatornán
         Console.WriteLine("Most balról fogjuk hallani. Nyomj egy gombot a folytatáshoz!");
         Console.ReadKey();
         waveOutDevice.Play();
         hang.Position = 0;
         Console.WriteLine("A hang balról szólalt meg. A program gombnyomásra bezáródik");
         Console.ReadKey();
         waveOutDevice.Dispose();                                   //lejátszó bezárása
         hang.Dispose();                                            //fájlolvasó bezárása
         csatorna.Dispose();                                        //csatorna bezárása
      } } }
```

A végleges programkód

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Emgu.CV.Structure;
using Emgu.CV;
using Emgu.CV.Util;
using Emgu.CV.UI;
using Emgu.Util;
using Emgu.CV.CvEnum;
using Emgu.CV.Features2D;
using NAudio.Wave;
using NAudio.CoreAudioApi;

namespace teszt001
{
  class MainClass
  {
    public static void Main(string[] args)
    {
      Console.Clear();
      Console.WriteLine("Diplomamunka adatai:\n \n Készítette:\n Jakó Dénes E3NELW (Széchenyi István Egyetem 2012)\n \n Téma:\n Hanghatások alkalmazhatóságának vizsgálata\n navigációt segítő képfelismerő szoftver esetén\n \n Felhasználási terület:\n Gyalogátkelőhely tábla felismerés, vakok és gyengén látók számára\n");
      Console.WriteLine("\n ...\n");
      Console.WriteLine("A program elindult...");


      ImageViewer viewer = new ImageViewer(); //képnéző létrehozása
      viewer.Height = 480;
      viewer.Width = 640;
      Capture capture = new Capture(); //capture definiálás
      Image<Bgr, Byte> testcap = capture.QueryFrame();
      viewer.Image = testcap;
      Console.WriteLine("Capture inicializálás. Nyomjon egy gombot a folytatáshoz!");
      Console.ReadKey();

      //Ezen a ponton fontos a capture inicializáció, mivel idő kell a felálláshoz
      //Ha nem hagyunk elég időt, akkor a MatchedFeature résznél hibával megáll.

      MCvSURFParams surfParam = new MCvSURFParams(500, false);
      SURFDetector surfDetector = new SURFDetector(surfParam);

      int counter = 0;  //később zavarvédelem miatt
      float pan = 0.0f;


//A program a következő oldalon folytatódik...
```

//Program folytatása

```
        IWavePlayer waveOutDevice;
        WaveFileReader hang = new WaveFileReader("sound.wav"); // HANG
        WaveChannel32 csatorna = new WaveChannel32(hang);
        waveOutDevice = new DirectSoundOut();
        waveOutDevice.Init(csatorna);

        Console.WriteLine("Hangcsatornák beállítva...");

        Console.WriteLine("Képi beállítások...");

        Image<Bgr, Byte> minta = new Image<Bgr, byte>("minta.jpg"); //MINTA
        Image<Gray, Byte> minta2 = minta.Convert<Gray, Byte>();
        Image<Gray, Byte> modelImage = minta2;



        ImageFeature[] modelFeatures = surfDetector.DetectFeatures(modelImage, null);
        Features2DTracker tracker = new Features2DTracker(modelFeatures);

        Console.WriteLine("Mintaelem beállítva");

        Application.Idle += new EventHandler(delegate(object sender, EventArgs e)
        {

           Image<Bgr, Byte> frame = capture.QueryFrame();
           Image<Gray, Byte> frame2 = frame.Convert<Gray, Byte>();

           ImageFeature[] imageFeatures = surfDetector.DetectFeatures(frame2, null);
           Features2DTracker.MatchedImageFeature[] matchedFeatures = tracker.MatchFeature(imageFeatures, 2);

           matchedFeatures = Features2DTracker.VoteForUniqueness(matchedFeatures, 0.8);
           matchedFeatures = Features2DTracker.VoteForSizeAndOrientation(matchedFeatures, 1.5, 20);

           HomographyMatrix homography =
Features2DTracker.GetHomographyMatrixFromMatchedFeatures(matchedFeatures);

           if (homography != null)
           {
              System.Drawing.Rectangle rect = modelImage.ROI;
              System.Drawing.PointF[] pts = new PointF[] {
                     new System.Drawing.PointF(rect.Left, rect.Bottom),
                     new System.Drawing.PointF(rect.Right, rect.Bottom),
                     new System.Drawing.PointF(rect.Right, rect.Top),
                     new System.Drawing.PointF(rect.Left, rect.Top)};
              homography.ProjectPoints(pts);

              for (int i = 0; i < pts.Length; i++)
                 pts[i].Y += 0;
```

//A program a következő oldalon folytatódik...

```
//Program folytatása

            frame2.DrawPolyline(Array.ConvertAll<System.Drawing.PointF, System.Drawing.Point>(pts,
System.Drawing.Point.Round), true, new Gray(255.0), 5);

                counter++;

                float mintakozep = (pts[0].X + pts[1].X) / 2;
                float kepkozep = (frame2.Width) / 2;

                if (counter >= 3) //zavarvédelem: kizárólag 4 egymást követő matchelésnél keletkezik log a konzolon
                {
                   DateTime CurrTime = DateTime.Now;

                   if (mintakozep > kepkozep)
                   {
                      Console.WriteLine("Találat -J- {0:T} - PTS:{1} - PAN:{2}", CurrTime, pts[0], pan);
                      pan = (float)(0.5 * (mintakozep - kepkozep)) / kepkozep;
                   }
                   else
                   {
                      Console.WriteLine("Találat -B- {0:T} - PTS:{1} - PAN:{2}", CurrTime, pts[0], pan);
                      pan = (float)-0.5 * (1 - (mintakozep / kepkozep));
                   }

                   csatorna.Pan = pan;
                   waveOutDevice.Play();
                   hang.Position = 0;
                   counter = 0;
                }
             }
             else
             {
                counter = 0;
             }
             viewer.Image = frame2;
          });
          viewer.ShowDialog(); //képnéző mutatás
          Console.WriteLine("Video szekvencia megállítva");
          waveOutDevice.Dispose();
          hang.Dispose();

       }
    }
}
```

A kiadott programoknál az Application files alatt is láthatjuk, hogy mely fájlokat adtuk hozzá a projekthez. Az adathordozón mellékelt program esetében is visszaellenőrizhető.

A program ismert és javításra szoruló hibái (Known issues)

Természetesen minden kiadott program (minimálisan is) tartalmaz hibákat, nem várt működéseket. Ennek a megjavítására általában egy következő verzióban kerül sor, ahogy a szoftver életciklusa halad előre. A fent bemutatott program egy diplomamunka része, amely prezentálja a képfeldolgozás mai állását a technológiában, és bemutatja az audio panorámázási lehetőségeket elsősorban vakok és gyengén látók navigációja céljából. A program természetesen, mint minden szoftver életciklusban felmerülő igények, fejlesztő oldali ötletek szerint továbbfejleszthető, módosítható.

#1: An unhandled exception of type 'System.IndexOutOfRangeException' occurred in Emgu.CV.dll (Additional information: Az index a tömb határain kívülre mutatott.)

- **Leírás:** Ez a hiba ritkán a program futtatása közben jelentkezhet.
- **A hiba forrása:** Ha nincs beolvasott képkocka, akkor a tömb méretén túlmutat a SURFTracker. Biztosítani kell, hogy indításnál a kamera inicializáció megtörténjen, illetve a capture során ne történjen „filmszakadás”.

#2: A program nem detektál gyalogátkelőt (illetve táblát) abban az esetben, ha a felhasználó egyenesen érkezik az átkelőhöz (nem merőlegesen út mentén). Ezzel a problémával a fejlesztés során számoltam, mivel táblajelzés csak az autósoknak áll fent. A gyalogátkelő pedig országonként változik, nem szabványos, így ez egy újabb kihívás lehet.

Minimális gépigény:

CPU: Intel Celeron 2.0 GHz

RAM: 1 GB

Csatlakoztatott USB Webkamera (640x480 felbontás)

Hangkártya (sztereo)

Framework: Microsoft .NET Framework 3.5

Telepítési információk:

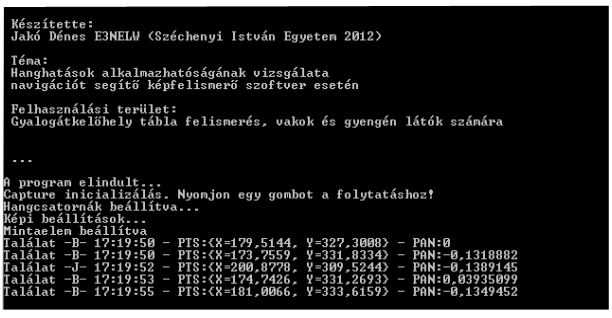
Telepítés előtt csatlakoztassuk a webkamerát a számítógéphez.

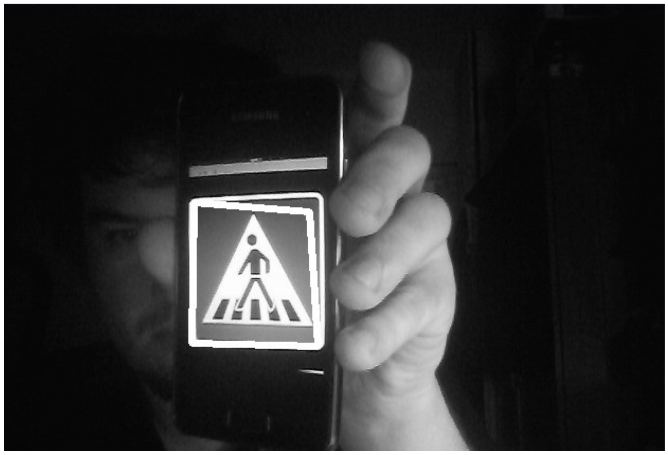
A minimális gépigénynek megfelelést követően indítsuk el a setup.exe filet. Célszerű, hogy telepítés mellett internetkapcsolatunk legyen, ugyanis a .NET keretrendszer telepítése ekkor történik meg (ha eddig nem volt telepítve).

Ezt követően a ZEBRAPROJEKT4 program telepítése veszi kezdetét. (fogadjuk el a tanúsítványt, kattintsunk a Telepítés gombra). A folyamat végén a program automatikusan elindul. Kilépés a kamerakép bezárásával lehetséges. **Ügyeljünk rá, hogy a program futtatása közben ne a terminálablakot zárjuk be elsőként, hanem a képnézőt!**

Ha az IndexOutOfRange hiba merülne fel, abban az esetben indítsuk újra a programot. Jelenleg erre a problémára hivatalos EMGU fejlesztői fórumon nincs megoldás. Ha betartjuk ennek a kezelését, folyamatos képet biztosítunk, akkor nem merül fel várhatóan hiba a program futtatása közben.

A program tesztelve lett x86 és x64 környezetben is különböző webkamerákkal.
A programot „házilag” is letesztelhetjük, csupán ki kell nyomtatnunk, vagy képként kimentenünk mobileszközre egy gyalogátkelő képet.





Összefoglalás

A szakdolgozat részletesen mutatta be a célkitűzések és programkritériumok mentén haladva a képfeldolgozás mai technológiáját, mind elméleti és mind gyakorlati megvalósításban. Téma szerint olyan pontokat érintett, amely a program létrejöttéhez feltétlenül szükségesek (a hivatalos OPEN CV jegyzet 577 oldalból áll). Gyakorlati oldalon az olvasó megismerkedhetett az OPEN CV elveivel, illetve EMGU CV csomagjaival, amelyekkel az elméleti ismeretek a C# programozási nyelv és könyvtárcsomagok segítségével alkalmazhatóvá váltak. A program a látássérültek navigációjának segítése céljából íródott elsősorban, azonban más területeken is felhasználható (például személygépjárműbe építve gyalogátkelő detektálására). A szakdolgozatot olvasva törekedtem a folyamatos és lépésről-lépésre történő ismeretátadásra mind elméleti és ezzel párhuzamosan gyakorlati oldalon. Ennek segítségével az olvasó saját maga is képes képfeldolgozó alapprogramokat írni. Természetesen a navigációhoz fontos audio fejlesztések is hasonló módon kerültek a programba, amelyeket szintén a megszokott részletességgel ismertettem. Az audio témakörben az alapvető lejátszási folyamatok, csatornadefiniálás, és panorámázási műveletek gyakorlati megvalósítását hoztam közelebb az olvasóhoz. Remélem a dolgozat megfelelő alapként szolgál majd a későbbiekben az említett képfeldolgozási és audio műveletek továbbgondolásához, és bőven merít ötletet a hasonló projektek kivitelezéséhez. Természetesen a könyvtárcsomagoknak létezik kereskedelmi verziója is, amely hatékonyabb feldolgozást ígér azok számára, akik kellő gyakorlatot merítettek, és tisztában vannak az EMGU CV műveletekkel, és szeretnék ötletüket és szoftverüket piaci körülmények közé vinni.

Summary

This thesis introduces the image processing methods used nowadays in theoretical and in practice. For the development part, the EMGU CV library (an OPEN CV wrapper) is used, which contains many methods to create programs, modify image properties, recognize objects and give navigation aid for the visually impaired users, detecting zebra crossings, and traffic signs. The topics are ordered by difficulty, and helps to create similar programs step by step using tutorials written for each topic. The program code is written in C# is created for demonstration purposes, but a suitable starting point for engineers and programmers to create basic programs in image transformation and feature recognition fields. Each method is presented with detailed description for the field of usage. From the basic solution setup, reference addition, image input examples, through segmentation processes to feature detection (with recognized feature points), the necessary fields are presented. Not only computer vision development is demonstrated, but some audio features are also included using Naudio library packages. For the navigation, base functions for audio playback, panning, and channel operations are implemented with useful tutorials. The program can be freely used and refactored for further development creating computer vision and audio projects (the final code that is released is shown at the end of the thesis, and for each topic sample program codes can be found with highly detailed description comments) . The EMGU CV library has a commercial version which is more effective, and faster for targeting these type of markets.

Irodalomjegyzék témakörök szerint:

Naudio leírások és dokumentáció a fejlesztésekhez:

http://naudio.codeplex.com/documentation

EMGU CV leírások és dokumentáció a fejlesztésekhez:

http://www.emgu.com/wiki/index.php/Tutorial

http://www.emgu.com/wiki/files/2.3.0/document/Index.html

http://www.emgu.com/wiki/index.php/Download_And_Installation

http://fewtutorials.bravesites.com/entries/emgu-cv-c/level-1---lets-make-a-camera-application

http://www.emgu.com/forum/

OPEN CV:

http://en.wikipedia.org/wiki/OpenCV

Gary Bradski & Adrian Kaehler – Learning OPEN CV (O'Reilly Media 2008)

(ISBN: 978-0-596-51613-0)

http://opencv.willowgarage.com/wiki/

Általánosan a képfeldolgozásról:

http://nyf.beckground.hu/incoming/kepfeldolgozas/FA_KJ_jegyzet2.pdf

http://e-oktat.pmmf.hu/kepeshang_10_fejezet

Gauss-szűrő, képi transzformációk:

http://rs1.szif.hu/~szorenyi/elm/bioselm3.htm

http://en.wikipedia.org/wiki/Gaussian_blur

http://stackoverflow.com/questions/7782941/gaussian-noise-in-emgucv

http://kepfelprojekt.uw.hu/Projekt/3_gauss.htm

http://www.emgu.com/wiki/files/1.4.0.0/html/15f22443-ef80-1e33-4f68-a1dd9fc1f370.htm

http://hu.wikipedia.org/wiki/Interpoláció

http://myopencv.wordpress.com/2008/04/09/thresholding-with-an-opencv-function/

Szegmentáció:

http://mazsola.iit.uni-miskolc.hu/DATA/segedletek/kepfeld_multm/vargaz/digim-fundament.htm

http://www.inf.u-szeged.hu/~kato/teaching/segmentation/01_segmentation.pdf

Detektálási módszerek:

http://imrannazar.com/content/img/android-sobel-example.png

http://en.wikipedia.org/wiki/Blob_extraction

http://bsd-noobz.com/opencv-guide/60-1-counting-coins-in-an-image

http://www.inf.u-szeged.hu/~kato/teaching/DigitalisKepfeldolgozasTG/10-ShapeDetection.pdf

http://www.inf.u-szeged.hu/~kato/teaching/segmentation/03_edgedetection.pdf

https://wiki.sch.bme.hu/bin/view/Valaszthato/KerteszKerdesek

http://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm

SIFT/SURF:

http://www.inf.u-szeged.hu/~kato/teaching/DigitalisKepfeldolgozasTG/08-KeypointCorrespondence.pdf

http://en.wikipedia.org/wiki/Scale-invariant_feature_transform

huro-sift.googlecode.com/svn-history/r2/.../magyar.osszefoglalo.doc – (Tárolt változat)

http://web.eecs.umich.edu/~silvio/teaching/EECS598/lectures/lecture10_1.pdf

http://www.vision.ee.ethz.ch/~surf/eccv06.pdf

huro-sift.googlecode.com/svn-history/r29/.../HURO.report.HUN.doc – Tárolt változat

Homográfia:

iar.bmfnik.hu/2007_2008/dplusz/doku/irodalomkutatas.doc

www.sztaki.hu/~sziranyi/ITK_3D/Diák.../Varga_Homografia.ppt

http://plus.maths.org/issue23/features/criminisi/homography.gif

Kép körbeírás/leírása, egyéb:

http://www.inf.u-szeged.hu/~kato/teaching/DigitalisKepfeldolgozasTG/11-ShapeDescriptors.pdf

http://www.uvt.bme.hu/targyak/kterv_I/kterv_I_II_kgysch/kterv_I_II_kgysch_5_6.pdf (zebra)