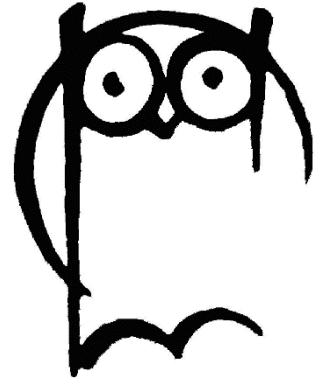


Debreceni Egyetem

Műszaki Kar

Villamosmérnöki és Mechatronikai tanszék



Második Generációs Wavelet Transzformáció Alkalmazása 2D Képeken, NI LabVIEW Segítségével

Készítette: **Puskás Dániel** mechatronikai mérnök hallgató

Belső konzulens: Piros Sándor

Tanszékvezető: Dr. Husi Géza

Debrecen

2014. november

Tartalomjegyzék:

Tartalom

I. Bevezetés	3
II. NI LabVIEW programozói környezetről röviden.....	4
A. Képfeldolgozás szempontjából legfontosabb eszközei:	5
1. Mátrixmatematikai függvények:	5
2. Képfeldolgozó függvények:	5
II. Második generációs wavelet transzformáció:	6
A. Transzformálás	7
B. Inverz- Transzformálás	8
III. Gyakorlati megvalósítás.....	10
A. Transzformáló progamegység	10
1. Feldolgozás első lépése, a kép beolvasása. Beolvasó VI bemenete, elemei, futása, és kimenete.	10
2. Kép feldolgozásának előkészítése. Kép daraboló VI futása, be- és kimenetei.....	12
3. Transzformálás a 256x256-os kép darabokon majd azok összeállítása az eredeti méretükre. 17	
4. Transzformált adatok elmentése és megjelenítése	20
5. Eredmény és kiértékelés.....	20
B. Inverz- transzformáció lefolyása.....	22
1. A transzformált adatfájlok beolvasása.....	22
2. A kép feldarabolása az inverz- transzformációhoz.....	23
3. RGB színsávok egyesítése, a visszaalakított kép mentése	24
IV. Továbbfejlesztési tervek.....	24
A. PC-n futtatás optimalizálása	24
B. Tervek az FPGA-n történő futtatásra	25
C. Videojelek feldolgozása	25
V. Felhasználási lehetőségek.....	25
A. Video steamelés.....	25
B. Computer látás, objektum felismerés, elmozdulás detektor.....	26
VI. Összefoglalás	27
VII. Irodalomjegyzék	29

Ábra- és Táblázatjegyzék:

1. ábra, hibajelzés rosszul bekötött eszköz esetén	4
2. ábra, mátrix függvények a LabWIEV eszköztárából.....	5
3. ábra, támogatott képformátumok be és kimeneti eszközei	6
4. ábra, az eltorzult eredeti kép és a különbségképzés eredménye	6
5. ábra (a, eredeti kép) (b, transzformált kép)	7
6. ábra (a, JPEG beolvasó VI) (b, képadatokat feldolgozó VI)	10
7. ábra, színes képet RGB sávokra bontó VI.....	11
8. ábra, több elem halmazba rendezése	12
9. ábra (a, mátrix szélessége és magassága) (b, osztás egész számú eredménye és maradéka) ..	13
10. ábra, kép kiegészítése 256-al való maradék nélkül oszthatóságúvá	14
12. ábra, a kép kivágó lépései.....	15
11. ábra, kép daraboláshoz szükséges koordináta adatok.....	15
13. ábra, kivágó koordináta készítő ciklusok	16
14. ábra, kép daraboló függvény	16
15. ábra transzformálás első 6 lépése	17
16. ábra, transzformáló For ciklus páros futásokra	18
17. ábra, transzformáló For ciklus páratlan futásokra	19
18. ábra file kiíró VI.....	20
19. ábra, transzformálás előtti és utáni hisztogramok	21
20. ábra, transzformált adatok beolvasása.....	22
21. ábra inverz- transzformáló ciklus	23
22. ábra RGB színsávok egyesítése.....	24

I. Bevezetés

A Debreceni Egyetem Műszaki Karának Mechatronika hallgatójaként a tanrendem részét képezi a Mechatronika Projekt elnevezésű tantárgy, a tantárgy lényege, hogy egyedül vagy csapatban egy adott problémát a kar egyik munkatársa által felügyelve önálló munka keretében megoldjunk, és azt dokumentáljuk. A témaválasztás során Piros Sándor Tanár úr által kínált projekttema megragadta a képzeletemet, a téma Tanár úr biológiai sejtosztódás mechanizmusainak mintáján alapuló képfeldolgozási eljárásának az alkalmazása volt, NI LabVIEW programozói felület segítségével. A projektfeladatot sikeresen prezentáltuk csapattársaimmal, Piros Tamarával és Urszuly Dáviddal, ám a program melyet készítettünk kezdetleges volt és igen korlátozott lehetőségekkel rendelkezett. A munka befejeztével engem továbbra is érdekelt az eljárásban rejlő lehetőségek ezért tovább dolgoztam Piros Sándor Tanár úrral, aki közben jelentősen átdolgozta a program magját képező mátrixmatematikai eljárásokat.

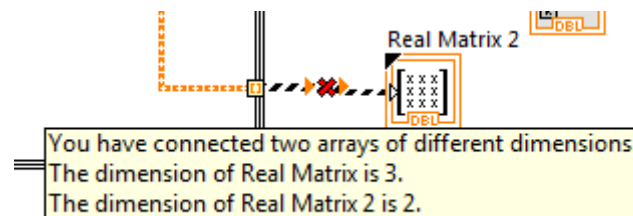
Így a dolgozatban a már új alapokra helyezett feldolgozó program működését kívánom részletekbe menően leírni. A program célja, hogy bármilyen két dimenziós (2D) színes képet feldolgozva olyan módon alakítsa azt át, hogy a kép jelentősen tömöríthetővé váljon az eredetihez képest, majd ezt az átalakított képet, képes legyen visszaalakítani eredeti formájába.

A program készítése során több szempontot kellett figyelembe venni, mivel a jövőbeni célok egyike a field-programmable gate array (FPGA) chipeken való futtatás, így a programnak a lehető legkisebbnek és a leghatékonyabbnak kellett lennie. Jövőbeni cél még a valósidejű videojel feldolgozás, kis sáv szélességű csatornákon lehetővé téve a videojel továbbítást. Valamint felhasználási lehetőséget látok a video streamelés és a computer látás területén.

II. NI LabVIEW programozói környezetről röviden

A National Instruments által forgalmazott grafikus fejlesztői környezet mely számtalan mérésadatgyűjtőhöz és vezérlő hardware-hez nyújt egyszerű illesztői felületet. A segítségével könnyen készíthetünk feldolgozó algoritmusokat valódi vagy akár szimulált mérőeszközökhöz, rengeteg beépített függvénnyel és feldolgozóegységgel rendelkezik, melyek építőköcek módjára elhelyezhető a munkaterületen és azonnal használhatók. A programelemeket VI-nak nevezi azaz Virtual Instrument-nek mely annyit tesz, hogy egy virtuális eszköz mely rendelkezhet bemenetekkel és a kimenetekkel. Ez az eszköz tartalmazhat bármilyen funkciót, és újabb VI-okat így több szinten strukturálva készíthetünk feldolgozó értékelő programokat.

A fejlesztői környezet nagy előnye a folyamatos hibaellenőrzés, mely azonnal figyelmeztet, ha a programban hiba található. A VI-ok és funkciókat egy áramkörhöz hasonlóan kábelekkel köthetjük össze így megadva az adatok és a változók kapcsolatait. Amennyiben két függvényt hibásan kötünk össze, azaz egyik funkció kimenetéről a másik funkció bemenetére érkező adat nem megfelelő akkor az összekötő kábelben egy piros (x) (1. ábra) jelenik meg jelezve, hogy a kapcsolattal baj van. A kurzorral az (x)-re navigálva információt kaphatunk egy felugró ablak formájában arra vonatkozóan, hogy a kapcsolat miért nem megfelelő, így gyorsan látható a hiba és orvosolható.

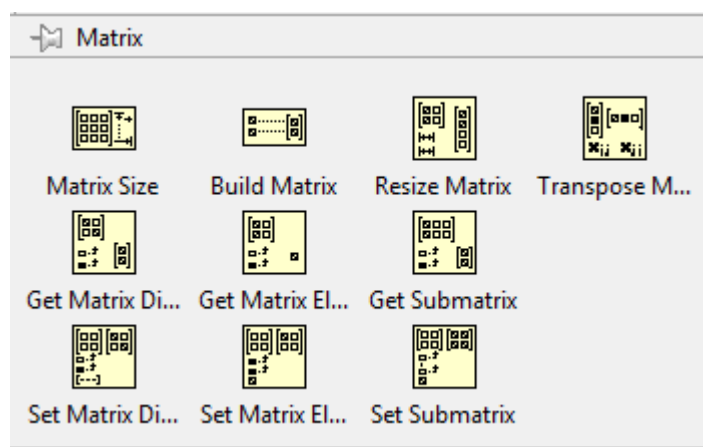


1. ábra, hibajelzés rosszul bekötött eszköz esetén

A. Képfeldolgozás szempontjából legfontosabb eszközei:

1. Mátrixmatematikai függvények:

A fejlesztői környezet rendelkezik, nagyszámú beépített mátrixmatematikai függvénnyel melyek segítik a nagy méretű 2D adattömbök feldolgozását, egyszerű függvények építésével lehetséges gyors és hatékony transzformálások elvégzése. A kifejezetten mátrix műveletekből a függvények között ugyan csak tíz darab található (2. ábra) de az egyszerű matematikai műveletek, mint például a szorzás vagy az összeadás a bemenettől függően automatikusan



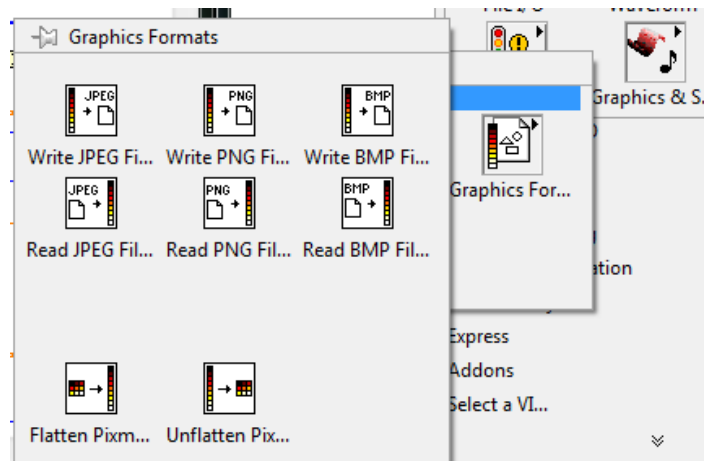
2. ábra, mátrix függvények a LabVIEW eszköztárából

alkalmas a mátrixok feldolgozására. Számos ilyen a háttérben dolgozó függvénnyel rendelkezik a LabVIEW melyek automatikusan a bemeneti adatnak megfelelően maguktól a helyes módon végzik el a számításokat, így nagyban megkönnyíti a programozó dolgát, a programozáshoz szükséges időt lecsökkenti.

2. Képfeldolgozó függvények:

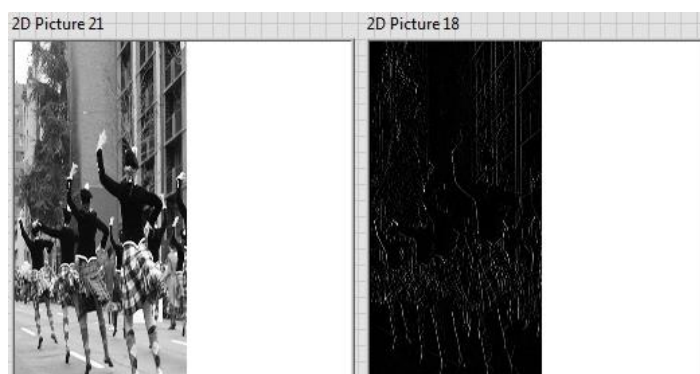
Számos előre elkészített függvényt tartalmaz a program, melyek segítik a felhasználót képek adattömbökké alakításában és az azokkal való manipulációban. Az alapeszközök között három formátumú kép feldolgozót találunk, ezek a formátumok: JPEG, PNG, BMP (3. ábra) Ezek az eszközök alkalmasak a képek pixeladatainak nyers adattá történő átalakítására. Valamint ezen eszközök párjaként megtalálhatjuk azokat melyek a nyers adatokból képesek a korábban említett három kiterjesztésű képet készíteni és azt menteni.[3]

II. Második generációs wavelet transzformáció:



3. ábra, támogatott képformátumok be és kimeneti eszközei

A digitális képek pixelekből állnak melyek egy 2D rácson elhelyezkedve rendelkeznek két koordinátával és a színre vonatkozóan a három alapszín szerinti színmélységgel. A transzformáció lényege abban rejlik, hogy feltételezzük azt, hogy egy átlagos képen ha a szélesség tengelyét (x) –nek nevezzük a függőleges tengelyt pedig (y) –nak akkor egy tetszőleges (y) értéket kiválasztva és azon végighaladva az (x) tengely mentén, a szomszédos pixelek általánosságban hasonló szín értékűek lesznek, kicsi lesz a deviációjuk. Így ha egy alul áteresztő szűrést alkalmazunk a szomszédos elemeken, például különbséget képzünk páros és a páratlan elemek között majd eltávolítjuk a páratlan adatokat és a különbségképzés eredményeit akkor egy az eredetihez képest fele széles képet és egy ugyanolyan szélességű különbségadatokból álló képet kapunk. Egyszeres átalakítástól az eredeti kép így eltorzul, (4. ábra) a következő lépésben a másik tengely szerint újra transzformáljuk a képet így újra az eredeti arányoknak megfelelő képet kapunk ám pontosan fele akkora, mint a kiindulási kép.



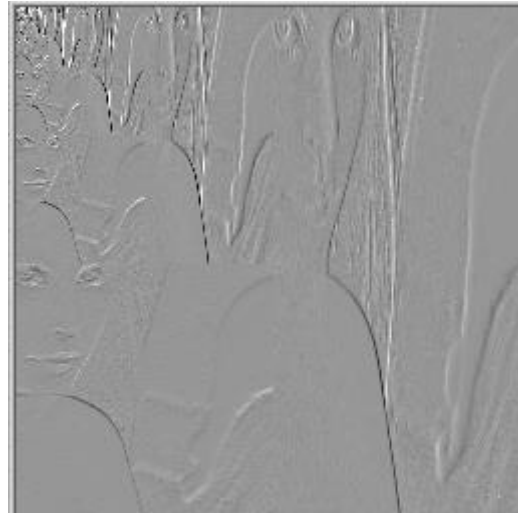
4. ábra, az eltorzult eredeti kép és a különbségképzés eredménye

Minden lépés között mentjük a különbségképzésből nyert deviációs adatokat, míg utolsó lépésként egyetlen pixel marad az eredeti képből és a hozzá tartozó lépésenkénti különbségadatok.[1] (5. ábra)

A. Transzformálás



a, eredeti kép



b, transzformált kép

5. ábra (a, eredeti kép) (b, transzformált kép)

Mátrixmatematika szabályainak segítségével ezek az átalakítások lépésenként egy szorzással elvégezhetőek. Létrehozhatunk egy transzformációs (**T**) mátrixot mellyel szorozva a pixeladatok (**P**) mátrixát megkapjuk a (**P'**) közelítési koefficienset és a (**D**) szub-mátrixot a részletek koefficiensével.

$$T \cdot P = \begin{bmatrix} P' \\ D \end{bmatrix} \quad 1$$

$$P \cdot T^T = [P', D] \quad 2$$

Az első egyenlet (1) az (y) tengely menti transzformálást a kettes (2) egyenlet pedig az (x) tengely szerinti transzformálást írja le. Látható hogy a transzformációs (**T**) mátrixot elegendő egyszer képeznünk, majd vehetjük a transzponáltját és azzal szorozva a másik tengely szerinti eredményt kapjuk.

Példaként az (x) tengely szerinti átalakítás látható a hármas egyenleten (3):

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} =$$

$$= \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{1,1} - a_{2,1} & a_{1,2} - a_{2,2} & a_{1,3} - a_{2,3} & a_{1,4} - a_{2,4} \\ a_{3,1} - a_{4,1} & a_{3,2} - a_{4,2} & a_{3,3} - a_{4,3} & a_{3,4} - a_{4,4} \end{bmatrix} \quad 3$$

és példa az (y) tengely szerint (4):

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix} =$$

$$= \begin{bmatrix} a_{1,1} & a_{1,3} & a_{1,1} - a_{1,2} & a_{1,3} - a_{1,4} \\ a_{2,1} & a_{2,3} & a_{2,1} - a_{2,2} & a_{2,3} - a_{2,4} \\ a_{3,1} & a_{3,3} & a_{3,1} - a_{3,2} & a_{3,3} - a_{3,4} \\ a_{4,1} & a_{4,3} & a_{4,1} - a_{4,2} & a_{4,3} - a_{4,4} \end{bmatrix} \quad 4$$

B. Inverz- Transzformálás

A folyamatot megfordítva visszakaphatjuk az eredeti adathalmazt, a következő műveletek segítségével veszteségmentesen visszaállíthatjuk a képet. A visszaállítás során a transzformációs mátrix inverzét kell használnunk az (x) tengelyre az (5) illetve az (y) tengelyre a (6) egyenlet szerint.

$$P = T^{-1} \cdot \left[\frac{P'}{D} \right] \quad 5$$

$$P = [P', D] \cdot (T^T)^{-1} \quad 6$$

Példa az (x) tengely szerinti visszaalakításra (7):

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}.$$

$$\cdot \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{1,1} - a_{2,1} & a_{1,2} - a_{2,2} & a_{1,3} - a_{2,3} & a_{1,4} - a_{2,4} \\ a_{3,1} - a_{4,1} & a_{3,2} - a_{4,2} & a_{3,3} - a_{4,3} & a_{3,4} - a_{4,4} \end{bmatrix} \quad 7$$

És (y) tengelyre (8):

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} =$$

$$= \begin{bmatrix} a_{1,1} & a_{1,3} & a_{1,1} - a_{1,2} & a_{1,3} - a_{1,4} \\ a_{2,1} & a_{2,3} & a_{2,1} - a_{2,2} & a_{2,3} - a_{2,4} \\ a_{3,1} & a_{3,3} & a_{3,1} - a_{3,2} & a_{3,3} - a_{3,4} \\ a_{4,1} & a_{4,3} & a_{4,1} - a_{4,2} & a_{4,3} - a_{4,4} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \quad 8$$

Az eljárást Piros Sándor sokkal mélyre menőbb részletességgel kifejti tudományos cikkében, ahol további lehetőségeket is számba vesz. Például lehetséges ezzel az eljárással több elem átlagának a képzése, így módosítva pedig a veszteséges tárolás lehetségessé válik. Én a veszteségmentes eljárásra építettem a programomat ezért a veszteséges módszerrel itt nem foglalkozok, mindazonáltal a programom kis módosítással alkalmas veszteséges átalakításra is.

[1]

III. Gyakorlati megvalósítás

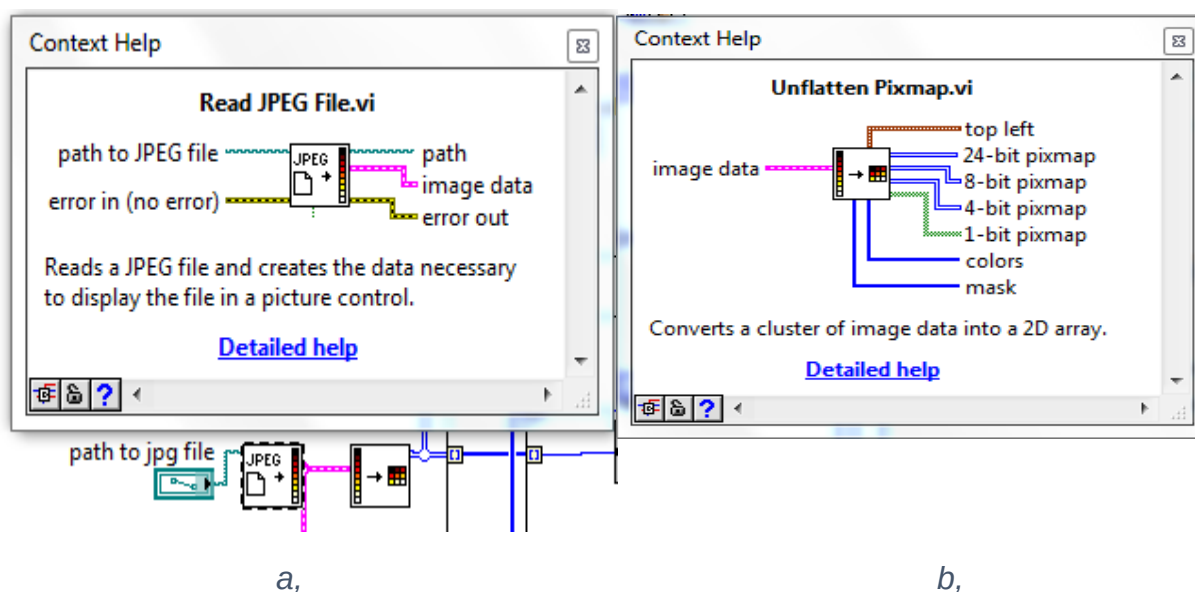
A képek transzformálásához és az inverz- transzformálásához két különálló programegységet alkottam melyek egymástól függetlenül futtathatóak. Ezáltal egy adatküldés esetén a küldő félnek elegendő rendelkeznie csupán a transzformáló, a fogadó félnek pedig az inverz-transzformáló egységgel.

A. Transzformáló programegység

A program működése lehetővé teszi bármilyen képbemenet feldolgozását a három már korábban említett képformátumok közül (JPEG, BMP, PNG), a kép méretétől függetlenül.

1. Feldolgozás első lépése, a kép beolvasása. Beolvasó VI bemenete, elemei, futása, és kimenete.

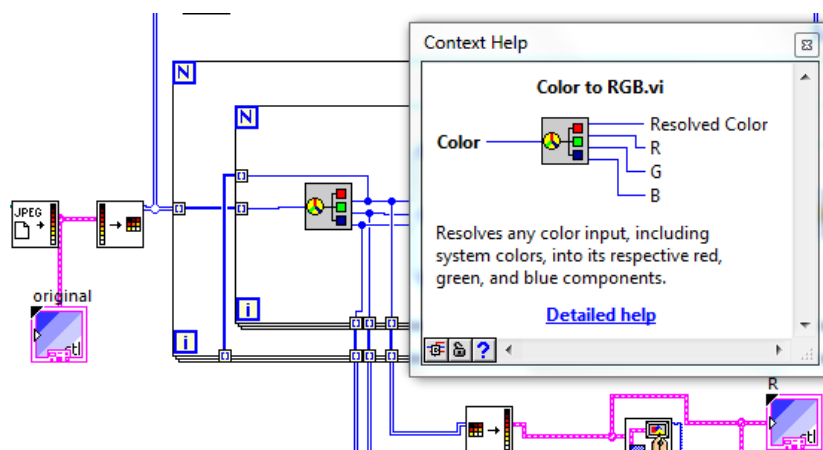
A következő példában (JPEG) kiterjesztésű képek transzformálásán fogom bemutatni a folyamatot. A transzformálni kívánt kép elérési útvonalát vagy begépeljük, vagy tallózzuk a merevlemezről. Ezt követően a (Read JPEG File.vi) nevű virtuális eszköz segítségével a megadott útvonalon található képet megnyitva azt feldolgozható adatsávokra bontja és azt egy hat elemű cluster-ként a kimeneten kiküldi. (6. ábra a,) A kimeneti cluster elemei egydimenziós tömbként tartalmazzák a kép minden információját az egyes pixelek színmélységére a pixelek színére és elhelyezkedésére vonatkozóan. Ezeket az adatokat közvetlenül képes feldolgozni az



6. ábra (a, JPEG beolvasó VI) (b, képadatokat feldolgozó VI)

(Unflatten Pixmap.vi) nevű virtuális eszköz mely automatikusan a bemeneti kép bitmélységének megfelelő kimeneten küldi ki a 2D pixeltérképet egy 2D halmaz formájában. (6. ábra, b,)

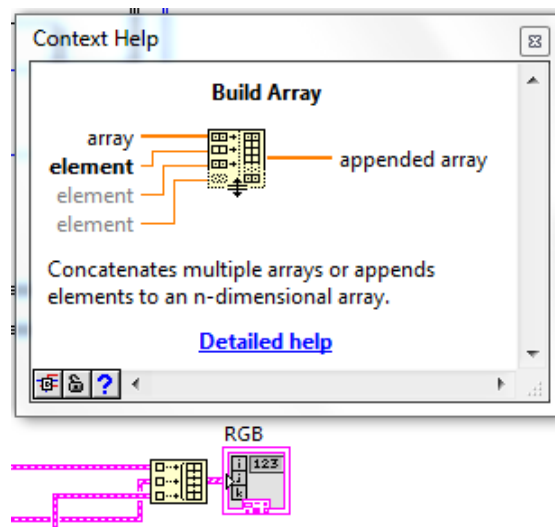
A színes képek felépítésüket tekintve három réteget tartalmaznak, a rétegek egy-egy alapszínt tartalmaznak. Ezek az alapszínek a piros, zöld és a kék, idegen nyelvű kifejezéssel ezeket RGB rétegeknek nevezik azaz Red, Green és Blue színeknek. Ezek a rétegek különböző szín mélységértékeket vehetnek fel pixelről pixelre. Ezeket a színmélység értékeket 8 bit hosszú integerekkel reprezentálják, tehát egy réteg egy pixele 0-255-ig bármilyen értékű lehet. Így a három szín rétegei egyenként 8-bit hosszú integereket tartalmaznak pixelenként. Ezeket a rétegeket a színes képekben minden pixelnél egy darab 32-bit hosszú integerrel tárolják. A (Color to RGB.vi) nevű eszköz, mely képes a 2D halaz minden 32 bit hosszú elemének az RGB színkészlet szerinti három komponensére bontani. Minden komponens 8 bit hosszú integer így minden pixel reprezentálhatóvá válik három 0-255-ig terjedő számmal. Az eszközt egy (For)



7. ábra, színes képet RGB sávokra bontó VI

ciklusban kell elhelyeznünk és ezt a (For) ciklust egy újabb (For) ciklusba kell tenni, erre azért van szükség, mert az eszköz egyszerre csak egyetlen pixeladatot képes feldolgozni. (7. ábra) Ezzel a strukturálással azt érjük el, hogy az eszköz bemeneti adatai indexelt formában érkeznek be. Sorrendben, a belső (For) ciklus minden futásakor a bemeneti 2D halmaz első sorának egy elemét dolgozza fel, majd mikor az első sor adatai elfogytak a külső (For) ciklus lép egyet és a belső (For) ciklus kezdi a feldolgozást újra, ám ezúttal a második sor elemein végig egyesével. A futás közben az elemek (Auto- Indexed Tunnel) segítségével lépnek be a függvénybe és lépnek ki onnan, így minden adat a kimeneti 2D halmazban az eredeti helyén található. A folyamat végén három darab 2D halmazt kapunk. RGB szabvány szerint felbontva a három alapszínre, a halmazok elemszáma az eredeti képpel azonos ám minden elem 8 bit hosszú integerrel lett reprezentálva. Ezzel maga a beolvasás befejeződött, az így kapott adatokat egy halmazba

rendezzük a továbbítás praktikussága miatt, kivezetjük a frontpanelre egy indikátor segítségével és azt a tartalmazó (VI) kimenetére kábelezzuk. (8. ábra)



8. ábra, több elem halmazba rendezése

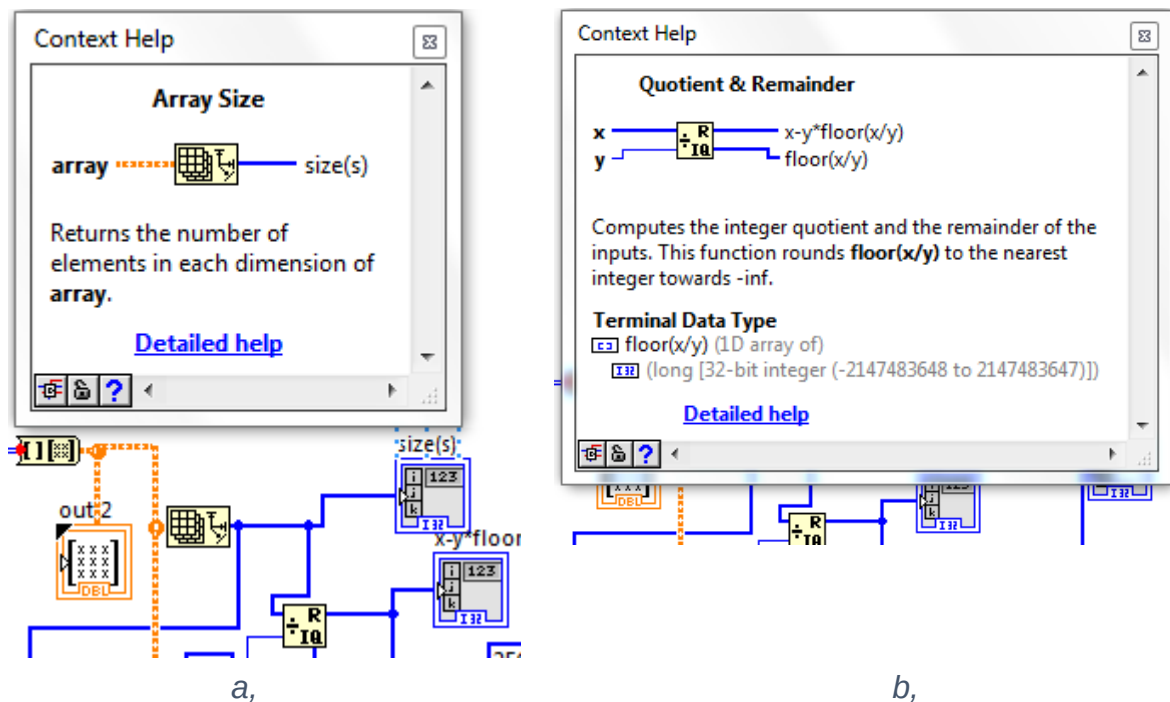
2. Kép feldolgozásának előkészítése. Kép daraboló VI futása, be- és kimenetei

A transzformáció hatékony futása és kezelhetősége érdekében maga a transzformációs eljárás kizárólag 256x256 pixel felbontású képet tud transzformálni. A célkitűzés a minimum HD felbontású képek feldolgozása volt, ezért a következő lépésben a program 256x256 pixel felbontású darabokra bontja a képet és majd azt továbbítja a következő feldolgozó eszköznek.

A képek feldarabolása nem okozna problémát, ha minden kép meghatározott szabványos felbontással rendelkezne, akkor elegendő lenne a szabványok kezelésére létrehozni adott eljárásokat. Azonban az interneten fellelhető képek többsége nem követ semmiféle szabványosságot, az elképezhető legfurcsább képarányokkal bármilyen felbontások előfordulhatnak. Ezért szükség volt egy módszerre, amely minden kép esetén egyedileg meghatározza a kép felbontását és megvizsgálja hány darab 256x256 pixel felbontású kis képre lehet bontani. Itt viszont problémába ütközhet az eljárás, hiszen ha a kép szélessége vagy magassága nem osztható maradék nélkül 256-al, akkor a sorok és oszlopok utolsó kivágott elemei hiányosak lesznek.

Ennek megoldására a legegyszerűbb módszernek az mutatkozott, hogy a nem megfelelő szélesség és magasság esetén a képet redundáns adatokkal egészítsem ki, hogy azok maradék nélkül oszthatóvá váljanak 256-al.

A kép adatai halmaz formában érkeznek a kiegészítő programblokk bemenetére, ezt konvertáljuk mátrix formátumúvá majd ezt az (Array Size) függvénnyel kiértékeljük, mely meghatározza a mátrix sorainak és oszlopainak elemszámát. (9. ábra, a,) Ezt a két adatot felhasználva a (Quotient & Remainder) függvény segítségével meghatározhatjuk, hogy a kép soraiban és oszlopaiban hányszor van meg maradék nélkül a 256. A Függvény az (**x**) bemenetén megkapja a kép szélességét és magasságát az (**y**) bemeneten pedig konstansként a 256-ot. Az $(x-y*\text{floor}(x/y))$ kimeneten megkapjuk az (**x**) és (**y**) tengely 256-al való elosztásából járó maradékokat. A $(\text{floor}(x/y))$ kimeneten az (**x**) és (**y**) tengely 256-al osztásának egész számú eredményét. (9. ábra, b,)

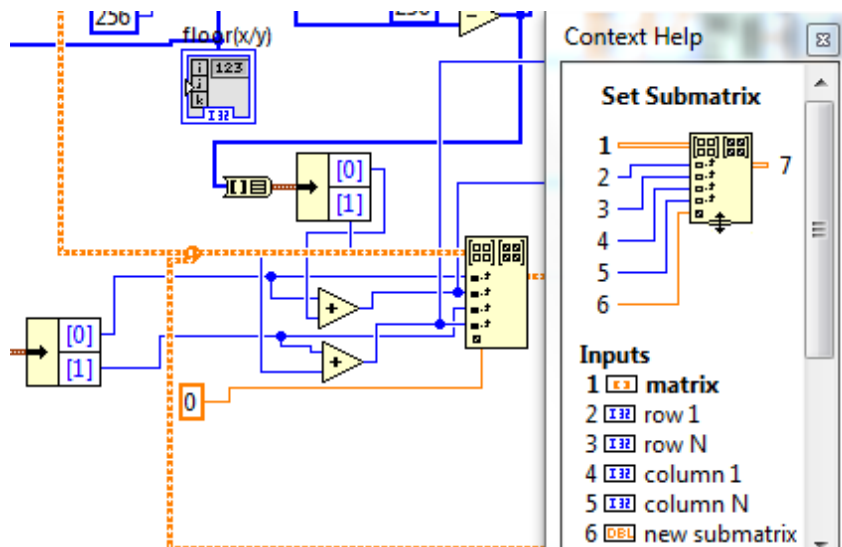


9. ábra (a, mátrix szélessége és magassága) (b, osztás egész számú eredménye és maradéka)

Így megkaptuk eredményül, hogy hány darabra 256x256-os részre kell bontanunk a képet a feldolgozáshoz, ez a szám mind (x) mind (y) tengely mentén egyenlő $(\text{floor}(x/z))+1$.

Következő lépésben ki kell egészítenünk a képet olyan felbontására, hogy az maradék nélkül legyen osztható 256-al. A korábbi osztás maradékát fogjuk erre felhasználni, a maradék értékét ki kell vonnunk 256-ból és Így megkapjuk pontosan hány elemmel többre van szükségünk mindkét tengely mentén ahhoz, hogy az osztás maradéktalanul elvégezhető legyen. Ezeket az adatokat felhasználva a (Set Submatrix) függvény segítségével kiegészítjük a képet. A függvény mátrix bemenetére az eredeti kép mátrixát kötjük, ezen kívül meg kell adnunk a mátrixban elhelyezni kívánt új elemek koordinátáját. A (row 1) bemeneten az új elemek sorainak kezdő koordinátáját kell megadnunk, ez egyenlő az eredeti kép sorainak számával,

ezután pedig a (row N) bemenetet adjuk meg ez pedig egyenlő lesz az eredeti kép sorainak számával plusz az osztás maradékának 256-ból vont különbségével. (10. ábra) Hasonlóan járunk el az oszlopok esetében is, a (column 1) bemenetre az eredeti kép oszlopainak számát kábelezzuk, a (column N) bemenetre pedig az eredeti oszlopok számát plusz az osztás maradékának 256-ból vont különbségét. Végül az új elemek értékét egy konstans (0) értékkel határozzuk meg a (new submatrix) bemenetre kábelezve. A (set Submatrix) függvény kimenet így az eredeti képhez képest nagyobb elemszámú, de ugyanakkor maradék nélkül osztható 256-al.



10. ábra, kép kiegészítése 256-al való maradék nélkül oszthatóvá

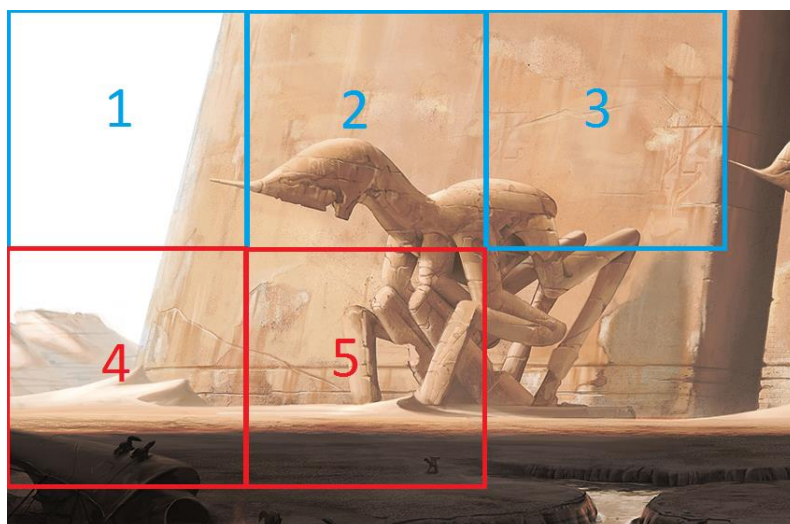
Az átalakított mátrixunk most már készen áll a darabolásra, ehhez egy (Get submatrix) elnevezésű függvény áll a rendelkezésünkre melyet egy ciklusban elhelyezve, a ciklus minden egyes futása során egy 256x256-os mátrixot vág ki a bővített mátrixból. Az első futás alkalmával a bal felső sarokban kezdve kivágjuk az első darabot ezután 256 oszloppal jobbra lépve a következőt és következőt ahányszor az oszthatóság vizsgálata alatt meghatároztuk. Ezt követően a kezdő sort 256-nál határozzuk meg és újból az előző folyamat zajlik le balról jobbra haladva az oszlopokon. A ciklus futásainak száma egyenlő a vízszintes és a függőleges oszthatóság szorzatával.

A (Get Submatrix) függvénynek szüksége van arra, hogy minden egyes futásnál megkapja a megfelelő koordinátaadatokat, hogy helyes méretben és helyes sorrendben vágja ki a 256x256-os darabokat. Ehhez generálnunk kell az összes bemenet számára megfelelő koordináta adatot (11. ábra) A vízszintes tengely menti koordináták minden lépésben 256-al nőnek, míg a függőleges tengely szerinti koordinátáknak konstansnak kell maradniuk. Mikor a kivágó végzett az első sor összes oszlopával akkor a függőleges koordinátáknak nőniük kell 256-al és

appended array 2

0	0	256	512	768	1024	0	256	512	768	1024
0	0	0	0	0	0	256	256	256	256	256
	256	512	768	1024	1280	256	512	768	1024	1280
	256	256	256	256	256	512	512	512	512	512
	0	0	0	0	0	0	0	0	0	0

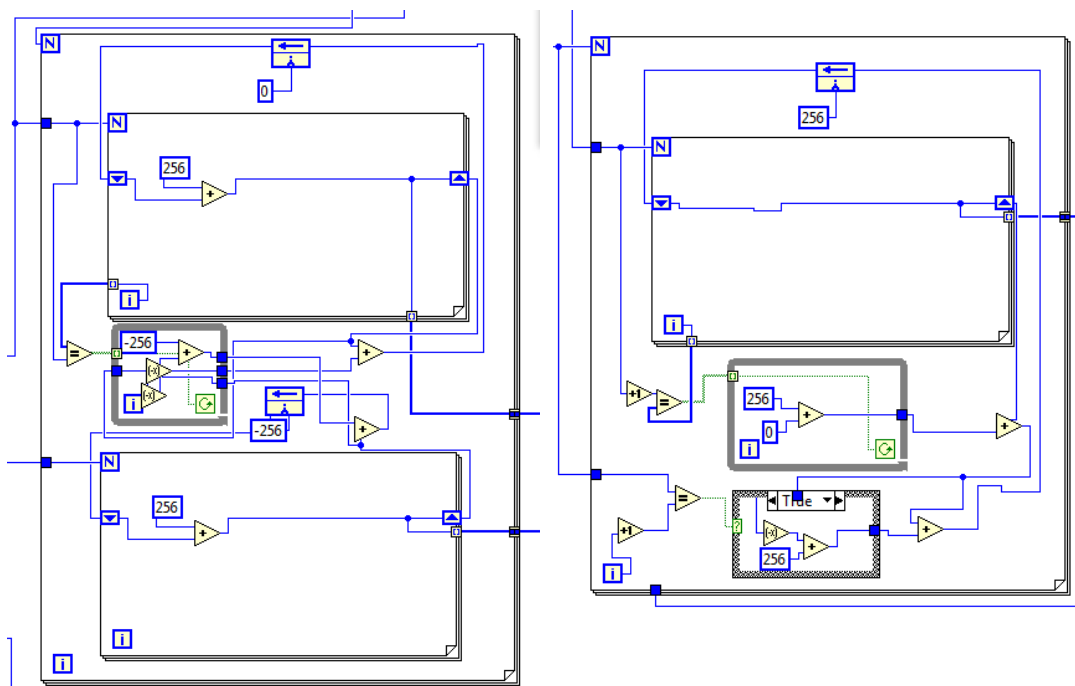
12. ábra, kép daraboláshoz szükséges koordináta adatok



11. ábra, a kép kivágó lépései

újából konstansnak kell maradniuk addig amíg az oszlopok mentén halad a program. A kivágás lefolyásának lépéseit a (12. ábra) szemlélteti.

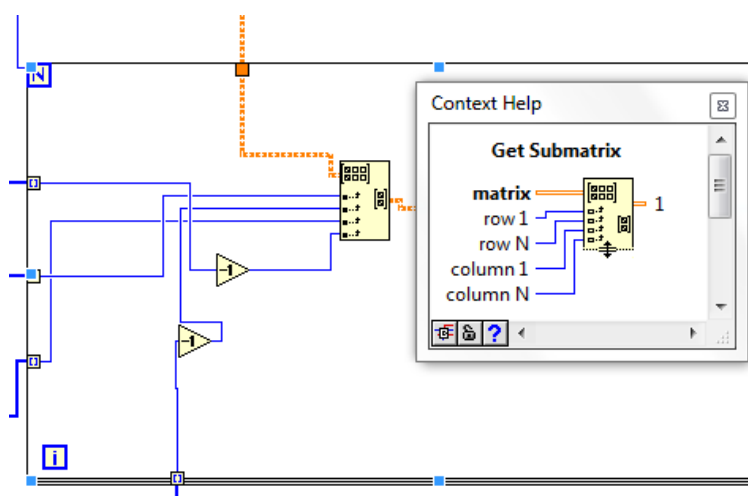
A koordináták képmérethez való igazodását úgy oldottam meg, hogy újból a (Quotient & Reminder) függvény eredményeit felhasználva, (For) ciklusok segítségével a megfelelő elemszámú halmazokat készítették. (13. ábra)



13. ábra, kivágó koordináta készítő ciklusok

Ezek a (For) ciklusok elkészítik az adott képmérethez illeszkedő kivágási koordinátákat, melyeket halmazban küldenek ki, és minden elemük indexelt. Ezeket az adatokat láthatjuk egy tetszőlegesen választott képre a 12. ábrán.

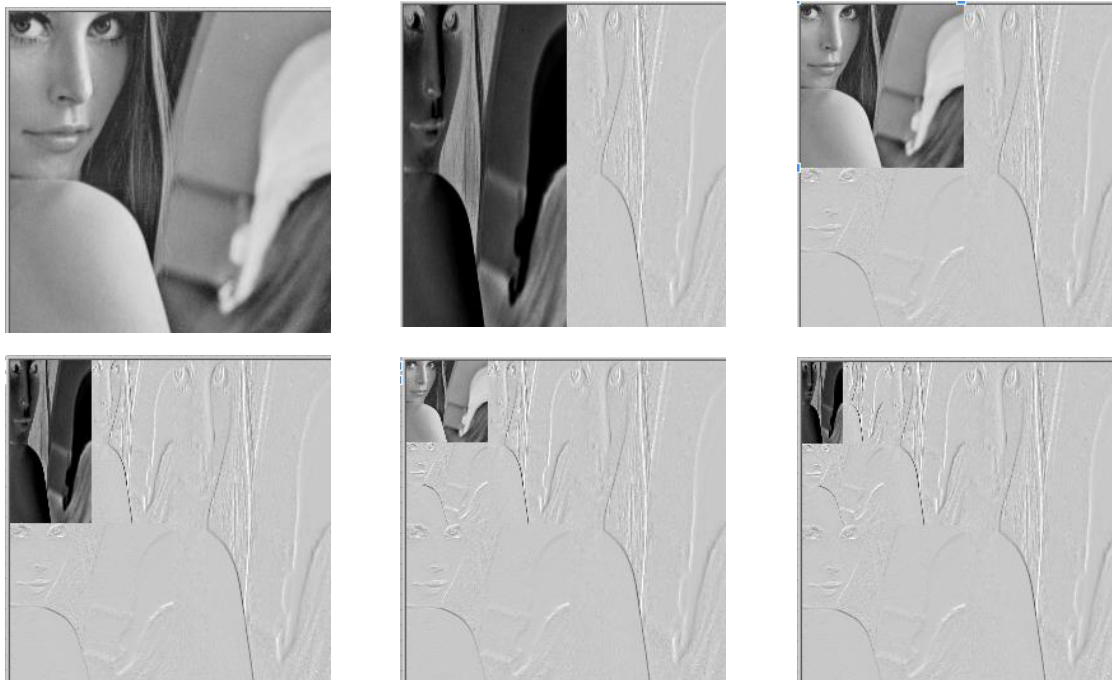
Ezeket a halmazokat a darabolásra használt (Get Submatrix) függvényhez könnyen hozzá tudjuk huzalozni, ami a többszöri futáshoz szintén egy ciklusban helyezkedik el. (14. ábra) A szükséges koordináta adatok minden futás előtt az indexelésnek megfelelően kerül kiolvasásra, tehát ahogy a 12. ábrán látható, a négy szükséges koordináta oszloponként kerül kiolvasásra.



14. ábra, kép daraboló függvény

Miután lefutott a darabolás, a kimeneten egy 3D mátrixot kapunk melynek két tengelye a függőleges és a vízszintes kiterjedésű pixeladatokkal feltöltött mátrix és a harmadik dimenzió tengelyén pedig a darabolás lépései szerint haladunk.

3. Transzformálás a 256x256-os kép darabokon majd azok összeállítása az eredeti méretükre.



15. ábra transzformálás első 6 lépése

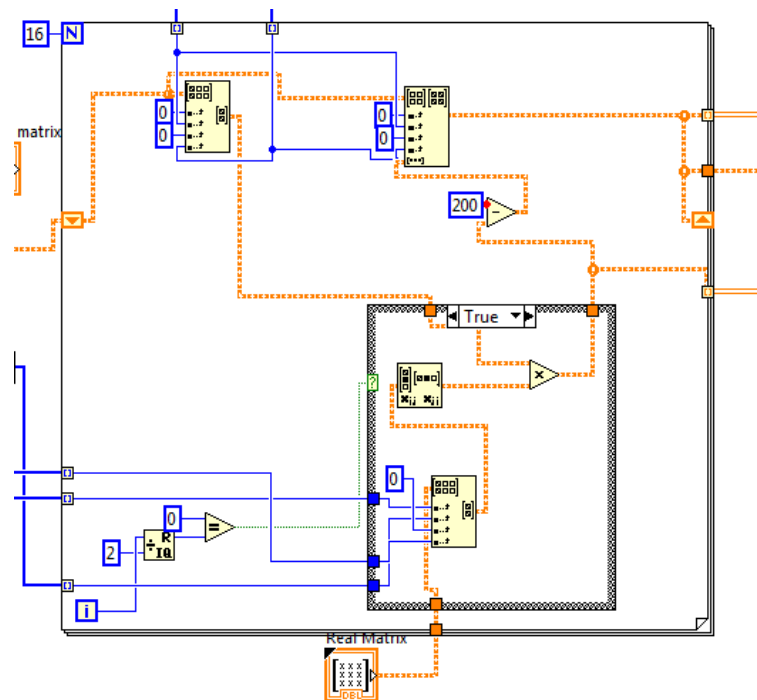
Az átalakítás első hat lépését láthatjuk a 15. ábrán melyeket kis módosítással jelenítettünk meg az ábrában látható képeken, a módosítás csupán kozmetikai jellegű mivel a képek transzformált részei képként megjelenítve közel teljesen feketék, a szemléletesség kedvéért ezeken módosítottunk egy állandóval.

A transzformáláshoz a III. Fejezet A. pontjában leírt matematikai módszereket kell alkalmaznunk a képet reprezentáló mátrixokon. A magyarázat könnyebb érthetőségének kedvéért eltekintünk attól, hogy a program egymás után akár több száz 256x256-os mátrix feldolgozását is végezheti, a folyamat csupán ismétlés kérdése ezért egyetlen futást fogunk bemutatni.

A transzformáláshoz szükségünk lesz egy sor transzformációs mátrixra melyekkel megszorozva a kép mátrixát, megkapjuk az átalakított képet. A transzformációs mátrixok a III. fejezet A. pontjában leírt formájúak, viszont két lépésenként más méretű mátrixra van

szükségünk. Az első lépésben a (**T**) mátrixunk 256x256- majd 128x128-, 64x64-, 32x32-, 16x16-, 8x8-, 4x4-, 2x2-es mátrix. Ezeket egy VI segítségével készítettem el a fő program legfelső futási rétegében, így ennek csupán egyszer kell lefutni a program inicializálását követően, ezután felhasználható minden egyes kép transzformálására.

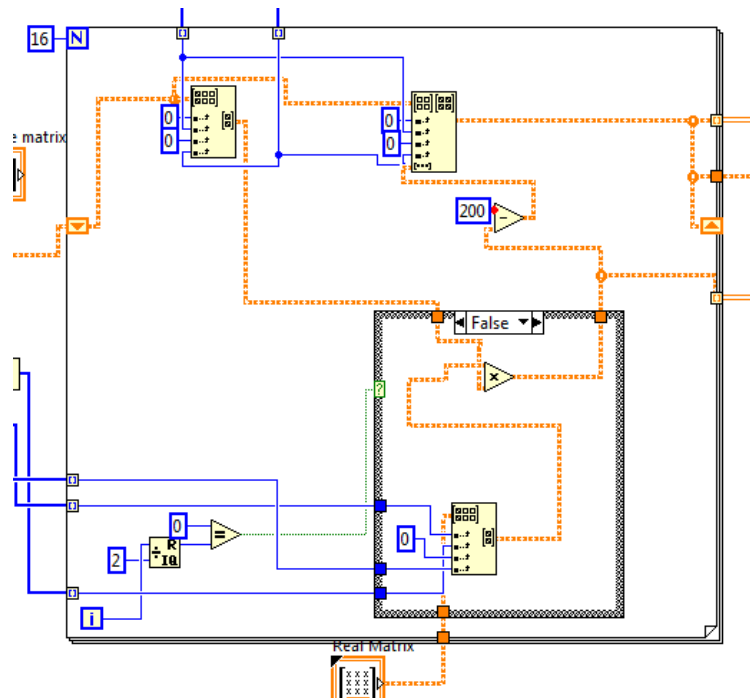
A transzformálandó mátrix egy (For) ciklus (Shift) regiszterébe érkezik, innen egy kivágó függvény az első lépésnél kivágja a teljes képet, gyakorlatilag csupán továbbítja a (Case



16. ábra, transzformáló For ciklus páros futásokra

Structure) felé, mely egy igaz hamis logika alapján eldönti, hogy a (**T**) transzformációs mátrixszal- vagy a transzponált (**T**) mátrixszal szorozza-e a kép mátrixát. A (Case Structure) állapotát futásonként változtatjuk, első futásnál, a (For) ciklus (**i**) iterációs terminálja nulla ezt elosztva 2-vel nullát kapunk. Tehát a következő egyenlőség vizsgáló függvény szerint melynek egyik bemenete nulla a másik pedig a 2-vel osztott terminál értéke, igazat fog jelezni így a (**T**) mátrix transzponáltját fogja szorozni a kép mátrixával. (16. ábra) Ez azt jelenti, hogy egy 256x256-os mátrixot szoroztunk 256x256-os mátrixszal, ennek az eredménye egy (Set Submatrix) függvénybe fut, mely felülírja az új adatokkal a kiindulási mátrix rácselemeit. Ezt az adatot ezután továbbítja a (Shift) regiszterbe, ahonnan a (For) ciklus következő futásakor beolvashatja. A második futás alkalmával a kivágó függvény egy 256x128-as mátrixot vág ki és ezt továbbítja a (Case) struktúrának. Ekkor már azonban az (**i**) terminál 1-re lépett Így ez osztva kettővel már 0,5-öt fog maradéku adni és ez mivel nem egyenlő nullával így a (Case)

struktúra a hamis állapotába vált. (17. ábra) Ez azt jelenti, hogy ezúttal a 256x256-os (**T**) mátrixszal fogja megszorozni a kivágott 256x128-as mátrixot. Ennek a szorzásnak az eredménye egy 256 sor- és 128 oszlopszélességű mátrix mellyel felülírva az előző futás adatait nem ír felül számunkra hasznos részek, tehát a 128-as oszloptól a 256-os oszlopig elhelyezkedő adatok, érintetlenek maradnak. Ez a folyamat megjelenítve látható a 15. ábrán. Ez a folyamat



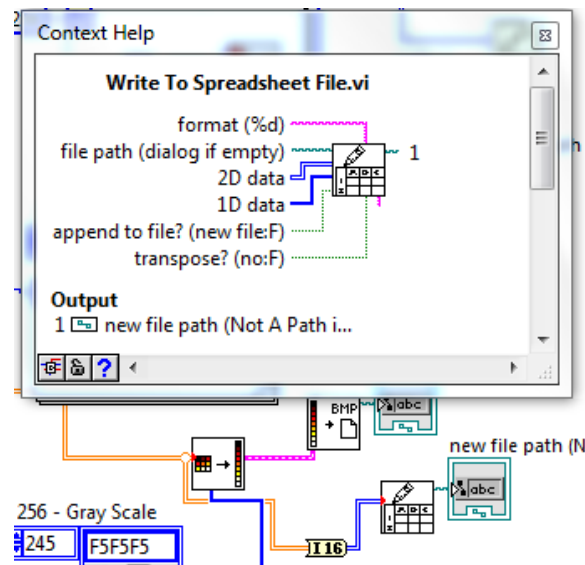
17. ábra, transzformáló For ciklus páratlan futásokra

ismétlődik 16 lépésen keresztül míg végül a kiindulási (**T**) mátrix egyetlen eredeti pixelre redukálódik és a hozzá tartozó transzformációs adatokra.

Ezt a folyamatot alkalmazzuk az eredeti nagy méretű kép minden kivágott darabjára, és az adatokat továbbítjuk egy VI-nak amely ezekből a 256x256-os mátrixokat egyesíti. Ez az egyesítés is egy (For) ciklus segítségével történik és a már megismert (Set Submatrix) függvény felhasználásával. A folyamat felhasználja a darabolásnál elkészített felosztási koordinátákat, Így azokat nem kell újra elkészíttetni.

4. Transzformált adatok elmentése és megjelenítése

A kapott transzformált mátrixot merevlemezre mentjük a (Write To Spreadsheet File.vi) eszköz segítségével. A választás azért esett erre a formátumra mivel a LabWIEV kimeneti eszközei közül ez bizonyult a leg helytakarékosabbnak. (18. ábra) A szemléltetés kedvéért a kapott



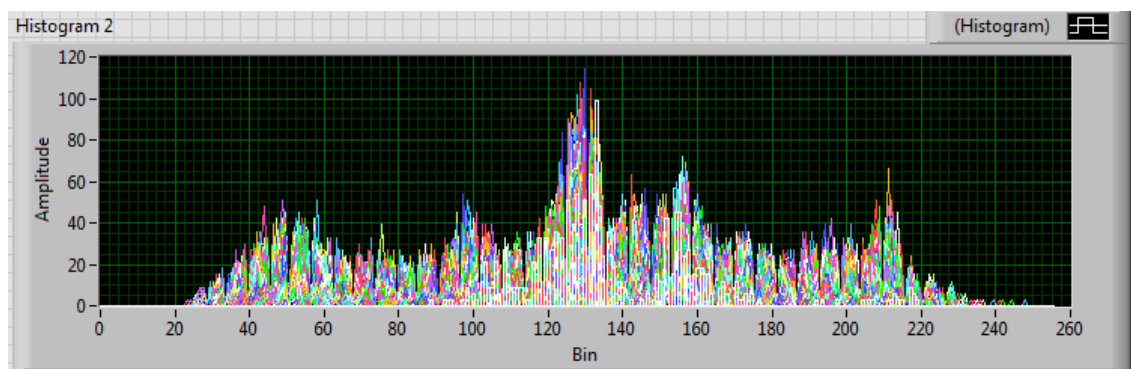
18. ábra file író VI

eredményeket kép formátumban is menthelyük és tárolhatjuk is. Viszont ekkor ügyelnünk kell arra, hogy ne történjen adatvesztés, a képformátum jellegéből fakadóan, ugyanis a transzformált mátrixunk tartalmazni fog értékeket -255-től 255-ig terjedően. A szabványos 8-bit színmélységű képek viszont csupán 0-tól 255-ig tartalmazhat értékeket és ilyen módon elveszíthetjük a negatív elemeket, ezért is érdekesebb adatfájlban tárolni a transzformáció eredményét.

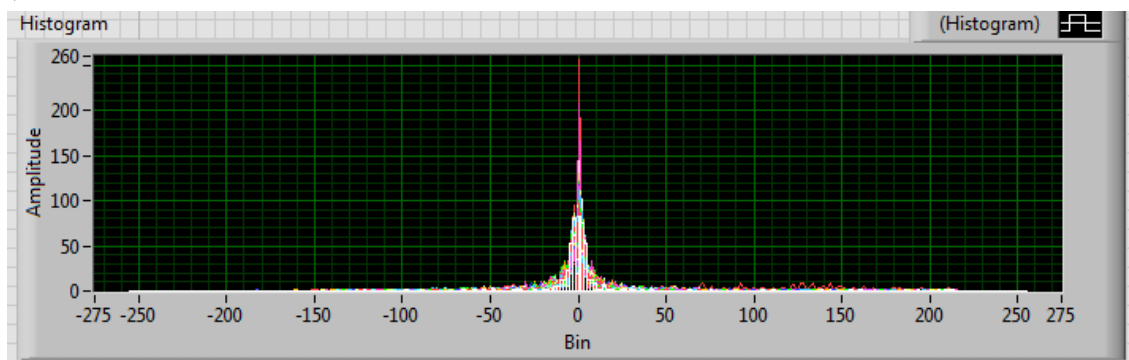
5. Eredmény és kiértékelés

A transzformációnk egy tetszőlegesen megválasztott felbontású képfájl feldolgozásával indul, ezt a képet három darab (Piros, Zöld, Kék) szín sávra bontjuk. A Szín sávok mindegyike 8-biten kódolt, tehát elemei 0-255-ig vehetnek fel értéket. a transzformációnk végén az eredményt 16-biten kódolva kapjuk ám az értékek csupán -255-től +255-ig terjedő értéket vehetnek fel. Az eljárás lényege abban rejlik, ahogyan megváltoztattuk a felvett értékek számosságát. Ezt nagyon szemléletes változás, ha összehasonlítjuk a transzformálás előtti (a.) kép hisztogramját és a transzformálás utániét kép (b.) hisztogramját a 19. ábra segítségével.

a,



b,



19. ábra, transzformálás előtti (a,) és utáni (b,) hisztogramok

Az ábrákon a vízszintes tengely a mátrix rácselemeinek az össze felvett értékét mutatja, a függőleges tengely pedig az értékek előfordulásának számát mutatja.

A hisztogramokból az olvasható ki tisztán, hogy az átalakítás előtt az pixelek értékei viszonylag véletlen eloszlási mintát mutatnak, viszont transzformálás után az értékek túlnyomó része a -50-től +50-ig terjedő értékeket vesznek fel.

Ez még önmagában nem jelent tömörítést, sőt valójában a kapott három adatfájl együttvéve egy kevéssel még nagyobb, is mint a kiindulási képünk volt. Viszont az adat jellege miatt egy dinamikus bitrátájú tömörítési módszerrel a transzformált képünk sokkal nagyobb mértékben tömöríthetővé válik.

B. Inverz- transzformáció lefolyása

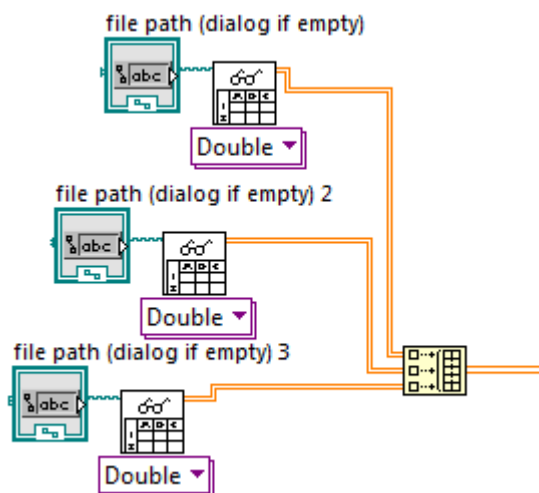
A folyamat nagyban hasonlít a transzformálás lépéseire, lényegében annak fordított lefutása. A bemenet a transzformálásból kapott három fájl, a három szín szerinti átalakított adat fájl. Kimenatként pedig a három fajta natív képformátumot választhatunk, a visszatranszformálás eredményeképpen a három színsávot egyesítjük így az eredeti képpel teljesen megegyező színes képet kapunk.

Az inverz- transzformáló algoritmus teljesen különálló programot képez a transzformáló algoritmustól, nincs szüksége a futáshoz a transzformálás három kimeneti fájlján kívül semmilyen adatra. Ezért egy interneten vagy hálózaton történő adattovábbítás esetén a fogadó félnek elegendő az inverz- transzformáló programmal rendelkeznie a visszaalakításhoz.

1. A transzformált adatfájlok beolvasása

A három szín adatfájljait tallózzuk vagy megadjuk az elérési útvonalukat, majd a (Read From Spreadsheet File.vi) eszköz beolvassa az adatokat és 2D halmazként továbbítja a kimenetén, a három file adatai egy 3D halmazba rendezzük a könnyebb kezelhetőség érdekében. (20. ábra)

Ezt a halmazt bekábelezzük egy (For) ciklusba ezáltal indexelve kapja a halmazokat és egymás után feldolgozza a három színsávot.

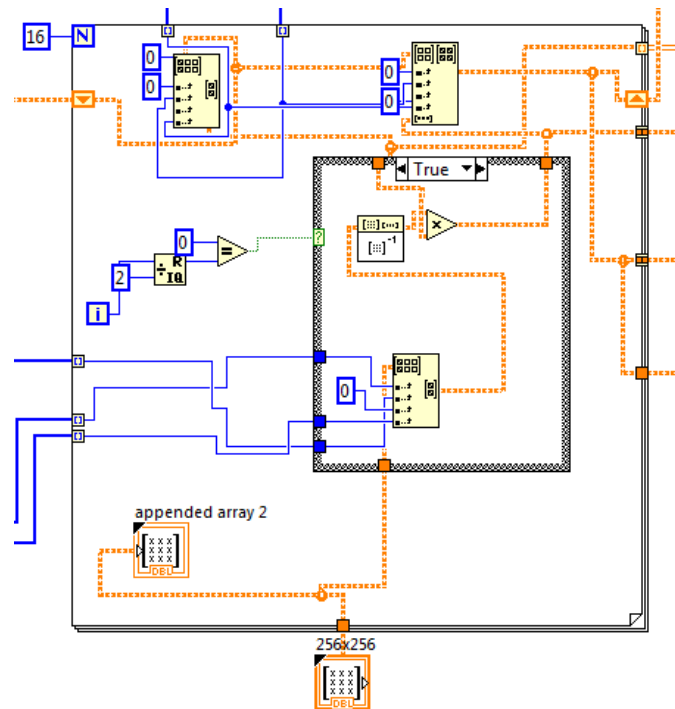


20. ábra, transzformált adatok beolvasása

2. A kép feldarabolása az inverz- transzformációhoz

Hasonlóan a transzformáláshoz a visszaalakítás is egyszerre egy darab 256x256-os méretű mátrixon történik ezért ugyanazzal az eljárással, amit már ismertettünk feldaraboljuk a kép mátrixait. A következőkben ismét csak egy színsáv és egy futási ciklussal foglalkozunk ezentúl a leírás egyszerűségének céljából.

A 21. ábrán látható hogy a visszaalakítás a transzformálás mintájára történik, a lényeges különbség a (Case) struktúrában található, ahol páros futások alkalmával a (T) transzformációs

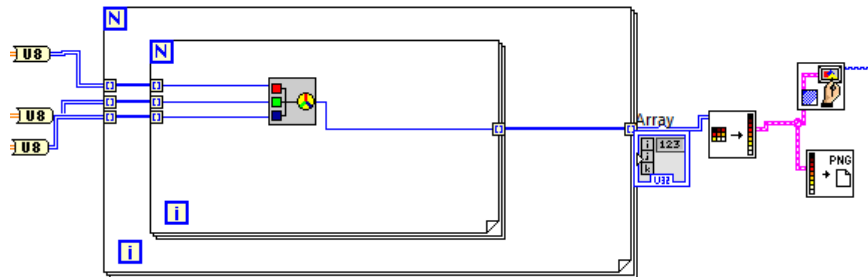


21. ábra inverz- transzformáló ciklus

mátrix inverzét szorozzuk a visszaalakítani kívánt mátrixszal. A páratlan lépések alkalmával pedig a visszaalakítani kívánt mátrixot szorozzuk a (T) transzformációs mátrix transzponáltjának inverzével. A (For) ciklus futásai során a kivágás 2x1-es mátrixszal kezdődik majd 2x2-, 4x2-, 4x4-, 8x4-, 8x8-as mátrixok kivágása követi egymást egészen 256x256-os elemszámig, minden kivágott darab szorzásra kerül a megfelelő (T) mátrixszal és utána a bemeneti mátrix ezekkel az eredményekkel felülírásra kerül. Ezt nyugodtan megtehetjük hiszen a visszaalakítás lépcsőzetesen történik, és szükségünk van minden előző lépés adatára a helyes lefutásához. A folyamat végeztével az adatok továbbításra kerülnek egy VI-hoz amelyik a 256x256-os darabokat újraegyesíti. Ekkor a három színsáv még külön adathalmazt képez, a színsávok egyesítése ezután következik.

3. RGB színsávok egyesítése, a visszaalakított kép mentése

A három színsávot az (RGB To Color.Vi) eszköz segítségével egyesítjük, ez hasonlóan működik az inverz párjához, ezt is egy kétszeres (For) ciklusban kell elhelyeznünk a helyes futáshoz.(22. ábra) Az eszköz végighalad a visszaalakított három színsáv mindegyik elemén,



22. ábra RGB színsávok egyesítése

melyek 8-biten kódoltak és 0-tól 255-ig vesznek fel értékeket. A kimente pedig egy 32-bit integerekből álló halmaz amit a (Flatten Pixmap.Vi) segítségével képadatokká alakítunk és a (Write PNG File.vi) segítségével PNG formátumú képfájllá alakítunk és mentjük a merevlemezre. A PNG-n kívül menthetjük még JPEG és BMP formátumba is a visszaalakított képünket.

IV. Továbbfejlesztési tervek

A. PC-n futtatás optimalizálása

A transzformálás PC-n futtatva tökéletesen működik, az elméletet sikeresen alkalmaztam és működő alkalmazást alkottam belőle. Mindazonáltal lehetséges a program további optimalizálása a jövőben. A program jellegéből fakadóan a ciklusok csak kis adathalmazokkal dolgoznak egyszerre, ezért ha lehetséges, akkor meg szeretném próbálni a program párhuzamosítását. Ha a feldarabolt kép darabjait párhuzamosan lehetne transzformálni, akkor jelentősen felgyorsulhatna a lefutás ideje.

A számítástechnikában nem olyan régen megjelent új technológia a PC-k videokártyájának erőforrásainak kihasználása arra, hogy besegítsen a processzor számításaiba. Ez a technológia különösen nagy potenciával bír a programom szempontjából mivel a videokártyák GPU-ja (Graphic Processing Unit) nagyszámú magokkal rendelkezik, ezáltal sok szálon futtathat egyszerre számításokat, és a videokártya memóriák sávszélessége és írási olvasási sebessége is

általánosan nagyobb, mint a rendszermemóriáé. Így a programom egyszerre több szálon futva még kevesebb időt és erőforrást venne igénybe, valamint a tervezett valós idejű videojel transzformáláshoz lehet, hogy nélkülözhetetlen is.

B. Tervek az FPGA-n történő futtatásra

A következő lépés minden egyéb fejlesztés előtt az FPGA-ra való átírás. A 2015-ös tavaszi félév folyamán mindenképpen szeretném alkalmazni az eljárást FPGA-ra a szakdolgozatom kutatómunkája részeként, meg kell ismernem az FPGA programozás sajátosságait és kifejleszteni a lehető legjobb algoritmust amely képtömörítésre képes ebben az eltérő futtatási architektúrában. Valamint ekkor már szeretnék méréseket végezni a futásidő a memóriaigény és a hatékonyságot illetően.

C. Videojelek feldolgozása

A videojelek feldolgozásához a program futását a lehető legnagyobb mértékben optimalizálni kell majd, hogy ne csak az egyes képkockákat tudja transzformálni, hanem az egymást követő képkockák közötti transzformálás is megvalósulhasson a még nagyobb tömöríthetőség eléréseért. A videojel transzformálást szintén alkalmazni kell FPGA chipre, hogy megvizsgálhassuk az ipari felhasználási lehetőségeket.

V. Felhasználási lehetőségek

A. Video streamelés

A video streamelés egy ma igen elterjedt video megosztási és fogadási módszer az interneten keresztül. A streamelés lényege, hogy ha egy video fájlt akarunk megnézni az interneten keresztül akkor nincs szükség arra hogy az egész videót letöltve azt a merevlemezről nyissuk meg egy video lejátszó program segítségével. Ehelyett a videó megnyitása után a jel folyamatosan érkezik a lejátszó eszközre és lejátszás közben tölti le a videojelet a számítógépünk, ezáltal nem kell, hogy rendelkezünk a video összes adatával elegendő az

éppen nézett részlet és a következő pár másodperc vagy perc adataival rendelkezünk. Ez mára egy jól kiforrott technológia ám ahhoz hogy a szolgáltató többféle felbontásban tudja kínálni a streamelt videót, a szolgáltató szerverein többféle formátumban és felbontásban kell tárolni a videót, és a felhasználó lejátszó eszközétől és internet kapcsolatának sebességétől függően kell küldeni a video adatfolyamát.

A transzformációs módszeremmel lehetséges egy képet visszaalakítani eredeti felbontására, de dönthetünk úgy, hogy a teljes felbontás helyett az eredeti felbontás felére konvertáljuk vissza a képet. Az inverz- transzformálás leírásánál bemutattam, hogy a 256x256-os képeket lépésenként kapjuk vissza kezdve egy 2x2-es majd egy 4x4-es, 8x8-, 16x16-, 32x32-, 64x64-, 128x128- és végül megkapjuk a 256x256-os képet. Ezért lehetséges megállítani a folyamatot az utolsó lépés előtt bármikor, így csökkentve az eredeti képhez képest a visszanyert kép felbontását. A lépések kettőnek a hatványainál állítható meg, mivel ezeknél a lépéseknél nincs képtorzulás a visszalakítás közben. Úgy gondolom, hogy ezzel a módszerrel lehetséges lenne egy olyan video streamelést létrehozni ahol a streamelt video a kiszolgáló szerveren csak egy formátumban kerül tárolásra. Ez az egy video file a lehető legnagyobb felbontású lenne és kérés esetén ezt a nagyméretű video fájlt küldené a felhasználóknak, az internet sebességére és a készülékre való tekintet nélkül. Az adatok viszont olyan sorrendben kerülnének kiküldésre, hogy először a 4x4-es felbontás visszaalakításához szükséges adatok kerülnének sorra, majd ezt követné a 8x8-as felbontáshoz szükséges adatok, majd a 16x16-os adatok, és így tovább a maximális felbontásig. Ezzel a módszerrel azt érhetnénk el hogy a felhasználó a lassú internetkapcsolat mellett is megkapja a videót a lehető legjobb minőségben amit az internetkapcsolata megenged. Ha valamilyen okból ingadozna az internetkapcsolat akkor is dinamikusán változhatna a felbontás aszerint amennyi sávszélesség rendelkezésre áll. A jó internetkapcsolattal rendelkező eszközök, amelyek viszont híján vannak a számítási kapacitásnak, egyszerűen olyan felbontásúvá transzformálják vissza a videojelet amilyenre a számítási kapacitásuk engedi.

B. Computer látás, objektum felismerés, elmozdulás detektor

A video eszközök felbontása évről évre egyre nagyobb, ma már kijelenthető, hogy a HD felbontású video rögzítés, ha nem is alapkövetelmény, de hamarosan azzá fog válni. Ez 1920x1080 pixelt jelent képkockánként, ez már önmagában is hatalmas adatsűrűséget, jelet főleg ha másodpercenként 30 képkocka fölötti rögzítési sebességről beszélünk. De közben

érkezik a következő lépcső a video rögzítés terén, a 4K szabvány, ami képkockánként 4096x3072 pixelt tartalmaz.

Így a videojelek feldolgozása a képfelismerő-, objektumkövető-, mozdulat érzékelő (gesture sensor)-, algoritmusoknak egyre nehezebb dolguk van, egyre nagyobb erőforrásokat igényelnek.

A transzformáló eljárásom ezeknél a nagy felontású képeknél azáltal segíthet, hogy kihasználjuk azt a tulajdonságát, hogy a már leírt módon kis felbontású képtől érkezünk lépésenként a teljes felbontásig. A visszalakatás során a kisebb felbontású még nem teljesen visszaalakított kép is teljes értékű és az említett computer látás algoritmusok használható adatot nyerhetnek arra vonatkozóan, hogy a videojel melyik részén történik valami érdekes, ami kiértékelésre érdemes lehet. A program ezt hamarabb megtudja határozni kisebb felbontás mellett és azokat a részeket melyek a releváns információt tartalmazznak, elsőként dolgozhatja fel.

Másik elképzelésem egy olyan rendszer, amely kis felbontásúra visszatranszformálja a videojelet és nagy felbontásban csak a lényeges rész dolgozza fel, így spórolhatunk az erőforrásokkal vagy megrovidíthatjuk a kiértékelés idejét.

VI. Összefoglalás

A kutatómunka célja az volt, hogy kiderítsem a matematikai alapok melyeket Piros Sándor tanárút dolgozott ki, alkalmazható-e NI LabVIEW programozói platformon. Miután a koncepció megvalósítását bizonyítottam, egy egyszerű 256x256-os méretű Grayscale (szürke árnyalatos) képen, a következő lépésben egy használható alkalmazás elkészítése volt a cél ami bármilyen méretű színes képet képes feldolgozni. A munka nagy része kísérletezéssel zajlott, a projekttel egy időben kezdtem foglalkozni a LabVIEW programozói felülettel, és kezdőként nehezek voltak az első lépések. Ahogyan haladtam a a program fejlesztésével, egyre ismerősebb lett a programozó platform és egyre könnyebb lett a haladás, közben felfedeztem újabb és újabb megoldási lehetőségeket korábbi problémákra. A fejlesztés során ahogyan új megoldásokat találtam az egyes lépések futtatására úgy azokat alkalmaztam a már korábban elkészített programrészekre, ezzel optimalizálva és továbbfejlesztve a program részeit.

Az elkészült programmal több futtatási tesztet végeztem, a program viszonylag kis számítási kapacitást igényel, de eközben nagy mennyiségű memóriát használ fel. Ennek azaz oka, hogy

a memóriát nem üríti a program egészen egy újabb futtatás megkezdéséig, a transzformálás három rétegben történik ezért a teljes képet tömöttetlen formában és hat példányban tárolja a program egészen a transzformáció befejezéséig. Ez kisméretű képeknél nem okoz problémát viszont 4K felbontású képekhez akár 1,5-2 GB memóriát is felemészt. Ennek kiküszöbölésére a jövőben dinamikus memóriakezelést kell beépíteni, hogy egyszerre a memóriában csupán egy 256x256-os kép adatainak tárolására legyen szükség.

A fejlesztés során rengeteg felhasználási lehetőségnek az ötlete adódott magának a program futásának felépítéséből, ezeknek az ötleteknek a megvalósítása következő kutatás keretében folytatódni fog.

VII. Irodalomjegyzék

- [1] Piros Sándor, Peter Korondi.: „Practical Implementation of Second Generation Wavelet Transformation on 2D Images by Transformation Matrixes,” Recent Innovations In Mechatronics, (megjelenés alatt)
- [2] Sarah Betz, Nirav Bhagat, Paul Murphy, Maureen Stengler.: Wavelet-Based Image Compression
https://www.clear.rice.edu/elec301/Projects00/wavelet_image_comp/index.html
letöltés: 2014.09.04
- [3] National Instruments Corporation 2003.: http://sukjaro.eu/ELFT-NI-palyazat/LabView_Manual.pdf letöltve.: 2014.09.11
- [4] M.A. EL-Sayed and T.A.-E. Hafeez, „New Edge Detection Technique based on Shannon Entropy in Gray Level Images,” International Journal on Computer Science and Engineering (IJCSE), vol. Vol. 3, no. NO. 6 June, June 2011.
- [5] C. Croonenbroeck, „Local Entropy Based Image Reconstruction,” Discussion Paper No. 312, January 2012.
- [6] S. Deutsch, A. Averbuch and S. Dekel, „Adaptive compressed Image sensing based on wavelet modelling and direct sampling,” in „SAMPTA’09, Marseille : France, 2009
- [7] N. Shakhakarmi, „Quantative Multiscale Analysis using Different Wavelets in 1D Voice Signal and 2D Image,” IJCSI International Journal of Computer Science Issues, Vols. vol. 9” no. Issue 1, No2” January 2012