Debreceni Egyetem
Informatikai Kar

Vizuális termékellenőrző rendszer kifejlesztése LabVIEW programozási környezetben

Témavezető:
Dr. Szabó István
egyetemi docens

Készítette:
Csontos József
programtervező matematikus

Debrecen
2007.

Tartalomjegyzék

1. Bevezetés

A vizuális termékellenőrzés célja az, hogy a termékek előállítása során felmerülő hibák felismerése automatizálva legyen. Napjainkban erre igen nagy szükség van, vegyük példának a naponta több száz nyomtatott áramkört előállító gyártósorokat, ahol - természetesen terméktől függően - igen nagy lehet az egyes termékek hibafaktora. A termékek automatizált vizuális ellenőrzésének gyakorlati jelentősége az, hogy elősegíti a gyorsabb és olcsóbb előállítást. Mára ennek a technológiai háttere elérhető lett, azaz a digitális képalkotás minősége és a számítógépek sebessége elérte azt a szintet, hogy a képek minősége a vizuális ellenőrzéshez megfelelő és a képeken végrehajtott vizsgálat sebessége valós idejű legyen.

Ezen diplomamunka célja a National Instruments cég fejlesztő eszközeit felhasználva a termékellenőrzéshez elengedhetetlen képfeldolgozási és elemzési módszerek tárgyalása, valamint egy vizuális termékellenőrző rendszer képfeldolgozó moduljainak az elkészítése.

A dolgozat első része a LabVIEW technológia ismertetéséről szól: hogyan néz ki a fejlesztői környezet, melyek a G adatfolyam programozási nyelv alapelemei, milyen eszközöket biztosít a LabVIEW egy projekt elkészítéséhez, hogyan néz ki az NI-IMAQ architektúra, valamint bemutatásra kerülnek a Vision szoftverek.

A második rész az NI Vision alapfogalmait ismerteti: a kép belső reprezentációja, a ROI és inzert definíciójának részletes megismerése fontos ahhoz, hogy az NI Vision képfeldolgozási és elemzési algoritmusainak működése érthető legyen.

A diplomamunka további fejezetei ezen eljárások közül a fontosabbakat mutatja be, példaképekkel szemlélteti működésüket, valamint olyan NI Vision-specifikus fejezetekre is kitér, mint például a részecske elemzés és a gépi látás.

A dolgozat végén a készített vizuális termékellenőrző rendszer (Visual Product Inspection System, VPIS) képfeldolgozó moduljainak felépítéséről lesz szó, és bemutatásra kerül két elkészített modul is, melyek a vizsgált termékeken található komponensek meglétének ellenőrzését és a vonalkódok lokalizálását, leolvasását végzik.

2. A LabVIEW ismertetése

A **LabVIEW** (**Lab**oratory **V**irtual **I**nstrument **E**ngineering **W**orkbench) a **National Instruments** által kifejlesztett platform, fejlesztői környezet és vizuális programozási nyelv egyben. Első verziója 1986-ban, eredetileg Apple Macintosh-ra készült. Későbbi verziói számos platformon (Microsoft Windows, UNIX, Mac OS) elérhetők, a legújabb verziója a 8.20-as, mely a LabVIEW 20. évfordulójának tiszteletére készült.

A LabVIEW külön érdekessége, hogy benne nem szövegalapú forráskódot írunk, hanem egy úgynevezett blokk diagramot rajzolunk. Ez lesz a „forráskód". Ezt az eszközt kifejezetten mérnökök számára mérési, automatizálási és folyamatirányítási célokra fejlesztették ki. Segítségével úgynevezett virtuális műszereket (**V**irtual **I**nstrument, **VI**) hozhatunk létre. Miért is jó ez? Képzeljünk el például egy gyártósort, ahol az egyes gyártási fázisokban a termékek fizikai paramétereinek tesztelésére számos mérőműszert használnak, amelyek ára elég magas, ráadásul sok helyet foglalnak el. Ha valamelyik alkatrészük meghibásodik, azt ki kell cserélni (vagy rosszabb esetben új berendezést kell vásárolni). Egyszóval az ilyen megoldásnak sok hátránya van. Erre a problémára adnak választ a virtuális műszerek. Azaz fogjunk egy számítógépet, tegyünk bele néhány digitalizáló, illetve jelgeneráló kártyát, telepítsük fel rá a LabVIEW-t, és máris elkészíthetjük a megfelelő virtuális műszereinket.

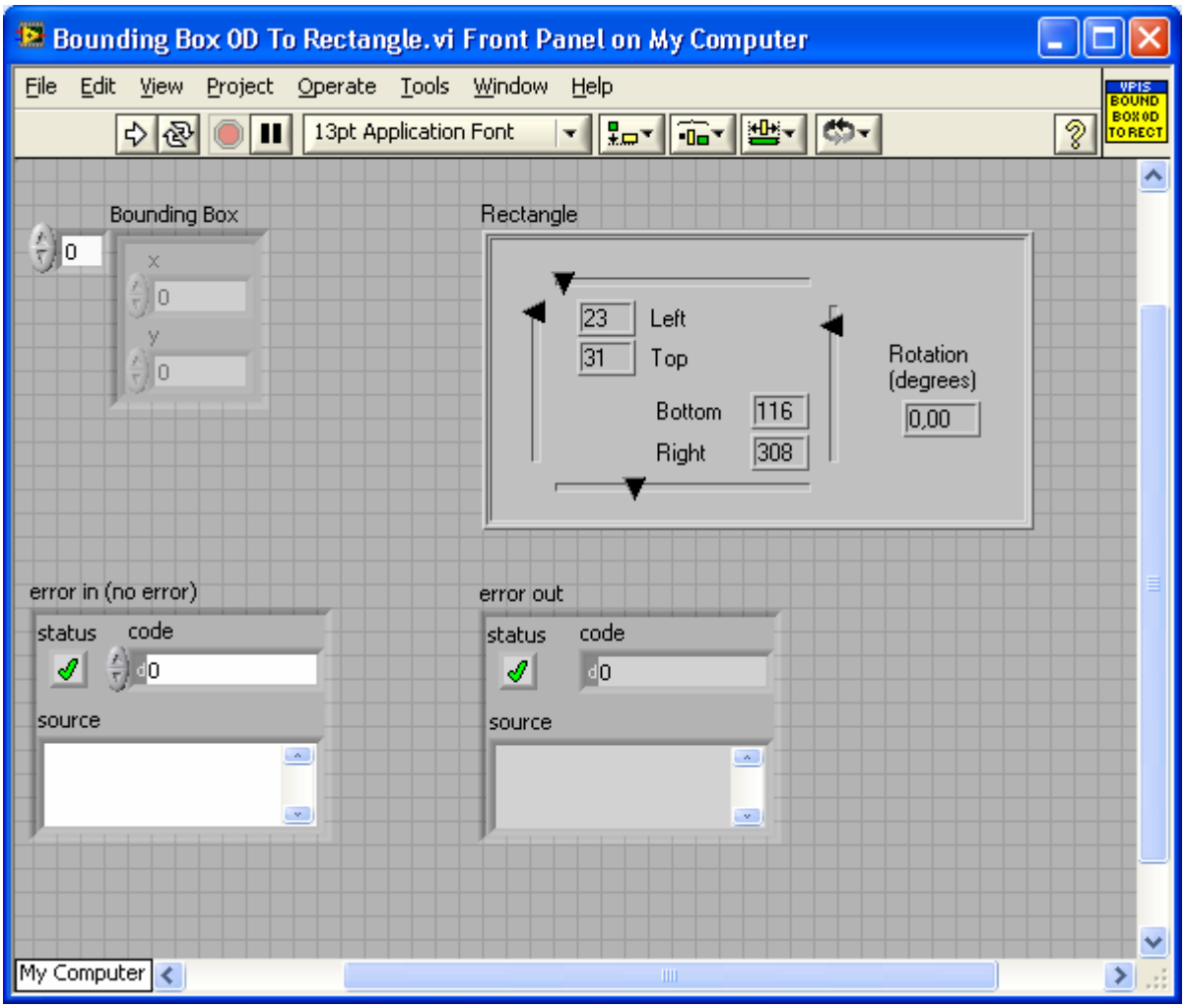
A LabVIEW programozási egysége a VI. A VI 3 fő részből áll, melyek a következők: előlap (**Front Panel**), blokk diagram (**Block Diagram**) és egy ikon, amely magát a programozási egységet jelképezi. Az előlapon hozható létre a felhasználói interfész, melynek kialakításakor számos komponensből válogathatunk. A másik fő rész a már fentebb említett blokk diagram.

2.1. Alapfogalmak

2.1.1. Az előlap

Az előlapi komponenseknek két nagy csoportja van: kontrollok és indikátorok. Az előbbiek a bemenő adatok bevitelére szolgálnak, míg az utóbbiak a kimenetet jelenítik meg.

Ezeket a komponenseket a kontroll palettán találjuk. Itt találhatóak még a díszítő elemek is, melyekkel szebbé tehetjük a programunk felhasználói felületét.



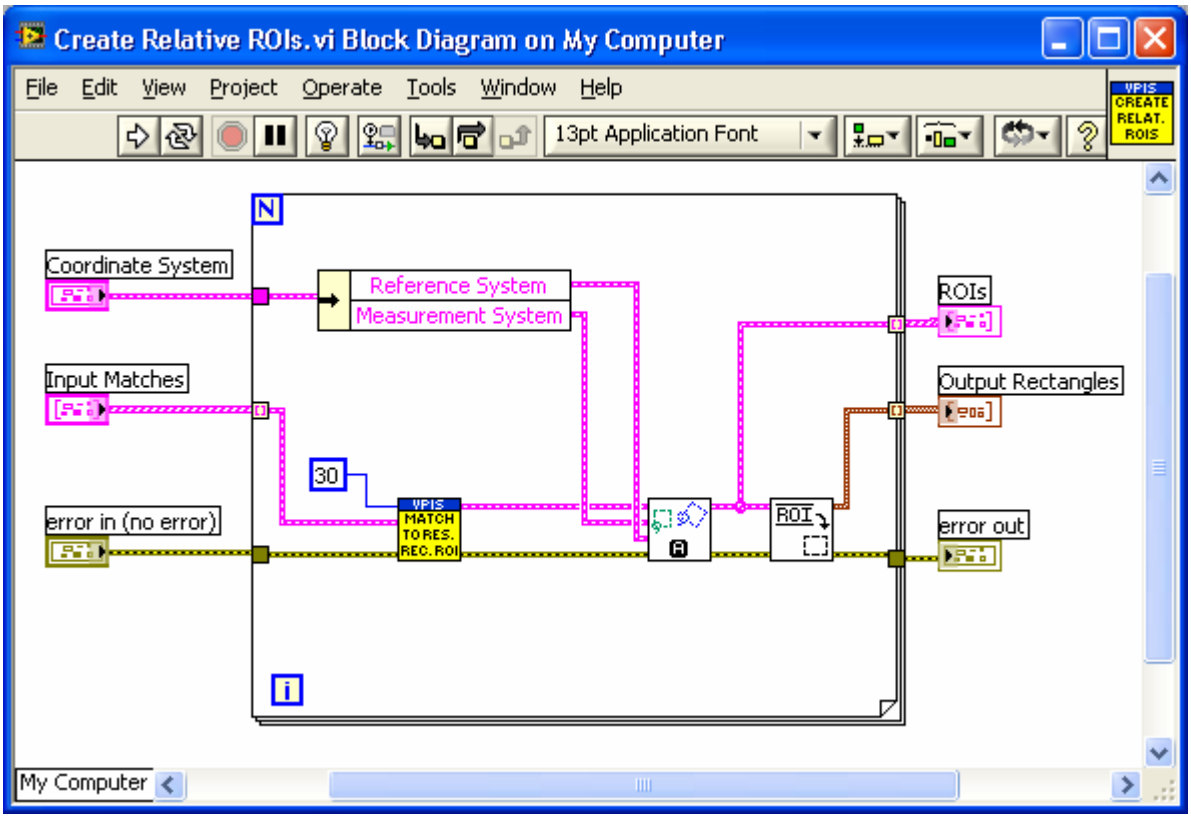
1. ábra. *Előlap*

2.1.2. A blokk diagram

A LabVIEW egy adatfolyam programozási nyelvre épül, melyet **G**-nek hívnak. A blokk diagramon - amely tulajdonképpen a forráskód - a programozó úgynevezett csomópontokat köt össze huzalokkal. A blokk diagramon a csomópontok egyszerű vagy összetett műveletek végrehajtására szolgálnak. Több fajtájuk van: függvények, struktúrák és SubVI-ok. Ezeket a függvény palettán találjuk. Egy csomópontnak nulla, egy vagy több bemenete és nulla, egy vagy több kimenete lehet. A csomópontokat huzalok kötik össze, melyek az adatáramlást biztosítják. A csomópontok bemeneti és kimeneti pontjait

terminálóknak nevezzük. Egy csomópont akkor hajtódik végre, ha minden adat elérhető a bemeneti terminálokon. Ha egy adott pillanatban több ilyen csomópont is van, akkor azok elvileg párhuzamosan futnak le. A gyakorlatban ezek sorrendje indeterminisztikus.

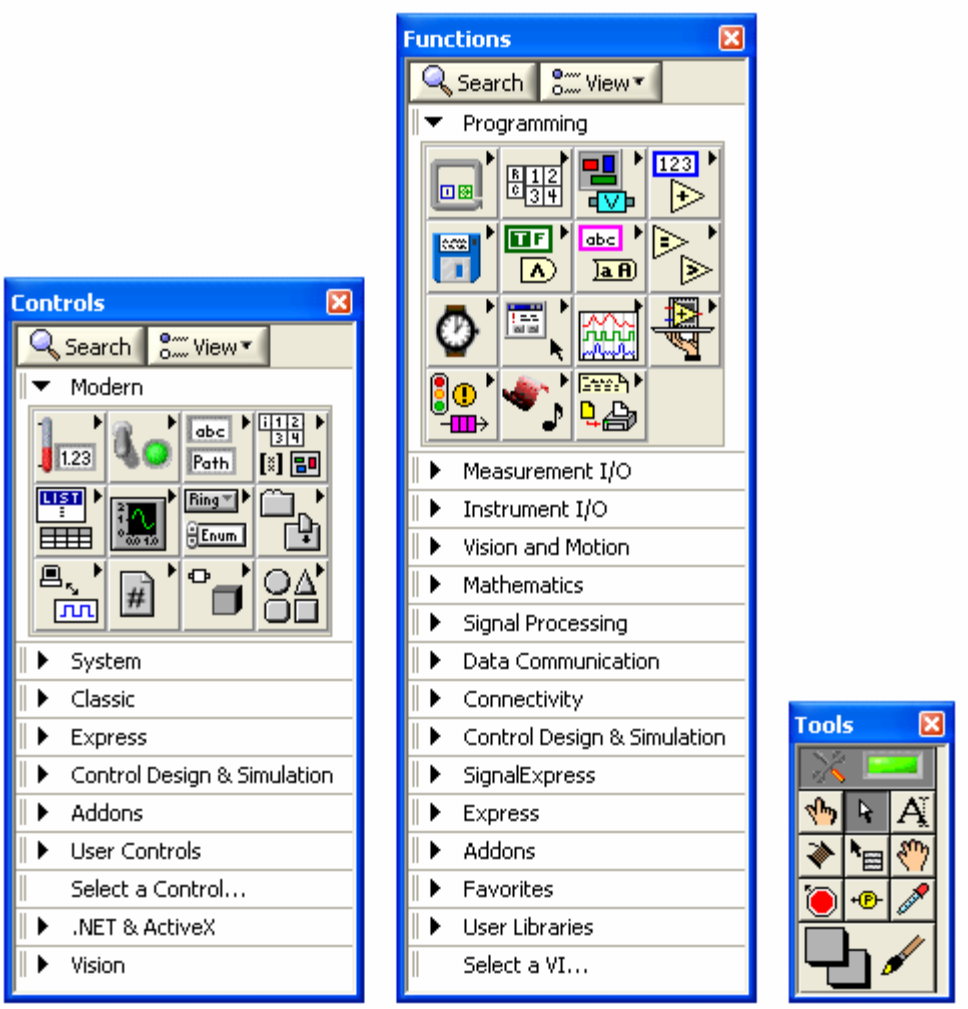
Mindezek miatt a LabVIEW-ban nagyon könnyű megvalósítani a párhuzamos programozást, ugyanis a többszálú működéshez szükséges nyelvi eszközök implicit jelen vannak, viszont a futási sorrend biztosítására további erőfeszítéseket kell tennünk. A programvégrehajtást a grafikus blokk diagram szerkezete határozza meg. A blokk diagram az adatáramlási sorrendben hajtódik végre, azaz az adat egy csomópont kimeneti termináljáról az adott csomóponttal közvetlen összehuzalozott bemeneti termináljára érkezik. Vagyis tévhit, hogy az adatáramlás balról-jobbra irányú, ez utóbbi csak egy konvenció, ami bár nem kötelező, de módszertanilag helyes ezt az irányzatot követni. Az adatok típusát a vezetékek színe és vastagsága jelzi.



2. ábra. *Blokk diagram*

2.1.3. Az eszközök paletta

Az eszközök palettát mindkét ablaknál tudjuk használni, segítségével az előlapon és a blokk diagramon található objektumokat tudjuk módosítani. Amikor LabVIEW-ban programozunk szinte mindig az egeret kell használnunk, ezért meg kell ismerkednünk az egérkurzor funkcióival. Az ezek közötti váltásra szolgál ez a paletta. Az eszközök neve föntről lefelé és balról jobbra haladva a következő: automatikus eszközválasztás, működtető eszköz, pozícionáló, címkéző, huzalozó, gyorsmenü eszköz, görgető eszköz, megszakítási pont, próba pont, színmásoló, színező.

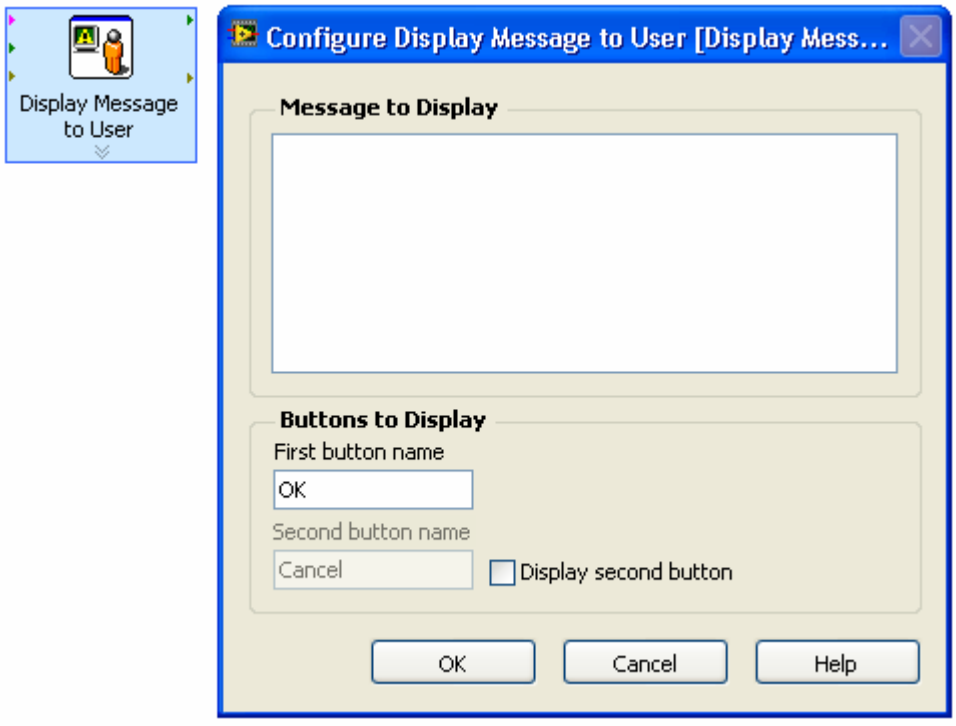


3. ábra. *Kontroll-, függvény- és az eszközök paletta*

2.1.4. Csomópontok

A csomópont tehát a VI végrehajtási egysége. A legegyszerűbb csomópont a függvény, amely a blokk diagram alapvető építőeleme. Egy függvénynek mindig halványsárga hátterű és fekete előterű ikonja van. Nem rendelkezik előlappal és blokk diagrammal, szemben a **SubVI**-al. Ez utóbbi az alprogramnak felel meg. Alkalmazásuknak számos előnye van. Modulárissá, jól átláthatóvá teszi a programunkat, amely megkönnyíti a tesztelést. Emellett az újrafelhasználhatóságot is támogatja.

A függvényekhez hasonlóan a LabVIEW-ban számos beépített SubVI létezik, azonban mi is könnyedén hozhatunk létre újakat. Itt kell megemlíteni a 7.0-ás verzióban bevezetett **Express VI**-t, amely a fejlesztő dolgát könnyíti meg azzal, hogy nemcsak huzalozással, hanem interaktív dialógusablakkal is konfigurálható.



4. ábra. *Express VI és a hozzá tarozó dialógusablak*

A csomópontok harmadik csoportját a struktúrák alkotják. Ezek a G nyelv vezérlőszerkezetei. A blokk diagramon úgy jelennek meg, mint egy keret. Egy struktúra által körülhatárolt diagram részt szubdiagramnak nevezünk.

A vezérlőszerkezetek első csoportját a szekvenciát megvalósító struktúrák alkotják. Közös jellemzőjük, hogy egy vagy több egymást követő keretből állnak, és az ezekben lévő szubdiagramok szekvenciálisan hajtódnak végre. Két fajtája van: a **Stacked Sequence Structure** és a **Flat Sequence Structure**. Az előbbiben a keretek – mint a kártyapakliban a lapok – egymást fedve helyezkednek el, míg az utóbbiban egymás mellett – mint a filmszalagon a képkockák – sorakoznak.

A második, szelekciót megvalósító csoportba három fajta struktúra tartozik: **Case Structure**, **Diagram Disable Structure** és **Conditional Disable Structure**. Szintén egy vagy több keretből állnak, és az elhelyezkedésük a Stacked Sequence Structure-höz hasonlatos. A Case Structure működése: az input terminálon megjelenő értéktől függően hajtódik végre valamelyik szubdiagram. A másik két struktúrának nincs input terminálja, a blokk diagram egy részének a letiltására valók.

A harmadik csoportot az iterációt megvalósító struktúrák alkotják. Ezek egyetlen keretből állnak, melyben elhelyezkedő szubdiagram a ciklusmag. Előírt lépésszámú ciklust a **For Loop**-pal hozhatunk létre. A számláló terminájára a kívánt lépésszámot kell huzaloznunk, az iterációs terminálja pedig ciklusszámlálóként szolgál. A lépésszámot azonban úgy is megadhatjuk, hogy egy tömb típusú bemenetre engedélyezzük az indexelést (Enable Indexing). Ekkor a tömbből lejön egy dimenzió, vagyis 1D tömbből skalár érték lesz (a tömb i. eleme), 2D tömbből pedig 1D tömb (a tömb i. sora). Az előbbi esetben minden elemre, az utóbbiban pedig minden sorra lefut a ciklus. Ugyanezt el lehet játszani a kimenettel is, ekkor a kimenet plusz egy dimenziós tömbként kerül ki. A **While Loop**-nak szintén két terminálja van. Az egyik a már ismertetett iterációs terminál, a másik a feltételes terminál, amely egy logikai értéket kap, és ez alapján állítja le a ciklust. Beállíthatjuk, hogy a ciklust igaz vagy hamis érték esetén állítsa-e le. A 7.1-es verzióban jelent meg a **Timed Loop**, amely az egyes iterációs lépéseket meghatározott időközönként hajtja végre.

A ciklusok alkalmazásakor gyakran szükség van olyan tárolóra, amelyben a ciklus minden egyes végrehajtásakor ki tudjuk olvasni annak az előző végrehajtáskori értékét, és a módosított értékét bele tudjuk írni. Erre két eszköz áll rendelkezésünkre: a shift regiszter (**Shift Register**) és a visszacsatoló csomópont (**Feedback Node**). A két eszköz működése egymással teljesen kompatibilis, és egymással tetszőlegesen felcserélhetőek. [cikk]

A negyedik csoportba az **Event Structure** tartozik, amely segítségével az eseményvezérelt programozást lehet megvalósítani a LabVIEW-ban.

Végül meg kell említenünk a **Formula Node** és **MathScript Node** struktúrákat. Előbbi segítségével - a C programozási nyelv szintaxisához hasonló módon megadott - matematikai kifejezéseket tudunk kiértékelni, míg az utóbbi - MATLAB szintaxisszerű - matematikai szkriptek kiértékelésére való.

2.2. Kódgenerálás

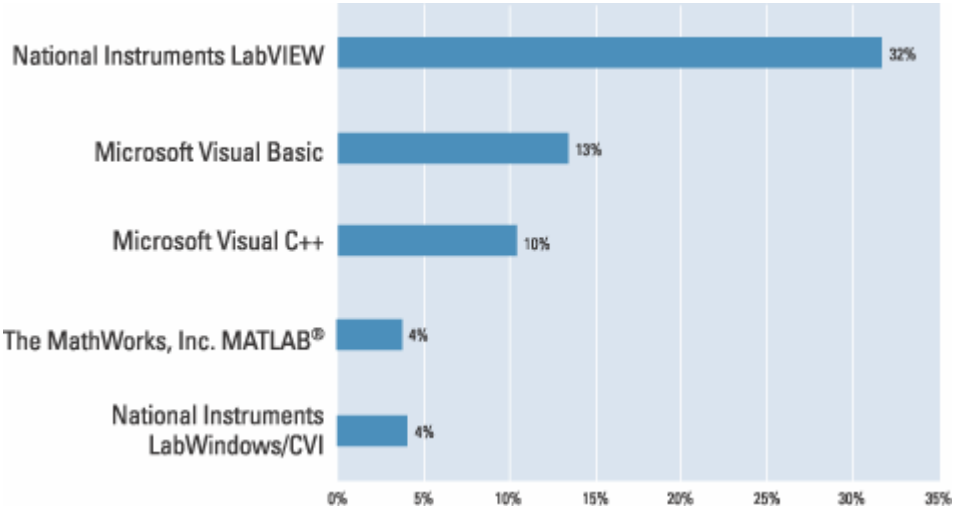
A LabVIEW nem egy interpretált nyelv, a beépített fordítója a C programokhoz hasonló futási sebességű natív kódot állít elő. Ezért futásidő-kritikus rendszerek gyors megvalósítására is az egyik legalkalmasabb fejlesztőeszköz. Mialatt a blokk diagramot szerkesztjük, a LabVIEW szigorúan ellenőrzi a grafikus kód szintaxisát. Futtatáskor vagy a kód mentésekor megtörténik a végrehajtható gépi kód generálása. Ez utóbbi esetben a lefordított kód mellé bekerül a forráskód is. A futtatható állomány a futásidejű motor (LabVIEW run-time engine) segítségével kerül végrehajtásra. A futásidejű motor alkalmazásának előnye az, hogy csökkenti a forráskód fordítási idejét és egy konzisztens felületet szolgáltat különböző operációs rendszerek, grafikus rendszerek vagy hardverkomponensek esetén, tehát hordozható kódot tudunk generálni a segítségével.

2.3. A programok futtatása és tesztelése

Az elkészült programokat az eszköztár futtatás gombjával tudjuk elindítani. Ha a programunk nem tartalmaz szintaktikai hibát, akkor ezen a gombon egy fehér nyíl látható, amely futás közben szaggatott fekete nyílra vált. Ellenkező esetben egy töredezett nyíl ábrája lesz látható. Ha ekkor a gombra kattintunk, megjelenik egy ablak a hibák listájával.

Teszteléskor jó szolgálatot tehet még az állandó futtatás gomb, melynek hatására a VI addig fut, amíg a végrehajtás leállítása gombbal le nem állítjuk, vagy a leállítás/folytatás gombbal átmenetileg felfüggesztjük a program végrehajtását. Ez a funkció főleg akkor hasznos, ha egy VI futását többször egymás után, különböző inputokra akarjuk megfigyelni. Az egyik leghatékonyabb tesztelési módszer a végrehajtás nyomkövetése. Ha aktiváljuk, akkor a vezetékeken kis buborékok fogják jelezni az adatáramlást, és a csomópontok által kiadott értékeket is megjeleníti. Mint a komolyabb fejlesztői környezetekben, itt is lehetőség van a program lépésenkénti végrehajtására, amit három gombbal szabályozhatunk. Mindezen

felül a vezetékeken elhelyezhetünk próba- és megszakítási pontokat is, melyek szintén nagymértékben megkönnyítik a hibakeresést.

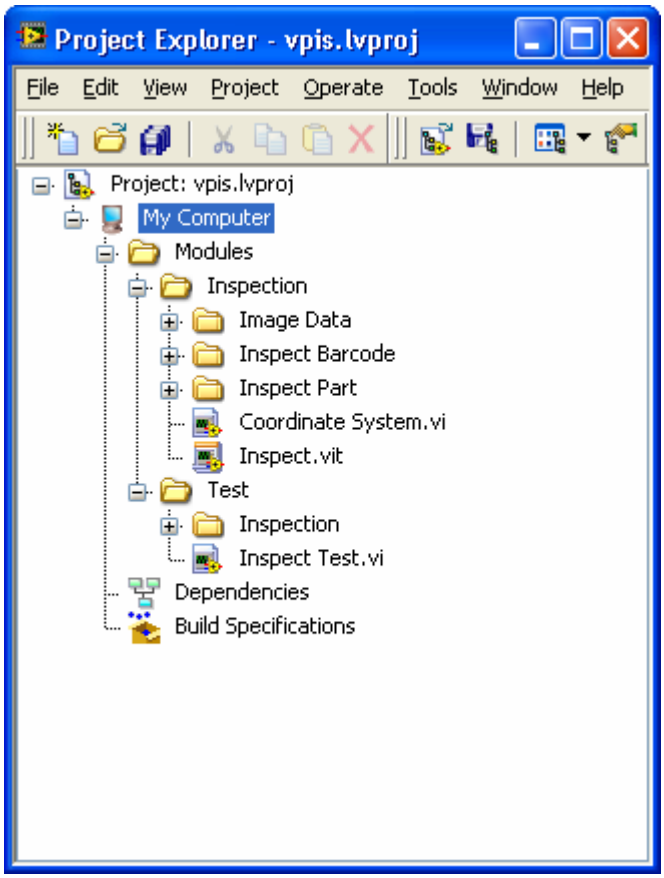


5. ábra. *Adatgyűjtésre és automatizálási célokra használt fejlesztői eszközök aránya*

3. Projektszervezés LabVIEW-ban

A LabVIEW 7.1-es, és az ettől korábbi verziókban a nagyobb alkalmazásokhoz tartozó állományok kezelése meglehetősen nehézkes volt, ugyanis ezen állományok projektbe szervezését manuálisan kellett megoldani. Ezen fájlok közé nem csak a VI-ok, hanem az adatállományok, hardverkonfigurációs beállítások, valamint a projekt dokumentációja is hozzátartozik. A 8.0-ás verzióban bevezetésre kerülő projektböngésző (**Project Explorer**) segítségével ezen fájlok kezelése és szervezése sokkal egyszerűbbé vált.

A projektböngésző ablaka négy fő részre osztható, ezek a komponensek egy hierarchikus listában találhatóak.



6. ábra. *Project Explorer*

Első ezek közül a projekt gyökere (**Project Root**), ez a többi komponenst tartalmazza, neve pedig megegyezik a projekt fájlnevével. A projekt elmentésekor a LabVIEW egy XML

formátumú projekt állományt (.lvproj) hoz létre, ami hivatkozásokat tartalmaz a projektbe tartozó állományokra, ezenfelül a fordítási és konfigurációs információk is itt tárolódnak.

A második a végrehajtó eszköz (**Target**), ami megfelelő kiegészítő modulok és illesztőprogramok hiányában (pl.: RT, FPGA, PDA, érintőképernyő, DSP vagy beágyazott modul) a saját lokális számítógépet, mint célhardvert jelenti a készülendő alkalmazás számára. Természetesen a projektböngésző lehetőséget biztosít az elkészült alkalmazásnak a megfelelő végrehajtó eszközre történő feltelepítésére is.

A harmadik komponens (**Dependencies**) segítségével azon elemeket tudjuk megtekinteni, amelyektől a projektben szereplő VI-ok függenek. Minden egyes végrehajtó eszközhöz külön függőségi viszony tartozik.

Az utolsó egység a projektböngészőben a fordítási specifikációk (**Build Specifications**) megadására szolgál. Itt tudjuk megadni, hogy az elkészült alkalmazást milyen formában szeretnénk terjeszteni. Ez lehet egy különálló futtatható állomány – ami természetesen igényli a LabVIEW végrehajtó motort -, telepítőkészlet, dinamikusan kapcsolódó könyvtár (DLL), forráskód-disztribúció vagy egy szimpla tömörített állomány.

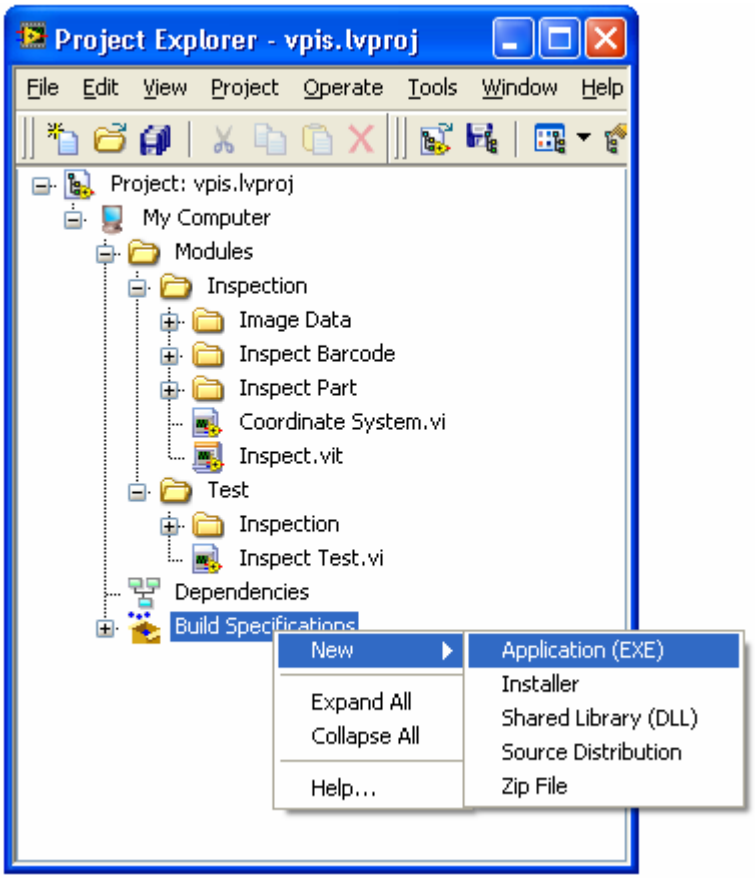
A LabVIEW projektböngészője a kapcsolódó állományok csoportosítását virtuális mappák segítségével valósítja meg. Ezen mappák logikai jellegűknél fogva nincsenek hatással a tárolóeszközön található állományokra és könyvtárakra. A fizikai állománystruktúra és a LabVIEW által kezelt logikai projektstruktúra szétválasztásának előnye az, hogy ugyanazon kódot több projektben is felhasználhatjuk és minden egyes projektben sajátos mapparendszert alakíthatunk ki kódunknak. Ez a sajátosság tehát lehetővé teszi a független fájlszervezést, de természetesen ez hátrány is lehet, ugyanis a fejlesztőnek - más integrált fejlesztői eszközökkel ellentétben - manuálisan kell a tárolóeszközön megjelenő könyvtárakat létrehoznia.

3.1. Az alkalmazáskészítő (Application Builder)

Az elkészített alkalmazásunk rengeteg állományból állhat. Ide nemcsak a végrehajtható állományok - jelen esetben VI-ok - tartoznak, hanem egyéb fájlok is, például adatbázisok és dokumentációk is. Ezeket az állományokat manuálisan összegyűjteni és egy disztribúciós csomagot készíteni fárasztó feladat lenne. Ugyanakkor problémát jelent az is, hogy a végfelhasználó számítógépén valahogyan biztosítani kellene azt, hogy mind azok a konfigurációs beállítások és modulok, amelyek a fejlesztő számítógépén rendelkezésre állnak, ugyanolyan formában megtalálhatók legyenek. Erre a problémára szolgáltat megoldást a

LabVIEW alkalmazáskészítője, amelynek segítségével a projektünkhöz tartozó fájlokból könnyedén készíthetünk disztribúciót.

Az alkalmazáskészítő használata során a fordítási specifikációk megadásánál említett lehetőségek közül választhatunk. Ezek közül a leggyakrabban használt választási lehetőség a különálló futtatható állomány (**Application**) készítése.



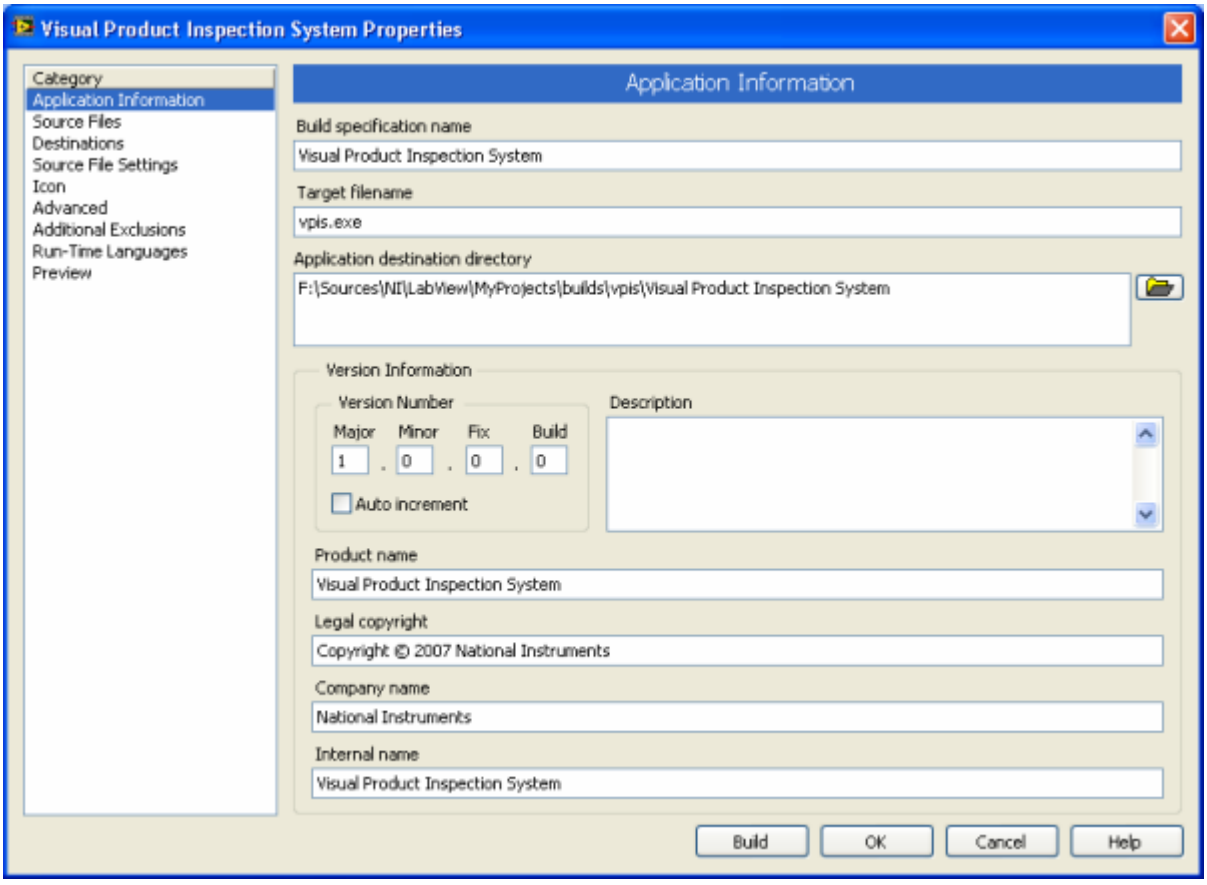
7. ábra. Az *alkalmazáskészítő indítása*

Miután kiválasztottuk az új alkalmazás készítésének lehetőségét, megjelenik az alkalmazáskészítő felülete. Ezen a felhasználói felületen a kívánt specifikáció beállításait tudjuk elvégezni. Egy projekthez több ilyen specifikáció tartozhat, amik a projektünk részeként kerülnek elmentésre.

3.1.1. Az alkalmazáskészítő tulajdonságai

Először is az alkalmazással kapcsolatos információkat adhatjuk meg, mint például a leendő futtatható állomány nevét, verzióját és rövid leírását.

A következő lépésben tudjuk meghatározni azt, hogy projektbe tartozó forrásfájlok melyike kerüljön fordításra a futtaható állományba. Kétfajta csoportba tudjuk tehát a megadandó fájlokat sorolni. Az első csoportba (**Startup VIs**) azokat a VI-okat kell megadnunk - legalább egyet kötelező megadni -, amelyek az alkalmazás indulásakor automatikusan megnyitásra kerülnek és ezen VI-ok fognak a futtatható állományba fordítódni. A másik csoportba sorolt (**Dynamic VIs and Support Files**) VI-ok és egyéb fájlok - például illesztőprogramok, súgófájlok vagy .NET alapú komponensek - amiket az alkalmazás használ, nem kerülnek bele közvetlenül a futtatható állományba, meghívásuk dinamikusan történik.



8. ábra. *Az alkalmazáskészítő tulajdonságai*

Lehetőségünk van arra, hogy célkönyvtárakat adjunk meg az alkalmazásnak. Alapértelmezett esetben a LabVIEW alkalmazáskészítője két könyvtárat ajánl fel, ahová a

fájlok kerülhetnek, de van lehetőség további könyvtárak létrehozására is. Ez egy nagyon jól használható lehetőség, ugyanis komplex szerkezetű szoftverek esetén - amik akár több száz állományból is állhatnak - az alkalmazás könyvtárszerkezete átláthatóbb lesz.

A célkönyvtárak megadása után, a forrás VI-ok és az ezeket tartalmazó mappák tulajdonságait tudjuk beállítani. Például, ha a projektben szerepel egy olyan VI, aminek a feladata a különféle hibaüzenetek megjelenítése, akkor erre a VI-ra beállíthatjuk, hogy legyen modális, azaz a felhasználó addig nem tudja a többi LabVIEW ablakot elérni, amíg a hibaüzenetet megjelenítő ablak be nem zárult. Természetesen nem csak a modális tulajdonság állítható be, hanem többek között a menü-, eszköz-, címsor és görgetősávok láthatósága és automatikus középre helyezés is módosítható itt.

A speciális beállítások kategóriában tudjuk beállítani például azt, hogy a futtatható állomány parancssori paramétereket tudjon fogadni, vagy a hibakeresést szeretnénk engedélyezni az alkalmazásban. A hibakeresés engedélyezésével a VI-ok blokk diagrammjait is tartalmazni fogja a futtatható állomány. Egy LabVIEW-ban készített alkalmazásban – ezen opció használatával – akár a hálózaton keresztül is kereshetünk hibákat a LabVIEW beépített hibakeresési eszközeit felhasználva. Ebben a kategóriában lehet kiválasztani azt is, hogy a fejlesztés során használt egyéni hibakód listát a LabVIEW az alkalmazásba másolja-e át.

Az alkalmazáskészítő továbbá lehetőséget biztosít az alkalmazás ikonjának módosítására és a LabVIEW futásidejű motor alapértelmezett nyelvének módosítására is.

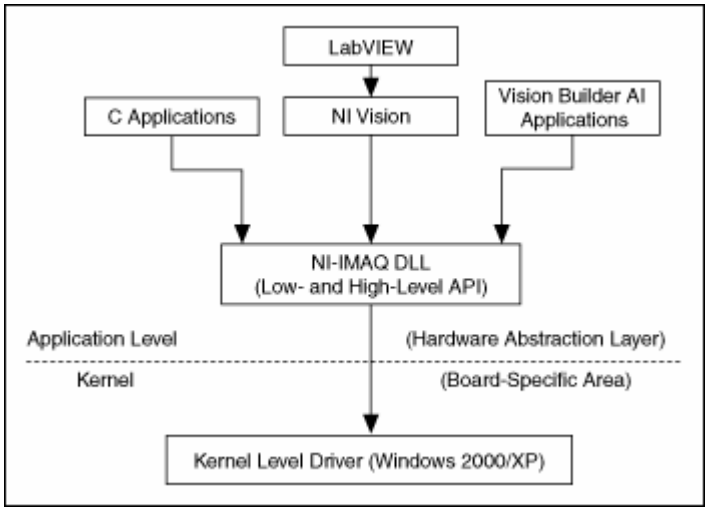
Miután a forrásállományokat és mappákat a követeléseknek megfelelően konfiguráltuk, az előnézet funkció segítségével ellenőrizhetjük, hogy minden szükséges állomány be fog-e kerülni az alkalmazásba és a kialakított mapparendszer megfelelő-e.

A kód sikeres lefordítása után lehetőségünk van arra, hogy telepítőprogramot (**Installer**) készítsünk az alkalmazásból. Ennek előnye az, hogy például az alkalmazáshoz szükséges illesztőprogramok és a megkövetelt konfigurációs beállítások is együtt kerülnek a célszámítógépre, így nem kell minden egyes gépen a beállításokat manuálisan elvégezni.

4. Képfeldolgozás NI alkalmazások segítségével

4.1. NI-IMAQ

Az NI-IMAQ segítségével valósítható meg a képgyűjtő eszköz és az alkalmazások közti kapcsolat. Az NI-IMAQ számos National Instruments által készített környezetből és egyéb programozási nyelvekből is elérhető és használható képgyűjtési, valamint analizálási célokra.



9. ábra. *Az NI-IMAQ architektúrája*

Az architektúra egy hardver-absztrakciós réteget tartalmaz, elkülönítve így a szoftveres API képességeket a hardverspecifikus információktól. A réteg előnye abban rejlik, hogy egy új képgyűjtő hardver használatba állításakor nem kell a teljes programkódot újrafordítani.

4.2. A Vision szoftvercsalád

Az ipari termékellenőrzés és képfeldolgozás területén a National Instruments alkalmazásai majd egy évtizede vezető szerepet töltenek be. Az NI Vision szoftvereknek kétféle változata létezik: az egyik a **Vision Development Module**, a másik pedig a **Vision Builder for Automated Inspection**.

4.2.1. Vision Builder for Automated Inspection

A Vision Builder AI egy olyan interaktív szoftverkörnyezet, amely lehetőséget nyújt automatizált ellenőrzési alkalmazások kifejlesztésére, konfigurálására és sebességmérésére programozás nélkül. Számos esetben nincs szükség komplex algoritmusra és vezérlési szerkezetekre ahhoz, hogy egy képfeldolgozó programot készítsünk. Ebben az esetben a Vision Builder AI-ban megtalálható képfeldolgozó egységek - mint például a szintre vágás, mintaillesztés és karakterfelismerés (OCR) – használatával rövid idő alatt el tudjuk készíteni a kívánt szkriptet, mint a felmerült probléma megoldását.

Az elkészült szkript - ami egy vbai kiterjesztésű állomány – fogja tartalmazni a feldolgozási algoritmus minden egyes lépését. A Vision Builder lehetőséget biztosít arra is, hogy az elkészült szkriptből LabVIEW VI-t generáljunk.

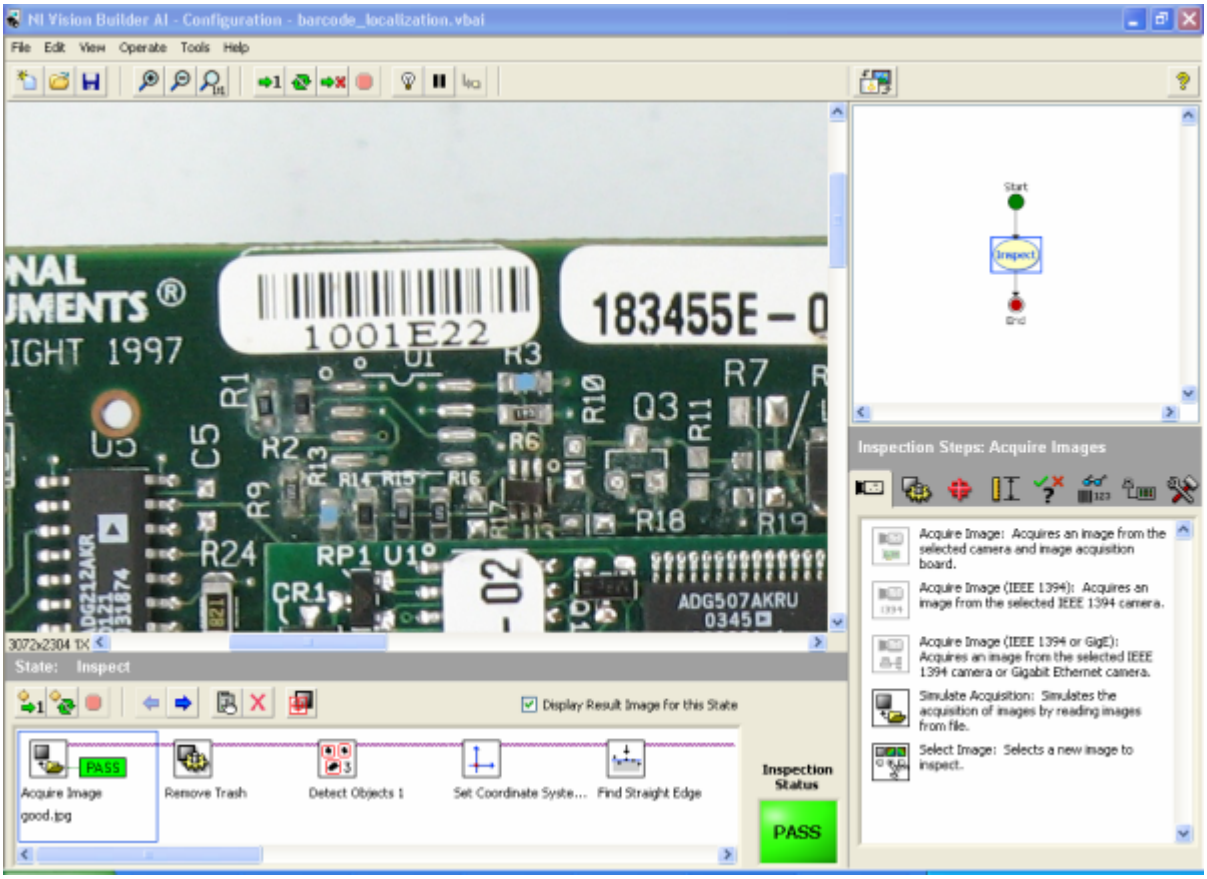
A Vision Builder AI kétféle - konfigurációs és ellenőrzési - módban képes működni. Az ellenőrzés konfigurálására és tesztelésére a konfigurációs felület, az elkészült megoldás felhasználására pedig az ellenőrzési felület használható.

Az ellenőrzések indítása mindkét felületen történhet. Lehetőség van arra is, hogy az összeállított lépéssorozatot nemcsak egyszer, hanem akár ciklikusan is futtathatjuk vagy az első hibás eredményig.

4.2.1.1. A konfigurációs felület

A konfigurációs interfész négy főbb részre osztható: főablak, áttekintő ablak, ellenőrzési lépések paletta és az állapot-konfigurációs ablak.

A főablakban látható a feldolgozandó kép, egyes ellenőrzési lépések tulajdonságai vagy az ellenőrzés állapot diagramja. Itt lehet definiálni a képen ROI-kat, egyes lépések paraméterei itt adhatók meg és az állapotdiagram itt szerkeszthető. Az áttekintő ablakban vagy a feldolgozandó kép miniatűr nézete vagy az ellenőrzés állapotdiagramja látható. Az ellenőrzési lépések paletta a Vision Builder-ben található képfeldolgozási egységeket mutatja. Ezen modulok használhatóak fel a készülendő ellenőrzésünkben. Az állapot-konfigurációs ablakban pedig az egyes lépéseket tekinthetjük meg.

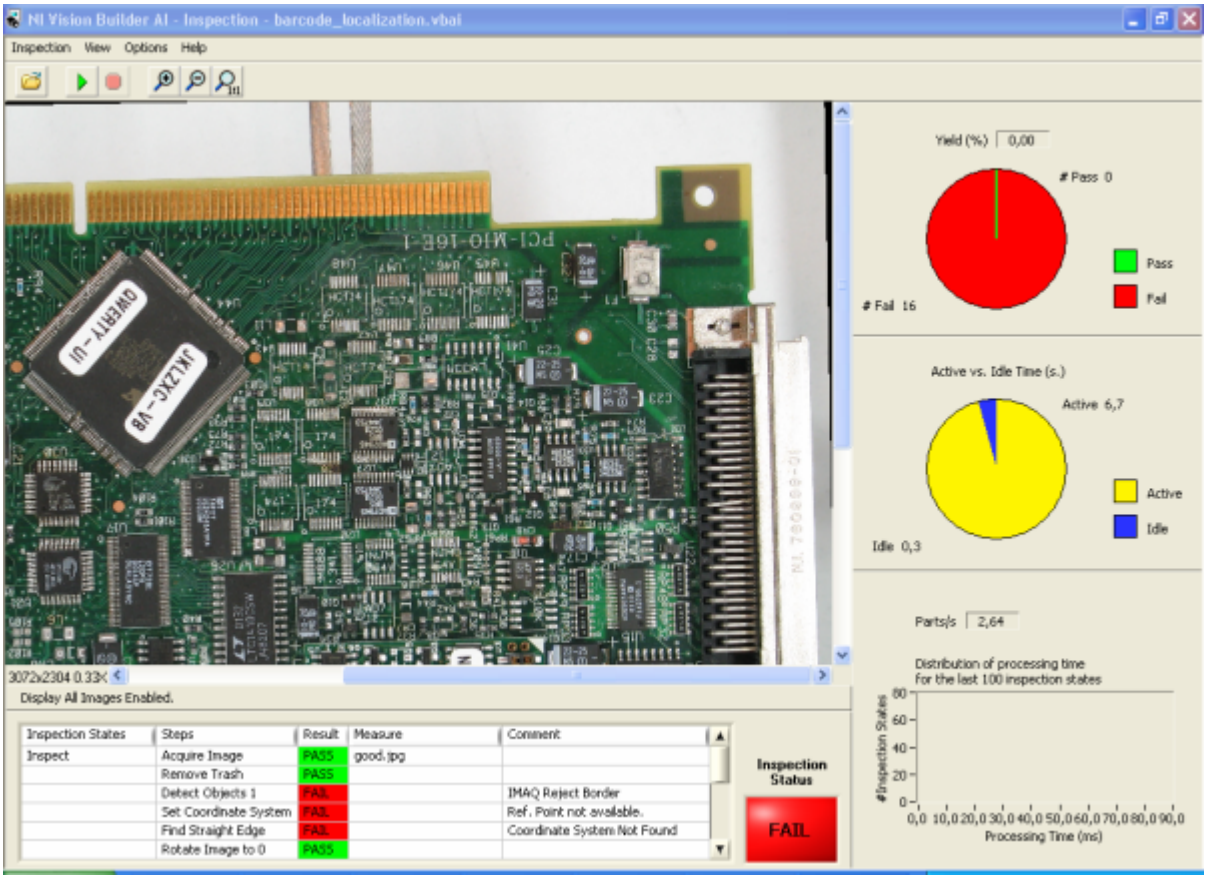


10. ábra. *Vision Builder AI - konfigurációs felület*

4.2.1.2. Az ellenőrzési felület

Az ellenőrzési felület három fő részre osztható: az eredmények panel, az ellenőrzési statisztikák panel, valamint a megjelenítő ablak.

Az eredmények panel szolgál arra, hogy az ellenőrzés egyes lépéseinek nevét, típusát és eredményét megjelenítse. A megjelenítő ablakban látható a feldolgozás alatt álló kép. Az ellenőrzési statisztikák panelen pedig az ellenőrzések sikerességének százalékos megoszlását, valamint az ellenőrzés üresjárati és feldolgozási idejét találjuk.

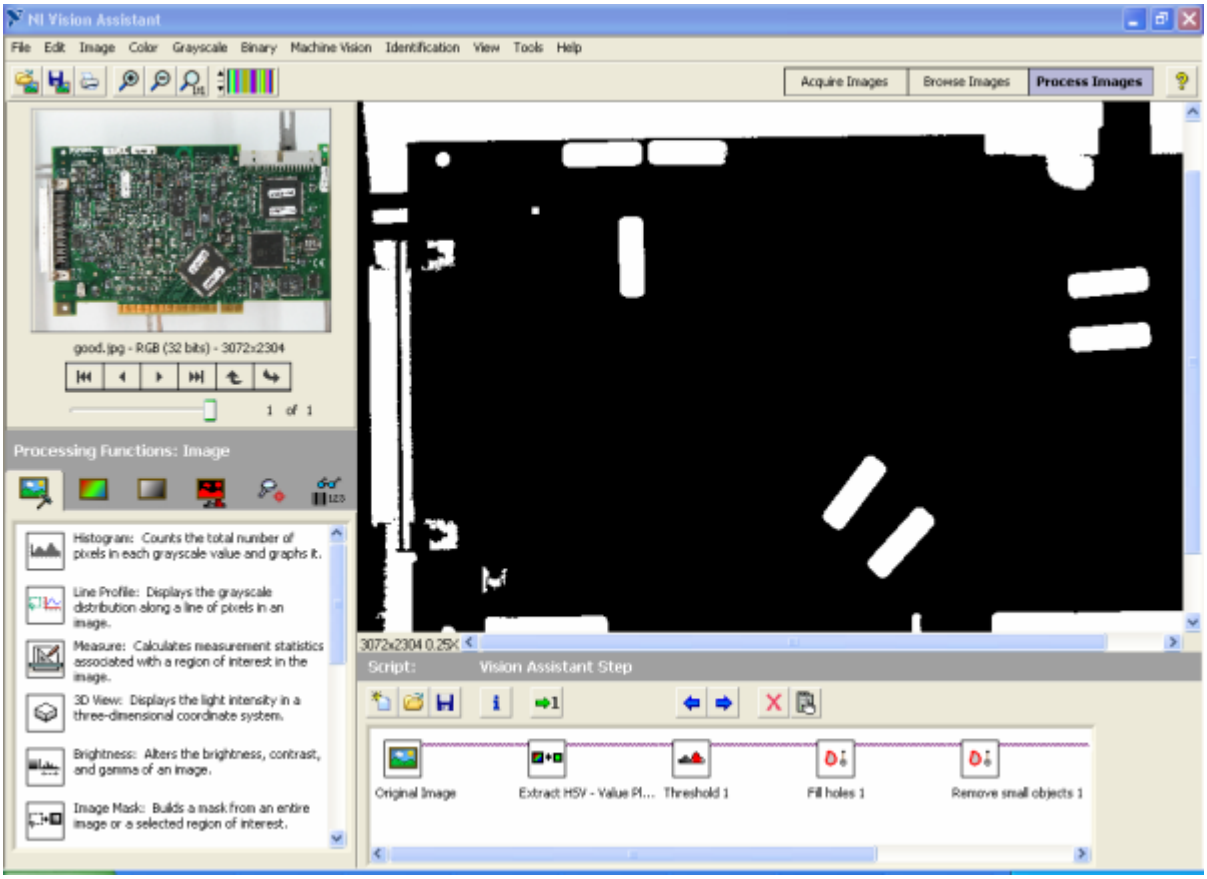


11. ábra. *Vision Builder AI - ellenőrzési felület*

4.2.2. Vision Development Module

A Vision Development Module nem más, mint képfeldolgozással kapcsolatos függvények gyűjteménye számos programozási nyelvhez, mint például LabVIEW, LabWindows/CVI, Microsoft C++, Visual Basic és .NET. A programozási eljáráskönyvtárak mellett a csomag része a **Vision Assistant** is.

A Vision Assistant első ránézésre nagyon hasonlít a Vision Builderhez, számos funkciójuk egyezik. A Vision Assisant inkább egy prototípuskészítési környezet, amely azt jelenti, hogy egy képfeldolgozó modul vázát interaktívan tudjuk elkészíteni, majd a kapott algoritmust - ami itt egy scr kiterjesztésű szkriptállomány - a kívánt programozási nyelv kódjára generálhatjuk. Az így kapott kód pedig részletesebben testre szabható, ami komplex alkalmazások esetén elengedhetetlen.

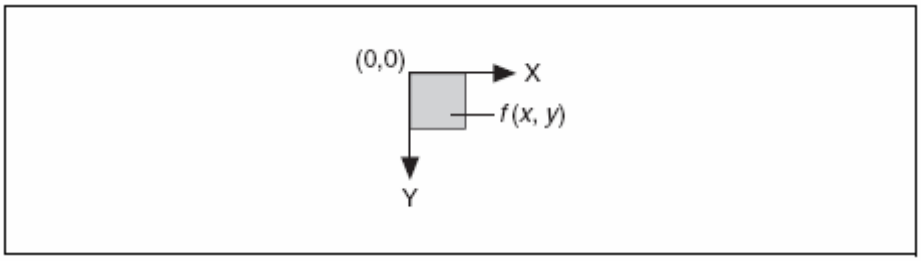


12. ábra. *Vision Assistant*

5. Alapfogalmak

5.1. A digitális kép definíciója

A digitális kép nem más, mint fényerősség értékekből álló kétdimenziós tömb, azaz egy mátrix. Egy digitális kép nem más, mint a fényerősség függvénye: *f(x, y),* ahol f az (x, y) koordinátájú képpont, azaz **pixel** fényessége. A továbbiakban kép alatt mindig digitális képet értünk. Megállapodás szerint a kép (0, 0) koordinátájú pontja a kép felső bal sarkában található.



13. ábra. *A digitális kép (0, 0) koordinátája*

A mintavételezés során a képszenzor mindig diszkrét számú pixellé alakítja az érzékelt képet, majd minden egyes pixelhez hozzárendeli a megfelelő szürkeárnyalatos vagy színes értékét, így meghatározva az egyes képpontok fényintenzitását vagy színét.

5.2. A digitális kép tulajdonságai

A digitális kép három alapvető tulajdonsággal rendelkezik: a kép felbontása, bitmélysége és a komponenseinek száma.

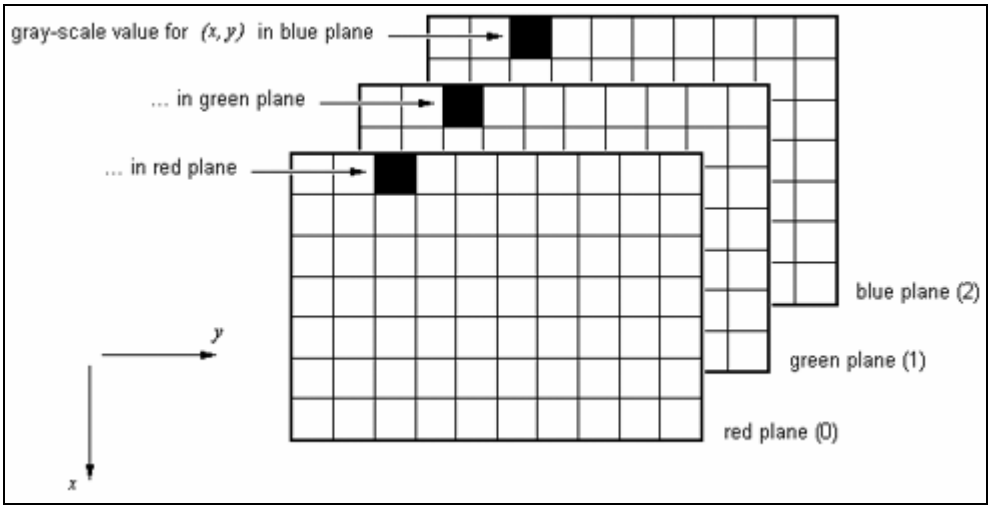
Egy kép - azaz egy mátrix – felbontásának nagysága a sorokban és oszlopokban lévő pixelek számától függ, azaz egy m oszlopból és n sorból álló kép felbontása m x n.

A kép bitmélysége nem más, mint a pixelértékek reprezentálásához szükséges méret bitekben megadva. Ha egy kép bitmélysége n, akkor a kép minden egyes pixele 2^n különböző értéket vehet fel. Például n = 16 esetén, egy pixel 65 536 féle értéket vehet fel, 0-tól 65 535-ig vagy -32 768-tól 32 767-ig. A jelenlegi NI Vision verzió (8.2.1) az utóbbi intervallumot támogatja 16 bites képek esetén. Az NI Vision 8, 10, 12, 14, 16 bites, lebegőpontos és színes

képet képes feldolgozni. Az NI Vision nem támogatja közvetlenül a 8 bitmélységnél alacsonyabb - 1, 2 és 4 bites – képek használatát, ugyanis ebben az esetben a kép bitmélysége automatikusan 8 bitre konvertálódik.

A képfeldolgozás során nagyon fontos szempont a helyes bitmélység megválasztása, ugyanis egyes műveletek során felesleges a szükségesnél nagyobb bitmélységű képet használni. Ez túlságosan memóriaigényessé és lassabbá tesz a feldolgozási folyamatot. Például egy képen található objektumok alakzatára vonatkozó információk meghatározása esetén, elég lehet a 8 bites bitmélység használata, ugyanakkor fényintenzitás mérése során célszerű minél nagyobb - 16 bites vagy lebegőpontos – bitmélység használata.

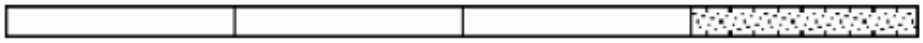
A komponensek száma adja meg a pixelmátrixok számát, amiből a kép felépül. Szürkeárnyalatos képek esetén egyetlen komponens létezik, színes képek esetén általában három. Pédául RGB színtér használata esetén a kép három színkomponensből épül fel, ezek a vörös, zöld és kék komponensek. Egy n komponensből álló kép esetén minden egyes pixel tulajdonképpen egy n dimenziós vektor lesz.



14. ábra. *RGB színkomponensek*

5.3. Képtípusok

Az NI Vision könyvtárak segítségével szürkeárnyalatos, színes és komplex képtípusokat tudunk kezelni. Habár az NI Vision mindhárom képtípust támogatja, bizonyos műveletek nem hajthatók végre egyes képtípusokon. Például egy komplex típusú képen nem hajtható végre az AND logikai operátor.

Image Type	Number of Bytes per Pixel Data
8-bit (Unsigned) Integer Grayscale (1 byte or 8-bit)	 8-bit for the grayscale intensity
16-bit (Signed) Integer Grayscale (2 bytes or 16-bit)	 16-bit for the grayscale intensity
32-bit Floating-Point Grayscale (4 bytes or 32-bit)	 32-bit for the grayscale intensity
RGB Color (4 bytes or 32-bit)	 8-bit for the alpha value (not used) / 8-bit for the red intensity / 8-bit for the green intensity / 8-bit for the blue intensity
HSL Color (4 bytes or 32-bit)	 8-bit not used / 8-bit for the hue / 8-bit for the saturation / 8-bit for the luminance
Complex (8 bytes or 64-bit)	 32-bit floating for the real part 32-bit for the imaginary part

15. ábra. *Képtípusok és a pixelek tárigénye*

5.3.1. Szürkeárnyalatos képek

A szürkeárnyalatos képek egyetlen színkomponensből állnak, a pixelek a következő reprezentációkkal rendelkezhetnek: 8 bites előjel nélküli egész, 16 bites előjeles egész, valamint egyszeres pontosságú, 4 bájton ábrázolt lebegőpontos érték.

5.3.2. Színes képek

Az NI Vision kétféle színteret használ a színes képek reprezentálására. Az egyik az RGB, a másik a HSL. Ezen színterek 3 komponensből állnak, de a színes képek pixelei négy értékből tevődnek össze. A negyedik érték az alfa csatorna, az NI Vision nem használja ezt a komponenst.

5.3.3. Komplex képek

Egy komplex kép egy szürkeárnyalatos kép frekvencia-információit tartalmazza. Ilyen típusú képet úgy lehet előállítani az NI Visionben, ha például egy szürkeárnyalatos képre alkalmazzuk a gyors Fourier-transzformációt (FFT). Komplex képeken végezhetőek el a frekvencia-tartományon értelmezett műveletek. A komplex képek pixelei két, egyszeres pontosságú lebegőpontos értékekből állnak, amelyek a komplex szám valós és képzetes részeinek feleltethetőek meg. Egy ilyen típusú pixelből a valós és képzetes részek, valamint az amplitúdó és fázis értékek nyerhetők ki.

5.4. Képfájlok

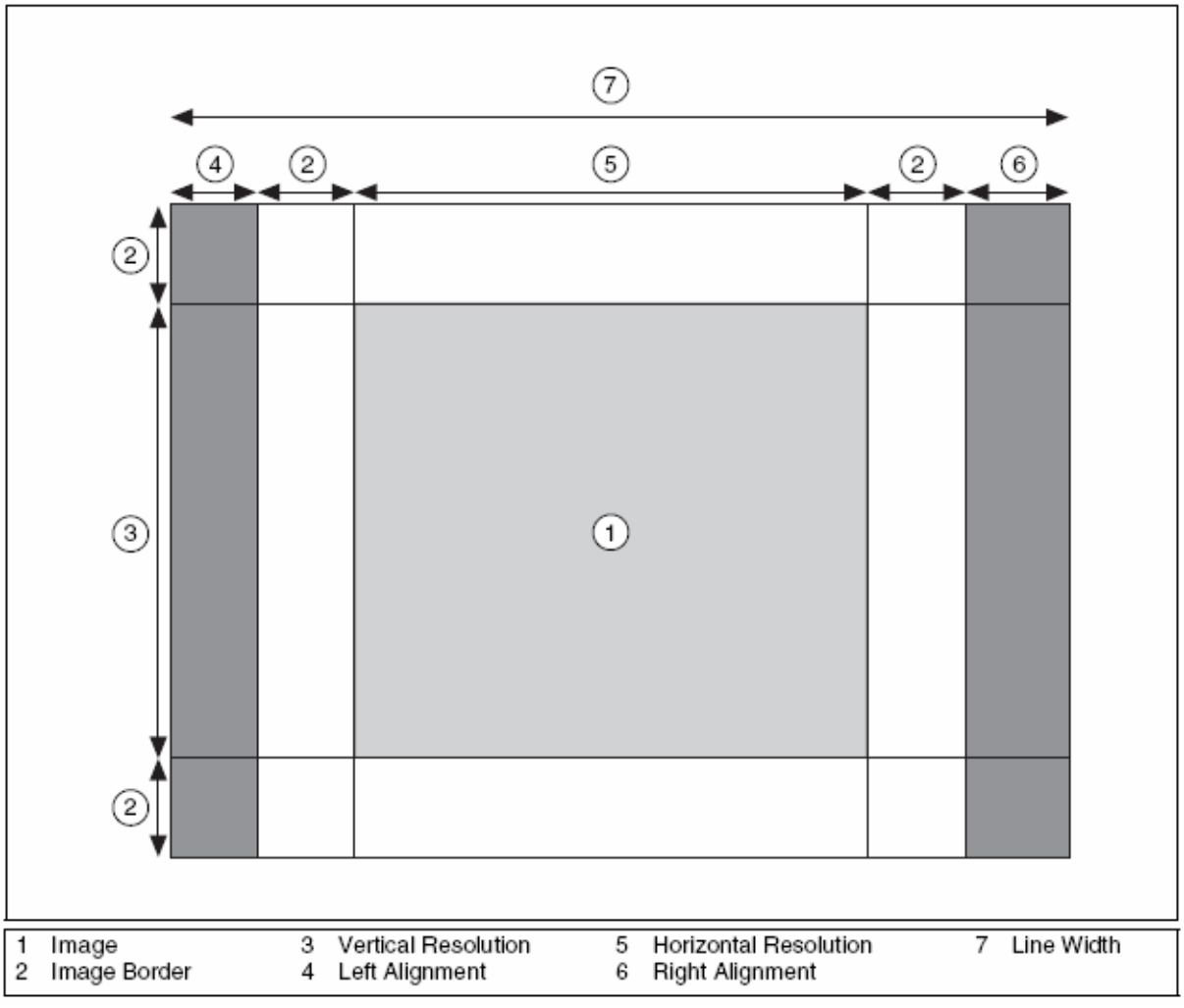
Egy képállományt egy fejléc és az utána következő pixelértékek alkotják. Az állomány formátumától függően a fejléc információkat szolgáltat a horizontális és vertikális felbontásról, a pixel definíciójáról, valamint a használt palettáról is. A képállományok ezen felül információt tárolhatnak különféle kalibrációs beállításokról és mintaillesztésekről is.

Az NI Vision a következő képformátumokat ismeri: BMP, TIFF, PNG, JPEG, AIPD. Ez utóbbi a National Instruments saját képformátuma, lebegőpontos, komplex és HSL képek tárolására szolgál. A szabvány 8 bites és RGB színes képek az ismert formátumokban

lehetnek, a 16 bites szürkeárnyalatos, 64 bites RGB és komplex képek pedig PNG és AIPD állományok lehetnek.

5.5. Az NI Vison képek belső reprezentációja

A 16. ábra illusztrálja az NI Vision képek memóriabeli reprezentációját. A kép pixelein túl, a tárolt kép plusz pixelekből álló sorokat és oszlopokat tartalmaz, ezekből áll a képkeret (image border), valamint a balra és jobbra igazítások (left, right alignment) is.



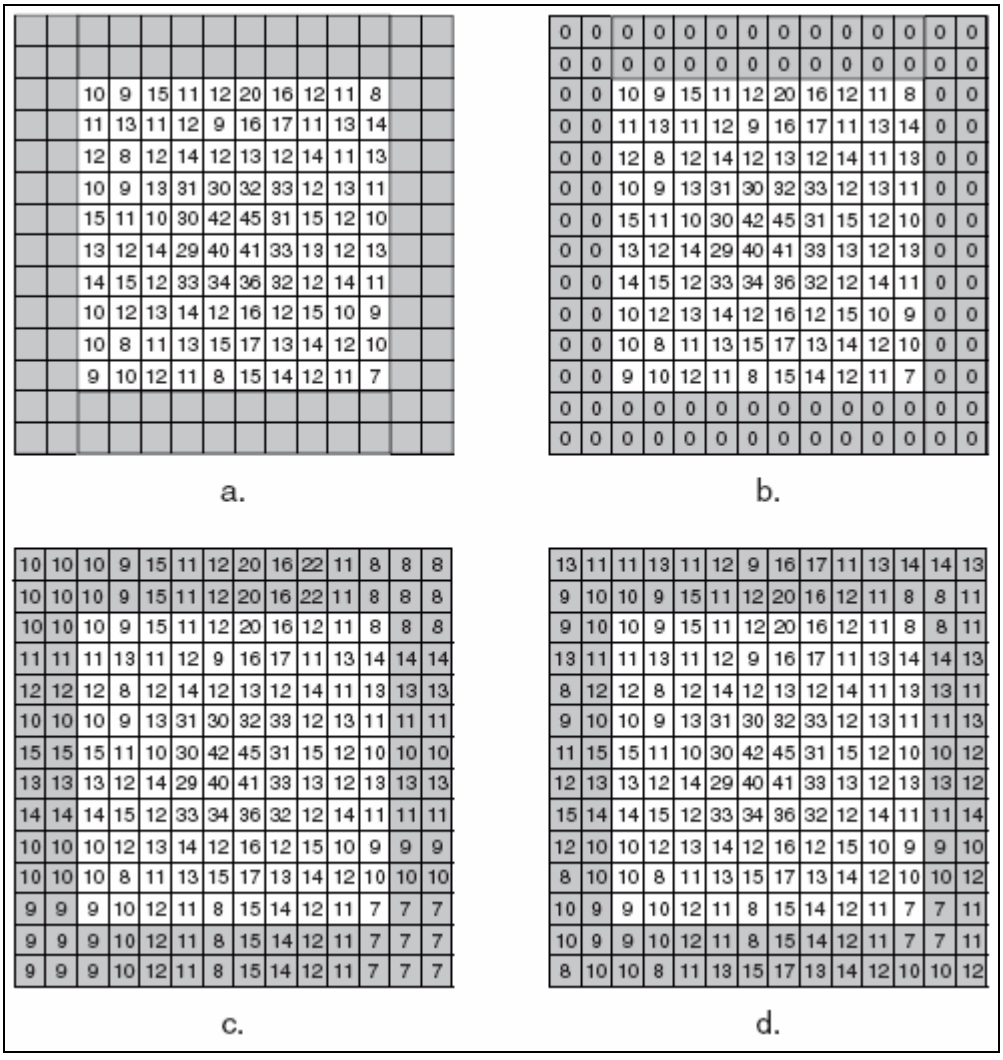
16. ábra. *Belső reprezentáció*

A képkeretet azon képfeldolgozási függvények használják, amelyek a feldolgozandó pixel szomszédságát is figyelembe veszik. Az igazítási blokkok mérete a kép szélességétől és a keret nagyságától függ. Ezen tartományok biztosítják azt, hogy a kép legelső pixele 32 bájtra

igazítódjon a memóriában. A sorszélesség (line width) a tárolt kép teljes horizontális pixelszáma, azaz a kép vízszintes felbontásán kívül a képkeret méretét és az igazítások nagyságát is magában foglalja. A sorszélesség egyezik a kép vízszintes felbontásával, ha a képkeret nulla és a vízszintes felbontás 32 bájt többszöröse.

5.5.1. Képkeretek

Az NI Vision háromféle módot biztosít a képkeret pixelértékeinek meghatározására. A 17. ábra illusztrálja ezeket a lehetőségeket. Az ábra a. része a kép pixeleit ábrázolja. A b. rész az alapértelmezett beállítás, azaz a képkeret pixelértékei azonosan nullák. Lehetőség van a kép szélein elhelyezkedő pixelértékek használatára a képkeretben, ahogyan ez az ábra c. részében látható.



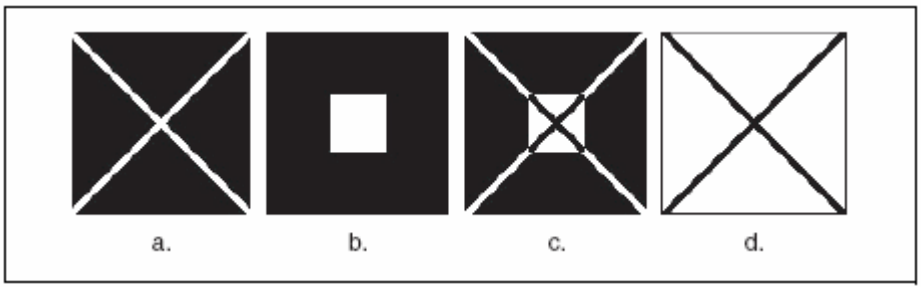
17. ábra. *A képkeret lehetséges pixelértékei*

A d. rész az utolsó opció, itt az egyes pixelértékek a kép széle mentén tükröződnek a képkeretbe. A képkeret pixelértékeinek megválasztása a képfeldolgozó eljárástól függ. Például éldetektáláskor a pixelek másolása vagy tükrözése pontosabb eredményt szolgáltat a kép szélein, mintha azok azonosan nulla értékűek lennének.

5.5.2. Képmaszkolás

A képmaszk - amely tulajdonképpen egy 8 bites bináris kép - segítségével a kép egyes részeit tudjuk elkülöníteni. A maszk méret kisebb vagy egyenlő lehet a feldolgozandó kép méretéhez képest. A képmaszk egyes pixelei határozzák meg azt, hogy a feldolgozandó kép mely pixelei fognak részt venni a feldolgozásban. Ez a következőképpen történik: ha a maszk egy pixelének értéke nulla, a feldolgozandó kép ugyanazon indexű pixele a feldolgozásban nem vesz részt, egyébként igen.

A teljes kép memóriában tárolása nagyon költséges, ezért célszerű a képnek csak egy kijelölt részét felhasználni és a maszkolást is csak ezen a területen végezni. Ezt a korlátozott területet az NI Visionben **ROI**-nak (**region of interest**) nevezik, erről a későbbiekben szeretnék említést tenni.



18. ábra. *Példa képmaszkolásra*

A 18. ábra illusztrálja a maszkolási technikát. Az ábra a. része a feldolgozandó bináris képet mutatja, a b. részben látható a képmaszk, a c. rész pedig az invertált maszkolt képet ábrázolja. A d. rész az eredeti kép inverzét mutatja.

5.5.3. A kép megjelenítése

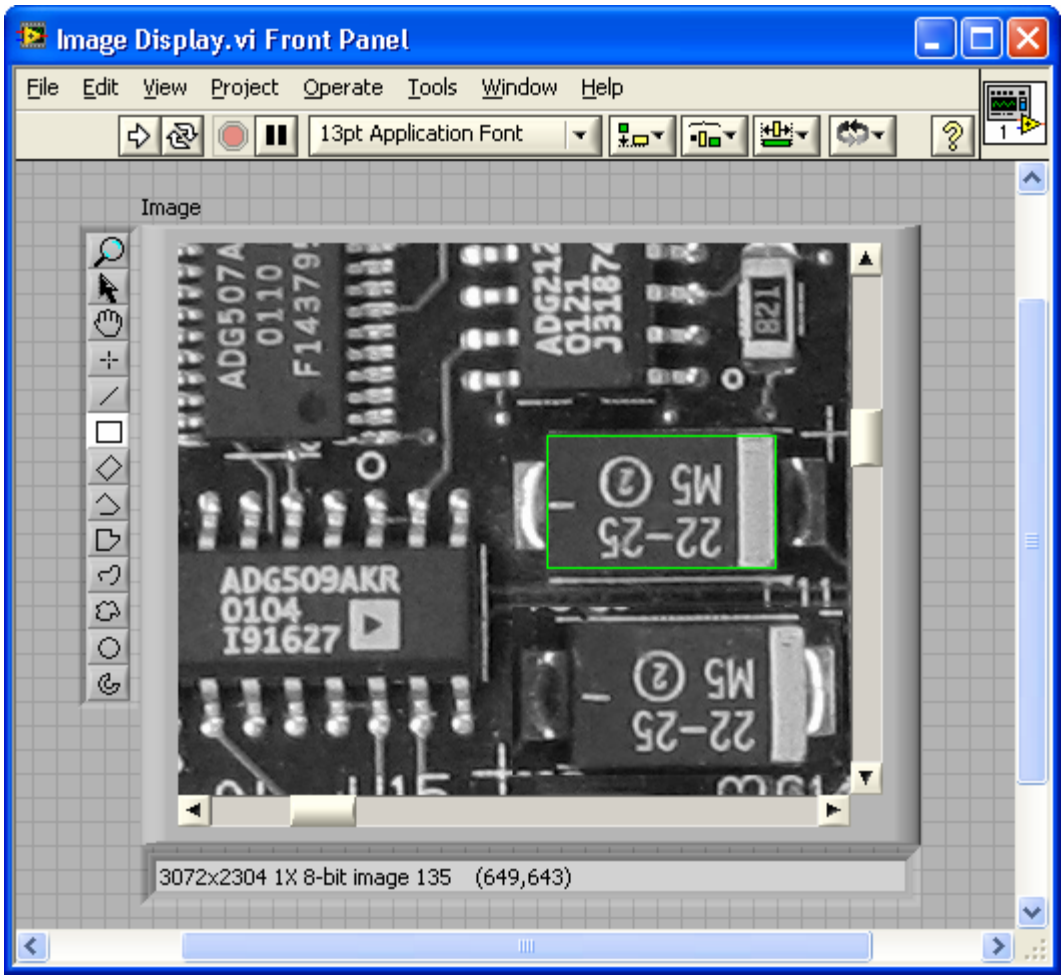
A képi megjelenítés az egyik legfontosabb komponens a vizuális termékellenőrző vagy egyéb képfeldolgozással kapcsolatos alkalmazások felhasználói interfészén. A

képfeldolgozás és képi megjelenítés egymástól elhatárolt komponensek. Az előbbi a kép létrehozásával, begyűjtésével és elemzésével kapcsolatos fogalom, míg az utóbbi a kép, mint adat megjelenítésével, valamint a felhasználó és a megjelenített kép interakciójára utal. Egy tipikus képkezelő alkalmazás például számos olyan képet tárol a memóriában, amit soha nem jelenít meg.

A felhasznált alkalmazásfejlesztési környezettől függően, a képeket a következő eszközökkel jeleníthetjük meg: külső ablak (LabVIEW, LabWindows/CVI), a LabVIEW saját képmegjelenítő kontrollja és CWIMAQViewer ActiveX kontroll (Visual Basic).

5.5.4. ROI

A ROI-k (**region of interest**) használata maximális interaktivitást biztosít az adott alkalmazás és felhasználója között.

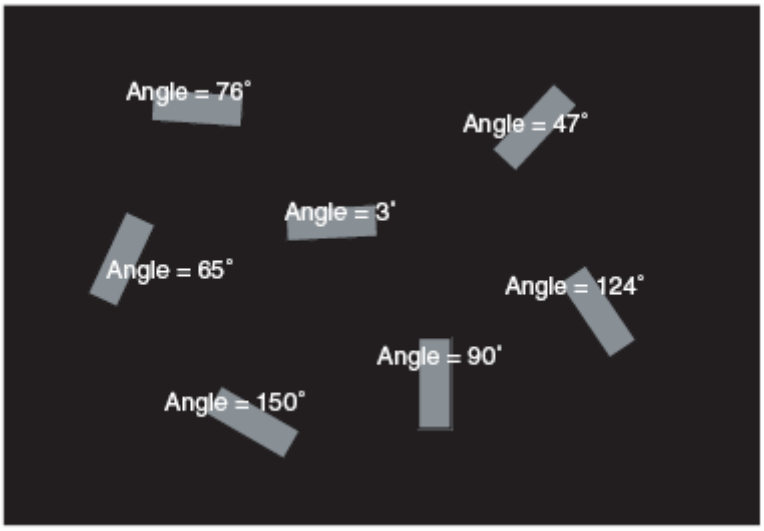


19. ábra. *Képmegjelenítés és ROI LabVIEW-ban*

A kép azon részét, ahol a kívánt képfeldolgozási műveletet akarjuk végrehajtani, ROI-nak nevezik. A ROI a képmegjelenítő eszközön jeleníthető meg - és akár itt is létrehozható -, erre a megjelenítők biztosítanak eszközöket úgynevezett kontúrok megrajzolásával. Természetesen programozott módon is történhet a létrehozás és megjelenítés. A ROI-kal a következő műveletek hajthatók végre: létrehozás, hozzárendelés, törlés, valamint képmaszk ROI-hoz rendelése és viszont.

5.5.5. Inzertek

Az inzert (**nondestructive overlay**) arra szolgál, hogy a megjelenített képre további információkat helyezzünk el anélkül, hogy ezen információk a réteg alatt elhelyezkedő pixelértékeket módosítanák. A képre szövegeket, vonalakat, geometriai alaktatokat lehet elhelyezni. Ezeket az információkat a képpel együtt PNG formátumban tudjuk elmenteni.



20. ábra. *Nondestructive overlay*

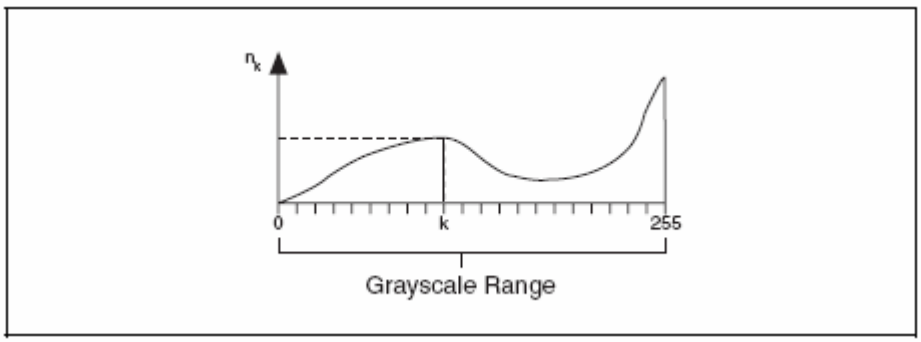
6. Képfeldolgozás és elemzés

6.1. Képelemzés

Maga a képelemzés nem más, mint a kép pixeleinek intenzitásértékein végzett mérések és statisztikai számítások. A képelemző függvények segítségével a kép tartalmi tulajdonságait tudjuk elemezni és a további feldolgozás során a kapott eredményektől függően döntést tudunk hozni, hogy mely képfeldolgozó függvényeket használjuk fel.

6.1.1. Hisztogram

A hisztogram segítségével összeszámolhatjuk és függvényszerűen ábrázolhatjuk az egyes szürkeárnyalat szinteken lévő pixelek számát. A kapott diagramból meg lehet állapítani, hogy a kép tartalmaz-e adott szürkeárnyalatú, különálló tartományokat valamint a képgyűjtési fázis beállításainak ellenőrzésére is használható.



21. ábra. *Hisztogram*

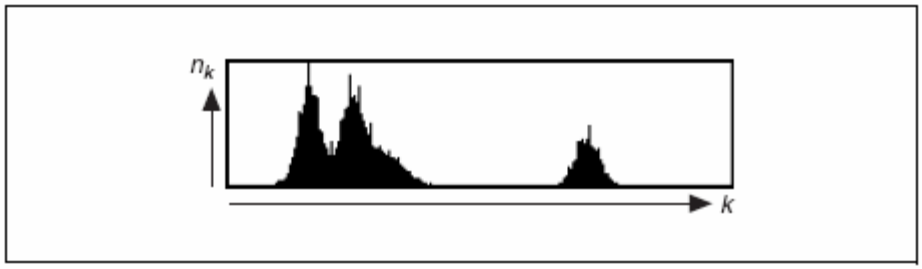
A hisztogram egy olyan H függvény, amely a [0, …, k, …, 255] intervallumon értelmezett pixelintenzitások eloszlását írja le:

$$H(k) = n_k,$$

ahol k a szürkeárnyalat értéke, n_k a k értékű pixelek száma a vizsgált képen. Ha n_k-kat összegezzük 0-tól 255-ig, akkor megkapjuk a kép pixeleinek a számát.

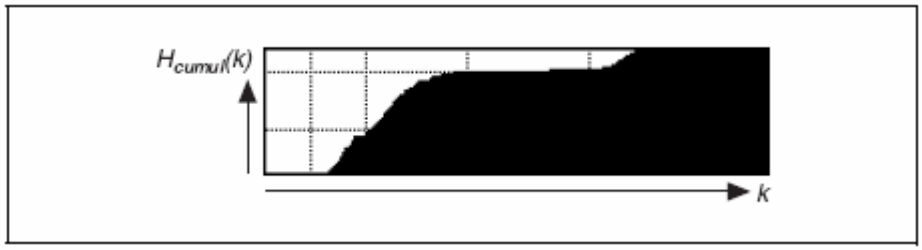
A vizsgált kép hisztogramjára nézve a következő két fontos ismertetőjelet tudjuk megállapítani:

- Telítettség - a képszenzor képrögzítési folyamata során, ha a rendelkezésre álló fény mennyisége nem elegendő, a készített kép alulexponált lesz, míg túl sok fény esetén túlexponált vagy telített. Ezen szélsőséges feltételek a további képfeldolgozási folyamatokban hibás eredményekhez vezethetek. Egy kép hisztogramjáról el lehet dönteni, hogy a kapott kép alul- vagy túlexponált-e. Egy alulexponált kép a feketéhez közeli értékekből túl sok pixelt tartalmaz, azaz a hisztogram függvény a szürkeskála értékek alsó részénél erősen csúcsosodik. Túlexponált kép esetén a hisztogram a szürkeskála felső részénél mutat nagyobb eloszlást.
- Kontraszthiány – egyes képfeldolgozási problémák megoldása során szükség lehet a képen található összefüggő tartományok detektálására és megszámolására. Erre az a stratégia, ha a keresendő tartományok és a háttér intenzitás-eltéréseit figyelembe vesszük. A 22. ábra egy olyan kép hisztogramját illusztrálja, ahol a képen két vagy több, jól elkülönített tartomány található. A képrögzítés során a hisztogram figyelembe vétele segíthet a további képfeldolgozáshoz szükséges megfelelő kontraszt beállításához.



22. ábra. *Lineáris hisztogram*

Kétfajta hisztogram számítási mód létezik: lineáris és kumulatív (halmozott) hisztogram. Mindkét esetben a függvény abszcisszája a szürkeárnyalat értékeket reprezentálja. Lineáris hisztogram esetén az ordináta tengely a pixelszámot jelöli, kumulatív hisztogram esetén pedig a k értéktől kisebb vagy egyenlő értékű pixelek százalékarányát jelzi.

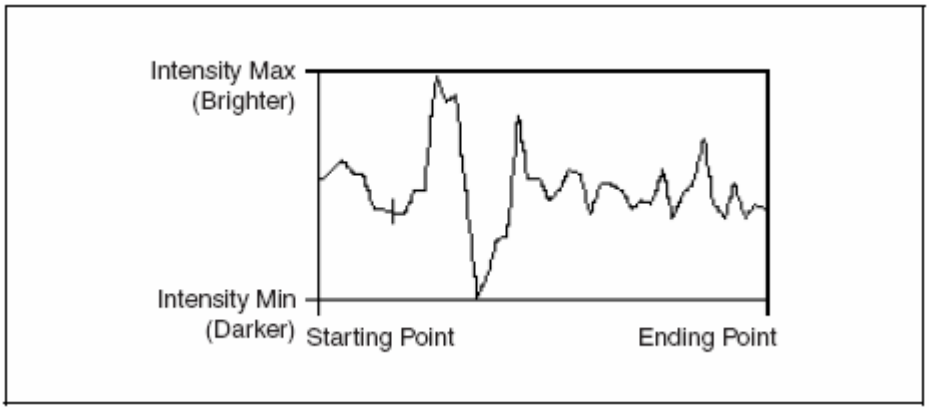


23. ábra. *Kumulatív hisztogram*

Színes képek esetén is értelmezhető a hisztogram fogalma, ekkor a hisztogram komponensenként van értelmezve.

6.1.2. Vonalprofil

Az NI Vision vonalprofil (**line profile**) eszközének segítségével egy vonal mentén lévő pixelértékek intenzitását tudjuk vizsgálni. A kapott értékeket a hisztogramhoz hasonlóan függvényszerű ábrázolja. Ezt az eszközt akkor célszerű használni, ha a vizsgált képen lévő objektumok elhatárolódását vizsgáljuk vagy pedig ismétlődő mintákat keresünk.



24. ábra. *Vonalprofil*

A vonalprofil segítségével kapott függvényen a csúcsok és völgyek a fényintenzitás növekedését és csökkenését jelentik a vizsgált képen. Ezek szélessége és amplitúdója arányos a hozzájuk tartozó tartományok nagyságával és fényintenzitásukkal.

Például egy világos és egyenletes intenzitású objektum vonalprofiljának függvényképe magas és egyenletes alakú lesz.

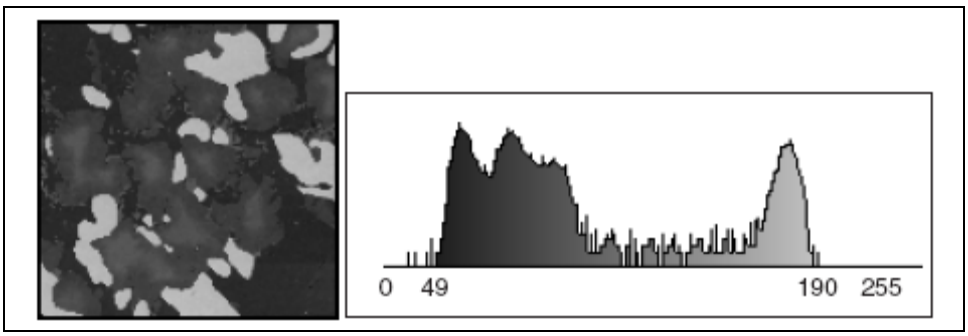
6.1.3. Intenzitásmérés

A szürkeárnyalatos képekről készíthetünk bizonyos statisztikákat. Az NI Vision intenzitásvizsgáló modulja segítségével például a kép egy tartományában meg tudjuk határozni a minimális, maximális és átlagos intenzitás értékeket.

6.2. Képfeldolgozás

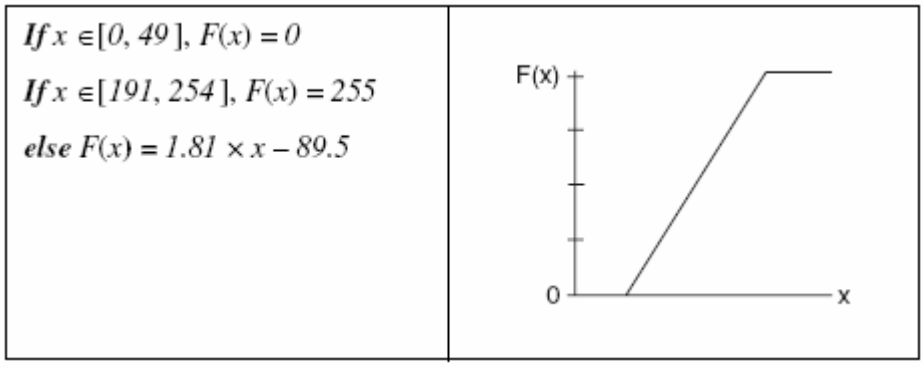
6.2.1. LUT

A LUT (**lookup table**) transzformációk olyan alapvető képfeldolgozási függvények, amelyek segítségével részleteket tudunk kiemelni azokban a képi tartományokban, melyek fontos információkat tartalmaznak. Ezek a transzformációk a következők: hisztogram-kiegyelítés, gamma, logaritmikus és exponenciális korrekciók.



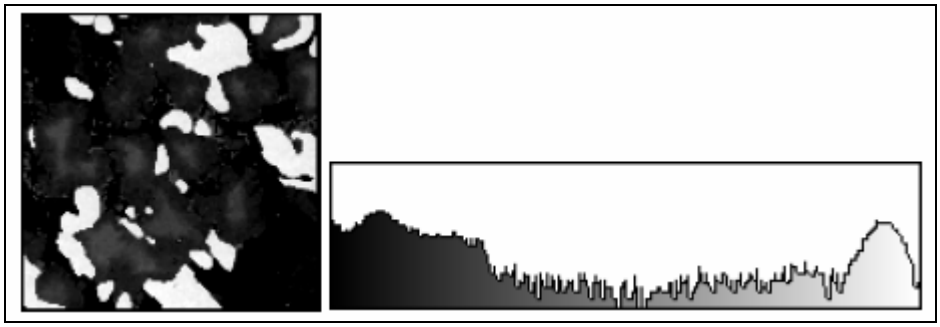
25. ábra. *LUT transzformáció előtti forráskép és hisztogram*

A LUT transzformációk a bemeneti kép szürkeskála értékeit konvertálják át más intervallumba, így az előállt kimeneti kép már más szürkeskálával fog rendelkezni. Az NI Visionben hét darab, előre definiált LUT transzformáció található. Vegyük példának a 12. ábrán látható képet! A forráskép lineáris hisztogramja szerint a [0, 49] és [191, 255] intervallumok között nincs értékes információ. Alkalmazzuk a 13. ábrán látható transzformációt! Azok a pixelek, melyek értéke kisebb, mint 50, legyen 0 és azok, melyek értéke nagyobb, mint 190, legyen 255, különben legye egyenlő az ábrán szereplő értékkel.



26. ábra. *LUT transzformáció*

Ez a transzformáció a 14. ábrán látható képet állítja elő. A forráskép pixelértékeinek [50, 190] intervallumát [1, 254]-re bővítettük. Az eredmény a 14. ábrán látható.



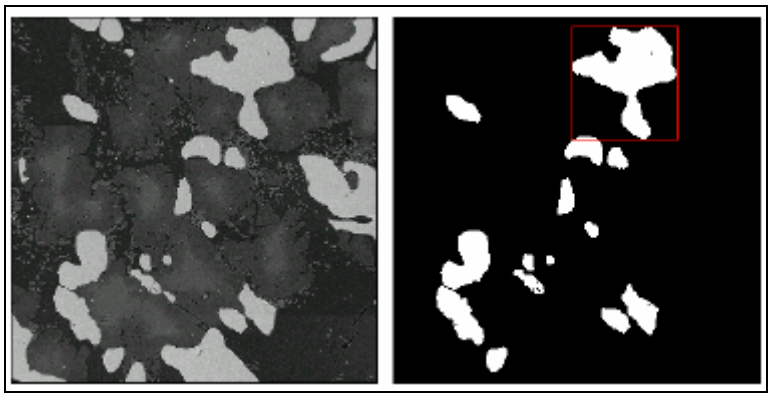
27. ábra. *A LUT transzformáció eredménye*

7. Részecske elemzés (Particle analysis)

A részecske elemző képfeldolgozási és elemzési eljárások arra valók, hogy segítségükkel összefüggő pixelcsoportokat - azaz kétdimenziós alakzatokat - detektálni tudjunk és ezekben a tartományokban bizonyos méréseket elvégezzünk. Ezeket a pixelcsoportokat az NI Vision **részecskéknek** nevezi. Tehát egy részecske nem nullaértékű pixelekből álló összefüggő csoport.

Egy tipikus részecske elemző eljárás kezdetben a kapott bemeneti képen keres és megpróbálja a kritériumoknak megfelelő részecskéket megkeresni a képen. Ha talál egyet, akkor arról részletes leírást készít. A keresési kritériumok segítségével tudjuk a részecskéket azonosítani és osztályozni.

Az 1. ábrán látható bináris képet úgy kaptuk, hogy a forrásképen végrehajtottunk egy szintre vágást, majd azon részecskéket, melyek a kép széleit érintették, eltávolítottuk. Az 1. táblázat azon értékeket mutatja, melyek a bináris képen látható téglalap segítségével bekerített részecskét jellemzik.



28. ábra. *Részecske elemzés*

Particle Measurement	Values
Area	2456
Number of Holes	1
Bounding Rect	
Left	127
Top	8
Right	200
Bottom	86

Particle Measurement	Values
Center of Mass	
X	167.51
Y	37.61
Orientation	82.36
Dimensions	
Width	73
Height	78

1. táblázat. *A vizsgált részecske tulajdonságai*

A részecske elemző eljárások használata előtt a képet bináris képpé kell alakítani, például szintre vágás segítségével. A kapott bináris képen morfológiai eljárásokkal javíthatunk, majd ezután végezhetjük el a méréseket.

7.1. Képszegmentálás

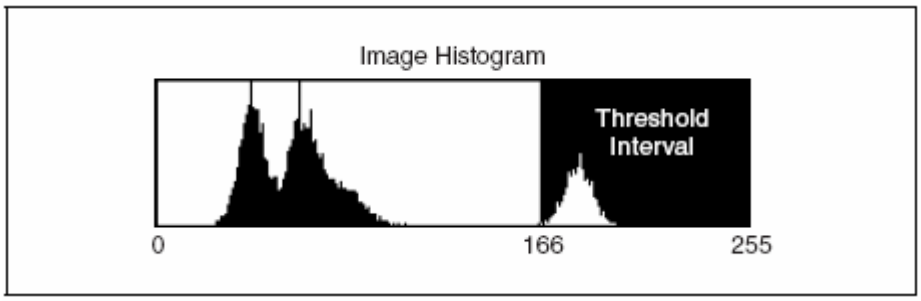
Szegmentálásnak nevezzük azt a folyamatot, amikor a képen objektumokat különítünk el a háttértől, így azok jól felismerhetővé és jellemezhetővé válnak. A legismertebb képszegmentálási eljárás a szintre vágás (**thresholding**).

7.1.1. Globális szintre vágás szürkeárnyalatos képeken

A szintre vágás általában az első lépés számos képfeldolgozó és elemző eljárás készítése során. Akkor érdemes használni, ha a képen olyan objektumokat (részecskéket) akarunk lokalizálni és elemezni, melyek jellegzetes struktúrával rendelkeznek.

A szürkeárnyalatos képen található részecskék mindegyike jellemezhető egy intenzitás tartománnyal, azaz a részecske pixeleinek értéke ebbe az intervallumba esik bele. Az ettől eltérő értékű pixelek nem fontosak, ezért akarjuk elkülöníteni őket.

A szintre vágás segítségével a megadott intervallumba eső pixelek értékeit 1-re, míg a többi pixelt pedig 0-ra állítjuk. A kapott kép tehát egy bináris kép lesz.



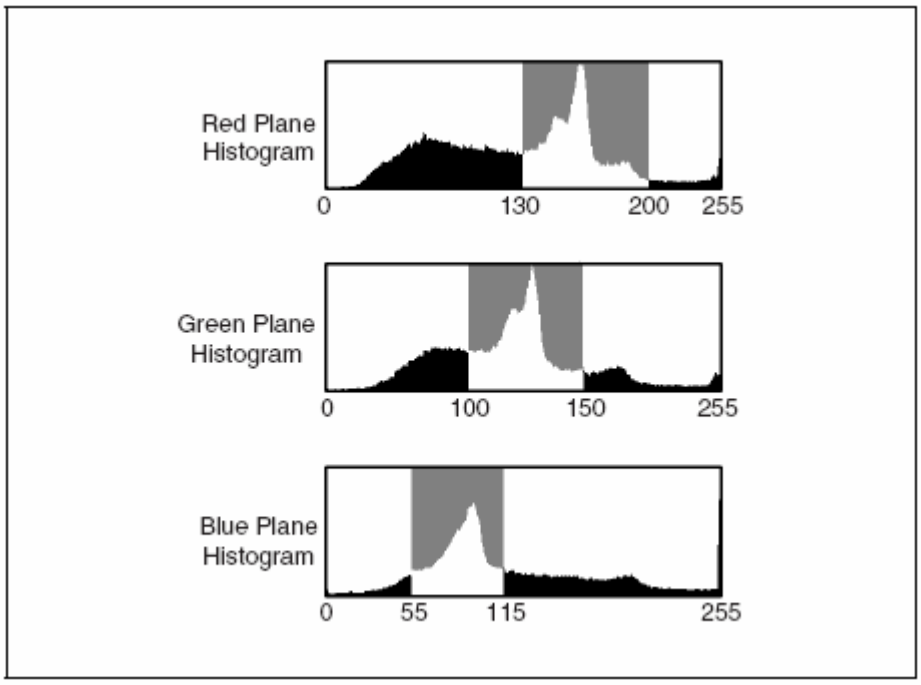
29. ábra. *Szintre vágás*

A 29. ábra egy olyan kép hisztogramját illusztrálja, ahol a 166-tól 255-ig tartó intervallumba esnek a részecske pixelértékei. A szintre vágás után ezek az értékek 1-re, a többi pixelérték pedig 0-ra fog változni.

Az NI Vision tehát lehetőséget biztosít a szintre vágás intervallumának manuális megadására, de lehetőség van ötféle, beépített, automatikus szintre vágó technika alkalmazására is, melyek a következők: **clustering**, **entropy**, **interclass variance**, **metric** és **moments**.

7.1.2. Globális szintre vágás színes képeken

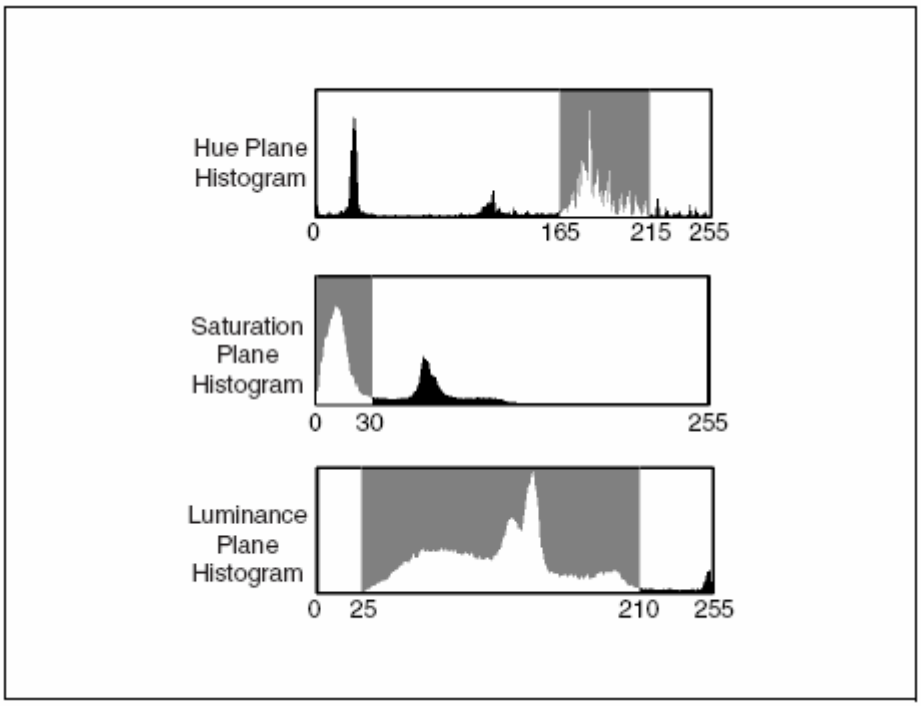
Színes képeken is végezhető szintre vágás, ez az eljárás is bináris képeket eredményez, hasonlóan a globális szürkeárnyalatos szintre vágáshoz. Az intervallum választása ebben az esetben komponensenként történik. A kapott bináris képen azon pixel értéke lesz 1, mely pixel komponensenkénti értéke beleesik a megadott intervallumokba. A 30. ábra egy RGB képen végrehajtott globális szintre vágást illusztrál.



30. ábra. *Globális szintre vágás RGB képeken*

Egy HSL színterű kép szintre vágása esetén első lépésben a vizsgálandó pixelek színárnyalat komponensének (hue) intervallumát kell meghatározni. A telítettség komponens (saturation) esetén az intervallum meghatározása általában az adott vizsgálati eljárástól függ. Mivel a fényesség komponens (luminance) csak az intenzitás értékekről hordoz információt,

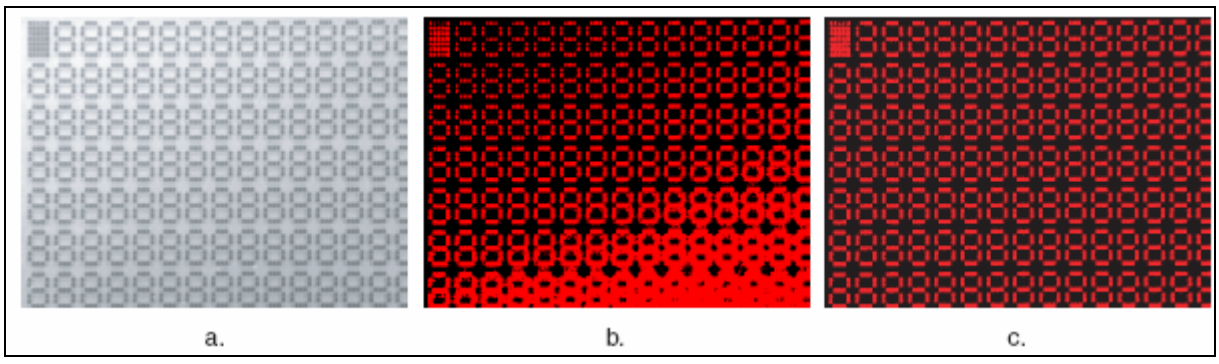
ezért célszerű a teljes intenzitás tartományt intervallumnak választani, így a szintre vágás ettől a komponenstől függetlenné válik. A 31. ábra egy ilyen típusú szintre vágást mutat be.



31. ábra. *Globális szintre vágás HSV képeken*

7.1.3. Lokális szintre vágás

A lokális szintre vágás az előző technikákhoz hasonlóan szintén bináris képet állít elő a fontosnak vélt képi információk háttértől való elkülönítése után. Az eltérés az, hogy a globális szintre vágás a teljes kép intenzitás értékeit veszi figyelembe, míg a lokális változat csak az egyes pixelek szomszédjainak intenzitását.



32. ábra. *A globális és lokális szintre vágás összehasonlítása*

Számos esetben a képrögzítéskor rendelkezésre álló fény nem egyenletes, így a vizsgált objektumon árnyékok és intenzitási eltérések jönnek létre. Az ilyen helyzetekben a globális szintre vágás nem megfelelően működik.

A 32. ábra a. részén egy folyadékkristályos kijelző látható. Ha egy globális szintre vágást alkalmazunk erre a képre, akkor a kapott eredmény nem lesz megfelelő, ugyanis a nem egyenletes megvilágítás miatt a kép jobb alsó sarkában számos olyan pixel fog az adott intervallumba beleesni, aminek - egyenletes megvilágítás esetén – a háttérhez kellene tartoznia. Az ábra c. része az eredeti képen végrehajtott lokális szintre vágást mutat, itt csak azok a pixelek fognak az előtérhez tartozni, melyek az eredeti képen az egyes LCD számjegyeket alkotják.

A lokális szintre vágás a vizsgált pixel szomszédos pixelein különféle pixelintenzitás-statisztikákat készít az intervallumokról, eloszlásokról, szórásokról és ezek kombinációjáról is. Az így kapott értéket - mely a vizsgált pixelhez tartozik - az algoritmus összehasonlítja az eredeti intenzitás értékkel, és ez alapján dönti el, hogy a pixel háttérelem-e vagy sem.

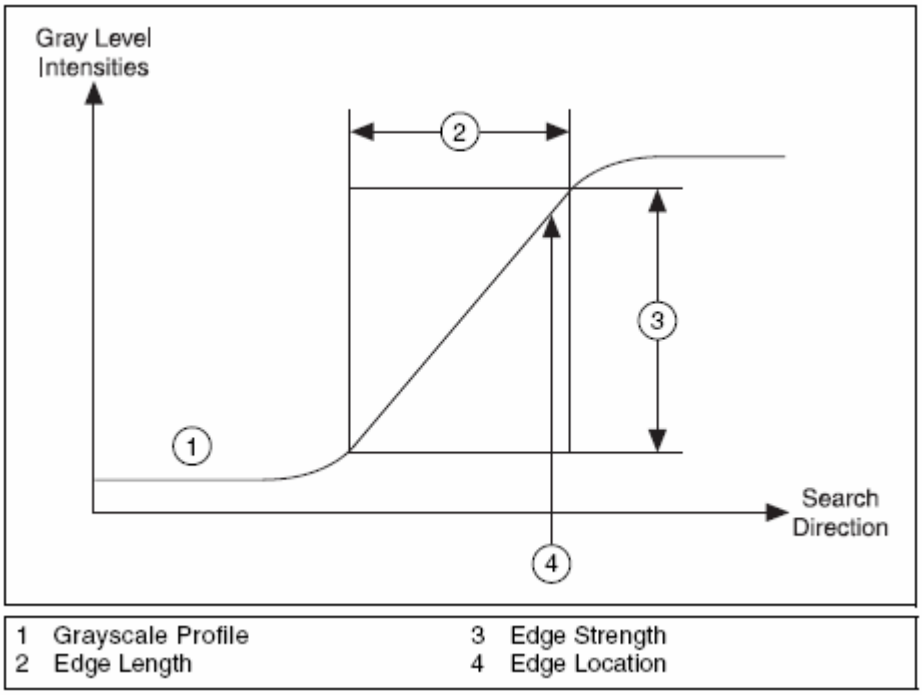
Az NI Vision lehetőséget biztosít arra, hogy a szomszédos pixeleket meghatározó keret megadása manuálisan történjen. Az alapértelmezett méret 32 x 32-es. Célszerű úgy megválasztani ezt a méretet, hogy a legkisebb részecske is - melyet a háttértől akarunk elkülöníteni - beleférjen a keretbe.

8. Gépi látás (Machine Vision)

8.1. Éldetektálás

Egy szürkeárnyalatos digitális képen egy él nem más, mint intenzitásértékek jelentős mértékű megváltozása szomszédos pixelek esetén. Ezek a hirtelen változások tulajdonképpen a képen látható objektumok körvonalait jelentik. Az éldetektálást (**edge detection**) egy előre megadott keresési tartományban végezhetjük el. A tartomány specifikálása az NI Vision-ben történhet interaktívan vagy programozott módon is. Az éldetektálás egy egydimenziós profil mentén történik a megadott keresési tartományon belül. Ez a profil lehet vonal vagy kör, ellipszis és téglalap határvonala. Az éldetektáló eljárás először analizálja a pixelértékeket egy egydimenziós profil mentén, majd beállítástól függően éleket keres a tartományban. Lehetőség van az összes létező él, csak az első él vagy az első és utolsó él detektálására is. Az éldetektálást kalibrálás, detektálás és illesztés esetén célszerű használni.

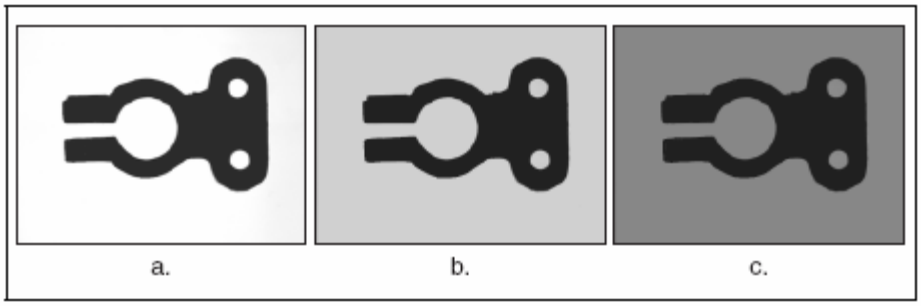
8.1.1. Éltulajdonságok



33. ábra. *Egy él jellemzői*

A digitális képeken megjelenő él a következő tulajdonságokkal rendelkezik (ezek szemléltetése a 33. ábrán látható):

- Élerősség – az él és a háttér szürkeárnyalatos értékének minimális eltérését definiálja. Szokásos élkontrasztnak is nevezni. Ezt a tulajdonságot nagyban befolyásolja a kép rögzítésekor fennálló megvilágítás mértéke, ugyanis ha a megvilágítás alacsony, a kontraszt nem lesz nagy a képen.
- Éltávolság – a maximális intenzitás-eltérést definiálja a háttér és él között, minél lassabban változik a háttér és az él közötti eltérés, annál nagyobb az éltávolság értéke.
- Élpolaritás – az él emelkedését, illetve lejtését definiálja. Az emelkedő él intenzitás-értékei növekvők, a lejtő élé pedig csökkenőek.
- Élpozíció – az él (x, y) koordinátája a képen.



34. ábra. *Eltérő kontrasztok hatása*

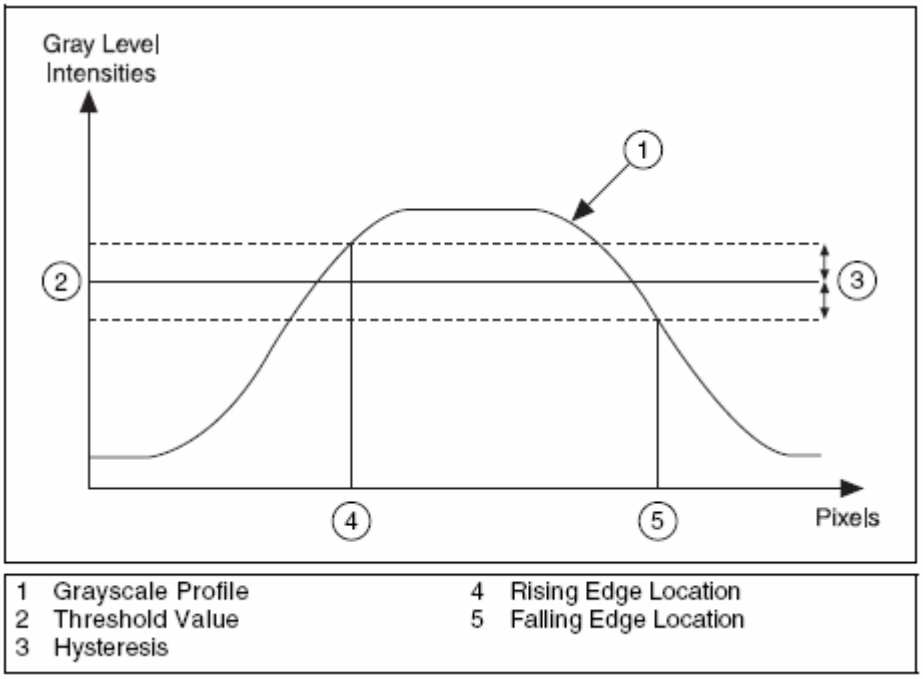
8.1.2. Éldetektáló módszerek

Az NI Vision kétféle éldetektáló módszert ismer, ezek közül az egyszerűbbik kerül bemutatásra. Mindkét módszer élerősséget számít a megadott profil mentén, pixelenként. Specifikálhatunk egy minimális kontrasztértéket, amely alatt nem akarunk éleket felismerni.

8.1.2.1. Egyszerű éldetektáló módszer

Az eljárás a profil mentén pixelről pixelre haladva a pixelértékeket vizsgálja. Amint egy olyan pixelhez ér, melynek értéke nagyobb, mint a megadott küszöbérték plusz egy hiszterézis érték, akkor emelkedő élt detektált a módszer. A hiszterézis érték segítségével különböző élerősségeket tudunk definiálni emelkedő és lejtő élek számára. Az emelkedő él detektálása után az eljárás ejtő élt keres, azaz az első olyan pixelt keresi, ahol az

intenzitásérték a megadott küszöbérték alá csökken. Az eljárás a profil végéig folytatódik. Ez a módszer jól működik minimálisan zajos képek és háttértől való erős elhatárolódás esetén.



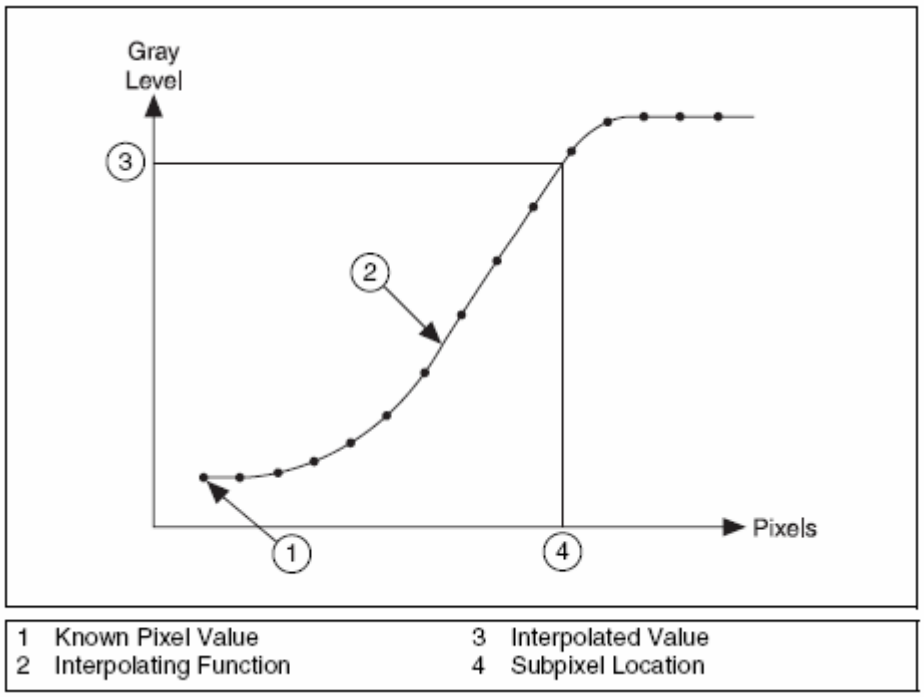
35. ábra. *Egyszerű éldetektálás*

8.1.2.2. Szubpixel pontosság

Amikor a vizsgált kép felbontása elegendően nagy, akkor a legtöbb mérő és vizsgáló alkalmazás pontos éldetektáló eredményeket szolgáltat pixel méretű pontosság esetén is. Azonban számos akadálya lehet annak, hogy a kívánt felbontást elérjük, például a rendelkezésre álló képrögzítő szenzor nem képes nagyobb felbontást produkálni. Ebben az esetben célszerű használni a szubpixel pontosságú (**subpixel accuracy**) keresést.

A szubpixel elemzés egy szoftveres metódus arra, hogy megbecsüljük azon pixelértékeket, amiket nagyobb felbontás esetén kaphatnánk. Az éldetektálás szubpixel pontossággal úgy működik, hogy az éldetektáló eljárás először egy magasabb rendű interpoláló függvényt illeszt a pixel intenzitásokra. A függvény tehát megpróbálja kiszámítani a meglévő pixelértékek közötti pixelek intenzitásokat. Ezután az éldetektálás a kapott értékeket felhasználva (szubpixel pontossággal) fog tovább működni. A 36. ábra egy harmadfokú spline interpolációt és a becsült pixeleket illusztrálja. A szubpixelek segítségével nyert információ használata meglehetősen pontos, de természetesen befolyásolják a

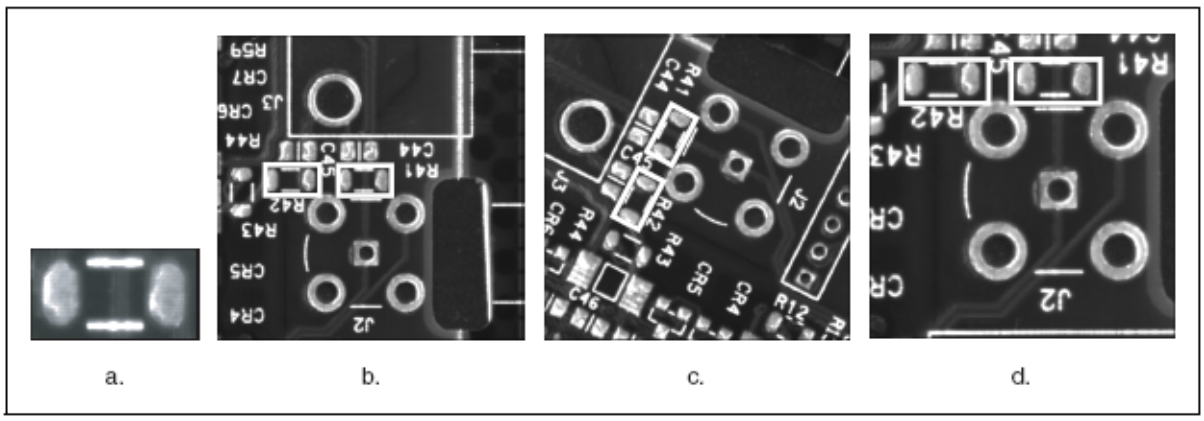
hatékonyságot külső tényezők is, például a megvilágítási körülmények. Célszerűbb - ha a lehetőség adott - a képfelbontást növelni.



36. ábra. *Szubpixelek értékének meghatározása interpolációval*

8.2. Minta- és geometriai illesztés

A mintaillesztés (**pattern matching**) a termékellenőrzésben egy nagyon gyakran alkalmazott eljárás. Segítségével szürkeárnyalatos képeken tudunk keresni olyan tartományokat, amelyekre az előre definiált mintát - más néven sablont - illeszteni tudjuk.



37. ábra. *Mintaillesztés*

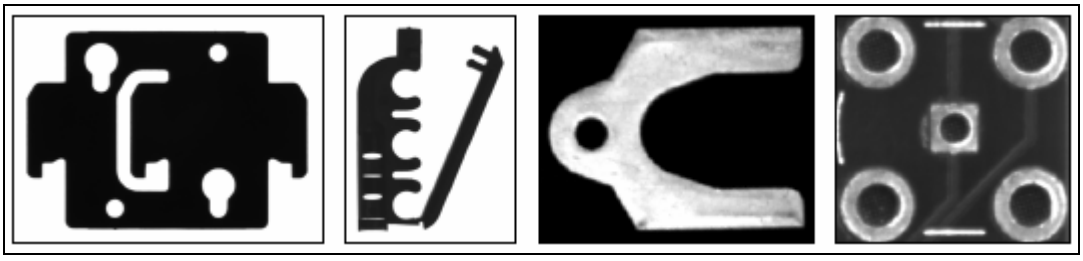
Először a keresendő képi objektumról készítünk egy mintát, majd a mintaillesztő algoritmus a vizsgált képen próbál a mintához minél hasonlóbb előfordulásokat keresni. Ezt pontozással viszi véghez, a nagyobb pontszámú - 0 és 1000 között, 1000 a tökéletes illesztés - előfordulások jobban hasonlítanak a keresett mintához. Számos tényező - mint például fényintenzitás, zaj, méret és orientáció - befolyásolja a keresés eredményét, ezért nagyon fontos az, hogy ezeket a mintaillesztő algoritmus a sikeres lokalizáció érdekében kezelni tudja. A 37. ábra a. részén található minta illesztése az ábra további részein látható.

8.2.1. Mintaillesztési technikák

Többféle mintaillesztési eljárás létezik, például a normalizált kereszt-korreláció (**normalized cross-correlation**), a piramis alakú illesztés (**pyramidal matching**), a méret- és elforgatás-független illesztés (**scale- and rotation-invariant matching**), valamint a képértelmezés (**image understanding**).

8.2.2. Geometriai illesztés

A geometriai illesztés (**geometric matching**) is egy mintát definiál, majd a vizsgált képen is ezt a mintát - pontosabban objektumot - keresi. Ezt a típusú illesztést akkor célszerű használni, ha a keresendő objektum megkülönböztethető geometriai tulajdonságokkal rendelkezik, erre példa a 38. ábrán látható. Viszont, ha a keresendő mintát inkább a textúrája, mint az alakzata jellemzi vagy a vizsgált kép túl sok élt tartalmaz, akkor a mintaillesztést érdemesebb használni.



38. ábra. *Geometriai illesztésre alkalmas objektumok*

9. A vizuális termékellenőrző rendszer

Ebben a fejezetben a készített termékellenőrző rendszer képfeldolgozó moduljainak felépítéséről, a feldolgozandó képek memóriabeli tárolásáról és a betanítási fázisról lesz szó, valamint részletes ismertetésre kerül a komponensek meglétét ellenőrző modul és a vonalkód-lokalizáló, leolvasó modul is.

9.1. A képfeldolgozó modulok felépítése

Egy képfeldolgozó modult a LabVIEW-ban egy SubVI reprezentál. A 39. ábrán látható VI be- és kimeneti termináljainak típusa - amit LabVIEW-ban a vezeték színe és vastagsága határoz meg - minden egyes ilyen vizsgáló modul esetén azonos, így a képfeldolgozó modulokat meghívó és végrehajtó VI (VI Server) a meghívott SubVI-ok működésétől függetlenül képes őket végrehajtani. Vegyük sorra a terminálokat!

Az első terminál (*Images*) típusa nem más, mint képeket tartalmazó egydimenziós tömb. A képfeldolgozó modul itt kapja meg a vizsgálandó képet, vagy például a keresendő alkatrészek mintaképét. A vizsgálandó kép nyers kép, a modul további átalakításokat végezhet rajta, például csökkentheti a színmélységét, ha a további feldolgozáshoz nem szükséges több.

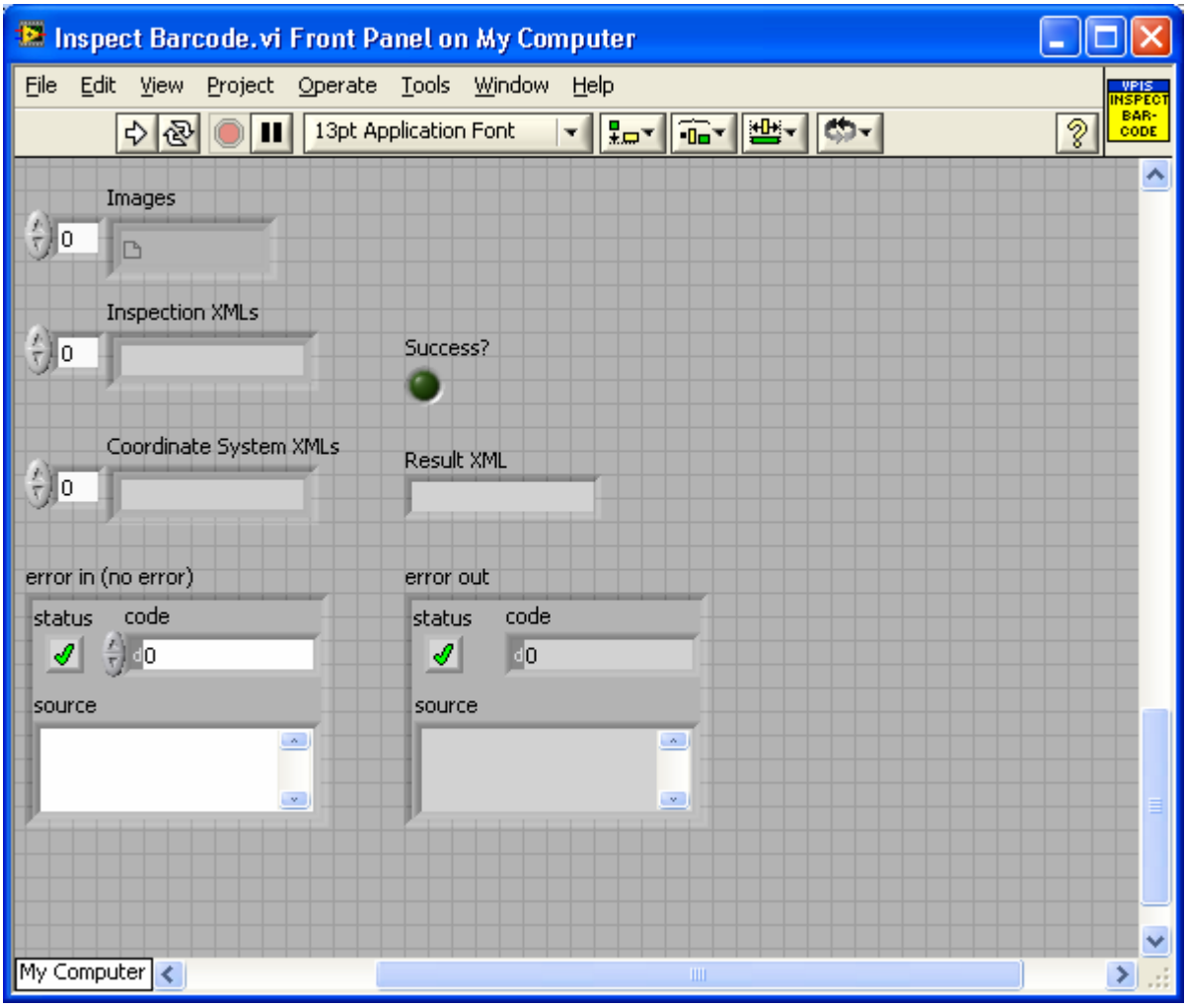
A következő terminál (*Inspection XMLs*) típusa egy LabVIEW-specifikus XML sztring lehet, itt kapja meg a modul a vizsgálandó tartományokat, például a vonalkód lokalizálása ennek segítségével történik, azaz nem kell az egész képen a megfelelő geometriai jellemzőkkel rendelkező címkét megkeresni. A vizsgálandó tartományok ROI-kra konvertálódnak a modulokban, és a vizsgálat ezekben a ROI-kban folyatatódik. Ez jelentősen megnöveli a feldolgozási sebességet.

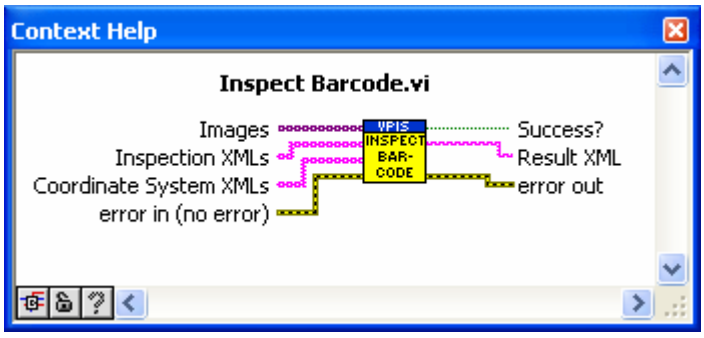
A ROI-k helyzete mindig egy viszonyítási ponthoz kötött. Ezt a pontot egy koordináta rendszer segítségével határozzuk meg, ami a következő bemeneti terminálon (*Coordinate System XMLs*) érhető el. Típusa szintén egy LabVIEW-specifikus XML sztring.

A modul legelső kimeneti terminálja (*Success?*) a vizsgálat eredményét szolgáltatja, például alkatrész meglétének vizsgálata esetén, ha a keresett alkatrész minden egyes tartományban megtalálható és az adott keresési paramétereknek eleget tesz, akkor a terminálra igaz logikai érték kerül.

A következő kimeneti terminál (*Result XML*) is a vizsgálat eredményét szolgáltatja, de konkrét eredményt ad, például a sikeresen leolvasott vonalkódot. Típusa szintén egy LabVIEW-specifikus XML sztring.

A be- és kimeneti hibaterminálok *(error in*, *error out*) szolgálnak az esetlegesen felmerülő hibák továbbítására.

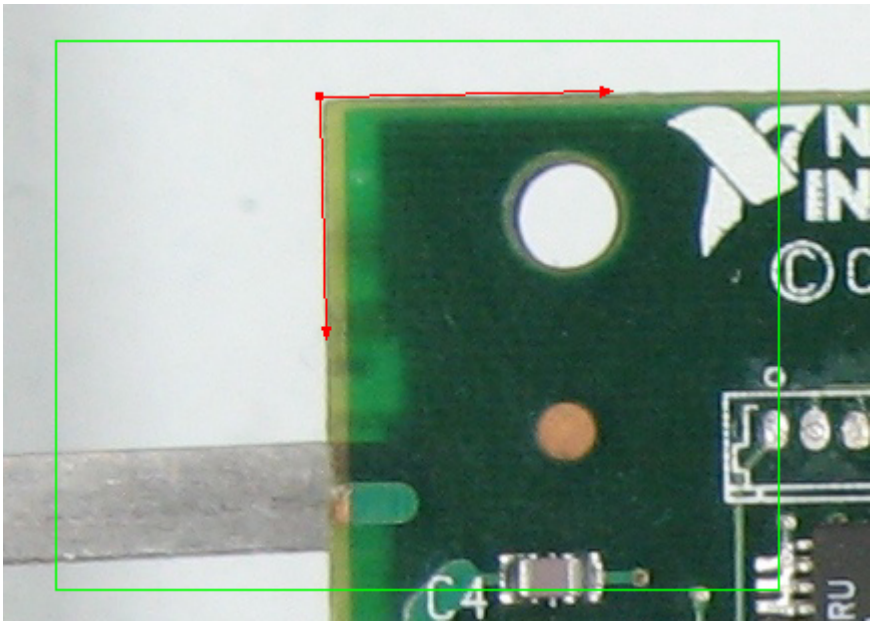




39. ábra. *A vonalkód-lokalizáló és leolvasó SubVI*

9.2. A betanítási fázis

A termékellenőrző rendszer működéséhez elengedhetetlen az egyes képfeldolgozó modulok működéséhez szükséges keresési tartományok definiálása. Vegyük példának a következőt: egy nyomtatott áramkörről készült digitális képen szeretnénk egy adott típusú kondenzátor meglétének a vizsgálatát elvégezni. Ekkor meg kell keresni és elmenteni - egy XML állományba a könnyebb feldolgozás végett - azokat a tartományokat, ahol ez a kondenzátor megtalálható. Ezeknek a tartományoknak a helyzete relatív kell, hogy legyen, azaz egy rögzített ponthoz viszonyítva kell őket meghatározni. Ezt úgy tehetjük meg, hogy feltételezünk egy olyan tartományt, amiben ez a pont minden egyes vizsgálati képen megtalálható és könnyen lokalizálható. A példánál maradva legyen ez a nyomtatott áramköri lapka bal felső sarka. Az NI Vision rendelkezik olyan beépített eljárással, aminek segítségével két, egymásra merőleges és egymást metsző él detektálásával a metszéspont lokalizálható. Ez a metszéspont lesz a vizsgálat koordináta rendszerének origója.



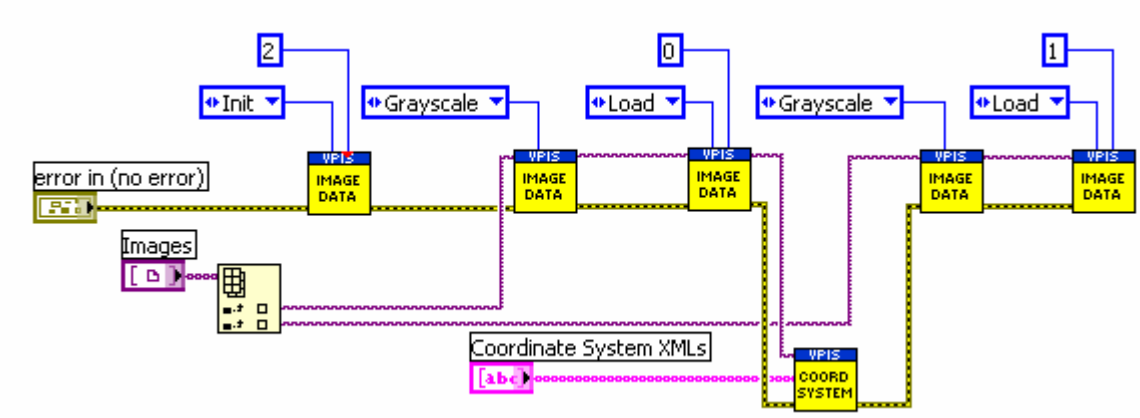
40. ábra. *Koordináta rendszer meghatározása*

Mivel a tartományok koordinátáinak távolsága ettől a ponttól függ, így ezt a pontot minden képfeldolgozó modul működése során meg kell keresni a vizsgált képen. Természetesen a sebesség növelése céljából, ha egy képen több vizsgálat kerül végrehajtásra, célszerű csak egyszer elvégezni a koordináta rendszer meghatározását.

9.3. A feldolgozandó képek tárolása

Az alkalmazás moduláris felépítése miatt célszerű egy olyan SubVI létrehozása, melynek feladata a vizsgáló modulok számára képekkel kapcsolatos funkciók szolgáltatása. Ezek a következők lehetnek: színkomponens elérése, szürkeárnyalatos kép létrehozása, képtömb inicializálása, törlése, kép betöltése tömbbe, kép mentése fájlba és kép tartományának lekérdezése.

Ezeket a funkciókat az **Image Data VI** valósítja meg a funkcionális globális változó technika felhasználásával. Ennek az a lényege, hogy a VI-ban használt shift regiszter a VI többszöri meghívása során is megtartja utolsó értékét, aminek az előnye jelen esetben az, hogy egy olyan VI-t hozunk létre, amely a kép tárolásával és konfigurálásával kapcsolatos funkciókat gyűjti össze. A funkcionális globális változó továbbá komplex adatstruktúrák kialakítására is alkalmas.



41. ábra. *A funkcionális globális változó lényege*

A 41. ábra egy olyan forráskód részlet, mely az Image Data VI felhasználását egy példán keresztül illusztrálja. Jelentése a következő: az első hívás során létrehoz egy kételemű tömböt, mely képek tárolására alkalmas. A második hívás a bemeneti képet szürkeárnyalatosra konvertálja, majd ez a kép a harmadik hívás során a korábban létrehozott tömb nulladik indexű eleme lesz. A VI negyedik hívása a bemeneti képet szürkeárnyalatosra konvertálja, majd tárolja a már meglévő tömbben első indexű elemként az ötödik hívás során.

9.4. Komponens meglétét ellenőrző modul

A komponens meglétét ellenőrző modul feladata az, hogy a vizsgált képen a megadott tartományokban megkeresse az adott alkatrészt mintaképét. Az algoritmus szövegesen a következőképpen néz ki:

Első lépésben inicializálja a korábban említett Image Data VI-t. Ez azt jelenti, hogy a bemenetre érkező vizsgálandó képet szürkeárnyalatosra konvertálja és tárolja, valamint ugyanígy jár el a szintén bemeneti mintaképpel is. A szürkeárnyalatosra konvertált vizsgálandó képen megkeresi a *Coordinate System XMLs* adatai alapján a viszonyítási pontot, azaz meghatározza a koordináta rendszert.

A második lépésben beolvassa azoknak a tartományoknak a koordinátáit, ahol a keresendő mintának lennie kell, valamint a mintaillesztési paraméterek is beolvasásra kerülnek.

Az előző lépésekben nyert információkat felhasználva megkezdődik a tartományokban a mintakeresés. A tartományok természetesen itt már a koordináta rendszerhez viszonyítottak.

A következő lépésben azokat a tartományokat, ha vannak olyanok, ahol a keresendő mintakép nem található, megjegyzi.

Végül a kimeneti *Success?* terminálra a vizsgálat eredménye kerül, ha voltak mintaképet nem tartalmazó tartományok, akkor ezen tartományok koordinátái a *Result XML* kimeneti terminálon megjelennek, valamint a lefoglalt erőforrások - jelen esetben a képek - felszabadításra kerülnek.

9.5. Vonalkód-lokalizáló és leolvasó modul

Ezen modul segítségével a vizsgált képen a megadott tartományban található vonalkód leolvasását lehet elvégezni. Habár az NI Vision tartalmaz vonalkód-leolvasó modult, mely több típus detektálására is képes, a sikeres működéshez szükséges a vizsgálandó tartomány átalakítása. Vegyük sorra az egyes lépéseket!

Első lépésben - a korábban említett modulhoz hasonlóan - inicializálja az Image Data VI-t a következőképpen: a bemenetre érkezező vizsgálandó képet szürkeárnyalatosra konvertálja, majd a kapott koordináta rendszer és tartomány-koordináták alapján

meghatározza a kép azon részét, ahol a vonalkód található. Ezután csak ezen a képrészleten folytatódik a feldolgozás.

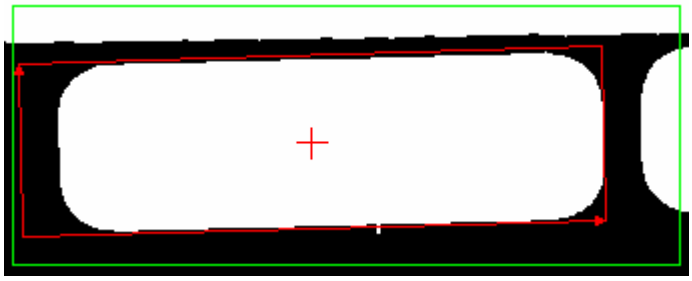
42. ábra. *A vonalkód címkéje a vizsgálandó tartományban*

A következő lépésben történik a címke pontos lokalizációja. Erre azért van szükség, hogy a vonalkód leolvasó rutinnak a vonalkód pontos helyét tudjuk megadni. Előfordulhat ugyanis az, hogy a címke nem párhuzamos az abszcisszával. Ebben a lépésben felhasznált képfeldolgozó eljárások paramétereinek megadása nagyban függ a vizsgált kép minőségétől és világosságától.

43. ábra. *Szintre vágás*

Egy lehetséges megoldás a címke pontos lokalizálására a következő eljárás: alkalmazzuk a szintre vágást a képre! Az így kapott bináris képen, amely a 43. ábrán látható, olyan felesleges részecskék találhatóak, melyek egy objektum - jelen esetben a címke - detektálását zavarják, ezért ezen részecskék morfológiai eljárássokkal eltávolíthatók.

Az így kapott kép már csak a lokalizálandó címkét tartalmazza. Mivel a címke könnyen azonosítható geometriai jellemzőkkel rendelkezik, ezért célszerű a geometriai illesztés használata, amely során egy a címkével megegyező alakzatú mintát adunk meg keresendő objektumként. A 44. ábra illusztrálja a geometriai illesztés eredményét, a 2. táblázat pedig az illesztés paramétereit.



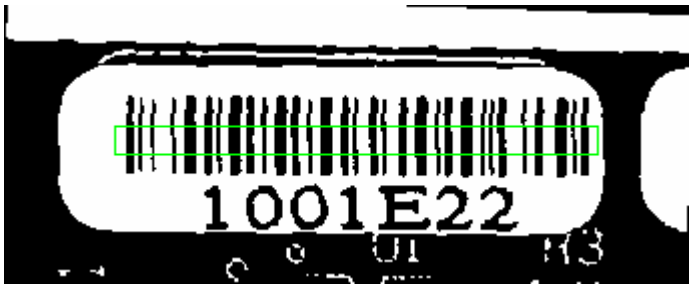
44. ábra. *A lokalizált címke geometriai illesztés felhasználásával*

Object #	1
X Position	153.73
Y Position	72.67
Angle	1.61
Scale	109.7
Score	964.28
Occlusion %	12.82
Template Target Curve Score	840.66
Correlation Score	690.06

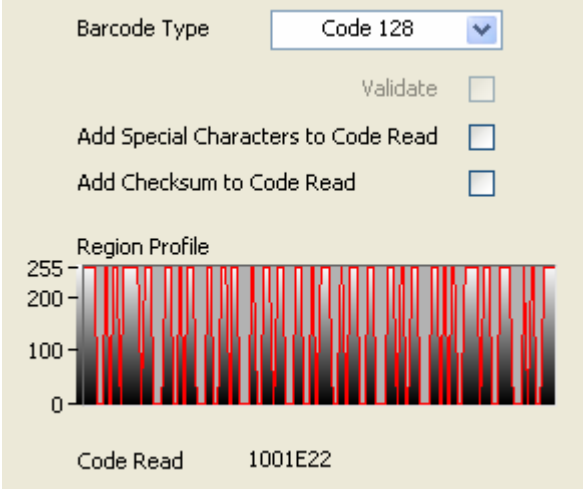
2. táblázat. *A lokalizált címke paraméterei*

Az eljárás csak akkor folytatódik, ha a geometriai illesztés sikeres, azaz pontosan egy objektumot talált. Ha ez igaz, akkor a 2. táblázatban is látható szög (**Angle**) paraméter alapján el kell forgatni az eredeti címkét - a 42. ábrán látható - úgy, hogy párhuzamos legyen az abszcisszával.

Ezután az előbb említett szintre vágást végrehajtjuk az elforgatott képre, így az NI Vision vonalkód-olvasó rutinja már alkalmazható erre.



45. ábra. A *vonalkód leolvasása*



46. ábra. *A leolvasott vonalkód tulajdonságai*

Végül a kimeneti *Success?* terminálra a vizsgálat eredménye kerül, és ha a leolvasás sikeres volt, akkor a leolvasott vonalkód a *Result XML* kimeneti terminálon XML formátumban megjelenik. Természetesen a lefoglalt erőforrások felszabadítására itt is sor kerül.

10. Összefoglalás

Az elmúlt közel ötven oldalon ismertetésre kerültek azon fejlesztői környezetek és eljárások, amelyek segítségével hatékony ellenőrző rendszer készíthető.

Egy termékellenőrző alkalmazás felépítése komplex, a képfeldolgozó egységeknek gyorsnak és testre szabhatónak kell lenniük, így biztosítva a széles körű felhasználási lehetőséget. A fejlesztési folyamat rendkívül időigényes, valamint alapos tesztelést kíván, kizárva az esetleges hibás ellenőrzési eredményeket. Ebből kifolyólag csak néhány ilyen modult sikerült elkészíteni, ezek a következők: vonalkódot lokalizáló és leolvasó, alkatrész meglétét és polaritást vizsgáló modul. A továbbfejlesztés természetesen folytatódik, ez nemcsak a képfeldolgozást végző modulokra érvényes, hanem az alkalmazás további egységeire is, mint például adatbázis-kapcsolat kiépítése, statisztikák megtekintése, egységes felhasználói felület létrehozása, jogosultságok kezelése és nyelvi lokalizáció megvalósítása. A cél egy olyan alkalmazásnak az elkészítése és a termelésben való használata, mely a konkurens, ilyen jellegű szoftverekkel szemben könnyebben konfigurálható és szélesebb körben használható.

11. Irodalomjegyzék

[1] Rick Bitter, Taqi Mohiuddin, Matt Nawrocki: *LabVIEW Advanced Programming Techniques.* CRC Press LLC, 2001.

[2] Thomas Klinger: *Image Processing with LabVIEW™ and IMAQ™ Vision.* Prentice Hall PTR, 2003.

[3] http://en.wikipedia.org/wiki/LabVIEW

[4] National Instruments LabVIEW Help

[5] NI Developer Zone (http://zone.ni.com)

[6] NI Vision Concepts Manual

[7] NI Vision Assistant Tutorial

[8] NI IMAQ Help