TDK Dolgozat 2014

Mobilis robot vizuális navigációja hibrid architektúrájú képfeldolgozó algoritmusok használatával

Készítette: Péter Gábor

Konzulens: Kovács Gábor

Budapesti Műszaki és Gazdaságtudományi Egyetem

Budapest, 2014

Tartalomjegyzék

Összefoglaló

Napjainkban egyre inkább teret nyer a különféle autonóm mobilis robotplatformok használata, és a közeljövőben a terület további fejlődése várható. Az ilyen egységek fejlesztésekor kulcskérdés a lokalizáció és a navigáció problémája, melyre a rendelkezésre álló számos lehetőség közül a vizuális elvű, képfeldolgozáson alapuló megoldások is használatosak. Ennek azonban szűk keresztmetszete a számítási kapacitás igény, melyet a mobilis egységek esetén mind a költség, mind pedig az autonomitást csökkentő energiafogyasztás korlátoz. Ezen problémára jelenthetnek megoldást az újszerű, hagyományostól eltérő architektúrákat használó képfeldolgozó rendszerek.

A dolgozat egy vizuális navigációt alkalmazó autonóm mobilis robot képfeldolgozási problémájának megoldását ismerteti. A minél rugalmasabb és kisebb költségű felhasználhatóság érdekében a navigáció passzív markereken alapul, melyeket eltérő színű sávok azonosítanak. Ennek megfelelően a képfeldolgozó rendszernek a kép kiolvasása után színszűrést, majd a már színenként szegmentált, bináris képeken a markerek megkeresését, azonosítását, valamint kamerához képesti pozíciójának meghatározását kell elvégeznie.

A kiinduláskénti rendelkezésre álló differenciális meghajtású platform egy hibrid, FPGA-t és beágyazott mikroprocesszort is tartalmazó irányítórendszerrel rendelkezik, mely egy alacsony költségű kameramodullal is kiegészítésre került. A hibrid architektúra lehetőséget biztosít a képfeldolgozási feladatok párhuzamos futtatására, és így a feldolgozási sebesség növelésére. Míg az FPGA párhuzamos műveletvégzésre képes, de memóriaterülete korlátozott, addig az általános célú mikroprocesszor a komplex, nagyobb tárigényű algoritmusok futtatására alkalmas. Az architektúra optimális kihasználására az elvégzendő műveletek két csoportra, pixel- és képszintű műveletekre oszthatók. A pixelszintű műveletekhez csupán az egyes pixelek adataira, míg a már bináris képeken futtatott képszintű műveletekhez a teljes kép ismeretére szükség van.

A dolgozatban ismertetett megoldás a kamerából sorfolytonosan érkező pixelek feldolgozását az FPGA-n valósítja meg, gyakorlatilag kiolvasásukkal egy időben, a kamera felbontásától és frissítési frekvenciájától független módon. A pixel szintű feldolgozás magába foglalja a színtérkonverziót és a HSL-színtérben történő küszöbözést, melynek során a markerszíneknek megfelelő bináris képek pixelei párhuzamosan állnak elő. Ezek a bináris képpontok azután egy közös használatú memóriaterületen keresztül kerülnek átadásra a beágyazott processzor számára, mely a további műveleteket a teljes képen végzi el. Mivel ezek az algoritmusok a már szín szerint szegmentált bináris képeket használják, ezért futási sebességük a kisebb teljesítményű beágyazott processzoron is megfelelő, így végrehajtásuk mellett a processzor képes a robot mozgásának irányítására is.

1. Bevezetés

Napjainkban egyre nagyobb teret nyernek a különféle mobilis robotplatformok. Ilyenek szállítják a munkadarabokat a gyártósorok gépei között, mobilis robotok cipelik a kórházi ebédet és szennyest, a legkisebb autonóm platformnak tekinthető robotporszívók pedig egyre inkább otthon hasznos segítőtársává válnak.

Önműködő mobilis egységek használatakor kulcskérdés a lokalizáció és a navigáció problémája, azaz annak megoldása, hogy a robot tudja, hol helyezkedik el és merre kell mozognia. A problémára számos megoldási lehetőség adódik, köztük rádiós alapú, ultrahangos vagy vizuális navigációs módszerek.

Az utóbbi években egyre több területen, mint például a minőségellenőrző cellákban vagy a felsőbb (illetve napjainkban már közép-) kategóriás gépkocsikban találkozhatunk kamera alapú megoldásokkal. Ennek oka a technológia fejlődése, mely az egyre gyorsabb és nagyobb kapacitású processzorok megjelenésével azonos műveletek elvégzése mellett drasztikusan lerövidítette a képfeldolgozó algoritmusok futási idejét.

Ezen processzorok azonban a sebesség növekedésével gyakran nagyobb villamos fogyasztásúak, ráadásul egyelőre áruk is elég borsos, így mobilis egységek számára nem mindig elérhetők. Az egyszerű sebességnövelés mellett a másik fejlődési irány a különféle, egy évtizeddel ezelőtt még szokatlannak számító architektúrák használata: többmagos processzorok, GPU-k, vagy akár FPGA-k alkalmazása.

A dolgozat egy mobilis robot vizuális alapú navigációjának megvalósítását mutatja be. A megoldás a fent említett elvet követve egy kis teljesítményű beágyazott processzor és egy FPGA-chip alkotta hibrid architektúrán került megvalósításra, kihasználva a rendelkezésre álló párhuzamosítási lehetőségeket.

A probléma specifikálása után a felhasznált hardver- és szoftverelemeket mutatom be. A 4. fejezet részletesen ismerteti a hibrid architektúrán megvalósított képfeldolgozási algoritmus lépéseit, míg az 5. fejezet a navigáció módszerével foglalkozik. Az eredmények értékelése után a dolgozatot a további fejlesztési lehetőségek összefoglalása zárja.

2. Marker alapú navigáció

A dolgozat alapját egy mobilis robot navigációs feladatának a megoldása jelentette. A követelmény egy előre megadott pálya követése, mely azonban ismeretlen környezetben található. Az egyszerű konfigurálhatóság és nagyfokú flexibilitás részeként fontos szempont, hogy új környezetben is gyorsan kijelölhető legyen az pálya. Egy ilyen rendszer sok helyen hasznos segítőtárs lehet, például múzeumokban, ahol új tárlatok esetén mindig új pálya kerül kijelölésre, valamint kórházakban, ahol például előre megadott útvonalak közül kell az aktuálisan kijelöltet követni. Jó példa erre a budapesti Honvéd kórházban kiépített rendszer. [13] Természetesen katonai alkalmazások is léteznek, ahol az értékes emberélet helyett a modern autonóm eszközök vesznek részt a veszélyes bevetéseken.

A mobilis egységek helyzetének meghatározására számos lehetőség áll rendelkezésre. A csupán relatív pozíciót szolgáltató dead reckoning módszerek, amelyek például a robot kerekeinek elfordulása alapján számítják a pozíciót, önmagukban nem elégségesek, mindenképpen szükség volt egy globális pozíció-meghatározási eljárásra is. A felmerülő lehetőségek közül a rádiójel alapú megoldások a magas költségek miatt elvetésre kerültek, a tervezett beltéri használat esetén pedig az önmagában amúgy sem elegendően pontos GPS-alapú megoldás sem jöhetett szóba.

Beltéri navigáció esetén a kézben tartható fényviszonyok lehetővé teszik a robusztus képalapú pozícionáló eljárások használatát. Ezen megoldások közül az egyik csoport esetén a kamera, vagy kamerák rögzített és előre ismert pozícióban találhatóak meg, míg a másik nagy csoport esetén a kamera magán a mobilis roboton kap helyet. A fix kamerás látórendszerek nagy előnye, hogy a kamera által látott környezet egy része nem változik, így egy referenciától való eltérés érzékelése sokkal kevesebb erőforrást igényel, mint a mobilis kamerával történő képfeldolgozás. Emellett biztosítható, hogy a kamera elhelyezése fizikailag megakadályozza a követendő robot esetleges elvesztését. Ez mennyezetbe szerelt, lefele néző kamerákkal könnyen megvalósítható, és amennyiben a robot nem túl alacsony, akkor majdnem biztosan kizárható, hogy valamilyen objektum kerül a robot és a kamera közé, mely zavarhatná a felismerést. A fix kamerás rendszer hatalmas hátránya ugyanakkor az ára, továbbá az előnyeként elmondható helyismeret egyből hátránnyá válik, amikor a rendszer újrakonfigurálása szükséges.

A mobilis robotokra szerelt kamerák esetén a telepítés költsége jóval alacsonyabb lehet, hiszen, míg a fix rendszer esetén térrészenként szükséges egy kamera, addig ebben az esetben robotonként van szükség egy darab érzékelőre. Nagy mozgástér és kevés, esetleg egyetlen robot esetén a robotra szerelt kamerás rendszer telepítése sokkal olcsóbb. További előnye e rendszernek a flexibilitás. Míg fix kamerák esetén az összes kamera áthelyezését igényli a rendszer, addig ebben az esetben mindössze a robot új munkatérben történő elhelyezése, és a képfeldolgozást segítő objektumok elhelyezésére van szükség.

Mindkét típusú eljárásban gyakran alkalmazott módszer, hogy az ismert, meghatározandó pozíciójú pontok képen való azonosítását fizikailag is megkönnyítik. Ehhez a környezettől a képen könnyen elkülöníthető objektumokat, ún. markereket használnak. A markerek kialakításukat tekintve lehetnek aktívak, amelyek általában infravörös fényt bocsátanak ki. Ezek előnye, hogy tág megvilágítási tartományban is robusztus azonosítást tesznek lehetővé, hátrányuk viszont, hogy tápellátást igényelnek. A passzív markerek nem bocsátanak ki fényt, a környezettől elütő színük és/vagy formájuk segíti azonosításukat a kameraképen.

Mivel a robotra szerelt kamera esetén a markerek megtalálása mellett a pozíció meghatározásához azok azonosítására is szükség van, a markereknek valamiféle azonosítót is hordozniuk kell. Ez megvalósítható a markerkomponensek struktúrájának megfelelő megválasztásával (hasonlóan pl. a QR-kódhoz), illetve színek használatával, vagy a két eljárás kombinálásával. A markerkomponensek struktúráján alapuló módszer hátránya, hogy kis felbontás vagy távoli marker estén igen érzékeny a képen található zajokra.

A minél rugalmasabb működés érdekében további cél volt a teljesen autonóm működés, melynek biztosításához a képfeldolgozás magán a robotplatformon történik. A markerek kiválasztásánál is fontos szempont volt az alacsony bekerülési költség és az egyszerűség, a képfeldolgozás robusztussága iránt támasztott követelményeket is mindvégig szem előtt tartva. A markerek kiválasztásánál nagy jelentősége volt a tervezett felhasználás jellegének, helyszínének. A robot beltéri használatra készült, így feltételezhető, hogy a talaj teljesen vízszintes lesz a felhasználás során, míg a megvilágítás vagy oldalról érkező természetes fény lesz, vagy pedig felülről érkező lámpafény. Ezen felhasználási körülmények ismeretében a választás hengeres markerekre esett. Ezek nagy előnye, hogy a forgásra invariánsak, így a roboton elhelyezett kamera minden irányból teljesen egyformának látja. Szükséges továbbá a markerek egymástól történő megkülönböztetése a sorrendiség miatt. Ezen

követelményeknek együttesen eleget tevő megoldás a színkódok használata. A megoldás előnye, hogy nem igényel nagy részletességű képet, mert a markereket csak a színek sorrendje azonosítja, cserébe mindenképpen színes kamerára van szükség a feldolgozás elvégzéséhez. A megfelelő színek kiválasztásával biztosítható, hogy az adott színek a környezetben csak ritkán, együttesen pedig lehetőség szerint egyáltalán ne forduljanak elő. A színek helyes kiválasztása a későbbi feldolgozást nagyban megkönnyíti, ezért kiválasztásukat a kamerával való tesztelés előzte meg.

A végleges színkódot négy sáv: egy piros, egy zöld, egy kék, továbbá egy sárga színű alkotja. Ez összesen 24 különböző marker létrehozását teszi lehetővé, mely a tesztek során elegendőnek bizonyult. A rendszer az egyszerű felépítés miatt flexibilis, így igény esetén a színek lecserélhetők, amennyiben az új munkatérben az itt megadott színek túl gyakran előfordulnak.

1. ábra - Felhasznált markerek

A robot pályáját markerekkel lehet kijelölni, ennek megfelelően a pálya eltárolása egy vektorban történhet, amely az útvonalat alkotó markerek azonosítóit tartalmazza a megfelelő sorrendben. A robot a navigációs feladat során markertől markerig halad, az aktuálisan közelített markert pedig a kellő közelség elérése után körívben kezdi kikerülni. Ennek a mozgásnak a megvalósításához szükséges a markerek irányának meghatározása a robot hosszanti tengelyéhez képest, mely praktikus okokból egyben a kamera optikai tengelye is, továbbá szükséges a markerek távolságának meghatározása a robothoz képest. A robot a visual servoing [11] elvet követve halad végig a kijelölt pályán.

3. A rendszer elemei

A következőkben a fejlesztés során felhasznált eszközöket ismertetem, kiemelve azoknak a megoldás során kihasznált tulajdonságait.

3.1.Mobilis robot platform

A fejlesztéshez a kiindulópontot a National Instruments Robotics Starter Kit II robotja[1], a DaNI jelentette. A robotplatform a Pitsco[2] cég által gyártott elemekből épül fel, kiegészülve egy National Instruments gyártmányú vezérlőkártyával. A platform egy differenciális meghajtású, alátámasztásként omnidirekcionális kereket alkalmazó szerkezet, melyen található egy Parallax Ping)))[3] típusú ultrahangos távolságmérő szenzor. Ez a robot elején, egy szervómotor által mozgatott konzolon került elhelyezésre, mely $\pm 90°$ -os forgatást tesz lehetővé a függőleges tengely mentén. A szenzor 2cm és 3m közötti kontaktus nélküli távolságméréseket tesz lehetővé. A robot hajtott kerekei kvadratúra enkóderrel ellátottak, melyek fordulatonként 400 impulzust adnak ki, így a tengelyfordulatok megfelelő pontossággal mérhetők. Ugyan jelenleg ezt az információt nem használtam fel, de a jövőbeli fejlesztéseknél ezek a mérések is felhasználhatók a robot pozíciójának és orientációjának meghatározásához.

3.2.Fedélzeti irányítórendszer

A robot vezérléséről egy National Instruments (NI) gyártmányú sbRIO 9632-es beágyazott vezérlő és adatgyűjtő kártya gondoskodik. Az sbRIO az NI RIO (Reconfigurable IO) termékcsaládjának egyetlen kártyára integrált tagja, mely egy FreeScale mikroprocesszor mellett egy FPGA-modult is tartalmaz. A gyártó célja a termékcsalád kifejlesztésekor az volt, hogy a ki- és bemenetek alacsony szintű, de nagy sebességet igénylő feldolgozását egy FPGA-n megvalósított algoritmus végezhesse, a további feldolgozást pedig már a mikroprocesszoron, a megszokott környezetben lehessen megvalósítani. Maga a termékcsalád általános felhasználású, nem kifejezetten képfeldolgozási feladatokra szolgál.

[1] http://sine.ni.com/nips/cds/view/p/lang/hu/nid/208018
[2] http://www.pitsco.com/
[3] http://www.parallax.com/product/28015

A kártyán található FreeScale processzor 400MHz-es órajelen működik, a Xilinx Spartan 3-as chip[4] 2 millió kapuval rendelkezik. Az sbRIO 9632-es rendszeren található modul legfeljebb 46080 logikai cella illetve 720kbit blokkmemória használatát teszi lehetővé.

A kártyán 110 darab 3.3V-os digitális I/O található, melyek 5V/TTL kompatibilisek. Analóg bemenetből 32 darab, 16 bites, maximum 250kS/s mintavételi frekvenciájú, kimenetből pedig 4 darab 16 bites, 100kS/s sebességű helyezkedik el a kártyán. További eszközök, ún. C-modulok csatlakoztatására további bővítő portok állnak rendelkezésre.

Az sbRIO modul 10/100-as Ethernet interfésszel is rendelkezik. A programozás, illetve a futásidejű monitorozás és adatgyűjtés így TCP/IP hálózaton keresztül végezhető, de a megfelelő szoftvermodulokkal a kártya akár HTTP- vagy FTP-szerverként is funkcionálhat.

3.3.Kameramodul

A robotplatform eredetileg nem rendelkezik a képek elkészítéséhez szükséges kamerával, így első kiegészítésként egy kameramodul illesztése történt meg. Érzékelőként az Aptina[5] cég MT9V034 típusú CLCC tokozású CMOS szenzora került felhasználásra [1]. A kamera WVGA, azaz 752*480 felbontású képek készítésére képes, mindezt igen meggyőző 60-as másodpercenkénti képkocka szám mellett. Ennek a típusnak létezik fekete-fehér, illetve színes (Bayer–maszkos[6]) változata is, a fejlesztés során ez utóbbit használtam.

A kamera 3.3 Voltos tápfeszültségről üzemel, ami lehetővé tette a szintillesztés nélküli használatot az FPGA szintén 3.3 Voltos I/O struktúrájával. A szenzor illetve a modul nem tartalmaz órajel-generátort, ennek előállítása az FPGA feladata lett, ajánlott értéke 26,67 MHz. Az FPGA alap órajele 40 MHz, amelyből előosztó és szorzó segítségével pontosan beállítható a kívánt érték. Az eszköz fogyasztása teljes kihasználtság esetén is kevesebb, mint 160 mW, így az sbRIO I/O portjain található 5 voltos csatlakozók bármelyike alkalmas a megfelelő tápteljesítmény biztosítására. A szenzor konfigurálása I2C buszon történik, a képek kiolvasása pedig történhet soros, illetve párhuzamos módon is. A modul eleve rendelkezésre állt, rajta három hüvelysor formájában a használatához szükséges összes csatlakozási pont kivezetésével. A kamera legfontosabb paramétereit az 1. táblázat foglalja össze.

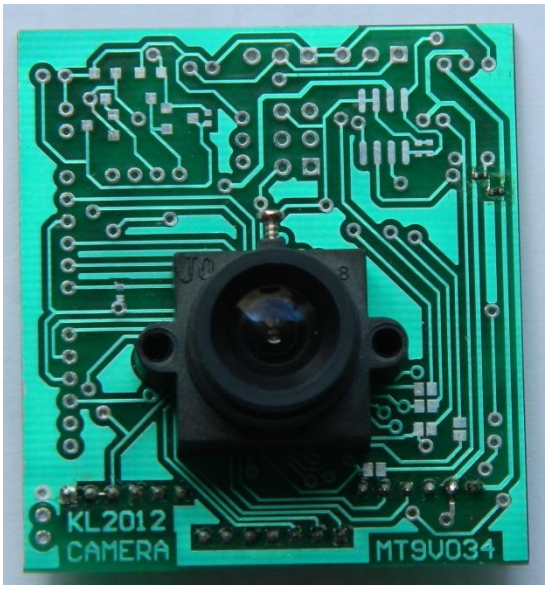
[4] http://www.xilinx.com/products/silicon-devices/fpga/spartan-3.html
[5] https://www.aptina.com/
[6] *A Bayer maszk a szomszédos 2*2 pixeles szenzorterületek előtt található Kék-Zöld-Zöld-Piros szűrők összessége [10]*

Paraméter	Érték
Szenzorformátum	1/3 inch
Aktív tartomány mérete	4.51mm(H) x 2.88mm(V)
Aktív pixelek száma	752H x 480V
Pixelméret	6.0 x 6.0µm
Színszűrő	RGB Bayer maszk
Zártípus	TruSNAP™ Global shutter
Maximális Master órajel	27MHz
Teljes felbontás	752 x 480
Frame rate	60 fps (teljes felbontás esetén)
ADC felbontás	10 bit
Érzékenység	4.8V/lux-sec (550nm)
Dinamikatartomány	>55dB
Tápfeszültség	3.3V ±0.3V
Fogyasztás	<160mW
Működési hőmérséklet	-30°C-tól +70°C-ig
Tokozás	48 pines CLCC

1. táblázat - A kamera paraméterei

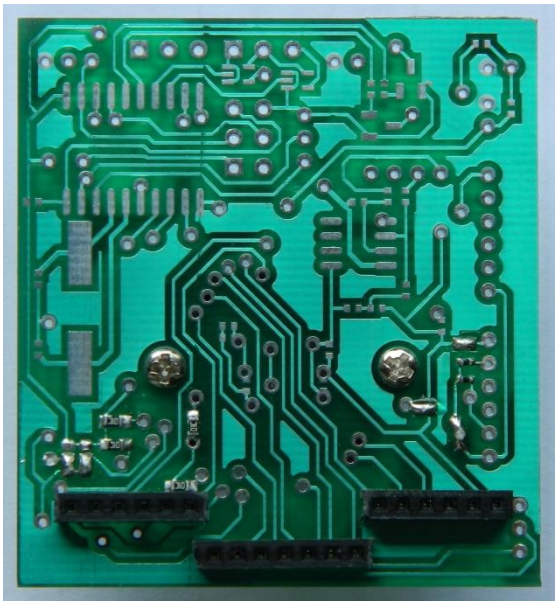
A paraméterek közül kiemelendő a Global Shutter[7], mely biztosítja, hogy a megszokott Rolling Shutter-rel ellentétben az összes pixel azonos időben kerüljön rögzítésre, ezáltal a kép gyors mozgás esetén is konzisztens marad, mely egy mobilis robot navigációja esetében fontos szempont.

[7] https://www.aptina.com/products/technology/aptina_global-shutter.jsp



2. ábra - A kameramodul az optika felől nézve

A párhuzamos adatbitek közül a tárolási kapacitás hiányában csupán az ADC felső 8 bitje került felhasználásra, így egy képkocka eltárolásához elegendő egy bájtnyi terület, míg 10 bit esetén bájtszervezésű adatstruktúrák használata esetén már két bájtra lenne szükség képkockánként, mely megengedhetetlen pazarlás az amúgy is szűkös memóriaméret miatt.



3. ábra - A kameramodul csatlakozó felöli oldala

A 16:9 arányú képet szolgáltató kamera 90 fokkal elforgatva került felszerelésre, így a kép szélessége lecsökkent, de magassága megnőtt. Mivel a kamera egy egy helyben elfordulásra

képes robotplatformon került elhelyezésre, ez nem okoz problémát, vízszintes síkban a látótér pásztázása egyszerűen megvalósítható.

3.4.LabVIEW

A kamerakép feldolgozására a National Instruments grafikus VHLL (Very High Level Language, nagyon magas szintű nyelv) fejlesztőkörnyezetét, a LabVIEW szoftvert használtam. Az azonos hardver és szoftvergyártó miatt a kompatibilitási problémák jelentősen lecsökkennek, mint ahogy a támogatottság is egységes.

Ez a gyakorlatban azt jelenti, hogy általánosan a National Instruments által gyártott programozható hardvereken elérhetőek a PC-s alkalmazásokhoz használt függvények és funkciók, az eltéréseket a célhardverek sajátosságai befolyásolják. Emiatt a szolgáltatások egy része egyes hardverek esetében nem elérhető, míg a hardverekben található speciális perifériák és tulajdonságok miatt egyes esetekben plusz funkciók is elérhetőek. Az alapvető struktúrák viszont minden esetben ugyanazok, mely rengeteget egyszerűsít a fejlesztésen. A LabVIEW magas szintű nyelvét használva például egy egyszerű, semmilyen speciális perifériát nem használó függvényt megírva az futtatható FPGA-n, de akár beágyazott processzoron, vagy a PC-n is. Mindezt anélkül, hogy a „kódban” bármilyen módosítást lenne szükséges végrehajtani.

Természetesen a magas nyelvű programozási nyelvek bizonyos helyzetekben hátrányt jelenthetnek, amikor a tervező a hardver ismeretében tudja, hogy mi lenne az ideális implementáció, miközben a fordító teljesen más megfontolások alapján sokkal kevésbé optimális kód generálására képes csak. Ez a jellegű probléma főleg az FPGA programozásánál okoz gondot, ahol a szintézist végző Xilinx eszközlánc számára csak azt lehet beállítani, hogy méretre, vagy sebességre optimalizáljon, más beállítási lehetőségre nincsen lehetőség. Általánosságban azonban elmondható, hogy a fejlesztési időt nagyon hatékonyan csökkenti az az elv, hogy minden architektúrának közös a nyelve és egy-egy speciális esettől eltekintve nem okoz hátrányt a magas szintű nyelv használata.

A szoftver felépítését tekintve moduláris, az egyes feladatkörökre külön telepíthető modulok állnak rendelkezésre.

Az alacsony szintű feldolgozáshoz a LabVIEW FPGA modult használtam, ebben találhatók meg azok a függvények, melyekkel el lehet érni az FPGA-hoz csatlakozó I/O lábakat, illetve a

különböző memóriákat. A beágyazott processzor programozásához a Real-Time modul kínál lehetőségeket. Az általános célú Freescale processzor programozása egyébként teljesen megegyezik a PC-re történő programírással, azzal a különbséggel, hogy a projektben a fájlok a Real-Time Target alatt jelennek meg.

A képfeldolgozáshoz a Vision eszköztárat került felhasználásra, mely az általánosan elérhető adattípusokat kiegészíti egy kép adatstruktúrával, továbbá különböző képfeldolgozási rutinokkal is kiegészíti a palettát.

4. Képfeldolgozás

A következőkben a vizuális navigáció alapját jelentő képfeldolgozási feladat megoldását, a hibrid architektúrát kihasználó képfeldolgozó algoritmus elemeit ismertetem. Először áttekintem a feldolgozás lépéseit, majd részletesen ismertetem az FPGA-n és a beágyazott processzoron megvalósított komponenseket.

4.1.A képfeldolgozási algoritmus

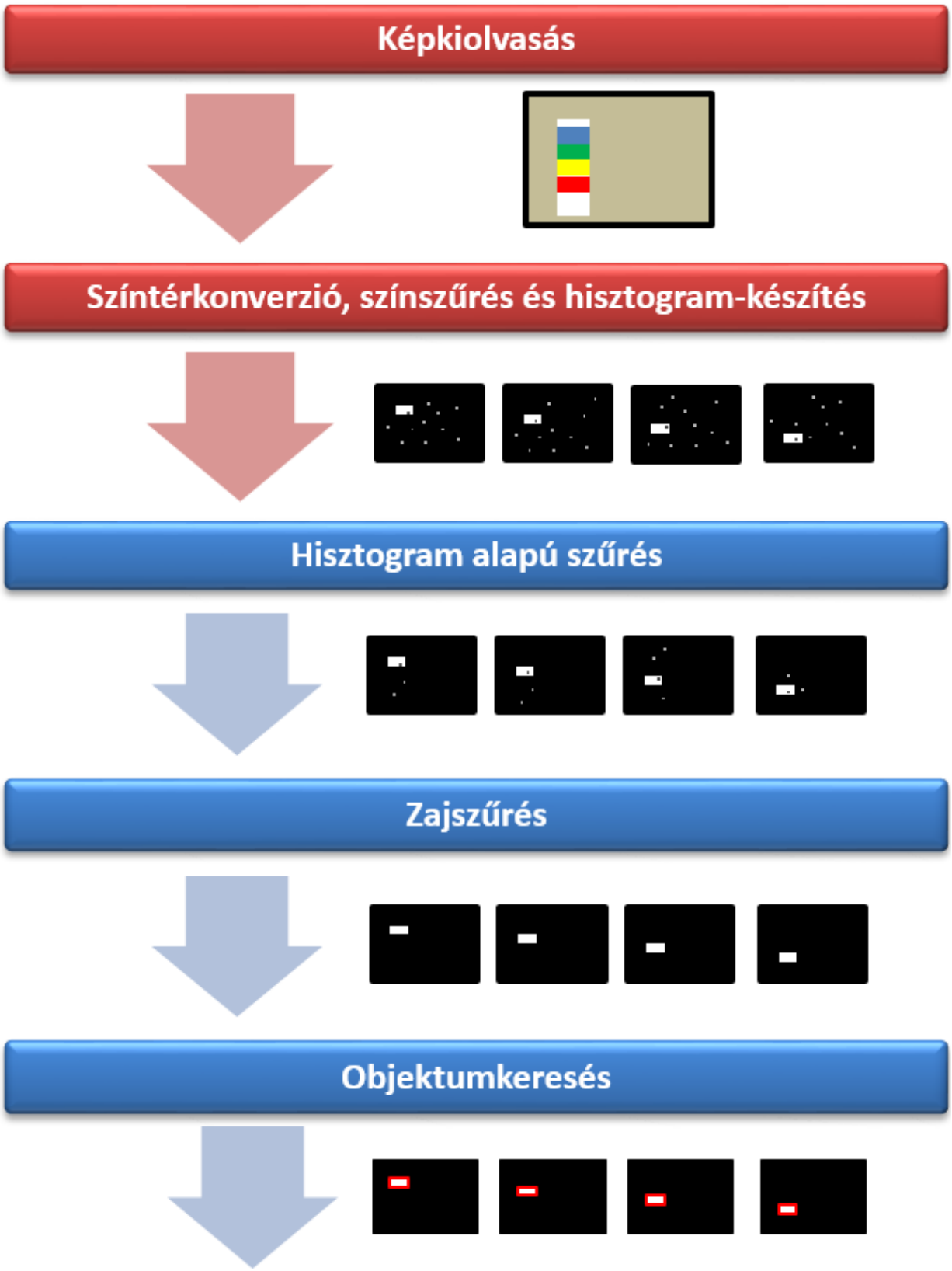
A képfeldolgozás célja, hogy a 2. fejezetben ismertetett marker alapú navigáció számára folyamatosan biztosítsa a kamera által látott markerek színkódját, kamerától vett távolságát, és az optikai tengelyhez képesti relatív helyzetét. Ennek első lépése a kép kiolvasása a kamerából, melyet egy szín alapú szegmentáció követ. A szegmentált képeken ezután objektumkeresés következik, az utolsó lépés pedig a markerek azonosítása.

A rendelkezésre álló hibrid architektúra lehetővé teszi, hogy az egyes fázisok párhuzamosan fussanak, egyrészt magán az FPGA-n belül, másrészt az FPGA és a FreeScale processzor között megosztva. A kamerakép kiolvasásával párhuzamosan az FPGA képes a pixel szintű, azaz a teljes kép ismeretét nem igénylő műveletek elvégzésére. A műveletek másik része a kép-szintű műveleteket jelenti, melyeknél szükséges az egész kép ismerete. Ide tartozik a képen végzett zajszűrés, illetve az objektumkeresés is.

A kamera modul közvetlenül az FPGA-hoz csatlakozik, így az FPGA sorfolytonosan olvassa a szenzor által digitalizált fényintenzitás-értékeket. Az FPGA-architektúra előnyeit kihasználva egy képpont kiolvasásával egyidőben az előzőleg kiolvasott képpontokon elvégezhetőek bizonyos feldolgozási műveletek is. Azonban ezek körének határt szab a szintetizálható blokkmemória maximális mérete, mely a rendelkezésre álló eszköznél csupán a kép egy kis részének egyidejű tárolását teszi lehetővé. Így az FPGA-n csak azon műveletek végezhetők el, azok viszont a kiolvasással párhuzamosan, melyek csupán néhány sornyi pixelértéket igényelnek.

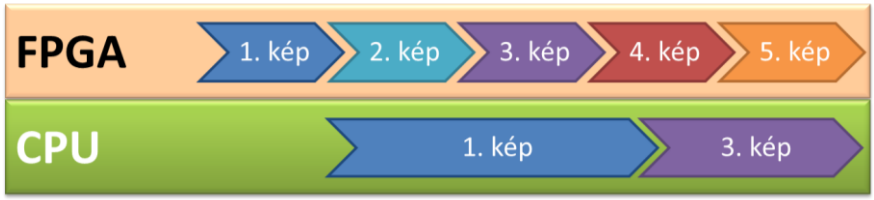
A képszintű műveletek, mint például az objektumkeresés, a megfelelő kapacitású memóriával rendelkező beágyazott processzoron kerülnek megvalósításra. Az FPGA a processzorhoz tartozó, közös használatú memóriaterületre továbbítja a pixel-szinten már feldolgozott képeket, ahonnan a processzor DMA használatával olvashatja ki azokat. A processzor kép-szintű műveletvégzésével teljesen párhuzamosan az FPGA elkezdi feldolgozni a következő képet, így a hibrid architektúra két feldolgozóegysége teljesen párhuzamosan

végzi a számításokat, mely a másodpercenként feldolgozott képkockák számát jelentősen megnöveli az egymás utáni műveletvégzési módhoz képest.



4. ábra - A képfeldolgozás folyamatábrája

Az FPGA feladata a kép kiolvasása, és ezzel párhuzamosan a négy markerszín szerinti küszöbözött bináris képek előállítása, majd ezek továbbítása a processzor számára.



5. ábra - Párhuzamos műveletvégzés hibrid architektúrán

A processzor feladata a megkapott bináris képeken a markerek azonosítása és identifikálása. Az első beérkező képet leszámítva a két feldolgozóegység teljesen egy időben végzi a saját feladatát. A képfeldolgozási feladat elemeit, és azok megosztását a hibrid architektúra elemei között a 5. ábra mutatja be.

4.2.Pixelszintű műveletek az FPGA-n

A kameramodul közvetlenül az FPGA-hoz csatlakozik, így minden egyes pixel mindenképpen áthalad rajta, mielőtt eljutna a beágyazott processzorig. A beérkező pixelek sorfolytonosan érkeznek, így felhasználva az FPGA sajátosságait, nagyon alacsony késleltetés mellett teljesen párhuzamosan feldolgozási feladatokra is használható. A pixelszintű műveletek FPGA-n történő implementációjával jelentősen csökkenthető a továbbítandó teljes adatmennyiség, továbbá a feldolgozás egy része úgy történik meg, hogy mindössze 4 órajelnyi késleltetéssel, azaz 154ns-al az utolsó szubpixel kiolvasása után a processzor már a pixel-szinten feldolgozott képet kapja meg az eredeti kép helyett. Összehasonlításul ez az idő a felhasznált valós idejű processzornak alig több, mint 6 órajel-ciklusa.

A processzoron elvégzendő objektumkeresés gyorsítása miatt az FPGA-s feldolgozás kiegészült egy hisztogram-képzéssel is, melynek segítségével kiválaszthatók a kép azon oszlopai, melyeken nagy bizonyossággal nem találhatók markerek.

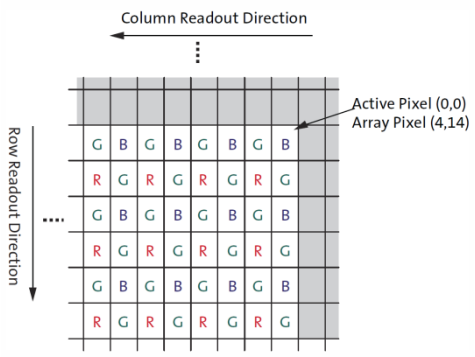
4.2.1. Pixelek kiolvasása

A képfeldolgozás legelső lépése a pixelek sorfolytonos kiolvasása a kameramodulból. Az Aptina MT9V034 kameramodul képkiolvasási szekvenciáját a 6. ábra mutatja be. Ezen az ábrán egyfelől látszik, hogy az egy érzékelővel rendelkező színes kameráknál megszokott módon a vörös, zöld és kék szubpixelek Bayer-maszkos elrendezésben adják ki a kimeneti színes képpontot, illetve látható, hogy a kiolvasás sorrendje sorfolytonos.

Mint látható, a 2×2-es méretű maszk miatt színes kép egyetlen sorának előállításához két sornyi szubpixel-érték feldolgozása szükséges. A páratlan sorokban érkező kék és zöld szubpixeleket egy megfelelő méretű FIFO-ban tárolva a páros sor szubpixeleit olvasva áll rendelkezésre a kimeneti képpont meghatározásához szükséges mind a négy szubpixel. A páros sorokban a piros szubpixelek kiolvasásakor áll rendelkezésre mind a három színcsatorna, melyek együttesen adják a kimeneti pixelt. Memória-takarékossági okokból a zöld csatornát a két szubpixel átlaga adja az implementáció során:

$$(R, G, B) = (R, \frac{G_1 + G_2}{2}, B)$$

A feldolgozás során interpolációt nem használtam, így az effektív felbontás a szenzor natív felbontásának a negyedére csökken. A kiolvasáshoz szükséges műveleteket a 2 táblázat foglalja össze.



6. ábra - Bayer maszk és a kiolvasás sorrendje [1]

Állapot	Aktuális pixel színe	Művelet
1	Kék	Aktuális pixel eltárolása a kék FIFO-ba Ugrás a 2. állapotba
2	Zöld	Aktuális pixel eltárolása a zöld FIFO-ba Ha nem a sor utolsó pixele: Ugrás az 1. állapotba Különben: Ugrás a 3. állapotba
3	Zöld	Zöld pixel kiolvasása a FIFO-ból Átlagolás elvégzése az aktuális értékkel Az átlagolt zöld pixel tárolása egy regiszterben Ugrás a 4. állapotba
4	Piros	Kék pixel kiolvasása a FIFO-ból Zöld pixel kiolvasása a regiszterből RGB-HSL konverzió megkezdése Ha nem a sor utolsó pixele: Ugrás a 3. állapotba Különben: Ugrás az 1. állapotba

2. táblázat - A Bayer maszk dekódolási folyamata

4.2.2. Színtérkonverzió

A markerek azonosítására használt színek szegmentálásához elengedhetetlen színenként egy-egy megfelelően paraméterezett szűrő. A robusztus működést szem előtt tartva számolni kell a megvilágítás intenzitásának a változásával. RGB színtér esetén egy adott szín paraméterei különböző megvilágítások esetén egy forgásparaboloidban helyezkednek el. Ahhoz hogy egy szűrő megbízhatóan működjön, ennek a forgásparaboloidot a paramétereit kellene meghatározni, ami azonban rendkívül komplikált feladat.

A gyakorlatban elterjedt megoldás egy másik színtér, mégpedig az árnyalatot (Hue), telítettséget (Saturation) és megvilágítást (Lightness) alkalmazó HSL-színtér alkalmazása, melybe az RGB-színtér elemei egyértelműen leképezhetők. Mivel a színtér egyik eleme a fényesség, így arra egy tág tartományú szűrőt definiálva a változó megvilágítás szempontjából jóval robusztusabb szűrés valósítható meg.

Mivel a színterek közötti konverzió egy-egy értelmű függvénnyel írható le, ezért a legegyszerűbb és leggyorsabb megoldás az lenne, ha minden egyes RGB-színtérbeli értékhez egy memóriatáblából kikeresnénk a hozzá tartozó HSL-színtérbeli értéket. Ehhez azonban 8 bites csatornánkénti felbontást használva három darab 255^3 elemű LUT-ra (Look Up Table) lenne szükség, ami jócskán meghaladja a felhasznált FPGA kapacitását.

A színtér-konverzió azonban elvégezhető a [4] irodalomban ismertetett algoritmus felhasználásával is, mely az R, G, B-vel jelölt vörös, zöld és kék komponensekből az alábbiak szerint állítja elő a HSL-színtérbeli H, S, L paramétereket:

$$R' = R/255$$

$$G' = G/255$$

$$B' = B/255$$

$$C_{max} = \max(R', G', B')$$

$$C_{min} = \min(R', G', B')$$

$$\Delta = C_{max} - C_{min}$$

$$H = 60° \times \left(\frac{G' - B'}{\Delta} mod 6\right), ha\ C_{max} = R'$$

$$H = 60° \times \left(\frac{B' - R'}{\Delta} + 2\right), ha\ C_{max} = G'$$

$$H = 60° \times \left(\frac{R' - G'}{\Delta} + 4\right), ha\ C_{max} = B'$$

$$S = 0, ha\ \Delta = 0$$

$$S = \frac{\Delta}{1 - |2L - 1|}, ha\ \Delta \neq 0$$

$$L = (C_{max} + C_{min})/2$$

Jelen implementációban sem az első három sorban elvégzett normálás, sem a H csatorna számításánál levő 60-as szorzás sem került ilyen formában implementálásra. Ennek oka, hogy a kimenetek, azaz a H, S, L paraméterek kívánt felbontása is 8 bit, mely megegyezik a bemeneti RGB csatornák felbontásával.

A fenti algoritmus FPGA-n történő implementálását azonban nehezítik a benne található osztás műveletek. Ezek végrehajtási ideje nem determinisztikus és relatíve nagy, így használatuk kerülendő a megcélzott nagy frekvenciájú feldolgozás esetén. Célszerű ezért ezen műveleteket az adott érték reciprokával való szorzásra cserélni. Mivel az osztók mindig a 0-255 tartományban vannak, ezek értékei előre is kiszámíthatók és tárolhatók egy ennek 255 elemű LUT-ban, melyből egyetlen órajel-ciklus alatt kiolvashatók. Az inverzek értékeinek eltárolásakor a 0-hoz tartozó érték is ábrázolható maradt a lebegőpontos típussal, mely az inicializálás utolsó lépéseként végül az 1-hez tartozó értékkel azonos, 65535-ös szaturált értéket vett fel az U16-os számábrázolás során.

Mivel a konverzió bemenő paraméterei és kimenő paraméterei is 8 bites számok, célszerű elkerülni a számítás közben is a törtszámokat, melyek használata nem kíméli az erőforrásokat. A megoldást a logikai eltolás operátor jelenti, melynek segítségével az inverzeket még a LUT-ban történő eltárolás előtt vissza lehet „skálázni” az egész tartományra. Figyelni kell viszont arra, hogy a visszaskálázás a műveletvégzés utolsó lépése legyen, ellenkező esetben számítási hibák lépnek fel. Az eltolások 2^{11}-en nagyságúak, és a PC-n végzett tesztelés során a lebegőpontos számábrázolással, osztásokkal megvalósított algoritmushoz képest nem mutatkozott semmiféle eltérés az inverzeket és eltolásokat alkalmazó számítások esetén, tetszőleges bemenő RGB paraméterek esetén.

A LUT inicializálása fordítási időben történik a PC-n kiszámolt értékek alapján, így az osztások helyett szorzásokká egyszerűsödnek a műveletek, melyek már egyszerűen elvégezhetőek. A magas frekvencia okozta szűkös időzítések betartása érdekében a függvény két részből áll. Az első rész eredményeit a második rész shift-regisztereken keresztül kapja meg, így a két rész pipeline működés szerint fut az FPGA-ban.

A piros szubpixel beérkezését követően egyből megkezdődik az RGB-HSL konverzió, közben pedig párhuzamosan kiolvasásra kerül a kamera-modulból a következő zöld szubpixel.

4.2.3. Szűrés

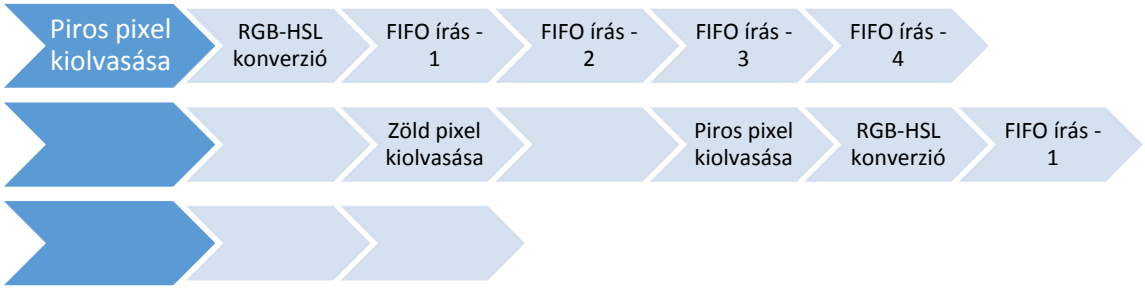
A szűrés célja a markereken található színsávok elkülönítése a háttértől és egymástól. Bemenetét a sorfolytonosan érkező, az előzőek szerint HSL színtérbe konvertált pixelek adják, kimenete pedig 4 bit, melyek mindegyike azt jelzi, hogy az adott bemeneti pixel azonos-e az egyes markerszínekkel. A négy küszöbözendő markerszín a piros, a zöld, a kék és a sárga. Ugyan a következőkben négy szűrő implementálását mutatom be, könnyen belátható, hogy a párhuzamosítás következtében a módszer tetszőleges számú szűrő egyidejű megvalósítását teszi lehetővé.

A HSL színtérbe való áttérés lehetővé teszi a szűrők egyszerű paraméterezését. A kívánt színek szűréséhez a három csatornán szükséges a megfelelő alsó és felső paraméterek egyidejű kiválasztása, melyek között a szűrő átereszti az adott pixelt a küszöbözött, bináris képre. A szűrőfeltételek teljesülését az FPGA csatornánként párhuzamosan vizsgálja, majd az egyes eredmények logikai ÉS kapcsolatát képzi, mely megadja a küszöbözés eredményét:

$$B_{red} = (H_{Rmin} \leq H \leq H_{Rmax}) \wedge (S_{Rmin} \leq S \leq S_{Rmax}) \wedge (L_{Rmin} \leq L \leq L_{Rmax}),$$

ahol $B_{red} = (0; 1)$ a piros színhez tartozó bináris képpont, H, S, L a képpont HSL-színtérbeli paraméterei, $H_{Rmin}, H_{Rmax}, S_{Rmin}, S_{Rmax}, L_{RMin}, L_{Rmax}$ pedig a vörös színszűrő H, S illetve L csatornára vonatkozó áteresztő sávjának alsó és felső határai.

Kihasználva az FPGA párhuzamos műveletvégző képességét, mind a négy markerszín szűrése egy időben történik, mellyel párhuzamosan az FPGA már egyben a következő pixel kiolvasását is végzi.



7. ábra - Pipeline konverzió és FIFO írás

Az FPGA-n implementált szűrők paraméterei akár futási időben is módosíthatóak a processzor és az FPGA közti kommunikációnak köszönhetően.

4.2.4. Hisztogram

A CPU-n való képfeldolgozás első lépése az objektumkeresés, mely régiónövesztéses módszerrel történik. Ez a módszer jelentős zajjal terhelt képek esetén igen sok időt is igénybe vehet, így célszerű a képet az objektumkeresés előtt zajszűrésnek alávetni. Természetesen felmerülő ötlet, hogy a hibrid architektúra előnyeit kihasználva ez a művelet is az FPGA-n, a kép kiolvasásával párhuzamosan történjen, azonban az előzőekben már ismertetett memóriakorlátok határt szabnak az ismert zajszűrési módszerek használatának.

Kihasználható azonban egy, a markerstruktúra által biztosított tulajdonság: mivel a marker színsávjai egymás felett helyezkednek el, a kép azon oszlopaiban, ahol a markerek

megtalálhatók, mind a négy színből található bizonyos számú pixel. Az objektumkeresés tehát jelentősen gyorsítható, ha a képből kiszűrjük azon oszlopokat, melyben valamely markerszínből egy adott értéknél kevesebb pixel található.

A kiszűrendő oszlopok kiválasztása egy egyszerű hisztogram küszöbözésével történhet. Mivel azonban a kép a kamera felől soronként érkezik, ezért a hisztogram elkészítéséhez színenként egy-egy regisztert kell rendelni a kép oszlopaihoz, aminek szintetizálására az eddigiek mellett már nem volt lehetőség a hardver korlátai miatt. A problémát a kamera 90 fokos elfordítása jelentette. Ilyenkor ugyanis a sorok és oszlopok helyet cserélnek, azaz a kép pixelei oszlopfolytonosan érkeznek, azaz a hisztogram elkészítéséhez elegendő négy számláló használata. Mivel az FPGA-n implementált, az előzőekben ismertetett műveletek invariánsak a kép elforgatására, így azok változtatás nélkül alkalmazhatók, különbséget csupán a beágyazott processzoron a képek (pixelmátrixok) összeállításának módja jelent.

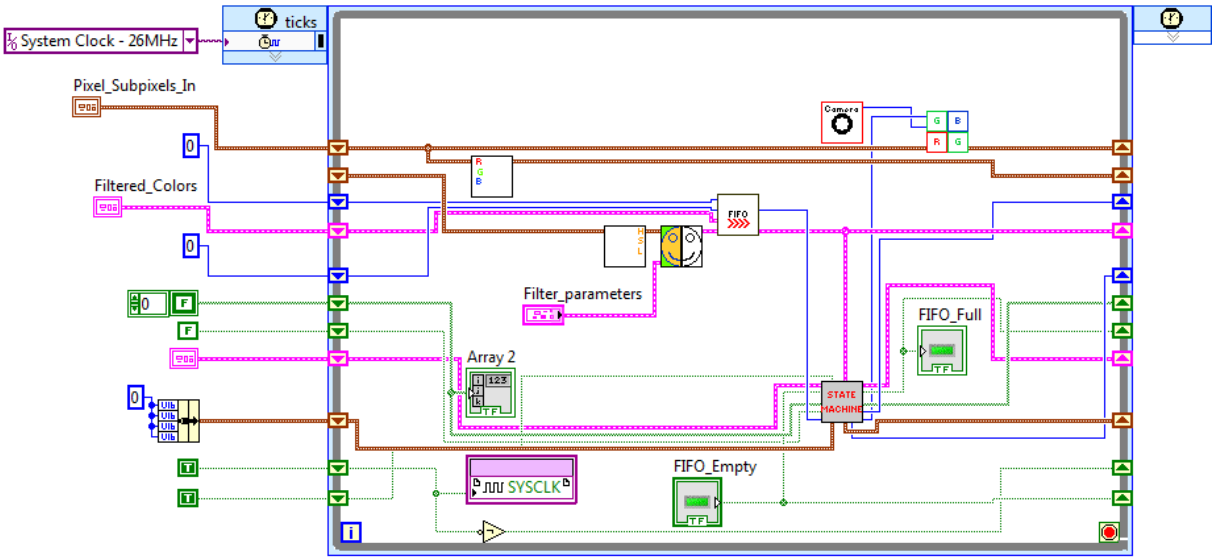
A hisztogram előállításához oszloponként minden színhez tartozik egy számláló, mely mutatja, hogy az adott markerszín hányszor fordult elő az adott oszlopban. Amennyiben ez az érték mindegyik szín esetén átlép egy megadott küszöbértéket, az azt jelenti, hogy az adott oszlop jó eséllyel egy marker vetülete, így érdemes belevenni az objektumkeresés által ellenőrzött területbe.

$$Hist = (C \leq R_{Counter}) \wedge (C \leq G_{Counter}) \wedge (C \leq B_{Counter}) \wedge (C \leq Y_{Counter}),$$

ahol $Hist = (0; 1)$ az adott oszlophoz tartozó hisztogram érték, C egy konstans, melynek nagysága 5, $R_{Counter}$, $G_{Counter}$, $B_{Counter}$ és $Y_{Counter}$ pedig az egyes markerszínekhez tartozó oszloponkénti pixelszámláló értéke.

A 8. ábra mutatja az FPGA által végzett műveletek egymásutániságát. Az időzítésekről a StateMachine feliratú blokk gondoskodik, mely számolja a sorokat, oszlopokat, és olvassa a kamerából érkező vezérlőjeleket. A kamera pixelei 8 bites számok formájában érkeznek, az egyes bitek olvasását és összefűzését a Camera feliratú függvény végzi. Az állapotgép időzítése alapján az aktuálisan beérkező szubpixel eltárolását végzi a GBRG feliratú függvény. Itt történik meg a Bayer maszk dekódolása, az RGB értékek kiszámítása a kimeneti pixelekre. Logikai sorrendben ezt követi az RGB-HSL konverzió, mely függvény két részből áll a pipeline műveletvégzés lehetővé tételéhez. Az első rész az RGB feliratú függvény, a második pedig a HSL feliratú. A színtérkonverzió után a szűrés következik, melyet egy félig színes, félig fekete

fehér smile jelöl. Ezt követi a pixelhez tartozó utolsó művelet, a kimeneti FIFO írása. A hisztogram előállítása magában az állapotgépben kapott helyet, így az nem látható külön függvényként. A FIFO_Full és a FIFO_Empty bináris előlapi szervek az FPGA és a CPU közötti szemaforok szerepét látják el, míg az Array2 feliratú indikátor a hisztogram. A Filter_parameters vezérlőelemen keresztül a processzor menet közben állíthatja a küszöbözéshez használt szűrők minden paraméterét.



8. ábra - Az FPGA-n futó képfeldolgozás LabVIEW blokkdiagramja

4.3.Képszintű műveletek a beágyazott processzoron

Az FPGA által rendelkezésre bocsátott küszöbözött képeket és a hisztogramot felhasználva a beágyazott processzor feladata az objektumkeresés megvalósítása, valamint ennek eredménye alapján a markerek identifikációja. Az azonosított markerek paramétereit ezután a navigációs modul használja fel, mely a képfeldolgozással párhuzamosan futva gondoskodik arról, hogy a robot a markerek által kijelölt pályán haladjon. A megfelelően robusztus működéshez elengedhetetlen, hogy a markerek azonosításának időigénye minél kisebb legyen, valamint az eredménye megfeleljen a valóságnak.

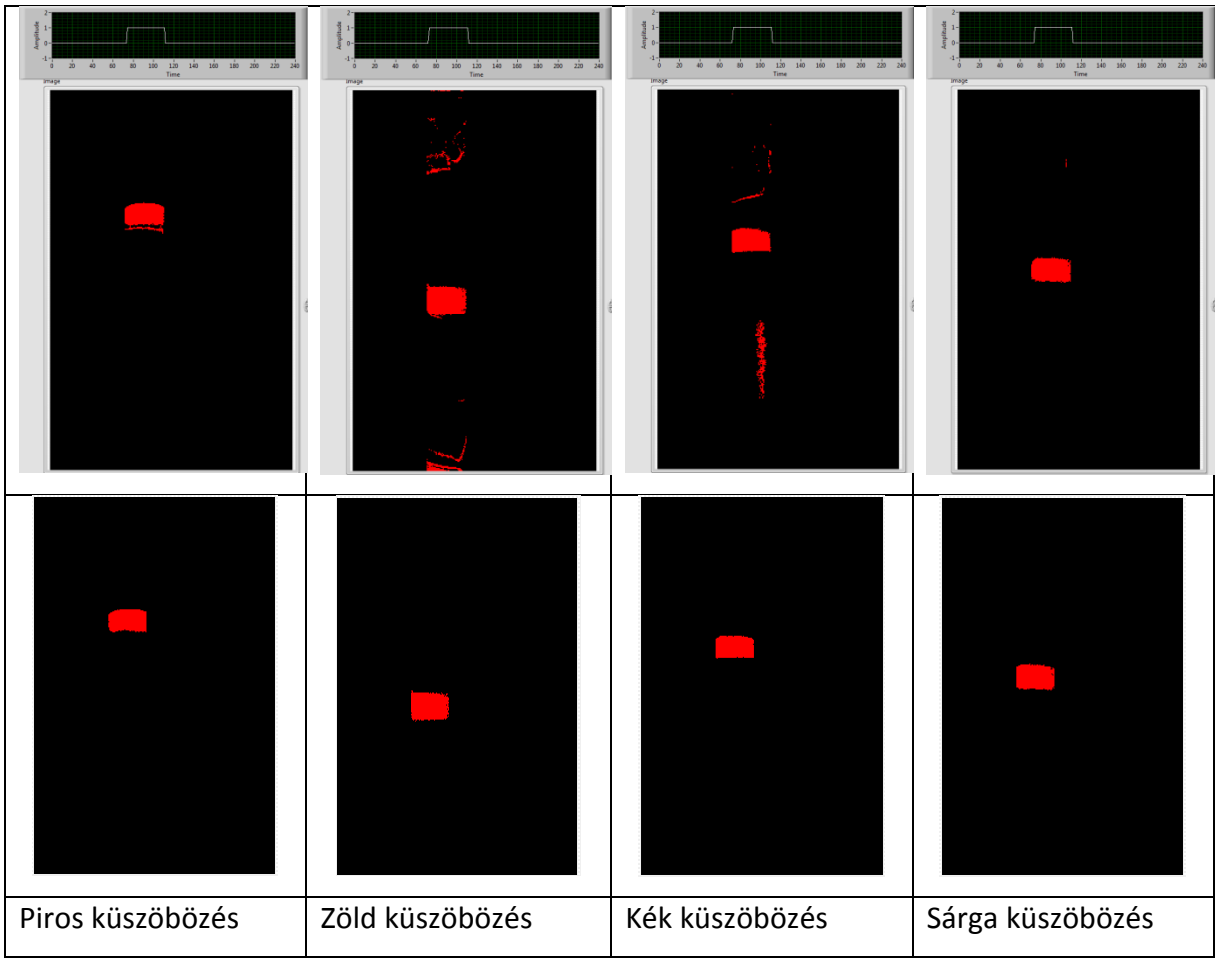
4.3.1. Objektumkeresés

Az objektumkeresés célja a markerek sávjainak megfelelő, összefüggő terület megkeresése a színenként szegmentált bináris képeken, majd a talált objektumok helyzetének meghatározása. A kamera és a szűrők nem tökéletes volta okozta zajok miatt elengedhetetlen a zajszűrés alkalmazása.

Első lépésként a beérkező küszöbözött képek maszkolása történik meg a hisztogram-vektor segítségével. Ez megfelelően paraméterezett szűrők esetén a zaj jó részét egy gyorsan elvégezhető lépésben megszünteti. A művelet végrehajtása során a küszöbözött képek azon oszlopaiban, melyekhez tartozó hisztogram-vektor értékek zérus értéket vesznek fel, az oszlopok teljes tartalma törlésre kerül. Csak azok az oszlopok maradnak érintetlenek, amikhez tartozó hisztogram-vektor érték 1 értékű volt, mely azt jelenti, hogy az adott oszlopban mind a négy markerszín kellő mennyiségben megtalálható. A hisztogram-vektor segítségével megtisztított képeken ezután még erózió műveletek kerülnek végrehajtásra, hogy a markerek oszlopaiból, valamint az ezek fölött vagy alatt lévő háttérből átszűrődő zaj is elnyomásra kerüljön.

Az erózió operátor [14] egy bináris képeken végrehajtható morfológiai művelet. A kimeneti kép egy kernelnek nevezett elemnek a képpel való konvolválásának az eredménye. A kernel elem egy 3*3-as négyzet, melynek minden eleme 1-es. Ezt a bináris képen konvolválva azok az értékek lesznek zérusok, amiket ez a négyzet képes úgy lefedni, hogy a középső elem alatt lévő érték a beérkező képen 0 volt (a kimeneti kép szélessége és magassága 2-2 pixellel kisebb, mint a bementi kép). Így a kis kiterjedésű, szigetszerű zajok eltűnnek a képről, azonban a nagy kiterjedésű, valószínűsíthetően hasznos képkomponensek megmaradnak. A művelet egymás után tetszőleges darabszámban elvégezhető, vagy a több kisebb lépés összevonható nagyobb méretű kernel alkalmazásával.

Az előkészített, szűrt képeken ezután végrehajtásra kerül az objektumkereső függvény. A függvény kimenetét a szűrt képeken található objektumok képezik. A további feldolgozás szempontjából lényeges adatok az objektumot körülvevő téglalap határoló oldalai, illetve a téglalap középpontja.



Piros küszöbözés	Zöld küszöbözés	Kék küszöbözés	Sárga küszöbözés

9. ábra - Küszöbözött képek az erózió művelet elvégzése előtt, és után

4.3.2. Markerek azonosítása

A képfeldolgozás során végzett műveletek eredményét felhasználva, a markerek identifikálásával szolgáltatja a program a navigációs modul számára az információt. Az identifikáció során meghatározásra kerül a marker színkódja, a kamerától mért távolsága, valamint a marker-kamera egyenes optikai tengelytől mért szöge.

Ennek első lépéseként a program meghatározza a négy bináris képen talált objektumok közül az egy markerhez tartozókat. Ehhez a program az egyes objektumokat az x irányú pozíciójuk alapján válogatja ki. Az első megtalált objektum kiválasztása után a többi szín esetén megvizsgálja, hogy vannak-e olyan objektumok, melyek középpontja a már kiválasztott objektum határoló téglalapjának x irányú oldalain belül található. Ezt a feltételt addig hívja

meg iteratívan, ameddig nem talál négy olyan objektumot, melyre teljesül a feltétel. Amennyiben még maradnak további objektumok, akkor a keresés nem áll le. A kamera által látott objektumok száma még az objektumkeresés után sem ismert, így ebben a lépésben dinamikusan történik a markerek eltárolása. Amikor elfogynak az objektumok, a keresés lezárul. A függvény kimenetét a megtalált markerek színkódjai, illetve adatai adják. A már eltárolt markerek színkódjának meghatározásához a benne szereplő sávok középpontjainak y koordináta szerinti sorba rendezése szükséges, mely megadja a színkódot. Az erre szolgáló alfüggvény létrehoz egy 4 elemű segédtömböt, melyben eltárolja a 0,1,2 és 3 számokat. Ezek sorban a piros, zöld, kék és sárga markereket jelentik. A talált objektumok y koordinátái színenként érkeznek a függvényhez, tehát az első y koordináta a piros, a második a zöld színhez tartozik és így tovább. A középpontok magasság szerinti buborékrendezése során a cseréket a program a segédtömbön is végrehajtja, így a segédtömb kimenete a sorba rendezés végén megadja a vizsgált marker színkódját.

5. Navigáció

A navigáció célja, hogy a robotot a kameraképen megtalált markerek közül a sorban következőhöz vezessük, majd annak megközelítése után az előre definiált pálya következő pontját kijelölő markert keressük meg. Ehhez egyrészt ki kell nyerni a képen megtalált markerek pozíció- és orientáció-információit, majd pedig el kell vezetni a robotot a kiadódó pozícióba. Ez utóbbi művelet a visual servoing elven alapul.

5.1.Navigációs adatok kinyerése a képinformációból

A képfeldolgozás során meghatározott markerek közül az éppen közelített marker robothoz képesti helyzete adja a szabályozás alapját, így a távolságnak és aktuális iránytól való eltérésnek a szöge fontos adatok a navigáció számára.

A marker távolsága a képen látott magasságából kerül meghatározásra. A marker magasságának meghatározása az alsó sáv alja és a felső sáv teteje közötti differencián alapul. Minél közelebb van egy marker, ez a távolság annál nagyobb. Az optika torzításait elhanyagolva a kapcsolat lineáris a differencia és a távolság között, a fennálló arányosság együtthatója negatív. A marker irányának meghatározásához az x irányú középpontok kerülnek nagyság szerinti rendezésre, majd a két középső elem átlaga szolgáltatja az eredményt. A kamera látószöge fekvő állapotban körülbelül 60°, azonban elfordítva ez az érték lecsökken nagyjából 45°-ra.

A képfeldolgozás alapú távolság meghatározást hatékonyan képes kiegészíteni az ultrahangos távolságmérő szenzor bevonása a pontosabb eredmény érdekében. Az így létrejövő szenzorfúzió a kalibrálást is elősegíti, mivel a Ping))) szenzor karakterisztikája ismert, így meghatározható az a pont, ahol a robot a marker közelítését abbahagyja és megkezdi a fordulási és kikerülési manővert. Természetesen a mérési pontatlanságok miatt nem konkrét pontok, hanem tartományok kerülnek meghatározásra mind az ultrahangos szenzor, mind a képfeldolgozás alapú távolságmérés esetén.

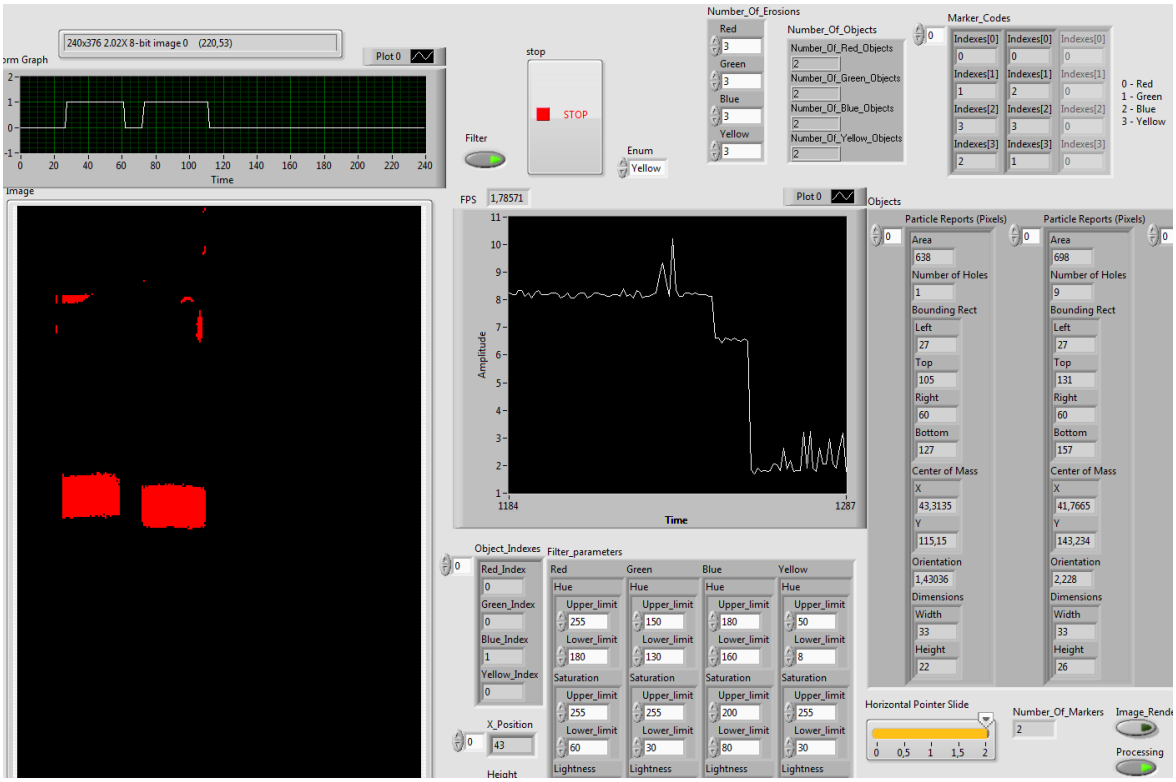
5.2.A navigáció elve

A pályát kijelölő markerek ellenőrzőpontokként szolgálnak a robot számára, a cél az ezeken való végighaladás oly módon, hogy a robot egy előre megadott távolságot elérve a közelítés helyett elkerülő manővert hajt végre az éppen elért marker körül. Az elkerülés pályája előnyös módon egy körív, melynek középpontja maga a megközelített marker. Ez garantálja, hogy a megfelelő távolság megválasztása esetén a robot nem megy neki a markernek üzemszerű körülmények között.

A közelítési fázisban a szabályzó úgy igyekszik vezetni a robotot, hogy az aktuális marker mindig a kép centrumában legyen, vagyis a legrövidebb úton történik a marker megközelítése, ennek az eljárásnak neve visual servoing. A markertől való előre meghatározott távolság elérése után, melyhez a képfeldolgozás mellett az ultrahangos mérések eredményei is felhasználásra kerülnek, a robot megáll, és elfordul 90°-ot. Ezután a dead reckoning elvet követve megkezdi a marker megkerülését, közben a pálya következő pontját kijelölő marker után kutatva.

6. Eredmények

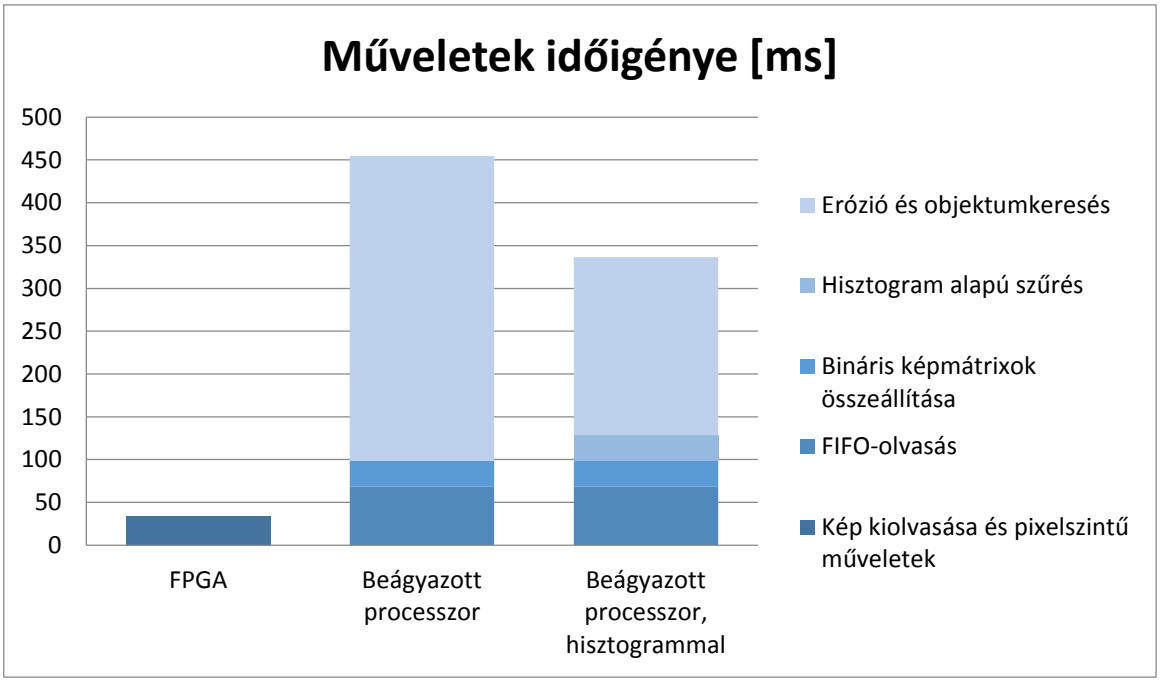
Az implementált program és a feldolgozás hatékonyságának vizsgálatát álló robotra szerelt kamerával végeztem. A kamera látómezejébe különböző színkódú markereket tettem többféle pozícióba. A feldolgozási sebesség vizsgálatához a processzoron futó folyamatban a relatív időmérést lehetővé tevő Tick count blokkokat helyeztem el, melyek segítségével az egyes képfeldolgozási műveletek időigényét pontosan meg tudtam határozni. Az időmérést felhasználva a mérésekből az derült ki, hogy az erózió művelet erőteljes használata egy ideig javít a feldolgozás sebességén és hatékonyságán, azonban előfordul, hogy egyes, főként vízszintes kiterjedésű zajok területe nagyobb, mint magának a küszöbözött markernek a területe. Ilyen esetekben a hisztogram sokkal hatékonyabb megoldást nyújt, mivel csak azokat az oszlopokat engedi át, amelyben az FPGA-n végzett számítások alapján jó eséllyel található marker. A jelenlegi kamerával a tág szűrőparaméter-intervallumok miatt a legjobb eredmény a hisztogramos szűrés és egy 3-as nagyságú erózió együttes használata nyújtja.



10. ábra - Azonosított marker – RBYG

A 10. ábra szemlélteti a feldolgozás eredményét kettő darab, a kamera látóterébe helyezett marker esetén. A bal felül található hisztogramon szépen kirajzolódik a markerek pozíciója az x tengely mentén. A hisztogram alatt látható küszöbözött kép a sárga szín szerinti eredményt mutatja, már a hisztogram alapján szűrt kép formájában. A képen ezután az erózió

operátor a beállított három alkalommal fut le, majd megkezdődik az objektumkeresés. A Number_Of_Objects előlapi megjelenítő elemen láthatóak a talált objektumok darabszámai, mely a várakozásoknak megfelelően kettő darab színenként. A jobb felső megjelenítő szolgál a talált markerek színkódjainak megjelenítésére. Eszerint a kamera előtt egy piros-zöld-sárga-kék és egy piros-kék-sárga-zöld színkombinációval ellátott marker található, mely megfelel a valóságnak. A kép közepén látható másodpercenkénti képkockaszám kijelzőn az ugrásokat az egyes feldolgozási műveletek bekapcsolása okozta. Kettő marker esetén a feldolgozás több ideig tart, mint egyetlen marker esetén, mely annak tudható be, hogy az objektumkeresés függvény ebben az esetben tovább tart a nagyobb objektumszám miatt.



11. ábra - Képfeldolgozási műveletek időigénye

A 11. ábra mutatja a képfeldolgozás különböző fázisainak időigényét. A bal oldali oszlop mutatja az FPGA által egyetlen képkockával töltött feldolgozási időt, mely 34,2ms ideig tart. Ez alatt elkészül mind a négy markerszínre a küszöbözött kép, valamint ezzel párhuzamosan elkészül a későbbi feldolgozást gyorsítani hivatott hisztogram is. A második és a harmadik oszlopok a beágyazott processzor képfeldolgozással töltött idejét mutatják. A középső oszlop a hisztogram használat nélküli feldolgozási időket szemlélteti, míg a jobb oldali, a hisztogramot, mint zajszűrőt alkalmazó megoldás időbeli lefolyását mutatja. A processzor által végzett első fázis a FIFO kiolvasása, mely mindkét esetben azonos ideig, 68,5ms-ig tart. Ebben a fázisban a beágyazott processzor csupán annyit tesz, hogy kiolvassa DMA segítségével a

360960 elemet a közös használatú memóriából, kiolvassa a hisztogramot, majd azonnal visszajelez az FPGA irányába egy szemafor segítségével, melynek hatására az FPGA a következő kép kezdetekor elkezdi annak feldolgozását. A következő fázis a kiolvasott bináris vektor decimálása, majd a négy színhez tartozó vektorok tömbbé formálása. Ez a fázis 30ms ideig tart.

A következő lépés csak a hisztogram felhasználása esetén kerül végrehajtásra, ez a hisztogram felhasználása maszk-ként. Ennek a lépésnek a célja a zajszűrés, mely az objektumkeresést hivatott gyorsítani. Látható, hogy a hisztogram használata 30ms ideig tart, ami főként annak tudható be, hogy a hisztogramnak a kép méretével megegyező méretű tömbbé alakítása a LabVIEW sajátosságai miatt egy 376 elemű ciklust igényel. Az utolsó fázis az erózió és objektumkeresés, melyek együttes idejét szemlélteti a legfelső időszelet. A 30ms-os előnyből a hisztogram nélküli feldolgozás 100ms-os hátrányig esett vissza, ami jól mutatja, hogy az objektumkereső algoritmus futási idejét nagyban befolyásolják a zajok, és a kezdeti képen található kisebb-nagyobb objektumok. A hisztogram feladata a zajok szűrése és látszik, hogy ezt igen hatékonyan teszi: A feldolgozás sebessége 130ms-al javult a hisztogram nélküli esethez képest, ami még úgy is megéri a használatát, hogy a hisztogram vektorból a hisztogram tömb előállítása 30ms-ig tart.

7. Továbbfejlesztési lehetőségek

A feldolgozás sebességégének javítására kézenfekvő ötlet az sbRIO 9632 kártya lecserélése a család egy nagyobb kapacitású tagjára. Itt elsősorban nem a processzor teljesítménye, hanem az elérhető DMA FIFO-k száma lényeges. Amennyiben az FPGA és a processzor közti kommunikációra több csatorna is használható lenne, akkor a beágyazott processzornak nem kéne értékes (és számottevő) időt töltenie a sorosan összefűzött bináris pixelek szétválogatásával, hanem a memóriából közvetlenül az egyes, már szűrt képeket olvashatná ki. Természetesen a nagyobb teljesítményű processzoron a jelenlegi algoritmus is gyorsabban futna, a nagyobb kapuszámú FPGA-chip használata pedig a szűkös memóriakorlátokon enyhítene.

A feldolgozás sebességét ugyan csak kevéssé, robusztusságát ellenben jelentősen javítaná egy jobb minőségű színszűrőkkel rendelkező kameramodul beszerzése. A jelenlegi, kifejezetten low-cost kamera Bayer-maszkjának színszűrői fakók, így a szűrőket kénytelen voltam széles áteresztő tartományúra választani, ami több zajt is átenged a szűrt képre. Ez a megoldás a képszintű műveleteken is eredményezne némi sebességnövekedést. Más irányú megoldás a szenzor cseréje helyett további szenzorokat bevonni a navigációba, és az azok által szolgáltatott adatok és a képfeldolgozási eredmények fúzióját felhasználni. A roboton található, dead-reckoning navigációt lehetővé tevő inkrementális adók és az ultrahangos távolságérzékelő abszolút távolságadata mellett az orientáció mérése és állandó értéken tartása egy egyszerű digitális iránytűvel volna megoldható. A szenzorfúzió alkalmazása megoldást adna akkor is, amikor a képfeldolgozás alapú vizuális navigáció csődöt mond, például amennyiben a markerek takarásba kerülnek.

Ábrajegyzék

Irodalomjegyzék

[1] A szenzor adatlapja regisztráció után elérhető az alábbi címen:

http://www.aptina.com/products/image_sensors/mt9v034c12stc/

[2] Az NI sbRIO 9632 kártya leírása:

http://www.ni.com/pdf/manuals/375052c.pdf

[3] A feszültségstabilizátor adatlapja:

https://www.sparkfun.com/datasheets/Components/LD1117V33.pdf

[4] RGB-HSL konverziós algoritmus:

http://www.rapidtables.com/convert/color/rgb-to-hsl.htm

[5] LabVIEW Course Manual:

http://ni.com/training

[6] NI DaNI adatlapja:

http://sine.ni.com/ds/app/doc/p/id/ds-217/lang/hu

[7] RGB színtér:

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

[8] HSL színtér:

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

[9] Valósidejű rendszerek:

http://www.tankonyvtar.hu/hu/tartalom/tkt/objektum-orientalt/ch05.html

Szerzők: Kondorosi Károly, Szirmay-Kalos László, László Zoltán

[10] Bayer maszk:

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

[11] Visual servoing:

S. A. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. IEEE Trans. Robot. Automat., 12(5):651--670, Oct. 1996.

[12] Visual servoing:

F. Chaumette, S. Hutchinson. Visual Servo Control, Part I: Basic Approaches. IEEE Robotics and Automation Magazine, 13(4):82-90, December 2006

[13] Mobilis robotok a Honvéd kórházban, videóriport, 2012

http://index.hu/video/2012/09/22/a_korhaz_ahol_a_robotok_az_urak/

[14]Morfológiai műveletek

Image Analysis and Mathematical Morphology by Jean Serra, ISBN 0-12-637240-3 (1982)