

Óbudai Egyetem
Neumann János Informatikai Kar
Alkalmazott Informatikai Intézet



TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

SZÁMÍTÓGÉPES PROGRAMOK KÉZGESZTUSOK ÁLTALI VEZÉRLÉSE

Szerző: **Molnár Gyula Máté**
mérnök informatikus BSc. szak, V. évf.

Konzulens: **Dr. Vámosy Zoltán**
egyetemi docens

Budapest, 2015.

TARTALOMJEGYZÉK

ABSZTRAKT	4
ABSTRACT.....	5
1. BEVEZETÉS	6
2. KÉZGESZTUST FELISMERŐ PIACI RENDSZEREK.....	10
2.1 FLUTTER APP (1) (2)	10
2.2 SAMSUNG SMART TV (3) (4)	11
3. A LEHETSÉGES MEGKÖZELÍTÉSI MÓDOK (5)	12
3.1 EGY KÉZGESZTUSFELISMERŐ RENDSZER LEHETSÉGES FELÉPÍTÉSE	12
3.1.1 AZ INFORMÁCIÓ MEGSZERZÉSE	13
3.1.2 A GESZTUS MODELLEZÉSE	13
3.1.2.1 A KÉZ MEGKERESÉSÉHEZ SZÜKSÉGES ELŐKÉSZÍTŐ TECHNIKÁK	13
3.1.2.2 A KÉP NORMALIZÁLÁSA	16
3.1.3 A KÉZ DETEKTÁLÁSA (11)	19
3.1.3.1 KINEMATIKUS MODELL (12).....	19
3.1.3.2 MEGJELÉNÉS ALAPÚ MEGKÖZELÍTÉS (13).....	19
3.1.3.3 ALACSONY SZINTŰ JELLEMZŐKÖN ALAPULÓ MEGKÖZELÍTÉS (12).....	20
3.1.3.4 MESTERSÉGES NEURÁLIS HÁLÓZAT ALAPÚ MEGKÖZELÍTÉS	21
3.1.3.5 GENETIKUS ALGORITMUSON ALAPULÓ MEGKÖZELÍTÉS	21
3.2 ALGORITMUSOK ÁTTEKINTÉSE	22
3.2.1 SIFT (14)	22
3.2.1.1 VALÓSÍDEJŰ KÉZGESZTUSFELISMERÉS SIFT SEGÍTSÉGÉVEL (16)	24
3.2.2 SURF (17)	25
3.2.3 FAST (18)	27
3.2.4 ORB (19).....	28
3.2.5 HAAR JELLEMZŐ ALAPÚ OBJEKTUMOSZTÁLYOZÁS (21)	30
3.2.6 KÉPI NYOMATÉKOK (22)	33
4. SAROKDETEKTÁLÓ ALGORITMUSOK TESZTELÉSE, EREDMÉNYEI.....	34
4.1 A TESZTELÉSEK MEGVALÓSÍTÁSA	34
4.2 EREDMÉNYEK.....	35
5. HAAR JELLEMZŐ- ÉS KONTÚR ALAPÚ ALGORITMUSOK TESZTELÉSE, EREDMÉNYEI	36
5.1 A MEGVALÓSÍTÓ PROGRAM.....	36
5.2 AZ IMPLEMENTÁCIÓ RÉSZLETEZÉSE.....	38
5.2.1 VEZÉRLÉS	38
5.2.1.1 HIBAKERESÉSI MÓD	38

5.2.1.2	FELHASZNÁLÓI MÓD.....	39
5.2.2	VÉGREHAJTÁS.....	39
5.2.3	KOMMUNIKÁCIÓS DLL	42
5.3	EREDMÉNYEK.....	43
5.3.1	A TESZTELÉS RÉSZLETEZÉSE	43
5.3.2	A TESZTELÉS EREDMÉNYEI	45
6.	TOVÁBBI MUNKA	47
7.	ÖSSZEGZÉS	48
8.	SUMMARY	49
	IRODALOMJEGYZÉK	50

ABSZTRAKT

A dolgozat témája az olyan számítógépes programok kézgesztusok általi vezérlése, melyek használatát az efféle kontaktus felhasználó-barátabbá, bizonyos környezetben pedig egyszerűbbé teszi. Társaságban vagy a kanapéra hátradőlve sokkal kényelmesebb a lejátszási lista vezérlése vagy a hangerő szabályozása, ha nem kell a számítógéppel közvetlen kapcsolatot létesíteni.

Manapság már szinte minden számítógép, legyen az asztali vagy hordozható, rendelkezik webkamerával. Az általánosnak számító használatuk (pl. videótelefonálás) mellett kreatívabb ötletekhez is segítő kezet nyújthatnak. A tárgyalt rendszer multimédiás alkalmazások vezérlését oldja meg - mint a lejátszott média manipulálása (indítás, szüneteltetés, lapozás) - a megfelelő statikus kézgesztusok felismerésével.

ABSTRACT

This paper is about controlling computer programs by hand gestures, which makes these applications user-friendlier and easier to use in specific situations. In situations, like being in company or sitting on the couch, can be a piece of cake to manipulate the tracklist or the volume without making any physical contact with the computer.

Nowadays almost every computer, like a PC or a laptop, has a webcam. Beside their general usage (ex. video calls), cameras can help in accomplishing more creative ideas. The created system can control multimedia applications (like start, pause, next, previous) by the proper static hand gestures.

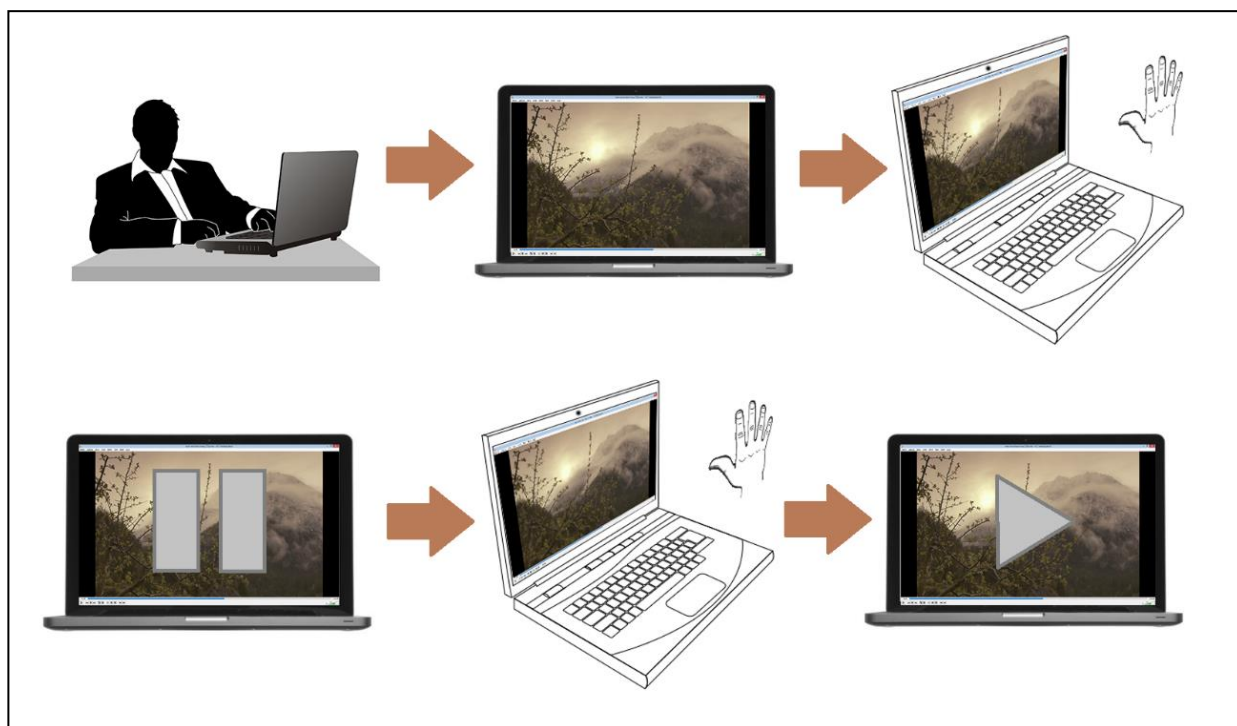
1. BEVEZETÉS

A mostanában jellemző, általános kapcsolata az embernek a gépekkel általában egy olyan perifériával történő interakció, amely nem használja ki az interaktivitást. Ilyen perifériák például a billentyűzet és az egér. Használatuk elsajátításához az embernek meg kell érteniük azok működését, mely elsöre szokatlan, életidegen. Ráadásul bizonyos körülmények között sokkal körülményesebb is az alkalmazásuk, példaként megemlíthető az, hogy oda kell menni a számítógéphez, annak apróbb vezérlése végett. Ezekben a helyzetekben jóval kényelmesebb, interaktívabb és a józan paraszti ész „szerint” is egyszerűbb más eszközökhöz nyúlni, ami közelebb áll az emberhez. A leginkább kézenfekvő megoldás a kezünk használata. Manapság már sok otthonban található olyan számítógép, amelyhez tartozik webkamera is, beépítve a piacon kapható, általános használatra tervezett hordozható számítógép esetén alapperiféria. Gyakorlatilag a két fontos eszköz így könnyen a rendelkezésünkre áll a számítógépes programok bizonyos funkcióinak emberbarátibb irányítására.

Természetesen, véleményem szerint, a programok teljes funkcionalitásának vezérlése nem oldható meg teljes körűen az imént említett technikával, ellenben dolgozatomban nem is ezt kívánom elérni. Kihívásként a multimédiás alkalmazások irányítását tűztem ki. Ezen alkalmazások vezérlése így - azon felül, hogy sokkal izgalmasabbá válik – új utakat nyit meg a felhasználói előtt, illetve én, mint az alkalmazás fejlesztője, rengeteg új ismeretre teszek szert a mélyebb szintű programozásban, valamint a népszerű multimédiás alkalmazások alkalmazásprogramozási interfészével való együttműködésben. Továbbá érdemes megemlíteni a kisegítő lehetőségeket igénylő számítógép felhasználókat is. Vannak emberek, akiknek a mindennapjaik teljesen máshogy telnek, mint az átlagembereknek, korlátozottabb a boldogulásuk. A látássérült emberek ebbe a kategóriába tartoznak, akikre kiemelt figyelmet kell fordítanunk, ugyanis folyamatos segítséget igényelnek bizonyos feladatok elvégzésében. Ha egy látássérült ember szemszögéből közelítjük meg a számítógép használatának problémáját, egészen más kihívással nézünk szembe, programozói szemszögéből is. Egy látássérült ember, habár az összképet, a nagyobb kiterjedésű tárgyakat még viszonylag elfogadhatóan érzékeli, a kisebbekkel már problémái adódnak. Egy számítógép esetében, a látássérült embernek, az egyik legnehezebb feladata a billentyűzet kezelésében rejlik, mivel kisméretű gombokról van szó, melyek nagyban hasonlítanak egymásra. Ha például egy ilyen felhasználó egy filmet néz a számítógépén és szüneteltetni szeretné azt, akkor vagy megkeresi a megfelelő gombot a billentyűzeten, esetleg a kis kurzort irányítja a jó pozícióba és kattint az egérrel, vagy egészen egyszerűen a megfelelő, egyszerű kézgesztust mutatja a képernyő, a

webkamera felé. A háttérben futó, ezt feldolgozni képes alkalmazás pedig elvégzi helyette a film szüneteltetését.

Az előzőekből következik, hogy a leginkább, széles körben használt multimédia-alkalmazások irányítását kívánom új szintre emelni, mint a VLC player, és a Windows Media Player. E programokban a legfontosabb funkciók: a lejátszott média szüneteltetése és folytatása, az előző illetve a következő fájlra való léptetés, valamint a hangerő csökkentése és növelése. Minden egyes funkciót egy megfelelő, statikus kézgesztus reprezentál az alkalmazásomban, azaz nincs olyan elvárás a felhasználó felé, hogy egy összetett mozdulatsort produkáljon a kamerának. Mindössze egy előre meghatározott kézjelet kell mutatnia. A mellékelt ábrán (1.1 ábra) egy egyszerű funkció scenáriója látható. A példa szerint a számítógép felhasználója egy filmet néz. A háttérben fut a gesztusfelismerő program, mely képes e multimédia-alkalmazás egyes funkcióit vezérelni. A film nézője, mivel szüneteltetni szeretné a filmet, a nyitott tenyerét a hordozható számítógépén megtalálható webkamera elé helyezi (megfelelő távolságon belül). Ekkor a gesztusfelismerő alkalmazás feldolgozza a kamera által készített képen megtalálható gesztust és a film lejátszását szüneteltetésre állítja. A felhasználó, később, folytatni szeretné a filmnézést, ezért ismét a nyitott tenyerét mutatja a webkamerának, majd a lejátszás folytatódik.



1.1 ábra: Az alkalmazás működése egy példán keresztül, a példa a filmnézés. A számítógép felhasználója szünetelteti a filmet, majd folytatja azt, a megfelelő kézgesztus webkamera felé mutatása segítségével. (saját ábra, filmrészlet: More Than Honey)

A kézgesztusok kamera általi felismerése azonban több problémát is fölvet.

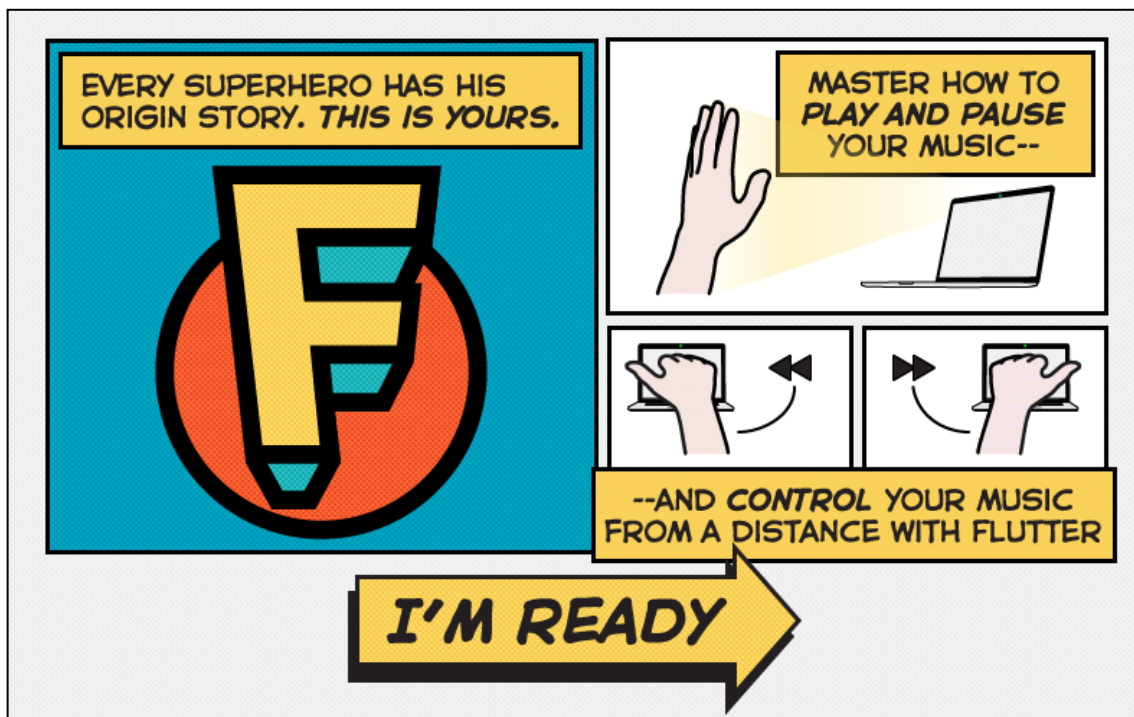
- Egyrészt, már probléma adódhat magukból az előre meghatározott kézgesztusokból is, abban az esetben, ha a képfeldolgozó rendszer túl szigorúan veszi figyelembe az egyes funkciókhoz eltárolt gesztusokat, azaz nem robosztus, nem toleráns még a képleképzésből adódó zajokra sem. Épp ezért egy ebből a szempontból robosztus rendszert szeretnék megvalósítani.
- Másrészt, megszorításokat kell meghatározni, ezek a következők.
 - A kéz távolsága a kamerától megfogalmaz egy elvárást az alkalmazás felhasználójától. Ez pedig az, hogy betartsa az előre definiált, ajánlott távolságot a kamerától, ugyanis csak ekkor tud olyan kép leképződni, amely teljes egészében és felismerhetően tartalmazza az adott gesztust. Ha viszont túl távol van a kéz, hasonlóképp információvesztés következik be, mivel a képen – ha egy átlagos webkamerát veszünk alapul – „elvész” a kéz, azaz az alacsonyabb felbontás miatt az nem lesz felismerhető.
 - Másrészt, igen nagymértékben eltérhetnek a körülmények abból adódóan, hogy mely napszakban, mely szobában, egyszóval milyen megvilágítási körülmények között áll elő a kép a kézgesztusról. Félhomályban vagy teljes sötétségben, illetve erős, beszűrődő fények esetén az alkalmazás használata korlátozottabb.

Az előbbieket összefoglalva tehát néhány alapvető viselkedést el kell várnia az alkalmazásnak a felhasználótól:

- ✓ az előre meghatározott kézgesztusok használatát
- ✓ a kamerától való ajánlott kéztávolság betartását
- ✓ a megfelelő fényviszonyok biztosítását
- ✓ a támogatott alkalmazások használatát

Az előző felsorolásból kiemelendő: a támogatott alkalmazások használata. Azért fontos erről is ejteni néhány szót, mert korábban említettem, hogy csak bizonyos, előre meghatározott programok működését lehet majd befolyásolni. Ehhez az adott alkalmazás API (alkalmazásprogramozói interfész) szolgáltatásainak igénybevétele szükséges. A rendszer képes fölismerni az éppen futó (támogatott) multimédia programot, ezen belül is a legutoljára aktívát, és ennek megfelelően elemzi a beérkező kézgesztusokat, a használatot még jobban könnyítve, jóval felhasználó-barátabbá téve. Természetesen lehetőség nyílik egy konkrét

alkalmazás figyelésének beállítására is, amennyiben a felhasználónak ilyen irányú igényei lennének (például, ha több alkalmazás is fut, mely támogatva van, mindig a preferált figyelembe vétele).



2.1 ábra: Részlet a Flutter App tutorialból

2. KÉZGESZTUST FELISMERŐ PIACI RENDSZEREK

2.1 FLUTTER APP (1) (2)

A dolgozatom alapötletét adó applikáció a Flutter App. Segítségével, kézgesztusokkal lehet vezérelni multimédiás alkalmazásokat és a Microsoft PowerPointot is. A Flutter egy startup vállalkozás fejlesztése volt, melyet 2013-ban a Google felvásárolt. Az alkalmazás statikus gesztusfelismerésen alapul (2.1 ábra). Ez azt jelenti, hogy mindössze a számítógéphez csatlakoztatott webkamera elé kell helyezni a felhasználónak a kezét egy, a kívánt funkciónak megfelelő, helyzetben (ez a megfelelő kézgesztus) és az utoljára aktív alkalmazás megfelelő funkciója hívódik meg. Multimédiás alkalmazások esetében ezek a következő funkciók lehetnek: szüneteltetés, lejátszás, előző, illetve következő médiára való ugrás.

Teszteltem használat közben a programot, összesen 100 gesztust mutattam váltakozva a kamerának, ez alapján születtek az eredmények (2.1 táblázat), melyek a következők. Nappali fénynél kielégítő a program működése, mely egyébként rendkívül gyors. A világos, fehér bőrszínhez közeli homogén háttér azonban sok tévesztést eredményez, szemben a sötét háttérrel, melyből könnyen kitűnik a kéz. Ha ez előbbi esetet nem vesszük figyelembe, jelentősen nő a jó találatok aránya. A hasonló gesztusokat, mint az előző és

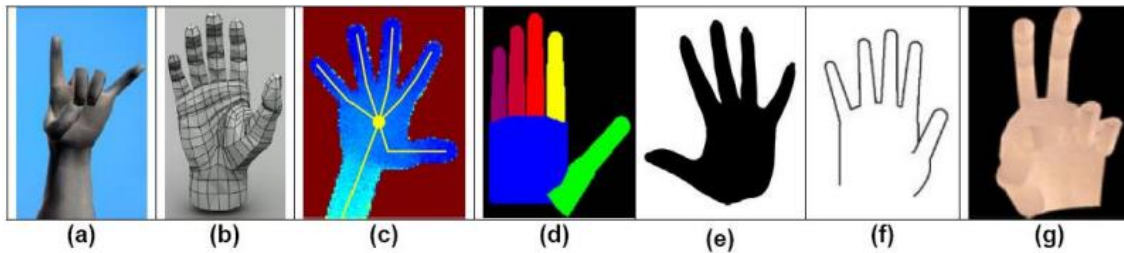
Háttér →	Homogén (világos)	Homogén (sötét)	Inhomogén	Összesen	Összesen (világos háttér nélkül)
Környezet ↓					
Nappali fény	33%	96%	77%	68,7%	86,5%
Esti lámpafény	30%	95%	50%	58,3%	72,5%
Este sötétben	76%	43%	67%	62%	55%
Összesen	46,3%	78%	64,7%	63%	71,34%

2.1 táblázat: A Flutter App tesztelésének eredményei, a sikeres találatok százalékban (saját táblázat)

következő médiára ugrás olykor összetéveszti, illetve a szüneteltetés/lejátszás gesztust (mely a nyitott tenyér) sem mindig ismeri fel a program, ez utóbbit is téveszti olykor. Processzorhasználatot tekintve csekély ráfordítást igényel, mivel mindössze 4-5%-on járátja a központi egységet.

2.2 SAMSUNG SMART TV (3) (4)

A Samsung 2012-től kezdődően az okos-televízióit kamerával, illetve mikrofonnal szereli fel. Ezek az interaktivitást hivatottak növelni és a televíziók funkcióját gyökeresen megváltoztatni. Gesztusfelismerésből a dinamikus került beépítésre a Smart TV-kbe, összetett mozgásra van szükség vezérlésükhöz. A képernyő előtti integetéssel hívható fel az okosteleviszió figyelme arra, hogy funkcióit használni szeretnénk. Ezután különböző lehetőségeink vannak, mint például a hangerő változtatása, váltás a csatornák között, a megjelenített képek méretének változtatása vagy a közösségi (pl.: Facebook) funkciók használata. Működtetni megfelelően jó megvilágításnál egyszerű, azonban az árnyékokra kevésbé toleráns rendszer.



3.1 ábra: Különböző kézmodellező technikák (5) (a) 3D térfogat modell (b) 3D geometrikus modell (c) 3D szkeleton modell (d) színes jelölő alapú modell (e) nem-geometrikus alak modell (f) 2D deformálható sablon modell kontúrja (g) mozgás alapú modell

3. A LEHETSÉGES MEGKÖZELÍTÉSI MÓDOK (5)

Az emberi kéz, felépítéséből adódóan, rengeteg gesztus kialakítására képes. Gesztusokat a mindennapi életünkben, egymással történő kommunikációnk során folyamatosan használunk. Ez a nonverbális kommunikáció része. A verbális kommunikációt nélkülözni kénytelen, fogyatékkal élő emberek pedig csak a gesztusokra támaszkodhatnak. Egyes alkalmazói munkájuk során is aktívan használják kezeik gesztusait. Gondoljunk például a jeltolmácsokra vagy a légiirányítókra a repülőgép indulása előtt. Azonban nem csak az emberek közti interakció során hasznosítjuk a gesztusokat. Különböző területeken, különbözőképpen bír ezek alkalmazása. Mind két-, mind háromdimenziós környezetben jelentős segítséget jelent a kéz általi vezérlés, egér és billentyűzet használata nélkül. Kétdimenziós térben használják például televízió funkciók kihasználására, háromdimenziós térben pedig 3D modellező szoftvereken belül a modell térbeli mozgatására, alakítására. Különböző megközelítési módokat kell használni a más-más alkalmazási terület eltérései miatt. A 3.1 ábra lehetséges 2D-s és 3D-s kézmodellező módszereket mutat be. Az elkészítendő rendszeremben 2D-s környezetben használom a gesztusfelismerést, illetve valóságos helyzetben feltételezem a kéz meglétét, azaz a felismeréshez speciális kesztyűt vagy színes jelölőket nem igényel az alkalmazás. E miatt az ezt kihasználó, illetve kizárólag a 3D-s felismeréshez használt módszereket nem fogom részletezni.

3.1 EGY KÉZGESZTUSFELISMERŐ RENDSZER LEHETSÉGES FELÉPÍTÉSE

A kézgesztusok felismeréséhez egy komplex rendszerre van szükség, amely azonban könnyen elkülönülő, egyszerű, egymással együttműködő modulokból tevődik össze. Mint minden képfeldolgozó rendszerhez, a gesztusfelismeréshez is először az információt kell beszerezni, mely jelen alkalmazás esetében a webkamerából érkező kép. Második lépésként a

keresett objektumot különítjük el a háttértől, azaz kezet keresünk és szegmentáljuk a képet. Harmadik lépésként eltávolítjuk a zajokat a képről (magához a szegmentáláshoz is használunk zajszűrést) és a használt módszernek megfelelően normalizáljuk azt. Negyedik lépésként jellemzőket keresünk a képen. A végső lépés a jellemzők megfelelő felhasználása egy kézgesztus felismerő eljárás segítségével.

3.1.1 AZ INFORMÁCIÓ MEGSZERZÉSE

A fejlesztendő rendszer tekintetében a bemeneti adatot a webkamera által szolgáltatott kép adja, tehát nincs más dolga a felhasználónak csak annyi, hogy a kamera elé helyezze a kezét az ajánlott pozícióba. A különböző igényű alkalmazások miatt azonban más megközelítések is léteznek. Például egy 3D modellező program egy igen pontos gesztusérzékelőt igényel. Tehát itt egy speciális kesztyű adatait használják bemenetként, mely ráadásul rendkívül gyors válaszidővel bír. Ha viszont a fényviszonyok szélsőséges változása esetén is helytálló rendszert tervezünk, akkor a Microsoft Kinect szenzora az, ami jó megoldás lehet. A tervezett gesztusfeldolgozó rendszer, széleskörű használhatósága miatt, a webkamerán túl más, kiegészítő eszközt nem igényel.

3.1.2 A GESZTUS MODELLEZÉSE

3.1.2.1 A KÉZ MEGKERESÉSÉHEZ SZÜKSÉGES ELŐKÉSZÍTŐ TECHNIKÁK

A bemeneti képen a kéz megtalálásához a háttérrel el kell különítenünk attól. Az alábbi módszerek közreműködhetnek ebben.

1. **Küszöbölés:** Ahogy a nevéből is adódik, a képen küszöbölést hajtunk végre egy meghatározott küszöbérték szerint. Így kapjuk meg az előteret, ami jelen esetben a kéz, illetve a háttérrel. Kézetektálás során jellemzően az Otsu küszöbölést szokták használni. Fontos azonban az, hogy egy ilyen, egyszerű küszöbölési módszer önmagában nem elegendő a szegmentáláshoz, ugyanis nem állja meg a helyét, mivel, például a jelenet megvilágításának változásából eredő képpont-intenzitásváltozások a küszöbölés korábban bevált küszöbértékeinek helyességét megkérdőjelezhetik.
2. **Bőrszín alapú megközelítés:** A szegmentálás során itt a bőr sajátos színét használjuk fel, mely használatával a háttértől könnyedén elkülöníthető a kéz. Különböző színterek segítségével dolgozhatunk ennél a módszernél. Ilyen lehet



3.2. ábra: A mozgás alapú küszöbölés hatása (7)

az RGB. Az egyik (6) felhasználási módja ennek a bemeneti kép piros-zöld pixelarányát veszi figyelembe. A fehér emberi bőrhöz tartozó említett arány ugyanis egy jól meghatározott tartományon belül mozog, így a kép többi részétől – elméletben - könnyen elkülöníthető. A bőrszín tartományt a következő szerint határozták meg: $1,05 < \text{PIROS/ZÖLD egy pixelen} < 4,00$. Magyarul az e közötti arány szerint küszöbölhető a kép. Sajnos ez a módszer azonban hátrányos megkülönböztetésben részesíti a más bőrszínnel rendelkező embereket. Szürkeárnyaltos képeket is felhasználhatunk.

3. **Kivonás:** Ez a módszer (7) abból indul ki, hogy egy gyakorlatilag statikus képet szolgáltató kamera képe miként változik meg mozgás hatására. Azaz változásokat keres a rákövetkező képkockában az előző képkockához képest. A bemeneti képsorozatot szürkeárnyaltossá alakítják, illetve zajszűrést hajtanak rajta végre. Majd az egymást követő képkockák különbsége alapján a statikus háttérrel elkülönítik a mozgó kéztől, tehát itt egy mozgást igénylő technikáról van szó. Ezután egy előre meghatározott, a szürkeárnyaltos bőrszín értéke szerint küszöbölik a képet. A módszer eredményét a 3.2. ábra szemlélteti.
4. **Statisztikai modell:** Ez egy valószínűség-számításon alapuló modell, mely minden egyes képpixelre meghatározza azok háttérbeli, illetve előtérbeli létének valószínűségét.
5. **Színnormalizálás:** Ennek a módszernek az alkalmazása lényeges szerepet tölt be azokban az esetekben, amikor egy előre meghatározott minta azonosítása a feladat a bemeneti képen. Webkamera bemeneti forrásként való használata bizonyos – korábban már említett – körülményeket hoz magával. Egyrészt a webkamerák eltérő minőségben szolgáltatnak képet, másrészt a felvétel közege legtöbbször változékony megvilágítással is bír. E technika a különböző helyzetek változó színintenzitásainak problémáját oldja meg úgy, hogy a megvilágítás különbözősége a feldolgozás során kevésbé jelentsen gondot.

5.1 *Átfogó színnormalizálás*: Erre a célra létezik egy ún. átfogó színnormalizálás (8), mely megoldást nyújt a fent vázolt problémára. Az ilyen folytán eltérő képek különbségeinek jelentős csökkentésére egy két szintből álló folyamatot kínál ez a módszer. Először is a képek szerkezetének olyan jellemzői kerülnek meghatározásra, melyek nem konstansok, majd ezeket a változókat eltávolítja az algoritmus. A legmeghatározóbb ilyen változók a következők: a fényforrások helyzete a jelenethez képest, valamint azok fényerőssége, illetve a fényforrások színbeli különbségeiből adódó jelenetbeli színkülönbségek. A másodikból könnyen alkotható matematikai modell, ha veszünk egy s skálázó faktort, mely a fényforrás fényintenzitását skálázza, és ekkor ez az s a képet is skálázza. Így a minden egyes pixelre megadott (r, g, b) színösszetevő (sr, sg, sb) alakban lesz felírható. Ebből már könnyedén invariánssá lehet tenni a bemeneti képet a beeső fényforrástól az alábbiak szerint:

$$\frac{r}{r + g + b}, \frac{g}{r + g + b}, \frac{b}{r + g + b}$$

3.1 egyenlet: *A kép a fényforrás intenzitásától invariánssá tétele (8)*

Az utolsónál is alkotható olyan modell, amely viszont a bemeneti képet a fényforrás színétől teszi invariánssá. Tegyük fel, hogy az r, g és b színcsatornák rendre az α, β és γ skálázó faktorokkal skálázottak. Vegyük két pont kameraválasztát, azaz képezzük pixelbe e két pontot, így megkapjuk (r_1, g_1, b_1) és (r_2, g_2, b_2) pixeleket, melyek azonos színű fényforrás esetén állnak elő. Azonban ahhoz, hogy két ugyanezen pont különböző színű fényforrás esetén képzett alakját megkapjuk, elég a fent említett színcsatorna skálázó faktorokkal kiegészíteni az imént kapott két pixelt. Ekkor a két pont alakja a következő lesz: $(\alpha r_1, \beta g_1, \gamma b_1)$ és $(\alpha r_2, \beta g_2, \gamma b_2)$. Ezek után felírható az alábbi kifejezés, melyben a három skálázó faktor kiesik:

$$\left(\frac{2r_1}{r_1 + r_2}, \frac{2g_1}{g_1 + g_2}, \frac{2b_1}{b_1 + b_2} \right), \left(\frac{2r_2}{r_1 + r_2}, \frac{2g_2}{g_1 + g_2}, \frac{2b_2}{b_1 + b_2} \right)$$

3.2 egyenlet: *A kép színforrás színétől függetlenné tétele (8)*

A felírt két összefüggés együttes alkalmazásával lehetséges csak mind a fényforrás erősségéből, helyzetéből, azaz a megvilágítás geometriájából, továbbá színéből adódó eltéréseket kiküszöbölni. A módszer eredményességét az 3.3 ábra

mutatja, melyen jól látható, hogy a megvilágítás tulajdonságainak változását a normalizálás megszüntette az eredményképeken.

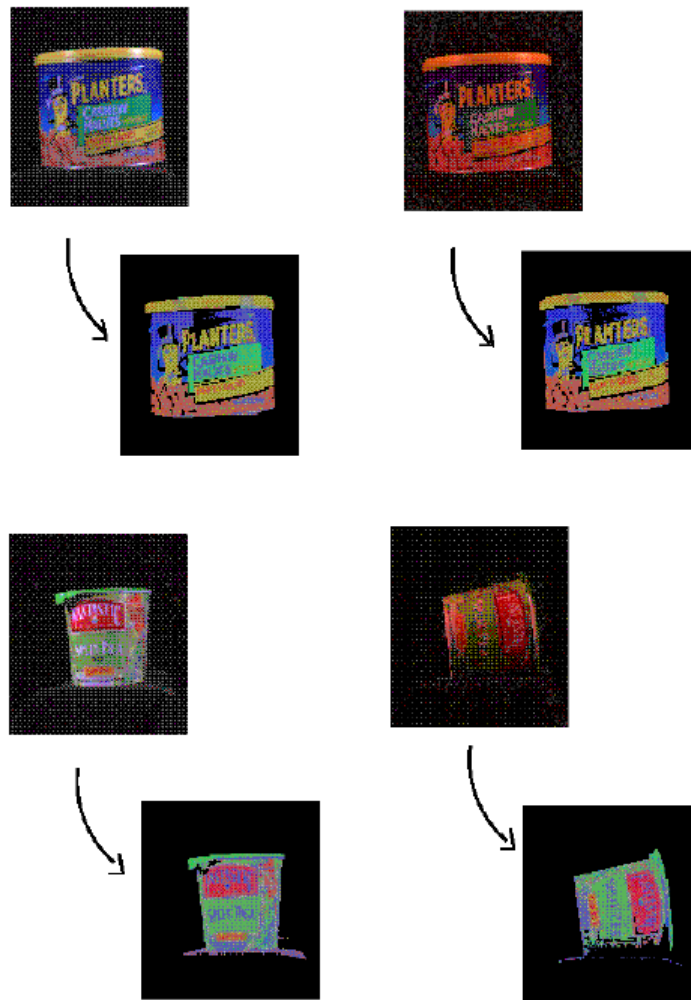
5.2 *Color Cluster Rotation* (9) (10): Ez a módszer egy olyan színnormalizációs technika, amely az ezt alkalmazó rendszerek bemeneti képeit gyakorlatilag érzéketlenné teszi a beeső fények miatt jelentkező különféle hatásokra, s ezáltal a hasonló képek közös jellemzői sikeresebben megtalálhatók. Ezen zavarok kiküszöbölése szükséges abban az esetben, ha a bemeneti kép rögzítésének körülményei változhatnak, mely sokszor előfordul. Elég, ha a megvalósítandó rendszert vesszük alapul: egy hordozható számítógép egyrészt különféle helyeken tartózkodhat, másrészt, mivel nagy valószínűséggel különböző napszakokban is használják, a megvilágítás igen változékony lehet. Az emberi agy az efféle megvilágításbeli különbségeket könnyedén korrigálja, korábbi tapasztalatai, folyamatos tanulása révén. Sajnos azonban a számítógép erre önmagától nem képes, így e komplex probléma megoldása csak átgondolt, bonyolult módszerek használatával lehetséges. A Color Cluster Rotation színnormalizálás technikája abból a tényből indul ki, hogy minden megvilágítás más-más színeloszlást eredményez a rögzített képen. A színeloszlást egy háromdimenziós színhisztogramként lehet szemléltetni, melynek három komponense a piros, a zöld és a kék, azaz az alapvető RGB színteret használja. Első lépésként az algoritmus kiszámolja a kép színintenzitásainak középértékét. Az adott színeloszláshoz kiszámolt kovariancia-mátrix legnagyobb sajátértékéhez tartozó sajátvektor határozza meg a színeloszlás fő irányát. Miután ezt kiszámolta az algoritmus, az alapötlet szerinti irányba indul tovább a végrehajtás, mely szerint az előző lépésben kiszámolt irányt az RGB színekocka szürkességi átlójába kell forgatni (amelyen az RGB komponensek értéke megegyezik). Ehhez az ún. Rodrigues formula kerül alkalmazásra. Lezáró lépésként az így kapott színeloszlás középpontja az RGB kocka főátlójának közepére tolódik, majd az alulcsordult (0-nál kisebb), illetve a túlcsordult (255-nél nagyobb) színtkomponensek „levágásra” kerülnek a klaszterből.

3.1.2.2 A KÉP NORMALIZÁLÁSA

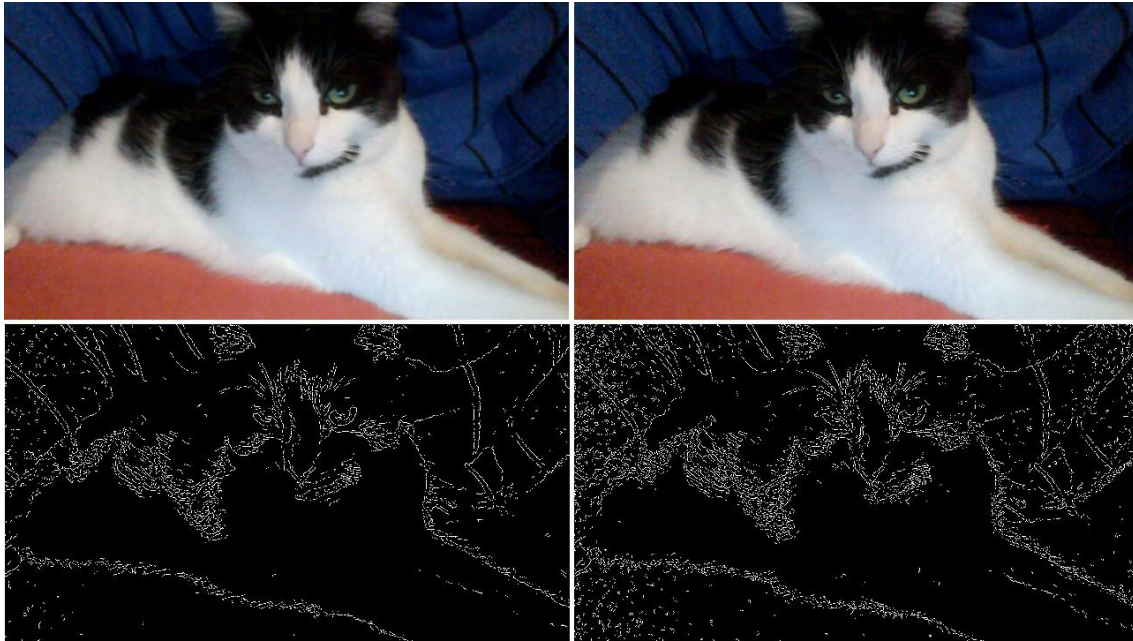
Minden képfeldolgozási probléma esetén szükség van a folyamat során, a képen keletkező zavarok eltüntetésére, mivel ezek zavarhatják a feldolgozást, a cél elérését. Esetünkben, mivel webkamera szolgáltatja a képet, nagy szükség van hatékony és gyors

zajszűrésre, mivel az efféle eszközök viszonylag gyenge képminőséget szolgáltatnak, így rengeteg képzaj keletkezik. Fontos, hogy olyan szűrőket válasszunk, melyek általános esetben képesek a zajok nagy részét eltávolítani. Ezen felül a kép azon részének normalizálása is szükség van, mely a feldolgozás szempontjából jelentőséggel bír. Jelen esetben ez a kéz területe. Tehát ebben a lépésben az előző lépésben megkapott kéz területén a legjelentősebb területeket határozzuk meg és ezt adjuk tovább feldolgozásra.

1. **Véletlenszerű zajok eltávolítása:** A leginkább előforduló, parányi, véletlenszerű zajok megszüntetésére a legalkalmasabbak a simító szűrők, amely lehet például a Gauss-szűrő. Ehhez kapcsolódóan a Gauss-piramis technika is



3.3 ábra: A képeken a konzervek különböző megvilágítási geometriával és megvilágítási színnel bírnak. Az eredeti állapotokat az 1. és a 3. sorban elhelyezkedő képek, az átfogó normalizálás utáni állapotot pedig a 2. és a 4. sorban lévő képek szemléltetik. (8)



3.4 ábra: A Gauss-piramis technika és a Gauss szűrő alkalmazásának különbsége (felső két kép balról jobbra: Gauss-piramis lefelé, majd felfelé skálázás és Gauss-szűrés; alul balról jobbra: az ezekre alkalmazott Canny-éldetektálás) (saját ábra)

megfelel a kép simításának. Az eredeti méret megtartásához ez utóbbi módszer két, egymással ellentétes viselkedésének (a lefelé, illetve felfelé skálázás) egymás utáni alkalmazásával járulhatunk hozzá, mindamellet pedig a zajok eltávolítása is végrehajtható. Fontos azonban megjegyezni, hogy a Gauss-piramis technika efféle használata, a Gauss-szűréshez képest, jóval jelentősebb hatást ér el. A 3.4 ábra jól mutatja – az élek tekintetében – hogy míg a sima Gauss-szűrés hatására is rengeteg zaj marad a képen, addig a Gauss-piramis technika alkalmazása ezeket redukálja, e mellett pedig a jelentős kontúrok meghagyja.

2. **Dilatáció és erózió (7):** A bemeneti képen a háttér, illetve az előtér, azaz a kéz meghatározása képzajok megjelenésével járhat. Ezek olyan nemkívánatos pixelek, melyeket az algoritmus előtérnek érzékelt, mert a zajeltávolítás után megmaradtak zavarokként a képen. A dilatáció a zavaró sötét, az erózió pedig a zavaró világos területek méretét csökkenti.
3. **Éldetektálás:** Az éldetektálás segít megtalálni egy képen az objektumok éleit, kontúrjait. Használatával egyszerűbben megtalálható a kéz, mint objektum a képen, mivel kontúrja az éldetektálás során megjelenik. A leggyakoribb éldetektáló módszerek a következők: Sobel maszk, Robert maszk, Prewitt maszk, LoG (Laplacian of Gaussian), illetve a Canny éldetektáló algoritmus.
4. **Méret egységesítés:** E lépésnek a lényege az, hogy a kapott, kivágott kezét tartalmazó képet egy előre meghatározott méretűre alakítjuk, s így ugyanolyan

méretű képet kapunk, mint amekkorák az összehasonlításához tárolt képek. Így jóval egyszerűbb a jellemzők komparálása.

3.1.3 A KÉZ DETEKTÁLÁSA (11)

Miután megkaptuk a zajszűrt, normalizált, azaz már feldolgozásra alkalmas képet, a legjelentősebb lépés következik, melyet a jellemzők kinyerésének hívunk. Emberi szemmel könnyen megtalálhatjuk az egyértelmű jellemzőit egy kézfejnek, azonban egy webkamerakép, a korábban már tárgyalt problémás körülmények okán, nem igazán alkalmas ezek felderítésére. A torzuló képi tulajdonságok miatt az egyes geometriai jellemzők csorbulhatnak, mint például a kézfej kontúrja, az ujjak láthatósága. Ugyanez igaz a nem geometriai jellemzőkre is, mint a bőrszín, amely a megvilágítás elégtelensége miatt megváltozhat. Ebből látható, hogy a legjobb megoldás az lehet, ha ráhagyjuk egy algoritmusra a megfelelő kézfejjellemzők automatikus kiválasztását.

3.1.3.1 KINEMATIKUS MODELL (12)

A kinematikus modell olyan, a kézfej mozgásához kapcsolódó paramétereket keres (a kézen), amely egy előre meghatározott, éllel határolt kéz képével képes kapcsolatot teremteni. Egyes felhasználásaiban például az ízületek szögei szolgáltatják a jellemzőket. A technika, amit használ, megvalósításában nehéz, mivel az éleket, mint jellemzőket sokkal nehezebb kinyerni egy tenyér esetében, mert nem egyértelmű maga a tenyér megtalálása. „A hatékony kinyerés érdekében ez a módszer igényli a homogén és nagy kontrasztú hátteret”¹, azaz a tenyér helyzetének igen egyértelműnek kell lennie, amely érzékelhetően a megvalósítandó rendszerem esetében nem az általános eset, azaz nem kedvező e módszer használata.

3.1.3.2 MEGJELENÉS ALAPÚ MEGKÖZELÍTÉS (13)

A megjelenés alapú megközelítés egyrészt lehet olyan, hogy a kézfejet annak mozgása alapján modellezi intenzitásképek segítségével. Jóval érzéketlenebb a nem homogén háttérre, így általános esetben célszerűbb a használata. Másrészt bizonyos esetekben az ujjak helyzete, mozgása szolgál a detektálás forrásaként. Kevésbé általános esetekben az ujjakat

¹ (5) p. 30

előre meghatározott színű jelölőkkel látják el, mely így azonban nem alkalmas „mindennapi használatra”.

3.1.3.3 ALACSONY SZINTŰ JELLEMZŐKÖN ALAPULÓ MEGKÖZELÍTÉS (12)

Ez a módszer az egyszerűsége törekszik, azaz nem célja egy teljes kézfej modell megalkotása. Ehelyett olyan alacsony szintű jellemzőket használ, melyek sokkal kevésbé érzékenyek a bemeneti kép lehetséges torzulásaira és gyorsan megkaphatók. „Ilyenek például a kézfej területének a súlypontja, a főtengely által meghatározott ellipszis alakú kézterület, illetve – a mozgásból kiinduló – optikai folyama a kézfej területének.”¹



3.5 ábra: Self-Growing and Self-Organized Neural Gas network, egy egyre több neuronnal bíró, folyamatosan növekedő hálózat, mely azonban csak a kinyújtott ujjakat képes megszámolni (11)

3.1.3.4 MESTERSÉGES NEURÁLIS HÁLÓZAT ALAPÚ MEGKÖZELÍTÉS

A mesterséges neurális hálózat (Artificial Neural Network – ANN) olyan problémák megoldására alkalmazható, melyekben jellemző minták fedezhetők fel, méghozzá olyanok, melyeket a nagy adathalmaz miatt az ember nehezen vagy pedig egyáltalán nem lenne képes észrevenni. Jó példa erre a hatalmas adatbázisokban alkalmazott ún. adatbányászat tevékenysége, melynek eredménye például egy adott cég jövőbeli üzleti stratégiáját is meghatározhatja. Az alapötlet az emberi agy működése, melynek alapját a neuronok szolgáltatják. Ez esetben is egy elosztott, egymással együttműködő feldolgozóegységek rendszeréről van szó. A kézdetektálás problémájára többféleképpen is felhasználták ezt a módszert, főleg e probléma egyszerűsítésére, adaptív megközelítésben. Tanuló algoritmusok fejlesztéséhez nyújthat még alapot, egy ilyen volt például a 3.5 ábra által szemléltetett SGONG algoritmus, mely azonban – természeténél fogva – nem volt képes minden kézgesztus felismerésére, mivel a kinyújtott ujjak száma alapján kategorizált.

3.1.3.5 GENETIKUS ALGORITMUSON ALAPULÓ MEGKÖZELÍTÉS

E módszer is – a mesterséges neurális hálózat mellett – a biológiát veszi alapul. Hasonlóképpen a neurális hálózathoz, ez a technika is robosztus algoritmusok tervezését teszi lehetővé, kevés kezdő megszorítással. A lényege, hogy az ún. populáción, mely a lehetséges megoldások halmaza, különféle műveleteket hajt végre, mint a reprodukció, keresztezés és mutáció, méghozzá egy kevésbé korlátozott környezetben. Ezzel lehetőséget teremt az optimális megoldás megtalálására. Nagy előnye még a módszernek, hogy párhuzamosan

képes működni, azaz ezzel is növeli a számítás teljesítményét. Egyes alkalmazásaiban például a bemeneti kép zajmentesítésére használták, más megközelítésekben kézgesztus szegmentálásra és felismerésre.

3.2 ALGORITMUSOK ÁTTEKINTÉSE

A rendszer végleges implementációja előtt egy átfogó kép megalkotása lényeges a létező algoritmusok köréről. Fontos a megvalósításban a zajokra való jelentős érzéketlenség, az algoritmus rugalmassága a jellemző változásokra. A megvalósítandó rendszer esetében érdemes olyan algoritmusokat kiválasztani, amelyek egy bizonyos szintig skálázásra (a kéz egy adott távolságtartományban jelenhet meg a kamera előtt), esetleg forgatásra is invariánsak. Ez utóbbi azonban nem szükséges feltétel, mivel jellemzően csak egy irányba „mutató”, statikus gesztusok alkalmazásáról van szó. Lényeges szempont - ezeken kívül – a feldolgozás gyorsasága. Habár a rendszer nem igényel valósidejű feldolgozást, mégis sok időt emészt fel az előfeldolgozás is.

3.2.1 SIFT (14)

A SIFT (Scale Invariant Feature Transform~Skálázás Invariáns Jellemző Transzformáció) algoritmus létrehozója olyan célt tűzött ki feladatánaként, hogy olyan lokális jellemzőket keressen egy bemeneti képen, melyek részben érzéketlenek a fényre, valamint invariánsak a skálázásra, a síkbeli forgatásra és valamelyest a térbeli forgatásra (20 fokig) is. A megközelítés a hagyományos mintaalapú összehasonlítás túlzott nagy időszükségletét próbálja csökkenteni úgy, hogy az összehasonlítást a kinyert jellemzőkön végzi el. A SIFT lényege, hogy a bemeneti képből lokális jellemzővektorokat készít, mely vektorok olyan jellemzők, melyek különbözők és, mint a módszer nevéből is adódik, invariánsak a skálázásra, továbbá a forgatásra is.

A forgatás invarianciához olyan pontokat választ ki az algoritmus a képen, melyek minimumok vagy maximumok. Ehhez az ún. Difference of Gaussian (DoG, ~Gaussok különbsége) (15) függvényt használja. Ez a függvény a bemeneti képből képpiramist épít, mely az eredeti kép átskálázott verzióit tartalmazza, az egyes szintek közötti kapcsolatot pedig újra-mintavételezéssel biztosítja. A DoG tulajdonképpen a sima Gauss konvolúciós szűrőt használja. Működését a következőkben tárgyaljuk. Első lépésként a bemeneti képen végrehajt egy Gauss szűrést. Második lépésként az első lépésben kapott képen egy ugyanolyan szűrést végrehajt, ezzel még elmosódottabb képet eredményezve. A Gaussok különbségét ezután e

második simított kép elsőből való különbségéből képi. Ezzel a piramis első szintjét eredményezi. Végül a piramis következő szintje a második lépésben meghatározott kép kettős lineáris (bilineáris) interpoláció segítségével való újra-mintavételezésével történik. Kicsinyítést hajt végre az algoritmus az eredeti képen, ezért létrejönnek olyan pixelek, melyek intenzitását meg kell határozni. Egy pontos közelítést kíván ezekre a pixelekre adni a kettős lineáris interpoláció, mégpedig az ismert pixel négy szomszédja alapján, s így kicsinyíti az adott képet. A piramis felépítése után a korábban említett minimumok és maximumok meghatározása következik. Minden egyes piramisszinten az egyes pixeleket az összes szomszédos pixellel összehasonlítja az algoritmus. Amennyiben a vizsgált pixel az adott piramisszinten szélsőérték, akkor az eggyel alacsonyabb piramisszint, a megtalált szélsőértékhez legközelebbi pixelének helye kerül meghatározásra. A megkapott pixel és annak szomszédjai is „átesnek” az előző szinten szélsőértékként előállt pixellel való intenzitás-összehasonlító vizsgálaton. Ha a korábbi pixel még így is szélsőérték (a most vizsgáltakhoz képest), akkor az eggyel magasabb piramisszinten hajtódik végre ugyanez a vizsgálat. Mivel ez az eljárás gyorsan diszkriminálja a nem megfelelő pixeleket, hatékony és gyors végrehajtással bír.

Minden egyes kulcspixelen meghatározásra kerülnek a kép gradiensei, illetve azok orientációja, a piramis minden szintjén az alábbi képletek szerint:

$$M_{ij} = \sqrt{(A_{ij} - A_{i+1,j})^2 + (A_{ij} - A_{i,j+1})^2}$$

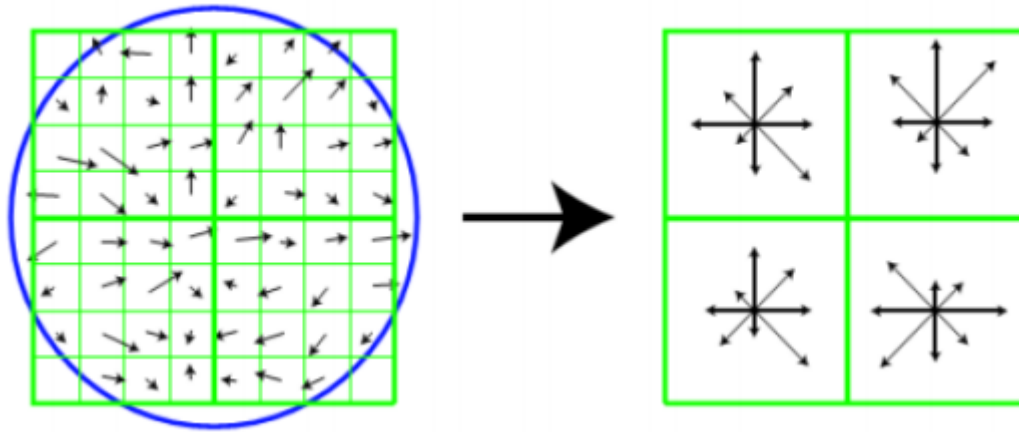
$$R_{ij} = \text{atan2}(A_{ij} - A_{i+1,j}, A_{i,j+1} - A_{ij})$$

3.3 egyenlet: *A gradiens nagyság és irány kiszámítása (14)*

Számításigénye ennek a lépésnek sem számottevő.

A kiszámolt gradiens nagyságok küszöbölése a legnagyobb meghatározott gradiens értékének 0,1-szeresével mérsékli a kép megvilágítás változásából eredő függőségét, mivel a környezeti tulajdonság változása a gradiens nagyságra, mintsem annak irányára van hatással.

Minden egyes kulcspixelnek orientáció-függetlennek kell tehát lennie, azonban a megvilágítás, illetve a kontraszt változása befolyásolja ennek stabilitását (nem elég az imént említett küszöbölés). Ezért az orientációk meghatározása olyan hisztogramban (ún. orientációs hisztogram) történik, melynek értékei a lokális gradiens irányok. A hisztogram 36 vödörből áll, mellyel magában foglalja mind a 360 foknyi forgatási irányt.



3.6 ábra: Az orientációk meghatározása (16)

A jellemző pontok lokális geometriai torzulásokra való robosztussá tételét is megoldja a SIFT algoritmus. Mivel mindegyik jellemző ponthoz az adott orientáció szerinti gradiens tartozik, ezért több orientációs síkot is számon kell tartani, melyek között simítás, illetve újra-mintavételezés teremti meg a kapcsolatot. Mivel előzőleg minden piramisszintre kiszámolta a módszer a jellemzőkhöz tartozó gradienseket és azok irányát, ezért azokat itt is fel lehet használni. Az adott piramisszinten megtalált jellemző pont 8 pixel sugarú körében lévő pixelek áthelyeződnek az adott orientációs síkba. Az aktuális irány pedig a kulcshoz viszonyítva kerül kiszámításra. Mivel a bemeneti kép különböző skálázásainál is el kell tárolni a jellemző pont 8 pixel sugarú körben lévő pixeleit, ezért az egyes jellemző pontokat igen hosszú leíróvektorok határozzák meg (jellemzően 160 elem nagyságúak).

3.2.1.1 VALÓSÍDEJŰ KÉZGESZTUSFELISMERÉS SIFT SEGÍTSÉGÉVEL (16)

A (16) forrásban taglalt rendszert a SIFT algoritmust használja, melynek megvalósítását a következőképpen érik el. Először két, különböző szélességű Gauss konvolúciós szűrőt alkalmaznak a bemeneti képen. Az egyes eredményeket a másik eredményekkel skálázzák, majd ezeknek a különbségét veszik. Az első képből, mely a homályosabb, vonják ki a második képet, mely pedig a kevésbé homályos. Gyakorlatilag e két kép kivonásával egy olyan eredményt kapnak, amely csak a kép térbeli frekvenciáit tartalmazza. Ennek a szűrőnek a neve: Difference of Gaussians (~Gauss-szűrések különbsége), mely egy sáváteresztő szűrő. (15) Ezekből a képekből több, eltérő méretűt is készítenek, amely így egy piramist, a DoG (Difference of Gaussian) piramist alkot. Ezután a piramis szintjein összehasonlítják és meghatározzák a szélsőértékeket, a kulcspontokat, subpixel pontossággal. Majd a nem megfelelő szélsőérték pontokat eltávolítják a megtalált kulcspontok közül. Végül a kulcspontok orientációját határozzák meg a lokális gradiens irányok és nagyságok alapján. Minden egyes kulcsponthoz egy nála másfélszer nagyobb ablak tartozik, melyen belül meghatározzák a legjellemzőbb lokális gradienst (3.6 ábra). Így a képen

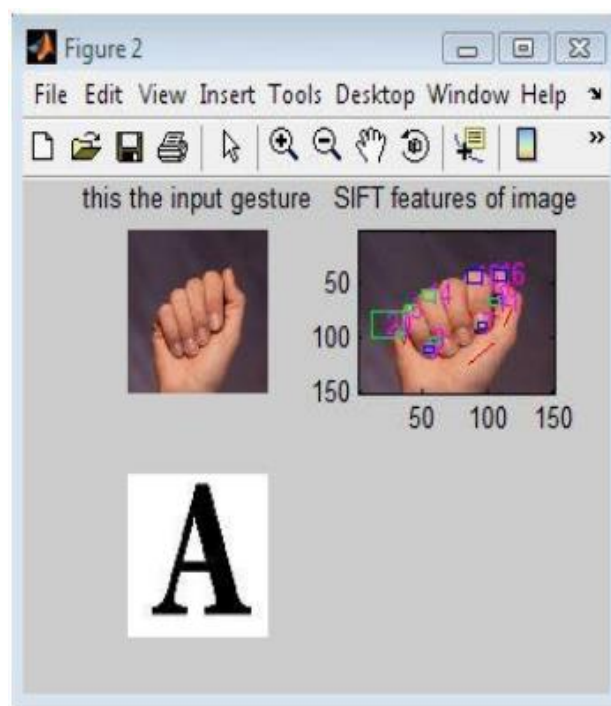
olyan jellemzők kerülnek meghatározásra, melyek, a korábban említettek szerint, érzéketlenek a skálázásra, forgatásra és a döntésre is.

Az algoritmus négy lépésből áll. Első lépésként a forrásból, mely egy videó, a képeket elmentik. Második lépésként az egymást követő képeken azt vizsgálják, hogy a képek mely részei nem hordoznak többletinformációt. A többletinformációt nem tartalmazó részeit levágják a képnek. A képek megmaradt részein a fentebb említett módszerrel megkeresik a SIFT jellemzőket, majd végül az előre eltárolt jellemzőkkel összehasonlítják azokat. Az eredményt a 3.7 ábra mutatja.

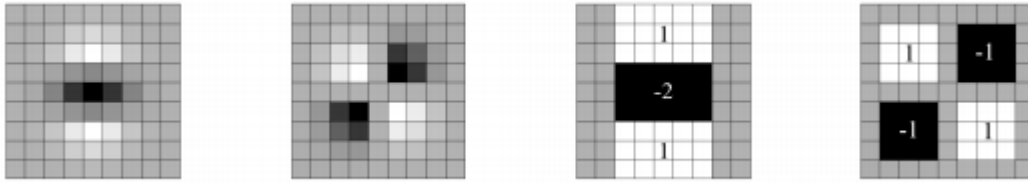
A módszer eredményeit tekintve jelentős, hogy nem érzékeny a zajra, illetve a korábban említett változásokra invariáns. Bemutatként egy 40 képkocka/másodperces videót használtak, melyben a 40 képkockán a jellemzők kinyerése 40 másodpercig tartott.

3.2.2 SURF (17)

A SURF (Speeded Up Robust Features ~ Felgyorsított robusztus jellemzők) a korábbi jellemző kereséseknél gyorsabb megoldást kínál ugyanehhez a feladathoz. Jellegzetes pontokat keres a bemeneti képen, melyek szomszédságát egy-egy jellemzővektorban tárolja el, melynek szintén jellegzetesnek, robusztusnak kell lennie. Végül pedig az összehasonlítandó képek jellemzővektorai kerülnek összehasonlításra, jellemzően e vektorok



3.7 ábra: A SIFT módszer eredménye (16)



3.8 ábra: Másodrendű Gauss szűrők (balra) közelítése boxfilterekkel (jobbra) (17)

közi távolság alapján. Az algoritmus megalkotói célul tűzték ki maguk előtt azt, hogy a gyors számítás oltárán feláldozzanak némi invarianciát, ráadásul ezt paraméterezhetővé is teszik. Például lehetséges az algoritmust úgy is használni, hogy az ne legyen forgatásra invariáns, csak skálázásra. Ehhez az ún. U-SURF-t (upright SURF ~egyenesen álló SURF) hozták létre. Ez egy opció.

A SURF jellemződetektora az ún. Hesse-mátrixon alapul. Ennek a determinánsát veszi alapul a jellemző helyének és arányának (skálázás) meghatározásakor. A Hesse-mátrixot itt egy 2x2-es mátrixként definiálják, melynek tagjai a bemeneti kép adott pixelénél vett konvolúciói a Gauss szűrő x , y , y_x , y_y másodrendű parciális deriváltjai szerint:

$$\mathcal{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix}$$

3.4 egyenlet: Hesse-mátrix (17)

Azonban a SURF Gauss szűrés helyett, boxfilterekkel közelíti a másodrendű deriváltakat, mellyel gyorsabban tud eredményt elérni, mivel ezek kiszámításához a bemeneti kép integrálképét használja fel (3.8 ábra).

Skálázásnál láthattuk a SIFT-nél, hogy képpiramist használt, itt is ez történik, de azzal ellentétben itt nem függ az egyik réteg kimenetétől a következő réteg bemenete, elég csak a megfelelő boxfiltereket alkalmazni az eredeti bemeneti képen. Ebből látható a párhuzamosítás lehetősége ebben a fázisban.

Jellemző pontok meghatározásához, a skálázási síkokat is figyelembe véve, az adott pixel szomszédjain a nem maximumok elnyomása történik meg egy 3x3x3-as körzetben. Ezután a Hesse-mátrix maximumát alapul véve közelít az algoritmus a képi és a skálázási térben.

A forgatás invarianciához minden egyes jellemző pont x és y irányában is kiszámolja az algoritmus a Haar-hullám (Haar-wavelet) választ, szintúgy a $6 \cdot (\text{skálázási faktor})$ sugarú

környezetében elhelyezkedő szomszédok esetében is. A válaszokat vektorok reprezentálják. A vektorok egy olyan koordináta-rendszerben írhatók le, ahol az x-tengely a vízszintes irányú válaszok erőssége, az y-tengely pedig a függőleges irányú válaszok erőssége. Ezután a megkapott vektorokból a legjelentősebbet választja ki az algoritmus, mégpedig úgy, hogy adott csúszó ablakon belül összegzi a vízszintes és függőleges Haar-hullám válaszokat. A leghosszabb vektor jelenti a jellemző pont orientációját.

Az egyes jellemző pontokhoz tartozó leírók meghatározása a következők szerint zajlik. Először az adott jellemző ponthoz kiszámításra kerül egy olyan négyzet, melynek középpontja maga a jellemző pont, mérete a skálázási mérettel függ össze és iránya a jellemző pont orientációja. Ezután ezt a négyzetnyi területet kisebb területekre (4x4) osztja az algoritmus. Minden kis területen, előre meghatározott számú és elhelyezkedésű mintapontoknál jellemzőket számol (Haar-hullám). Az így kapott kis területek válaszait összegzi, mellyel a jellemző vektor első adag adatát meghatározza. A második adag adat a válaszok abszolútértéke lesz. Így, a 4x4 kis négyzetre osztott eredeti négyzet miatt, egy 64 hosszú leíró vektor az eredmény. Az algoritmusban lehetséges rövidebb (36), illetve hosszabb (128), bővebb leíróvektorokkal is dolgozni. Előbbi nem olyan hatékony, mint az eredeti, viszont gyorsabb. Utóbbi pedig pontosabb eredményt ad, de valamivel lassabb.

3.2.3 FAST (18)

A FAST (Features from Accelerated Segment Test ~Jellemzők a felgyorsult szeletekből) tervezői egy olyan algoritmust szerettek volna létrehozni, amely bemeneti videón képes valósidőben, nagy hatékonysággal és emellett igen gyorsan jellemzőket megtalálni, összehasonlítani. Kategória szerint a sarokdetektálás csoportjába tartozik, azon belül is olyan sarokdetektort használ, mely egy-egy kisebb foltról állapítja meg a bemeneti képen, hogy az „úgy néz-e ki”, mint egy sarok. S mivel csak kisszámú pixelt vizsgál az algoritmus, ezért a számításigénye is jóval alacsonyabb. Tanításos algoritmust használ. Ennek magja a következő. A vizsgált pixelhez, mely legyen p , egy ún. Bresenham kört rajzol, mely diszkrét értékekkel közelít egy nem diszkrét kört. Ez a kör 16 pixelt foglal magába, melyek a későbbi számításokban szerepet játszanak majd. Ebből n pixelnek (n 12-nek lett választva) teljesítenie kell azt, hogy intenzitása egy bizonyos t küszöbnél jobban eltér a vizsgált p pont intenzitásától. Amennyiben ez megtörténik, p sarokpontnak számít. Az imént részletezett algoritmust néhány bemeneti képen alkalmazzák, sarokpontokat keresve. Minden egyes p pont körül lévő n pixelről megállapítható annak intenzitása, mely lehet világosabb, sötétebb és

ugyanolyan, mint a vizsgált p pont. Ha minden egyes p pixelre (P) az összes tréningképen elvégezzük az előbb elhangzott kategorizálást, akkor P három részcsoportha osztható: a világosabb, a sötétebb és az ugyanolyan intenzitású pixelekre. K_p jelöli p saroklétét, azaz lehet igaz vagy hamis. Az ún. ID3 döntési fa generáló algoritmust használja a FAST, kezdésnek azt a pontot választja ki p körül, amelyik a legtöbb információt adja p saroklétéről, K_p entrópiája (bizonytalanság) alapján. A legtöbb információt adó pixel kiválasztása után, az algoritmus végigmegy mindhárom korábban meghatározott részcsoportha. Akkor áll le a végrehajtás, ha az egyik részcsoportha entrópiája nullára csökken. Ennek hatására egy olyan döntési fa jön létre, amellyel könnyedén és pontosan meg lehet találni a tréningképeken detektált sarokpontokat.

Problémát okoz a sarkok detektálásában az, ha egymással szomszédos p pontokon vizsgálódunk, melyek egyazon sarokhoz tartoznak. Ennek kiküszöbölésére az ún. nem maximumok elnyomása módszert alkalmazták a FAST létrehozói, melynek működése a következő. Egy változót határoznak meg, értékül pedig a kiválasztott p pont és a körülötte lévő kör 16 pixelével egyenként vett abszolút differenciájának összességét adják. Ugyanezt megteszik a mellette lévő másik p pixellel is. A további számításokat pedig azzal a p ponttal folytatják, ahol ennek a változónak az értéke nagyobb.

A FAST egy igen gyors sarokdetektáló algoritmus, sajnos azonban nem elég robusztus a zajokra, illetve sok múlik azon, hogy milyen t küszöbértéket választunk meg. Továbbá nem tárol információt az egyes jellemző pontok orientációjáról.

3.2.4 ORB (19)

Az ORB (Oriented FAST and Rotated BRIEF ~Írányított FAST és forgatott BRIEF) a korábban ismertetett FAST jellemző pont detektorra és a BRIEF (Binary Robust Independent Elementary Features ~Alapvető bináris független jellemzők) leíróra épít, mert a fejlesztők célja volt a gyors és hatékony feldolgozás, minimális erőforrás-felhasználással és ezt mindkettő teljesíti. Továbbá a SIFT-hez és SURF-höz képest, az ORB ingyenesen felhasználható.

A BRIEF egy olyan jellemző leíró algoritmus, mely közvetlenül a képeken megtalálható jellemzőkből bináris karakterláncokkal leírható jellemzővektorokat készít, s így e jellemzővektorokkal történő számításokat nagymértékben gyorsítja. Ráadásul, mivel bináris számokról van szó, magától értetődően, könnyűszerrel tárolhatók problémamentesen a memóriában és a rajtuk végezhető műveleteket is hatékonyabban végzi – architektúrájából

adódóan – a számítógép. Ilyen művelet például a XOR (kizáró vagy), mely két bináris jellemzővektor összehasonlítására alkalmas, azaz e módszerrel elég ún. Hamming távolságot (a Hamming távolság megadja, hogy két karaktersor között hány karakternyi különbség van, mennyi karaktert kellene megváltoztatunk valamelyik sorban a másik sor eredményül kapásához) számolni, ami rendkívül gyors.

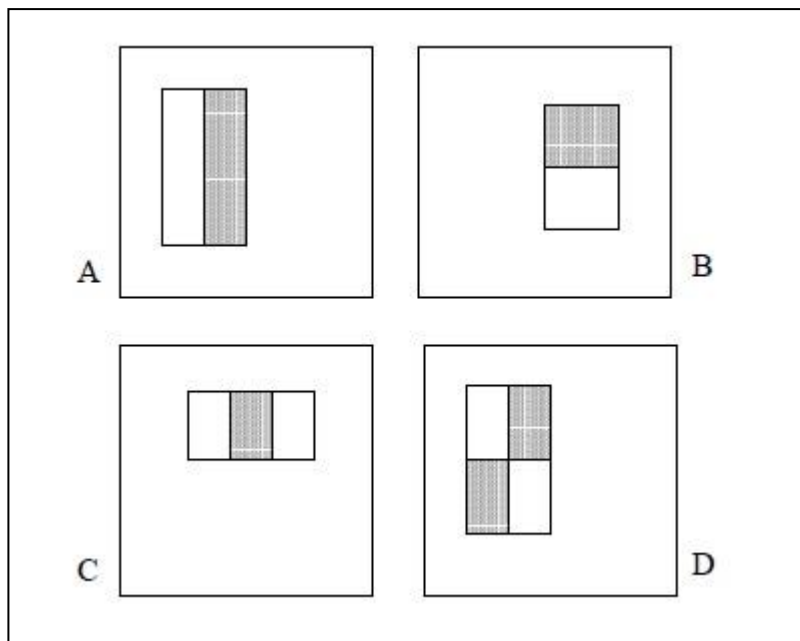
A BRIEF a korábbi jellemzővektor készítő algoritmusok gyengeségeit próbálta áthidalni, mint a SIFT alapvető lassúsága nagyméretű jellemzővektoraiból adódóan. S mivel a SIFT-re, illetve a SURF-re is épít, ugyanúgy foltalapú összehasonlítást végez, sajnos azonban forgatásra nem invariáns. Először az adott jellemzőt is magába foglaló folton belül kiválaszt egy-egy pontpárt. E pontpárok intenzitását hasonlítja össze. Amennyiben az első pontpár intenzitása alacsonyabb a másodikénál, az összehasonlítás eredménye 1 lesz, egyébként 0. Az adott jellemző foltnál végez n darab ilyen összehasonlítást és összesíti ezek eredményét. Végül ezek formálnak bináris jellemzővektorokat. (20)

Az ORB a FAST jellemzőpont detektort használja a bemeneti kép jellemzőinek megtalálására. Azonban a FAST egyrészt nem invariáns a forgatásra, másrészt nem rendelkezik (például a SIFT-hez hasonló) skálázásos, piramisos technikával, azaz nem tárol különböző méretű jellemzőket. Ezek az ORB-ben meg lettek oldva. Az orientáció hiányának problémáját például a következők szerint orvosolták az algoritmus készítői. Az összehasonlítandó foltok ún. intenzitással súlyozott súlypontjait számolják ki. A foltok közepén pedig az adott sarokpontok helyezkednek el. A kiszámolt súlypont és az e sarokpont közötti vektor iránya adja a későbbi összehasonlítás során az aktuális jellemzők orientációját.

A BRIEF-et, mint jellemzővektor meghatározót továbbfejlesztve, „bekormányozza” annak jellemzővektorait a továbbfejlesztett FAST által megtalált, irányított jellemzők alapján. Mivel – a korábbiakban részletezettek szerint – a továbbfejlesztett FAST egy-egy jellemző körülötti foltnál n darab összehasonlítást végez (x_i és y_i pontokon), ezeket egy $2*n$ nagyságú mátrixban el is lehet helyezni, az alábbiak szerint:

$$S = \begin{pmatrix} x_1, \dots, x_n \\ y_1, \dots, y_n \end{pmatrix}$$

3.5 egyenlet: S tesztmátrix (20)

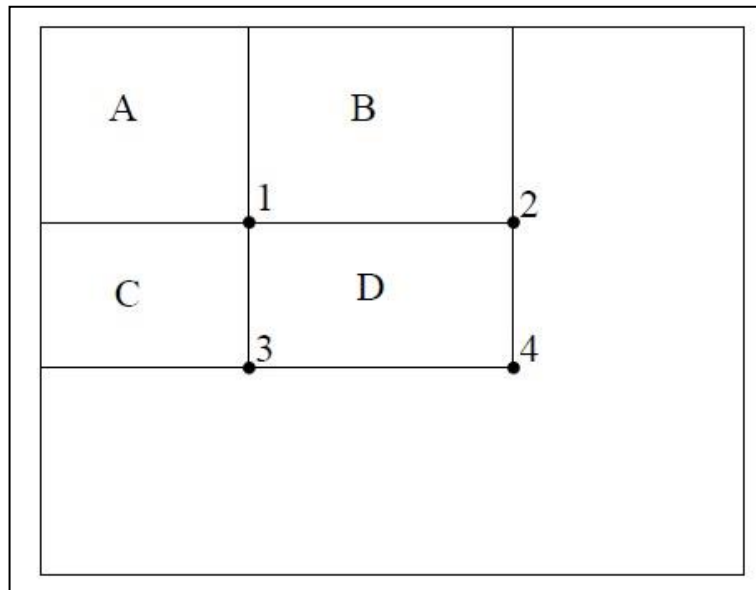


3.9. ábra: Példa téglalapjellemzőkre (21)

Ezután a korábban megtalált jellemzők orientációjával lehet „bekormányozni” ezt a mátrixot, azaz balról megszorozni azt az adott θ orientációval. A jellemzők orientációja kvantált érték, így a gyorsabb számítás érdekében előre kiszámolt LUT-ban (Look Up Table ~Keresési táblázat) tárol az algoritmus ún. BRIEF mintákat.

3.2.5 HAAR JELLEMZŐ ALAPÚ OBJEKTUMOSZTÁLYOZÁS (21)

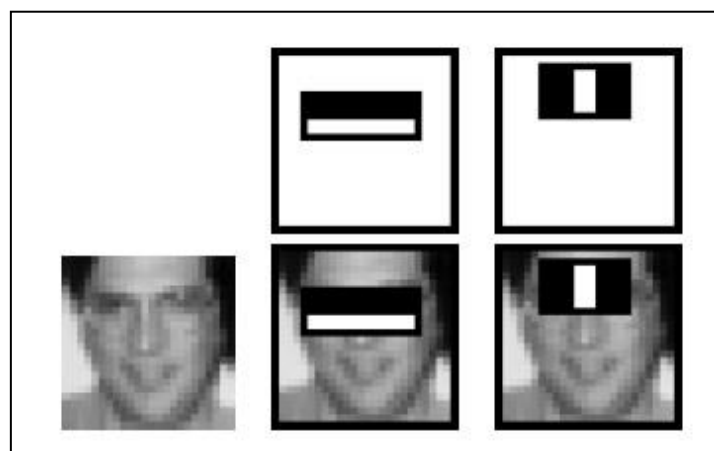
A Haar jellemző alapú objektumosztályozás Paul Viola és Michael Jones univerzális objektumdetektáló módszere, tehát bármilyen objektum felismerésére be lehet tanítani, mivel egy tanítható algoritmusról van szó. Lényege abban rejlik, hogy nem közvetlenül a pixelek intenzitásával dolgozik, hanem képi jellemzőkkel számol. Ezek a jellemzők téglalapok, melyek különbözőképpen helyezkednek el egymáshoz képest és mindegyik téglalap egy-egy pixelterületet fed le. Az alap rendszer háromféle jellemzővel dolgozik (3.9. ábra): két téglalapossal, melynél az egymás alatt vagy egymás felett elhelyezkedő téglalapok alatt lévő pixelek összegének különbségét veszik; három téglalapossal, melynél a két szélső, valamint a középső terület különbségét veszik; és négy téglalapossal, ahol az átlósan összetartozó két-két téglalap különbsége szolgáltatja az eredményt. Mindegyik, előbb felsorolt ún. téglalapjellemzőnél a kisebbítendő tag a szürke területek, a kivonandó tag pedig a fehér területek alatt elhelyezkedő pixelintenzitások összege. Egy alap, 24x24-es felbontású detektáló ablakot használva, az összes lehetséges téglalapjellemző ablakon belüli kiszámítása igen jelentős számítási teljesítményt igényelne, mivel ebben az esetben több tízezer ilyen jellemző létezik (több, mint 160.000), így csak a legjobbak használhatók fel. Az algoritmus



3.10 ábra: Az integrálképen adott téglalapterületek reprezentálása (21)

ereje az ún. integrálképben rejlik. Az integrálkép forrása az eredeti kép szürkeárnyaltos változata, (x,y) helyen vett értéke pedig a forráskép e pozíciója felett és balra lévő pixelintenzitásainak összege. Előnye az, hogy meghatározásával egy-egy képbeli téglalapterület intenzitás-összegének kiszámítása mindössze 4 elemi művelet segítségével végezhető el, az integrálkép megfelelő helyen vett értékeit felhasználva. Példaként a 3.10 ábra tekinthető, melyen például az A terület intenzitásösszege pontosan az 1-el jelölt pont értéke, a D téglalap területének intenzitásösszege pedig a következőképpen számítható: $4+1-(2+3)$.

A téglalapjellemzők a bemeneti képek osztályozására használhatók fel. Azonban közel sem mindegy, mely jellemzőket használja az algoritmus. E módszer ennek eldöntésére, azaz a megfelelő, robosztus téglalapjellemző kiválasztására, az ún. AdaBoost tanítóalgoritmust használja, melyet Yoav Freund és Robert Schapire alkotott meg. A nevében



3.11 ábra: Az AdaBoost által elsőként kiválasztott téglalapjellemzők arc-detektálás esetén (21)

rejlik, hogy erősít (boost=erősítés), esetünkben ez, a már meglévő ún. gyenge tanítóalgorithmus erősítését jelenti, ugyanis az AdaBoost több, önmagában gyenge tanítóalgorithmust használ, a végén pedig az algoritmusok kimeneteinek súlyozott összegét adja vissza. A Viola-Jones módszerben e tanítóalgorithmus feladata, hogy megtalálja azokat a téglalapjellemzőket, melyek a legjobban végzik a bemeneti képek osztályozását. Példaként az arcdetektálást lehet felhozni, mivel a szerzők is ezzel foglalkoztak. Ebben az esetben Viola és Jones AdaBoost implementációja két olyan jelentős jellemzőt használ fel az arcon, amely a legtöbb helyzetben, a képen szemben elhelyezkedő arcok esetében, állandó arcjellemzőket vesz figyelembe. A 3.11 ábra két robosztus jellemzőt mutat be. Az első esetében, mely az egymás alatt elhelyezkedő téglalapokkal dolgozik, az algoritmus az emberi arc azon sajátosságát használja fel, hogy a szemek vonala általában sötétebb a szemvonal alatti területnél. Ebből a megállapításból létrehozta egy jellemzőt, mely a két téglalapnyi intenzitásösszegek különbségéből képződik. A második esetben egy három téglalapos jellemzőről van szó. A szemek viszonylagos sötétsége itt az orrvonal világosságával kerül összehasonlításra.

A tárgyalt objektumdetektálási módszernél Viola és Jones, a hatékonyság mellett, az erőforrásigény optimalizálására is törekedtek. A detektálás során ugyanis nem csak azt vették figyelembe, hogy a lehető legkisebb hibával felismerjenek arcokat, hanem azt is, hogy ezt minél gyorsabban tegyék. Az AdaBoost tanítóalgorithmus eredménye egy sokkal kisebb elemszámú jellemzőhalmaz lett (kb. 6.000), azonban ennyi jellemző vizsgálata egy 24x24-es ablakban még mindig túl számításigénynek bizonyult volna. Ezért módszerüket továbbfejlesztették, még pedig egy ún. lépcsőzetes osztályozó megvalósításával, melynek legfőbb feladata a bemeneti kép adott, a keresett objektumot nem tartalmazó, területeinek kizárása. Ennek hatékonysága abban érezhető, hogy egy-egy bemeneti képen, legnagyobb valószínűséggel, a legnagyobb területek a keresett objektumot nem tartalmazzák. E továbbfejlesztés egy többlépcsős folyamat lett (innen a lépcsőzetes osztályozó elnevezés), melynek elve szerint több, egymást követő szint vizsgálta az aktuális ablakot, más-más jellemzőkkel, a lehető leggyorsabb negatív terület (azaz a keresett objektumot nem tartalmazó) kizárással. A módszer szerint minél tovább jut egy ablak a vizsgálati folyamatban, annál nagyobb számításigény keletkezik, mivel a magasabb szinteknél már egyre részletesebb vizsgálat szükséges (azaz több téglalapjellemző kiszámítása). Azonban a szerencsés felépítésnek köszönhetően a negatív találattal bíró ablakok hamar kiesnek, így a végén már

csak a nagy valószínűséggel a keresett objektumot tartalmazók maradnak, melyeknek száma általában jóval kisebb.

Végezetül a betanított Viola-Jones objektumdetektor szisztematikusan végigvizsgálja a bemeneti képet egy adott méretű detektáló ablakkal. Mivel az arcok mérete nem mindig egyforma, ezért a vizsgálati ablak mérete folyamatosan nő, így egy jóval hatékonyabb algoritmust kapunk. Az ablak átfedésekkel végzi a detektálást, tehát mindig egy meghatározott deltával mozog tovább a bemeneti képen, hogy minden lehetséges objektumot legyen lehetősége észlelni. E szerzők által publikált módszer, a korábbiakhoz képest, kb. tizenötször gyorsabb futásidővel bírt.

3.2.6 KÉPI NYOMATÉKOK (22)

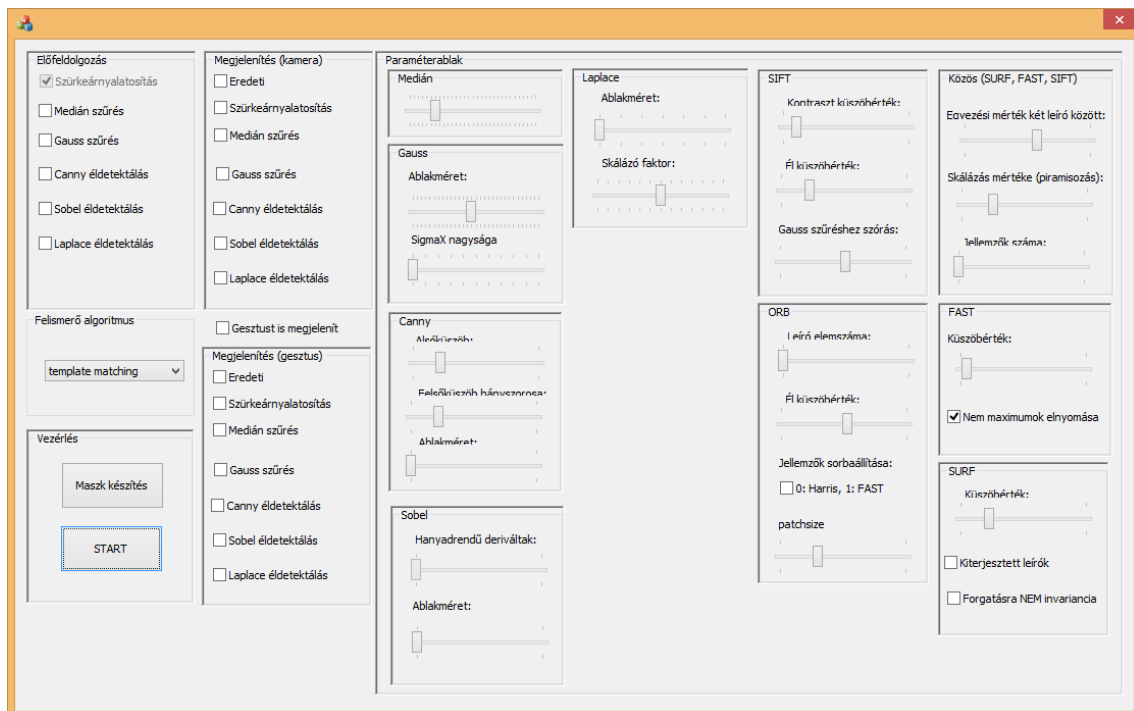
A képi nyomaték invariánsok fontos szerepet töltenek be a képfeldolgozásban objektum-felismerés esetén, jól lehet őket alkalmazni különböző minták felismerésénél, összehasonlításánál. Nevükből fakad geometriai invarianciájuk, mellyel a forgatásra, skálázásra, eltolásra is közel érzéketlenek, értékük csekély mértékben változik. Az invariáns képi nyomatékok hatalmas szerepet játszhatnak a robosztus objektum-felismerésben, amikor például egy kamera bemeneti képe az összehasonlítandó objektum forrása. Ilyen – és ehhez hasonló – nem ideális, általános esetekben nem határozható meg a keresett objektum mindenkor pontos kiterjedése pixeleken, illetve nem állandó orientációja sem lehet kiinduló pont. Mivel a sikeresen alkalmazott, skálázásnál használt, képi nyomatékok normalizáltak, így az imént említett, a vizsgált környezet változékonyságából adódó, nem meghatározható paraméterekre érzéketlenek. Minden geometriai invarianciát egy-egy nyomaték típus határoz meg. A nyomatékok alapja, a vizsgált képi objektum súlypontja, illetve a 0-ad rendű nyomaték, melyet tömegnek neveznek és a pixelintenzitások összegét jelöli. Eltolás invariancia esetén centrális nyomatékot használunk, skálázás invarianciánál normalizált centrális nyomaték használatos, mivel ez utóbbi esetben az objektum mérete, azaz területe is jelentősen változhat, mely befolyásolja a kapott nyomaték értékeket. A Ming-Kuei Hu által publikált, ún. Hu nyomatékok által, a geometriai forgatás invariancia is megjelent a képfeldolgozásban, az objektum-felismerés területén, melyek alkalmazása ugyancsak segít az eltolás, illetve a skálázás invariancia elérésében.

4. SAROKDETEKTÁLÓ ALGORITMUSOK TESZTELÉSE, EREDMÉNYEI

4.1 A TESZTELÉSEK MEGVALÓSÍTÁSA

A sarokdetektáló algoritmusok tesztelését egy C++ kódban írt program valósította meg, mely az OpenCV könyvtáraiban implementált függvényeket is felhasználta. A felhasználói felületet a Visual Studio MFC (Microsoft Foundation Class) alkalmazása adta.

A tesztelt algoritmusokat a lehető legjobban paramétereizhetővé tettem, mely paramétereket a felhasználói felületen (4.1 ábra) lehetséges valós időben állítani. Ilyen paraméterek a különböző küszöbértékek (pl.: a kívánt jellemzők száma), néhány egyéb, az algoritmusok működéséhez szükséges opció. Az alkalmazáson belül lehetőség van valós időben változtatni az előfeldolgozást és paramétereit. Ilyen előfeldolgozás lehet például a Gauss szűrés vagy a Medián szűrés. Minden részeredményt külön ablakban lehetőség van megjeleníteni. A célom az volt, hogy teszteljem a megismert algoritmusokat, megvizsgáljam azok viselkedését különböző helyzetekben, például a háttér változásakor (homogén, nem homogén). Bemenetként pedig egy kézzel készített képet használtam.



4.1 ábra: A tesztalkalmazás konfigurációs felülete (saját ábra)

4.2 EREDMÉNYEK

Bemeneti képként a folyamatos webkameraképet használom. A gesztus, amit keresek, egy betöltött képen helyezkedik el, homogén háttér előtt. Csak maga a kézgesztus látható rajta. Tapasztalataim alapján inhomogén háttér előtt nem hatékony az algoritmus, túl sok fals pozitív találatot ad a jellemzők párosításakor. Ez zajszűrés (Gauss szűrő) használatával orvosolható egy bizonyos mértékig, azonban túl nagyméretű szűrő használata esetén (ha már nagyon elmosódik a kép), nem talál releváns jellemzőket egy algoritmus sem. Továbbá a kívánt jellemzők számának változtatása is befolyással bír a működésre. Túl kevés jellemző esetén nem talál elég párosítást az adott algoritmus egy pozitív találathoz, a túl sok jellemző pedig túl sok, általában nem korrekt párosítást talál és így fals pozitív eredményt hoz. Ezeket a felvetéseket a gyakorlat is igazolta. A mérések során előfordult pozitív, fals pozitív és fals negatív felismerés is. A zajok miatt a perspektív transzformáció sem tökéletes, azaz nem mindig rajzolja ki megfelelően a megtalált kéz körül a keretet, de a találat egyértelműen látszik.

A tesztelés azt mutatta, hogy leggyengébben a SIFT szerepelt (ráadásul a legtöbb erőforrást is ez használta föl), míg legerősebbnek az ORB bizonyult. Az ORB az egyes tesztelési környezetekben jelentős eltérést produkált a találati arány szempontjából, legjobban a zajszűrt homogén háttér előtt bizonyított, de a nem szűrt homogén háttér előtt is biztató eredmények születtek. Emellett még gyors feldolgozást is produkált, mely a FAST-ról is elmondható. A SIFT azon kívül, hogy igen lassú is volt, még említésre méltó helyes párosítása sem produkált inhomogén háttér előtt. Homogén háttér előtt a SURF talált egyezéseket, azonban a U-SURF (Upright SURF~Egyenesen álló SURF) használata, azaz a forgatás invariancia mellőzése, nem járt érezhető teljesítménycsökkenéssel. Érdekes módon, inhomogén háttér előtt a Gauss szűrő használata nem változtatta meg számottevően a pozitív eredmények számát, inkább a fals pozitívak száma csökkent valamelyest. Sajnos viszont a valós használatbeli eredmények alkalmatlanná tették ezeket a megközelítéseket kezdetektől, a sok fals pozitív találat, illetve a zajokra való túlzott érzékenység miatt. Emiatt más módszereket kerestem.

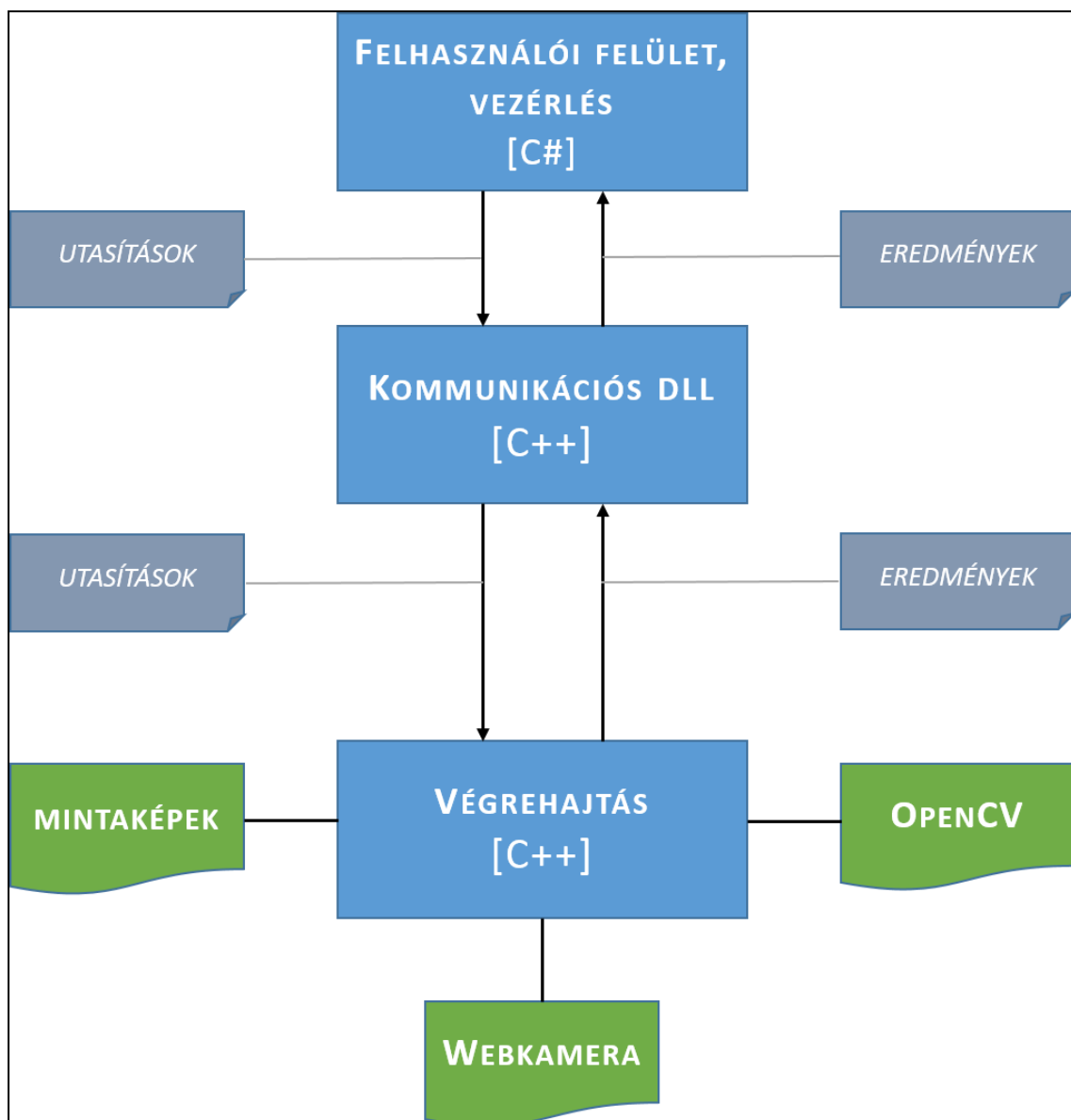
5. HAAR JELLEMZŐ- ÉS KONTÚR ALAPÚ ALGORITMUSOK TESZTELÉSE, EREDMÉNYEI

A sarokpont alapú objektumdetektálás nem kielégítő eredményei okán nagy hangsúlyt fektettem a Haar jellemzőkön, illetve a kontúrokon alapuló technikák mély tanulmányozására, majd alkalmazására. Ehhez, továbbá teljesen újragondoltam a megvalósítást programozói szinten is. A következőkben ezt a programot fogom részletezni, mely pedig maga a végleges alkalmazás.

5.1 A MEGVALÓSÍTÓ PROGRAM

A dolgozat témáját megvalósítandó, a korábbi tesztelés során tapasztaltak alapján terveztem meg a feldolgozást megvalósító alkalmazást (5.1 ábra). Azt tapasztaltam, hogy sokkal egyszerűbb az OpenCV-t a saját környezetében, a C++ nyelvvel kezelni. Nem kell hozzá saját wrappert (~csomagoló) írni, esetleg a már mások által megvalósított, szintén saját tapasztalat alapján, sok szempontból nehézkesen használható, EmguCV .Net wrappert használni. Így a program lényegi, képfeldolgozó, azaz végrehajtó modulja C++ alapokon nyugszik. A felhasználói felület lehetőségeinek tárháza és programozói oldali használhatósága sokkal kedvezőbb C#-ban, ezért a megjelenítés és a vezérlés ebben a programnyelvben történik. A két modul közötti kapcsolatot egy C++-ban írt DLL szolgáltatja. A kommunikáció az ebben megírt, majd C#-ból meghívott függvények segítségével történik. Mind a használt paraméterek, mind a visszatérési értékek a mindkét nyelvben ugyanazt jelentő adattípusok, mint például az egész szám típus (integer), illetve a két nyelv ekvivalens típusai (C#-nál a Byte, mely C++ nyelvben az unsigned char-nak felel meg). Minden ilyen függvény neve egyértelműen meghatározza azok viselkedését, így ez könnyű használhatóságot biztosít a C# oldali modul programozásánál és a C++ nyelvben megvalósított függvények meghívásánál. Fontos a gyors működés, így az alkalmazás felépítése is e szerint történt. C++ oldalon az OpenCV beépített típusai a dominánsak, azonban a C# modul mindössze egy-egy bájt folyamatot kap meg, melyet ezután Bitmap formátumú képpé alakít és megjelenít egy ablakban. Ez nem felügyelt (unsafe) programkód segítségével lehetséges rendkívül gyors memóriaműveletekkel. C++ oldalon a megvalósító függvények szintúgy egyértelmű feladatokat látnak el, továbbá összekapcsolhatók a hasonló függvény szignatúráknak köszönhetően, így tetszőleges, ha úgy tetszik, moduláris végrehajtási folyamatokat lehet általuk létrehozni. A megfelelő memória-felszabadítás teljesen megoldott, C#-ban is van lehetőség a megkapott

kép (byte folyam) memóriabeli kitisztítására. A C# oldali felhasználói felület két üzemmódot kínál fel. Egyrészt egy hibakeresési (debug), valamint a végleges működést reprezentáló felhasználói (user) módot. Hibakeresési módban lehetőség van az egyes funkciókat külön-külön tesztelni, illetve különböző végrehajtási folyamatokat is végigkövetni. Egyszerre több funkció is tesztelhető, lehetőséget biztosítva a valós idejű összehasonlításához. Ekkor minden egyes végrehajtási folyamat külön szálon (thread) fut. A programkód alapnyelve az angol, a könnyebb átláthatóság érdekében, mivel például az OpenCV függvényei is angol nyelvűek. Ez mind a függvények, mind a változók neveit is érinti, így marad konzisztens e szempontból is az alkalmazás.



5.1 ábra: A megvalósító program moduljainak kapcsolata (saját ábra)

5.2 AZ IMPLEMENTÁCIÓ RÉSZLETEZÉSE

Az előfeldolgozást és a feldolgozást végző függvények egy előre meghatározott metódus szignatúrával bírnak. A feldolgozás egyes lépései az OpenCV kép típusú változóján (IplImage) hajtódnak végre, így ez egy kötelező paramétere minden függvénynek. A köztes lépésekben e metódusoknak visszatérési értékük nincs, azonban a végleges eredményt egy egész szám típusú visszatérési érték szolgáltatja, tehát ez egy kitüntetett szerepű függvény. A DLL, azaz a C#-C++ kommunikációért felelős, ún. dinamikus csatolású könyvtár, metódusai a programnak a megfelelő adatátadáshoz szükséges szignatúrát használják. Amennyiben a működésük eredménye egy kép, úgy visszatérési értéként egy-egy unsigned char mutatót adnak vissza, melyet C#-ban Byte mutatóként lehet kezelni. Az alkalmazás a kép méreteit az inicializálásnál már megkapja (a DLL-en keresztül, a kamerától), a színcsatornák száma pedig, az egyes függvényeknél, lehet változó (1 vagy 3), esetleg egyértelmű (meghatározott, hogy szürkeárnyaltos vagy színes az eredmény), így a memóriából való kiolvasás nem vezet hibához. Egyszerű paraméter-beállítást végző függvéynél nincs visszatérési érték.

A továbbiakban a működéshez elengedhetetlen modulok kerülnek részletezésre.

5.2.1 VEZÉRLÉS

A vezérlést a - korábban említett - C# alkalmazás szolgáltatja. Fő feladata a felhasználóval történő interakció feladatainak ellátása, mint a felhasználói interfész utasításainak végrehajtása a DLL-ben megtalálható függvényekkel, illetve az eredmények rögzítése és közlése. A két mód, mellyel a program rendelkezik, alapvetően nem tér el egymástól, ha az alap működési logikát nézzük. Azonban a hibakeresési mód sokrétűsége a felhasználói módban már nem nyilvánul meg. Ez utóbbi egyszerű és puritán, egyedül a véglegesített használatot látja el és információt szolgáltat a kézfelismerés eredményességéről. Ha sikeres felismerés történt, akkor a megfelelő művelet hajtódik végre az adott célalkalmazásban.

5.2.1.1 HIBAKERESÉSI MÓD

Hibakeresési módban az összes tesztelési és korai fejlesztési időszakbeli funkciókat lehet használni. A célja az elkészült függvények kielégítő tesztelése, s mint a nevéből is következik, a hibák felfedezése, illetve a hatékonyság mérése. Minden funkció eredménye külön ablakban jelenik meg, párhuzamosan fut, külön szálakon. A cél az, hogy nyomon

lehesen követni az eredeti kamerakép és az egyes eredmények vagy az eredmények egymás közötti különbségét. Lehetőség van valósidőben állítani az egyes funkciók paramétereit, mint például színalapú szegmentálás esetén a szinterek küszöbértékeit.

5.2.1.2 FELHASZNÁLÓI MÓD

Felhasználói módban a végleges működés valósul meg. Az egész projekt célja az, hogy multimédiás alkalmazások egyszerű funkcióit lehessen vezérelni kézgesztusokkal. Az alkalmazás felhasználói része mindössze egy egész számot vár bemenetként, mely egyértelműen meghatározza a működés mikéntjét. Amennyiben egy nem negatív szám a paraméter, úgy az adott számhoz rendelt kézgesztust ismerte fel a rendszer, s így a megfelelő parancsot kell a programnak kiadnia az egyik multimédiás alkalmazásnak.

5.2.2 VÉGREHAJTÁS

A végrehajtás az OpenCV natív környezetében történik, azaz C++ nyelvben. Sokkal szabadabbnak tapasztaltam a fejlesztést így, mint EmguCV. NET wrappert használva, mert így minden függvényt közvetlenül tudok meghívni és nem kell plusz egy réteg működését megérteni és az ezzel járó hibákat is ki tudom így küszöbölni. Az implementálás során használtam mind a C++, mind az OpenCV beépített típusait és függvényeit. A következőkben bemutatom az előfeldolgozást és a feldolgozást végző fontosabb metódusokat.

- **CreateCamera:** A paraméterként átadott ID alatti kamerát állítja be, mint az elsődleges képforrást, a metódus. Továbbá eltárolja a beolvasott képkockák méretét, melyet a C# alatti vezérlő program is le tud kérdezni, így az eredmény képeket mindig helyesen tudja megjeleníteni. Ezek mellett a program működéséhez szükséges inicializálásokat is végrehajt a függvény.
- **GetFrame:** Szürkeárnyaltos vagy színes képet olvas be az elsődleges képforrásról.
- **DeleteImageData:** Lehetőséget biztosít a C# oldali vezérlő programnak a korábban megkapott byte folyamat, megjelenítés után, a memóriából kitakarítani.
- **DoPreProcessing:** A beolvasott kép előfeldolgozását hajtja végre. A zajok eltávolítása fontos feladat, mivel helytelen eredményt adhatnak sok esetben a használt algoritmusok, ha ezt nem hajtjuk végre.
- **DoColorSegmentation:** RGB színalapú szegmentálást hajt végre a bemeneti színes kameraképen. Minden pixelre megvizsgálja a 3 színcsatornára azt, hogy az

intenzitásértékek egy – korábban beállított, tapasztalat szerint a legtöbb körülmény között helytálló – intervallumon belül helyezkednek-e el. Amennyiben mindegyik színcsatornára teljesül ez utóbbi, akkor a vizsgált pixel fehér előtérpixel, ellenben pedig fekete háttérpixel lesz. Az eredmény bináris képen így csak a számunkra lényeges információ marad, mellyel sokkal könnyebb és hatékonyabb tovább dolgozni.

- **LoadTrainedFiles:** Az AdaBoost algoritmus által meghatározott jellemzőket tartalmazó XML-ek betöltése a programba. A betöltött XML fájlokat a kézdetektálás során használja az alkalmazás. Több különböző XML egyidejű használatával biztosítható az egyes XML-ek hiányosságainak kiküszöbölése, a különböző helyzetben megjelenő kezek detektálása (amennyiben megfelelő XML fájlokat használunk).²
- **FindHand:** A kézdetektálást végző függvény, melynek bemeneti paramétere a vizsgálandó kép, visszatérési értéke pedig a megtalált kézgesztus (előre rögzített) sorszáma vagy -1, amennyiben nincs találat. A módszer először a feldolgozáshoz szükséges műveleteket hajtja végre, mint a bemeneti kép szürkeárnyalatossá alakítása, illetve zajszűrés elvégzése Gauss szűrővel. Következő lépésként, a korábban betöltött, betanított XML-ek által inicializált Haar



5.2 ábra: A kéz kontúrja a konvex burokkal és a konvex hibákkal. A sárga és zöld vonalak konvex hibákat jelölnek, a zöldek a csuklónál elhelyezkedő konvex hibák. (saját ábra)

² XML fájlok forrásai: <http://www.andol.info/hci/1830.htm>, <https://code.google.com/p/advanced-tracker/downloads/detail?name=Hand.Cascade.1.xml&can=2&q=#makechanges>, <https://github.com/chenosaurus/drone/tree/master/xml>

osztályozókon lépked végig a végrehajtás, és a bemeneti képen elvégzi az ezek segítségével történő kezdetektálást. Az összes találat egy vector típusú változóba kerül, mely az ezeket reprezentáló téglalapokat tartalmazza. Ezzel is az erős találatok megtartása a cél, ehhez pedig a több betanított XML alkalmazása a kulcs. A következő lépés az összes megtalált téglalap egyenkénti vizsgálata, mivel minden téglalap tartalmazhat jó eredményt. Ebben a fázisban az egyes téglalapterületek pixeleinek bőrszín alapú küszöbölése (mely előre beállított paraméterekkel történik) történik meg. Az eredmény fehér és fekete pixeleket tartalmaz, melyből az előbbi a számunkra értékes, ami pedig az előtér. Ez után, a képen, a zárás morfológiai operátor kerül alkalmazásra. Ez akkor nagy segítség a program számára, amikor az egyes kézgesztusok területe hibásan kerül küszöbölésre, a zavaró fényviszonyok miatt, és apró fekete területek maradnak a szignifikáns területen. Mivel a morfológiai zárás először dilatációt, majd eróziót hajt végre, a zavaró területek eltűnnek és a kéz kontúrja sem torzul látható mértékben. Utolsó lépésként egy összetett folyamatot alkalmaz a függvény. A megmaradt területen kontúrkeresés (23) történik. Majd a MatchContour nevű implementált függvény kerül meghívásra, a legnagyobb megtalált kontúrt paraméterként megkapva. Ez a függvény a megkapott kontúr köré létrehoz egy konvex burkot (24), mely egy sokszög. Ezután meghatározásra kerül az összes olyan pont a konvex burok és a kontúr alapján, melyek miatt nem lehetséges teljesen jó konvex burok közelítést adni a kontúrra, mivel ekkor az már konkáv lenne. Ezek az ún. konvex hibák (5.2 ábra). A konvex hibákból a nagyon kicsik nem lényegesek, a többiből az a 2 az érdekes, melyek a csukló alját jelzik, ezeket a program megkeresi és az alatta lévő kontúrponthoz kitörli, így a Hu invariánsok alapján összehasonlító függvény helyes párosítást tud majd adni (mivel felesleges kontúrrész nem marad a kontúr vektorban). Végül a mintakontúrok és az előbbieken előkészített kontúr összehasonlítása történik meg. Amennyiben a legjobb találat jósága egy, előzőleg meghatározott, küszöbvel egyenlő vagy nagyobb érték, akkor a megfelelő kontúr, azaz kézgesztus, sorszáma lesz a visszatérési érték, egyébként -1. A FindHand függvény is ezt az értéket adja vissza, az azt meghívó metódusnak, melyet – a kommunikációs DLL-en keresztül – a C# vezérlő modul is megkap. A további műveleteket így az könnyűszerrel elvégezheti.

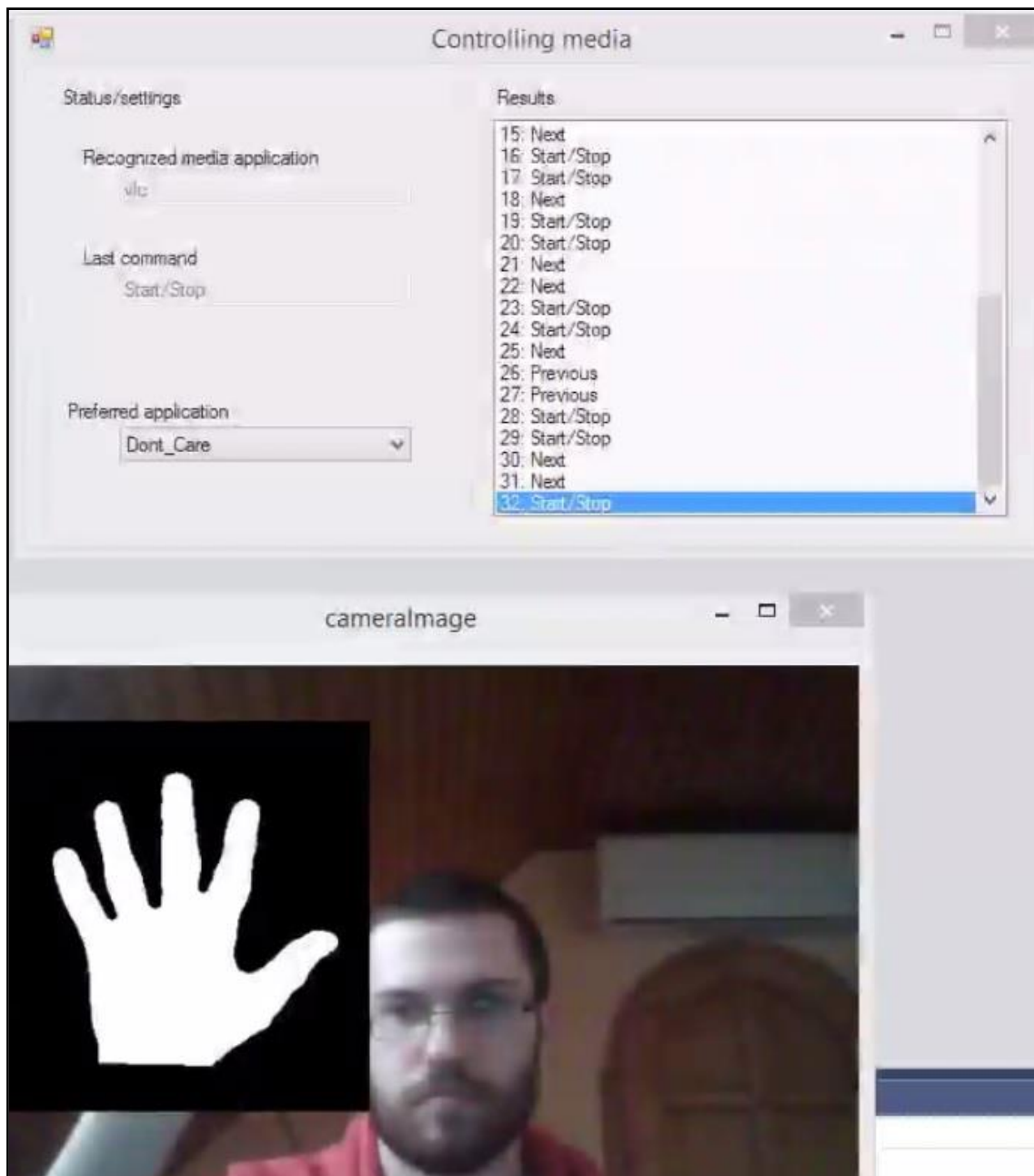
5.2.3 KOMMUNIKÁCIÓS DLL

Az ún. kommunikációs DLL a legfontosabb kapocs, mely nélkül az alkalmazás működésképtelen lenne. Elválasztja a vezérlést a megvalósítástól és az adott funkciónak megfelelően hívja meg a szükséges megvalósító függvényeket a C++ kódból. Így egy-egy hívásra egy teljes beállított végrehajtási folyamat futtat le. E köztes rétegben, mind a visszatérési értékek, mind a paraméterek esetében, a C#-al ekvivalens típusok használatosak, így a fejlesztés szempontjából könnyedséget jelent ez is.

5.3 EREDMÉNYEK

5.3.1 A TESZTELÉS RÉSZLETEZÉSE

A programot végleges, rendeltetésszerű használat mellett teszteltem le. Ez azt jelenti, hogy a kiadott parancs gesztusokat vizsgáltam alkalmazás vezérlési szempontból. Ebben az üzemmódban a legfontosabb az adott preferált média lejátszó alkalmazás megfelelő vezérlése. Továbbá fontos pont a személyfüggetlen működés is, ugyanis a végleges alkalmazásnak erre



5.3 ábra: A felhasználói mód felhasználói felülete működés közben, mely információt nyújt az adott médiaalkalmazás vezérléséről. Jobbra láthatók az indítás óta fogadott felhasználói parancsok. (saját ábra)

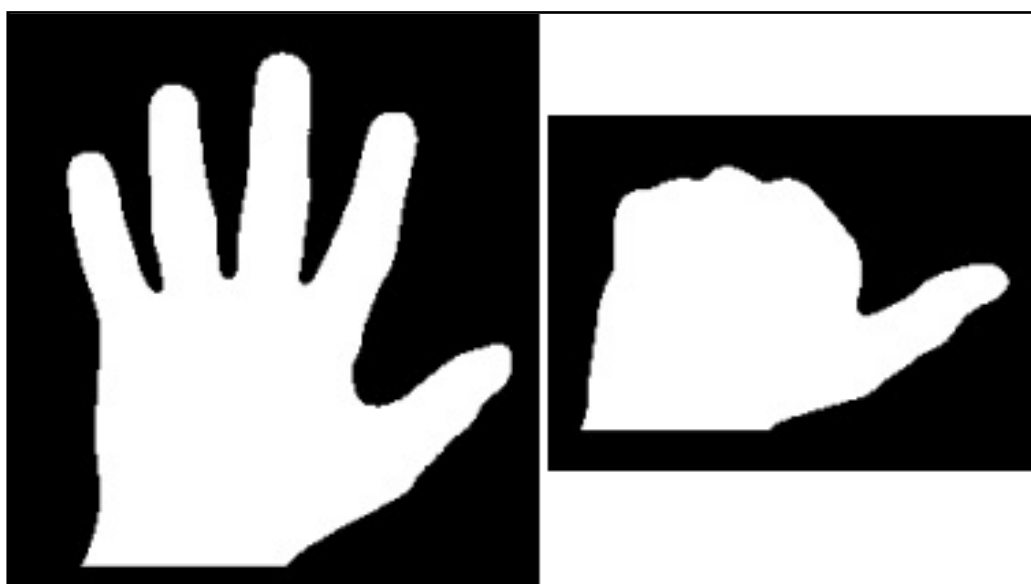
is képesnek kell lennie. Ez utóbbit segíti az előre beállított hibahatár, mely a mintaképek és a bemeneti kép kontúrjainak összehasonlításához kell, s mely még így is igen alacsony érték, a fals pozitív találatokat kiküszöbölése érdekében. Az OpenCV matchShapes függvénye (mely Hu invariánsokkal számol), visszatérési értékként 0-t ad, ha a vizsgált kontúrok teljesen megegyeznek, nagyobb értékű eredményének mértéke pedig a két kontúr eltérésének mértékével arányos. Tehát, ebből kiindulva, az említett küszöb egy 0-nál nem sokkal nagyobb szám, a 0,01, mely elég mozgásteret biztosít a személyfüggetlenség és a képzajok esetén is, a felismerés helyességében, a rossz találatok kiküszöbölése mellett. Mivel felhasználói módban a kényelmes használat (5.3. ábra) a fontos, ezért bevezettem egy bizonyos megszorítást is (így nem történik 1 másodperc alatt sok parancskiadás): egy meghatározott időn belül, 3 darab azonos gesztust kell a programnak felismernie a parancskiadáshoz. A tesztelést a felhasználói üzemmódban hajtottam végre, mely már a végrehajtási logikával összekapcsolt, végleges program. A program - alapértelmezetten - detektálja az éppen aktív médiaalkalmazást vagy az éppen futót. Lehetőség van egy preferált alkalmazás követésére, ebben az esetben az automatikus detektálás nem üzemel (csak a beállított alkalmazásra). A mutatott gesztusok felismerésével, a figyelt médiaalkalmazás vezérlése rendeltetésszerűen működött, a hibás gesztusokat a működési logika a legtöbb esetben korábban elvetette, így azok nem produkáltak helytelen parancskiadást. Ez azért lehetséges, mert a kontúrok összehasonlítása érzékenyen reagál bármily nemű jelentős változásra (mint a kontúrterület vagy kerület), így a Haar kézdetektálás fals pozitív eredményei a következő szintre már nem léphetnek. A személyfüggetlenséget, rajtam kívül, 3 ember tesztelte. A program, ez utóbbi esetben is, a korábbiakhoz hasonló működést produkálta. Teljesítmény szempontjából is érdemes megvizsgálni a működést. A futtatókörnyezet a következő volt: Intel Core i7-3537U 2.00GHz 2 magos processzor, illetve 8gb memória. A tényleges futás során másodpercenként kb. 20 képkocka került vizsgálat alá, tehát gyakorlatilag valós idejű feldolgozásról beszélünk. Ekkor 20mb memóriát használt az alkalmazás és átlagban 30%-os processzorhasználat volt jellemző. Úgy gondolom, hogy ebből a szempontból is fejlődnie kell a programnak, érdemes a feldolgozást még jobban átgondolni, javítani annak hatékonyságán.

Háttér →	Homogén (világos)	Homogén (sötét)	Inhomogén	Összesen	Összesen (világos háttér nélkül)
Környezet ↓					
Nappali fény	13%	75%	70%	52,67%	72,5%
Esti lámpafény	3%	50%	40%	31%	45%
Este sötétben	35%	39%	49%	41%	44%
Összesen	17%	54,67%	53%	41,56%	53,83%

5.1 táblázat: A tesztelés eredményei, a sikeres találatok százalékban (saját táblázat)

5.3.2 A TESZTELÉS EREDMÉNYEI

Az alkalmazás az inicializálási fázisban betölti a mintaképeket, melyek az összehasonlítás alapját fogják a működés során szolgálni. Ezeken mindössze a kézgesztusok találhatók, melyek fehér színűként jelennek meg egy fekete háttér előtt, kitöltött állapotban (5.4 ábra). Összesen 100 gesztust mutattam a kamera felé. Az eredményeket, előttük sorszámmal, a felhasználói mód jobb oldalán található lista elem tárolta. A tesztelést úgy építettem föl, hogy mindig változzanak a gesztusok, illetve a kéz helyzete is a kamerához képest, így hiteles eredményt szolgáltatva. Amikor egy-egy objektumot a program gesztusként értelmezett, annak kontúrja megjelent a kamerakép megjelenítési ablakában. Ha azonban nem volt sikeres a gesztuskeresés, akkor nem jelent meg plusz információ. Ez utóbbi következhetett a környezeti zavaró hatásokból vagy nem létező gesztus megjelenéséből egyaránt. A táblázat (5.1 táblázat) a mérés eredményeit szemlélteti nappali fénynél, esti



5.4 ábra: Példa minta kézgesztusokra – a képen a gesztusok kontúrjai kitöltött állapotban láthatók (saját ábra)

lámpafénynél, illetve este, lámpafény nélkül (csak a monitor fénylett). Mindegyik esetet felbontottam homogén, illetve inhomogén háttér előtti tesztelésre is. A homogén háttér világos, illetve sötét háttérrel is jelentett. Az eredményekből sok mindent meg lehet állapítani. Először is azt, hogy az inhomogén háttér kevésbé befolyásolja az eredményeket. Ez legfőképp a bőrszín alapú szegmentálás jótékony hatása lehet, mivel egy zavaros háttérben kisebb valószínűséggel jelenik meg nagy, összefüggő, bőrszínű objektum. Sajnos a bőrszín alapú szegmentálás mellékhatása a bőrszínhez hasonló színű objektumokkal való összeakadás. Ezért – jelen pillanatban – nem alkalmas a program ahhoz hasonló homogén háttér előtti használatra. Meglepő adat a monitor fénye előtti tesztelés eredménye, de a körülményeket átgondolva mégsem annyira különös. Sötétben ugyanis a háttér hatása sokkal kisebb jelentőségű, az előtér pedig felértékelődik. Összességében pedig az látható, hogy - a világos háttérrel kivéve - a legtöbb körülmény között képes az alkalmazás értékelhető eredményeket produkálni.

6. TOVÁBBI MUNKA

A tesztelések tapasztalataiból levonva a következtetéseket, a következő mérföldköveket határoztam meg a későbbi továbbfejlesztéshez.

- Fontosnak tartom a bőrszínhez közeli hátterekkel, objektumokkal fedésben lévő kézgesztus e színektől való függetlenítését. Mivel egy nagyobb intervallumot használok a bőrszín alapú küszöbölésnél (mindhárom színtérben), ezért esetenként olyan színek is előtérpíxelek lehetnek, melyeknek nem kéne annak lenniük. Ezen kívül, nem megoldott a színesbőrű emberek alkalmazásvezérlése sem, azonban erre is megoldást jelenthet a többkörös küszöbölés. Az alkalmazott Haar osztályozók képesek megtalálni a bemeneti képen a tenyeret vagy az öklöt is. Ez a program első lépése, viszont ez még nem elegendő, hiszen egyértelműbb jellemzőket kell keresni, mint a kontúr, mely ugyancsak megtörténik. Azonban a bőrszínhez hasonló színek zavart okoznak a küszöbölésben, azaz az „érdekes” terület kijelölése során. Ezt a hibát ki kell küszöbölni, mely többlet számításigénye miatt a teljesítmény rovására mehet. Ezért kívánatos egy olyan kizáró eljárást is létrehozni, mely korai szakaszban képes elvetni egy-egy, valószínűleg negatív találatot.
- Erősíteni kívánom a Haar XML alapú kézdetektálást is. Ehhez saját betanítást tervezek megvalósítani, különböző környezeti hatásokban, mint a lámpa fényénél vagy félhomályban, tehát összességében az alkalmazásban – jelen pillanatban – gyengeséget mutató kézdetektálási területeken.
- Optimalizáltabb működést tartok szükségesnek a továbbiakban. Elsődleges szempont a processzorhasználat csökkentése, melyet a végrehajtás kisebb átstrukturálásával, újragondolásával, véleményem szerint, jelentősen lehet majd csökkenteni.

7. ÖSSZEGZÉS

Feladatombként azt tűztem ki, hogy egy olyan szoftver tervezzek és fejlesszek, amely erős megszorításokat mellőzve képes egy egyszerű webkamera segítségével emberi, statikus kézgesztusokat felismerni, és a kapott adatok felhasználásával multimédia alkalmazásokat vezérelni. Az általános működés a kezdetektől fogva a legfontosabb szempont volt. A fejlesztés során sikerült megbirkóznom e feladattal, a rendszer erős magját lefektettem, habár további fejlesztésekre még szükség van a minden körülmények közötti robosztusabb működéshez.

A tesztelések és a használat során tapasztalt eredmények igen kecsegtetőek, a zavaró környezeti körülményekhez történő alkalmazkodást is figyelembe véve. Azonban a meglévő működési logika további erősítésére van szükség, adott környezetekben, mint a homogén háttér és a bőrszín kapcsolata vagy a zavaró fények hatása küszöböléskor. A vizsgált kereskedelmi kézgesztus-felismerő rendszer, a Flutter App, hasonlóan viselkedik homogén világos háttér esetén, mint az általam fejlesztett rendszer, azaz a legnagyobb gyengeségét ez utóbbi okozza számára. Az e körülmény alatti működés javítása egy fontos szempont a további fejlesztés során. A továbbiakban így arra fogok törekedni, hogy minél robosztusabbá tegyem a rendszert, alkalmazkodóvá az egymáshoz képest nagy kontrasztot mutató környezetekben.

8. SUMMARY

My task was to design and develop a software which has the capability of recognizing static hand gestures without any strong constraints and only with an assistance of a simple webcam, and which is able to use the received data to control multimedia applications. Creating a general solution has been the most important. During development I could deal with the task I undertook: I finished the main logic of the system, though there are further developments which are need to be done in order to have an application which can deal with most of the conditions and can be more robust.

The results I got, during the tests and while I was using the application, were proposing, despite the disturbing environmental conditions. Although the existing functional logic needs more improvement under specific conditions like light colored homogeneous background or avoiding disturbing light sources. The tested hand gesture recognizing commercial application, named as Flutter App, behaves the same, when there is a light colored homogeneous background, as the system developed by me, so this is its biggest weakness. To have a more correct operation, under the previous circumstances, some further development needs to be done. In the future development I am going to make the most robust application I can, which will adapt to the most different environmental conditions.

IRODALOMJEGYZÉK

1. *Hand Gesture Recognition Systems: A Survey*. **Arpita, R. S., Sanyal, G. és Majumder, S.** 2013., International Journal of Computer Applications (0975 – 8887), old.: 26-37.
2. **Cui, Y. és Weng, J.** *Appearance-Based Hand Sign Recognition from Intensity Image Sequences*. 2000.
3. *A Review of Vision Based Hand Gestures Recognition*. **Murthy, G. R. S. és Jadon, R. S.** No. 2, 2009., International Journal of Information Technology and Knowledge Management, 2. kötet, old.: 405-410.
4. *A Fast Algorithm for Vision-Based Hand Gesture Recognition for Robot Control*. **Malima, A., Ozgur, E. és Cetin, M.** 2006., In Proc. of 14th IEEE Signal Processing and Communications Applications.
5. *A Hand Gesture Recognition based Virtual Touch World*. **Gowtham, P. N. V. S.** 2012., Int. J. of Information & Education Technology, old.: 36-42.
6. *Real time Hand Gesture Recognition using SIFT*. **Gurjal, P. és Kunnur, K.** 2012., Int. J. of Electronics & Electrical Engg, old.: 19-33.
7. **Davidson, M. W. és Abramowitz, M.** *Difference of Gaussians Edge Enhancement*. Optical Microscopy, Digital Imaging and Photomicrography. [Online] [Hivatkozva: 2014. 03 10.] <http://micro.magnet.fsu.edu/primer/java/digitalimaging/processing/diffgaussians/index.html>.
8. **Finlayson, G. D., Schiele, B. és Crowley, J. L.** *Comprehensive Colour Image Normalization*. 1998.
9. *Intelligent Approaches to interact with Machines using Hand Gesture Recognition in Natural way: A Survey*. **Chaudhary, A., és mtsai.** 2011., International Journal of Computer Science & Engineering Survey (IJCSES), 2. kötet.
10. **Paulus, D., Csink, L., és Niemann, H.** *Color Cluster Rotation*. 1998.
11. **Lowe, D. G.** *Object Recognition from Local Scale-Invariant Features*. 1999.
12. **Bay, H., Tuytelaars, T. és Van Gool, L.** *SURF: Speeded Up Robust Features*. 2006.
13. **Rosten, E. és Drummond, T.** *Machine learning for high-speed corner detection*. 2006.
14. **Rublee, E., és mtsai.** *ORB: an efficient alternative to SIFT or SURF*. 2011.
15. **Calonder, M., és mtsai.** *BRIEF: Binary Robust Independent Elementary Features*. 2010.

16. *Flutter*. [Online] 2014. [Hivatkozva: 2014. 08 02.] <https://flutterapp.com/>.
17. *Google buys human-gesture recognition start-up Flutter*. [Online] 2013. 10 03. [Hivatkozva: 2014. 08 02.] <http://www.bbc.com/news/technology-24380202>.
18. **Samsung**. *All of the Smart TV Gestures*. [Online] 2013. [Hivatkozva: 2014. 08 02.] http://www.samsung.com/global/microsite/tv/common/guide_book_5p_vi/main.html.
19. **Katzmaier, D.** *Samsung Smart Interaction: Hands-on with voice and gesture control*. [Online] 2012. 04 09. [Hivatkozva: 2014. 08 02.] <http://www.cnet.com/news/samsung-smart-interaction-hands-on-with-voice-and-gesture-control/>.
20. *Rapid Object Detection using a Boosted Cascade of Simple*. **Viola, P. és Jones, M.** 2001.
21. **Csink, L. és Sergyan, Sz.** *Color Histogram Normalization using Matlab and Applications in CBIR*. SSIP'05, Szeged : ismeretlen szerző, 2005.
22. **Flusser, J., Suk, T. és Zitová, B.** *Moments and Moment Invariants in Pattern Recognition*. 2009.
23. **Abe, K. és Suzuki, S.** *Topological Structural Analysis of Digitized Binary Images by Border Following*. *COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING*. 1985.
24. **Sklansky, J.** *Finding the Convex Hull of a Simple Polygon*. 1982.