**Virtuális Egér**

**2. mérföldkő**

**1. Feladat kiírás**

Egy olyan rendszer megvalósítása, melyben kamera értelmezi a kéz és az ujjak mozgását. Ezek a mozgások megfeleltethetőek egy igazi egér eseményeinek, mintha egy touch pad-ot kezelnénk. Kamera elhelyezhető a monitor tetején, vagy egyéb helyen.

**2. Megvalósítás**

Olcsó webkamera segítségével, mely felülről figyeli a kéz gesztusait.

* *Kéz alapértelmezett állapota*: mutató ujj előre felé, kinyújtott helyzetben. A kéz mozgása megfeleltethető az egérmutató mozgásának.
* *Bal klikk esemény:* Hüvelyk ujj kinyitása egy bizonyos küszöb szög feletti állásba, a mutató ujjhoz képest.
* *Bal gomb dupla kattintás esemény:* Hüvelyk ujj kinyitása egy időintervallumon belül kétszer, a küszöb szög fölé.
* *Bal gomb nyomva tartás esemény:* Hüvelyk ujj kinyitása a küszöb szög fölé és a szög fölött tartása egy meghatározott időn túl.
* *Jobb klikk esemény:* Még nincs rá működő megoldás

A programozáshoz Microsoft Visual Studio 2010-et és OpenCV-t használunk (c++).

**3. Algoritmusterv**

1. kamera inicializálása (felbontás, fps)
2. minden frame-re:
3. szegmentálás (2D normalizált szintéren alapuló, vagy HSV szintéren alapuló)
4. ujjhegyek meghatározása (kontúr és konvex burok segítségével)
5. kurzor új pozícióba mozgatása
6. ha a kézgesztusból következik: kattintás
7. amíg a felhasználó le nem állítja a programot

**4. Szegmentálás**

Az algoritmus egyik legfontosabb lépése a szegmentálás. Az a cél, hogy egy olyan bináris képet kapjunk, amely már csak a kéz pixeleit tartalmazza vagy tökéletesen elkülöníthető a kéz más, zavaró objektumoktól. Ez egy nehéz feladatnak bizonyult és nem is sikerült tökéletes megoldást adni rá. Két féle bőr alapú szegmentálást implementáltunk:

* 2D normalizált szintéren alapulót és
* HSV színtéren alapulót

A legnagyobb gondot a fényviszonyok jelentik, hiszen a bőr máshogy néz ki homályos szobában, mint jól megvilágított helyiségben, így borzasztóan fontos a megfelelő megvilágítás. Ezért a következő megkötésekkel kell élni:

* Lehetőleg világos háttér legyen, de semmiképpen sem bőrhöz hasonló színű.
* Ne legyen túl erősen megvilágítva kéz.
* Ne legyen sötét a szoba (talán a LED-es webkamera feloldja ezt a megkötést)

## HSV szintéren alapuló szegmentálás

Az algoritmus első lépése egy RGB -> HSV szintér konverzió, amelyet a *cvCvtColor(RGBImg, HSVImg, CV\_BGR2HSV)* OpenCV függvény segítségével végzek el. Ezt követően egy intervallumos szegmentálást hajtok végre minden csatornára a *cvInRangeS (hsvImg, hsv\_min, hsv\_max, hsv\_mask)* függvény segítségével, ahol a *hsv\_min* és *hsv\_max* paraméterek vektorok, amelyek megadják a szegmentálási intervallumokat:



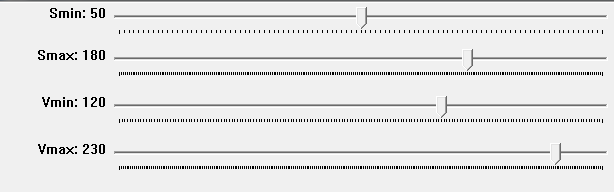
Az intervallumok a következő 3 kép alapján alakultak ki:



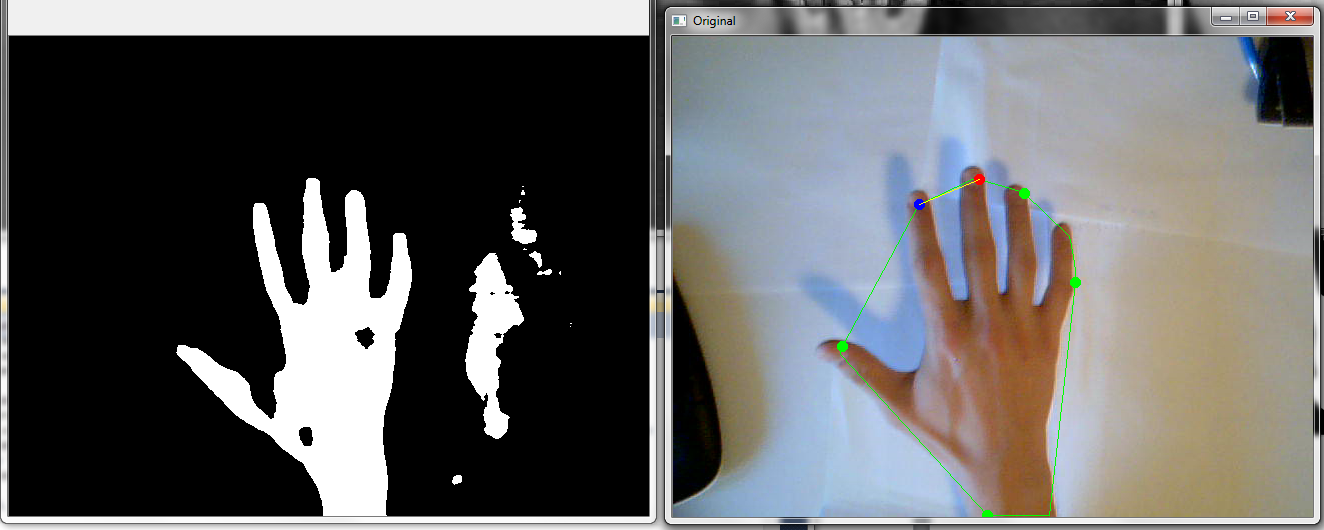
**H, S és V csatornák**

Jól látszik, hogy a H csatornán nagyon alacsony pixelértékeken található bőr, így akkor találtunk bőr-pixelt, ha 0 és 20 közötti a pixelérték. Az S csatornánál ez az [50, 180] intervallumra adódott, míg a V csatornán [120, 230] intervallumra.

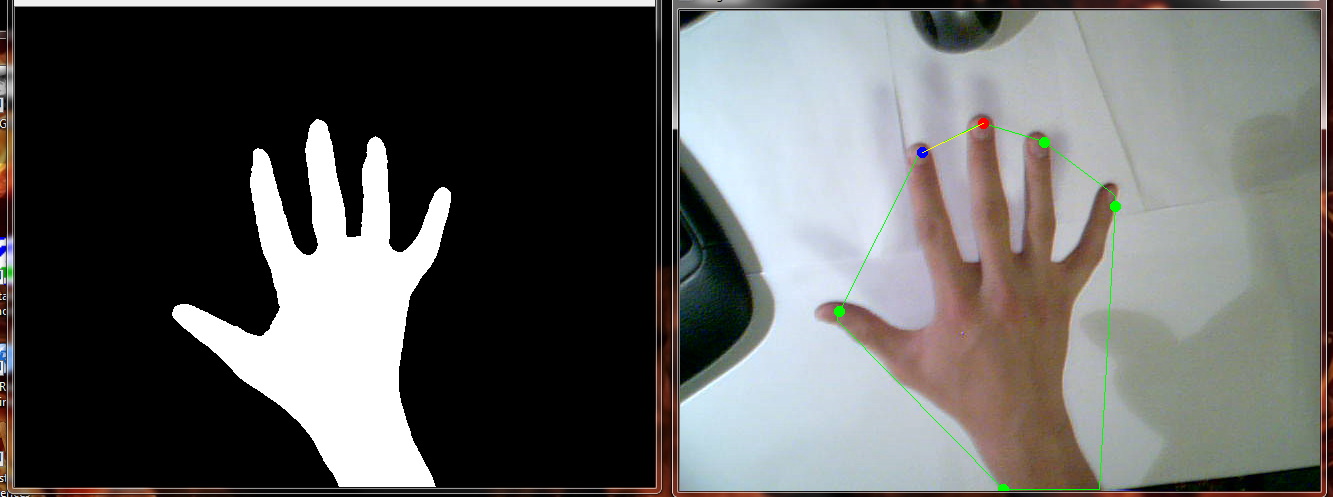
Természetesen nagyon komoly faktor a megvilágítás és az illető bőrárnyalata ezért a szegmentálás paraméterei egy csúszka segítségével változtathatóak az aktuális környezetnek megfelelően:



Utolsó lépésként pedig egy mediánszűrést hajtok végre 27-es ablakmérettel, hogy a fölösleg pontokat eltávolítsam. Ez a szegmentálási eljárás a következő eredményt hozta :



**Gyenge fényviszonyok mellett.**

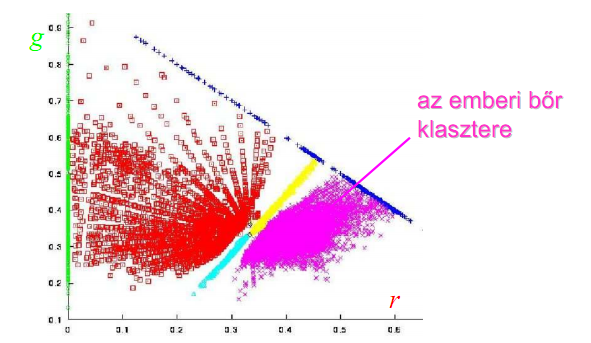


**Jó fényviszonyok mellett (mesterséges fény)**

## 2D normalizált színtéren alapuló szegmentálás

RGB -> 2D normalizált szintér konverziót hajtunk végre:

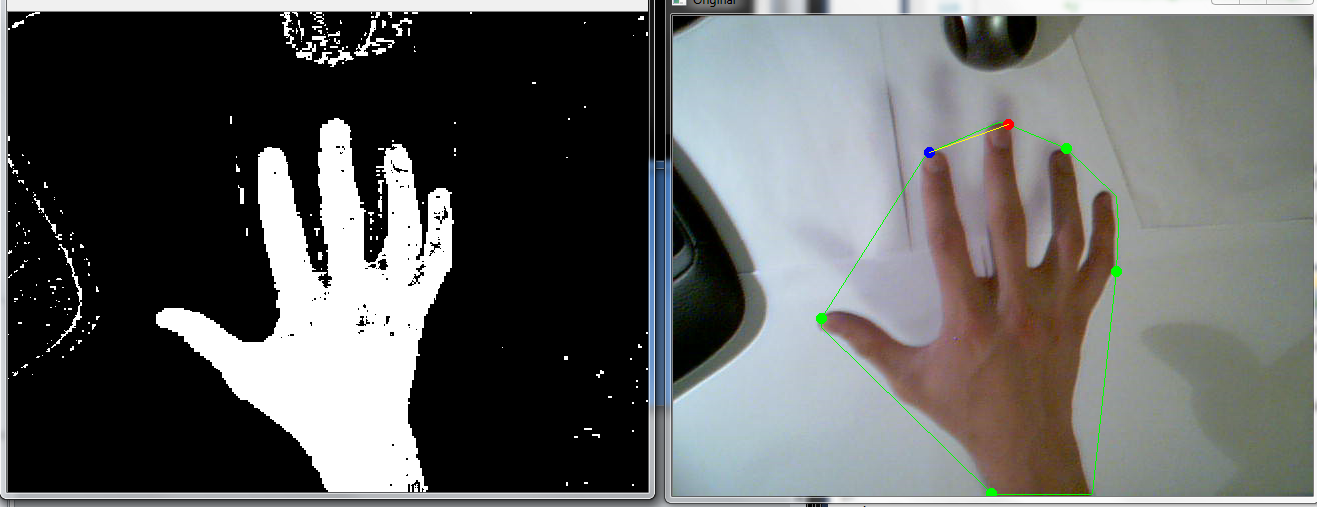
,



Az emberi bőr a [0.35, 0.55] g intervallumon és a [0.34, 0.55] r intervallumon helyezkedik el:

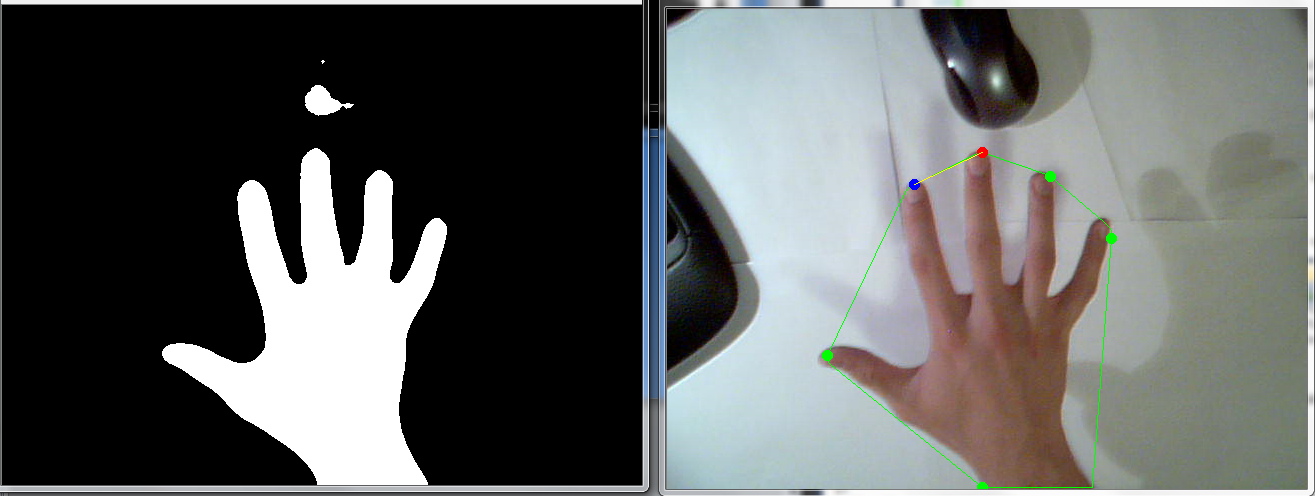


Miután a fenti képletet leprogramoztam a következő eredményt kaptam:



**RG szintérben történő szegmentálás (mesterséges fény mellett)**

Szemmel láthatóan sok olyan objektum maradt a képen, ami közel sem bőr. Egy 25-ös ablakméretű mediánszűrés után a következő eredményt kaptam:



**A szegmentált kép mediánszűrés után**

## Összegzés

A tesztek során az aktuális körülmények függvényében mindkét módszer szerepelt jól és rosszul is. Ezért arra az elhatározásra jutottam, hogy mindkét módszer belekerül a programba és a felhasználó döntése lesz, hogy melyiket használja. Az „r” gomb lenyomására a program a színteret vált (HSV az alapértelmezett).

Nagyon fontos, hogy az algoritmus megfelelő, jól elkülöníthető hátteret és megfelelő megvilágítást feltételez, ezek hiányában előfordulhat rendellenes működés.

**5. Kontúr, konvex burok, újhegyek**

**5.1. Kéz kontúrvonalának meghatározása**

Ha a szegmentált kép már adott, akkor következő lépésben megkeressük a kontúrokat. Ezt OpenCV-ben a cvFindContours függvénnyel tehetjük meg. A függvénynek többek között megadjuk a szegmentált képet, majd egy szekvenciában visszakapjuk a kontúrokat. Azért többet, mert a szegmentált képen a kezet szimbolizáló fehér területen kívül lehetnek még más területek is, mivel teljesen tökéletes szegmentálás a legtöbb esetben nem lehetséges. Ezen alakzatok területe tapasztalataink alapján azonban sokkal kisebb, mint a kéz területe. Ezt kihasználva adhatunk egy alsó korlátot a kontúrok által határolt területekre, ami alatt nem vesszük figyelembe az alakzathoz tartozó kotúrt. Így biztos, hogy csak a kéz kontúrvonalát kapjuk vissza. Terület számítására a cvContourArea eljárást alkalmaztuk.

|  |  |
| --- | --- |
| A kéz kontúrja | A kézen kívül más objektum is látható |

**5.2. Konvex burok számítása**

Konvex burok számításához nincs másra szükségünk, mint az előbb meghatározott, a kézhez tartozó kontúrvonalra. OpenCV-ben a cvConvexHull2 paranccsal tehetjük ezt meg, melynek egy paramétere van: a kontúrokat tartalmazó szekvencia.

|  |
| --- |
| Konvex burok |

A konvex burok gyakorlati hasznát a következő pontban tárgyaljuk.

**5.3. Ujjhegyek keresése**

Ujjhegyek keresésénél felhasználjuk a kéz kontúrvonalára illesztett konvex burkot. Olyan helyeken, mint például az ujjak hegyei, biztosak lehetünk abban, hogy a burok töréspontokat tartalmaz. Egy ujjhegyre azonban több töréspont is jut, ezért a pontokat „kiritkítjuk”. Jelen esetben ez úgy történik, hogy ha egy pont a következőtől adott távolságon belül van, akkor figyelmen kívül hagyjuk. Ezenkívül többféleképpen történhetne még a ritkítás, például klaszterezéssel.

A ritkítás után már minden ujjhegyet csak egy pont szimbolizál (természetesen nem kizárt, hogy ezeken a pontokon kívül lesznek még más pontok is). Ezeket a pontokat tároljuk (jelen esetben egy veremben), a legkisebb y koordinátájú pont lesz a kinyújtott mutatóujj ujjhegye, az őt megelőző pont pedig a hüvelykujj ujjhegye.

|  |  |
| --- | --- |
| Ritkítás után | Mutatóujj és hüvelykujj |

**6. Kurzor mozgása és kattintás**

**6.1. Mozgatás:**

*SetCursorPos()* segítségével, melynek meg kell adni paraméterként a képernyő egy pontját, az x és y koordinátákat.

Az x, y koordináták kiszámítása a következő:

1. szegmentált mutató ujj koordinátáinak megkeresése:

a konvex burok legkisebb y koordinátájú pontjának koordinátája lesz a mutató ujj egy pontja

1. a megtalált pont felskálázása a képernyőre:

x koordináta:

(Xmin\*képernyő szélesség)/kamera szélesség

y koordináta:

(Ymin\* képernyő magasság)/kamera magasság

(Xmin és Ymin a mutató ujj kordinátái)

1. Apró remegés kiküszöbölése:

x és y irányban, ha az elmozdulás nagysága egy megadott küszöb alatt van, akkor, változatlanul hagyjuk a koordinátákat.

Függvény meghívása után az egér kurzor a megadott koordinátára ugrik.

**6.2. Klikkelés:**

Az egér klikk eseményt a *mouse\_event()* függvény végzi el. Az első paraméterben adhatjuk meg, hogy milyen gombnyomás eseményt szeretnénk meghívni. Van külön gomb lenyomás és gomb felengedés.

Gomb események, melyekre szükségünk lesz a következőek:

MOUSEEVENTF\_LEFTDOWN - bal gomb lenyomva

MOUSEEVENTF\_LEFTUP - bal gomb felengedve

MOUSEEVENTF\_RIGHTDOWN - jobb gomb lenyomva

MOUSEEVENTF\_RIGHTUP - jobb gomb felengedve

Az eseményeket kombinálva megoldható az egérkezelés.

Egy szimpla bal gomb lenyomás művelet a MOUSEEVENTF\_LEFTDOWN esemény. Ezzel az eseménnyel objektumokat is megragadhatunk, elengedéshez pedig meg kell hívni a MOUSEEVENTF\_LEFTUP-t.

A bal klikk a MOUSEEVENTF\_LEFTDOWN és a MOUSEEVENTF\_LEFTUP egymás utáni meghívása. A dupla kattintás e két esemény egymás utáni meghívásának kétszeri megismétlése egy adott időn belül. Ezt az adott operációs rendszer szabja meg.

Jobb klikk elvégzése hasonlóan a bal klikkhez történik.

Bal gomb esemény megvalósításai:

Az esemény bekövetkezése a hüvelykujj kinyújtásával szabályozható. A mutató ujj és a hüvelykujj konvex burok béli koordinátái ismertek, ezért a két pont között számolhatunk távolságot.(). A távolság alapján megadunk egy küszöböt, aminél nagyobb távolság érték gomb lenyomásnak, azaz egy MOUSEEVENTF\_LEFTDOWN-nek fog megfelelni. A hüvelykujj visszazárása, tehát a távolság érték küszöbnél kisebb értéke meg fog hívni egy MOUSEEVENTF\_LEFTUP eseményt. Természetesen nem akarjuk, hogy az esemény a küszöb alatt állandóan bekövetkezzen, ezért egy boolean értékkel figyeljük, hogy volt-e előtte gomb lenyomás vagy sem.



Gombnyomás közben a kéz kicsit elmozdul, ezért, ezt kompenzálandó, ha lenyomjuk a bal gombot 1 másodpercre letiltjuk az egér elmozdulását, timer segítségével. Ez főleg a dupla kattintáskor segít sokat.

**6.3. Hiányosságok:**

* az egérkurzor mozgatásának pontossága
  + kézremegés
  + szegmentálásból adódó hibák
* a webkamera felbontás🡪 képernyő felbontás felskálázás során a webkamera képe nem használható ki teljes terjedelmében
  + konvex burok eltűnik, ha nincs megadott nagyságú kéz terület a képen
  + előző pont miatt, csak a kamera felső része használható pozícionálásra
  + a kép szélein pontatlan a burok illesztés
* kattintás során a kéz bemozdul
* jobb klikk kivitelezés

**Hivatkozások**

1. G. Bradski, A. Kaehler. Learning OpenCV: Computer Vision with the OpenCV Library (The OpenCV book)
2. S. Malik, J. Laszlo. Visual Touchpad: A Two-Handed Gestural Input Device. *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces (2004), pp. 289-296.*
3. Z. Zhang, Y. Wu, Y. Shang. Visual Panel: Virtual Mouse, Keyboard and 3D Controller with an Ordinary Piece of Paper. *Workshop on Perceptive User Interfaces (PUI 2001), Nov. 15-16, 2001. Orlando, Florida.*
4. V. I. Pavlovic, R. Sharma, T. S. Huang. Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence (1997), pp. 677-695*
5. Paláhyi Kálmán: Képfeldolgozás haladóknak 2. előadás
6. Hand detection: <http://www.andol.info/hci/830.htm>
7. Számos YouTube-on elérhető videó, kulcsszavak: virtual mouse, hand detection