Gesztusvezérelt egér szín- és alakzatdetektálással

Bézi Patrik

2021. 09. 16.

Tartalomjegyzék

1. Bevezetés

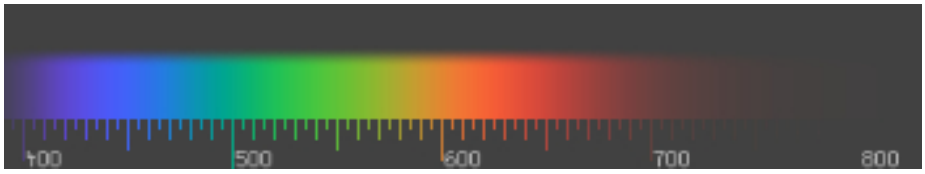
A számítógépes programozás és a képfeldolgozás olyan szintre fejlődött az elmúlt évek során, melynek során könnyen fejleszthetőek olyan szkriptek, amelyek a gépek mindennapi felhasználását elősegítik.
A dolgozatom során egy olyan szoftvert fejlesztek, amely a bemeneti képet szín és egyszerű alakzatszűrő segítségével képes feldolgozni úgy, hogy azzal az egér mozgását lehessen befolyásolni. Az ötlet egy egyszerű elven alapszik, miszerint legyen bármilyen tárgy alkalmas arra, hogy a gép egere felett könnyedén át lehessen venni az irányítást, legyen szó prezentálásról, szórakozásról, vagy egyszerű kényelmi szempontokról.
A szoftver legfőbb követelménye a valós idejű képfeldolgozás, vagyis hogy az adott tárgy és a számítógép egerének mozgása a lehető legnagyobb szinkronban történjenek. Ennek a sebességnek az eléréséhez fontos kiválasztani a megfelelő felbontást és a megfelelő műveleteket, amelyek nem okoznak különösebb gondot a számítási igényükkel még egy gyengébb teljesítményű gépnek sem.
Legelőször ki kell választani azt az eszközt, amellyel az irányítást el szeretnénk végezni. Ha ez megvan, el kell érni, hogy a kamera felismerje azt és lekövesse a mozgását.
A felismerés pontosságának fejlesztéséhez maszkolási eljárásokat is alkalmazni fogok, amelyek működését össze fogom hasonlítani a dolgozat készítése során.
A szoftver fejlesztése **Python 3** nyelven történt **Ubuntu 20.04** környezetben. A megoldáshoz kizárólag az **OpenCV** keretrendszer 4.5.2-es verzióját, illetve a **numpy** csomagot használtam fel.

2. Színmodellek

A háttértudás leírásához az alábbi irodalmakat használtam fel: [1] [2] [3] [4] [5]

2.1. Színek

Az elektromágneses hullámok közül nem mindegyiket érzékeljük szemünkkel, pedig természetük azonos. Az átlagos emberi szem kb. a $3,810^{-7}m$ és $7,610^{-7}m$, azaz a 380 nm és 760 nm közötti hullámhossztartomány érzékelésére képes: az ebbe a tartományba eső elektromágneses sugárzást nevezzük fénynek. Ez a látható tartomány természetesen emberről emberre változik. A különböző hullámhosszaknak a tiszta színek felelnek meg, amelyet az 1. ábra szemléltet.

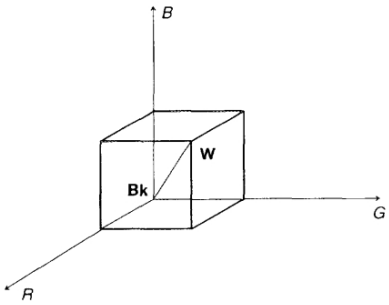


1. ábra. A látható spektrum és a fény hullámhossza nanométerben.[1]

A színek jelentősége hatalmas az egész informatikában. A belőlük elkészített színmodellek különböző feladatok elvégzésére tesz alkalmassá. Egy rajzolóprogram például nem ugyanazt a modellt használja, mint a nyomtatók. A következő két alfejetben bemutatom az RGB-, illetve a feladat megoldására használt HSV modelleket.

2.2. RGB modell

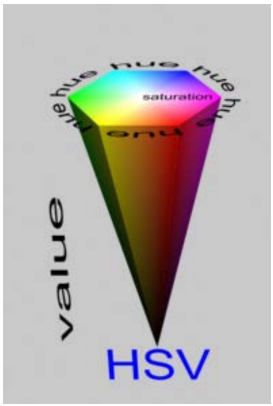
Színmodellnek nevezünk egy háromdimenziós testet, amely a 3 koordináta partikuláris leírásának segítségével leír egy színt.
Az RGB-re épülő talán a legismertebb ilyen modell, amely a vörös, zöld és kék színek keverésével segít meghatározni a színeket. A skála [0-255] intervallumon belül terjed mindhárom szín esetében. Minél nagyobbak ezek az értékek, az eredmény annál világosabb és minél jobban közelít a 0-hoz, annál sötétebb lesz.



2. ábra. Az RGB színkocka.[3]

2.3. A HSV modell

A HSV a hue(árnyalat), saturation(telítettség) és a value(fényerő) szavak rövidítéseiből tevődik össze. Ezek kombinációja határozza meg a színeket egy adott képen.
Egy adott színt sokkal könnyebb leírni ezen kódolás segítségével. Az elsődleges elképzelés a feladat megoldásáról, hogy minden értéknek legyen egy alsó és egy felső határa, amelyeket közötti színeket figyelembe fogok venni. Ha például bizonyos fényviszonyok között én egy sárga toll mozgását szeretném lekövetni, akkor ahogy mozgatom, a fent említett értékek folyamatosan változhatnak. Ebből kifolyólag célszerű megadni olyan határértékeket, amelyek között a sárga a még folyamatosan változó körülmények esetén is felismerhető lesz.



3. ábra. A HSV modell.[5]

Fogalmak

- Árnyalat: ez adja meg tulajdonképpen a fény hullámhosszát a spektrumban. Az OpenCV-ben ez az érték [0-179] tartományban terjed

- Telítettség: A szín tisztaságát adja meg. Ez annyit jelent, hogy az adott színárnyalat telítettebb formában élénkebb lesz, míg semlegesebb formában egyre jobban közelít a szürkeárnyalathoz. Ez az érték OpenCV-ben [0-255] tartományban terjed.
- Fényerő: Egy szín fényerejét adja meg. Ez az érték OpenCV-ben [0-255] tartományban terjed.

A fent említett három tulajdonság segítségével kiszűrhetővé válik az a spektrum, amelyet figyelembe szeretnénk venni a szoftver futása alatt. A legelső lépés ehhez, hogy az OpenCV beépített függvényét felhasználva az alap BGR kódolású képet át kell konvertálni HSV kódolásra. Ennek matematikai levezetése az OpenCV dokumentációja alapján a következőképpen alakul:

$$V \leftarrow max(R,G,B)$$

$$S \leftarrow \begin{cases} \frac{V-max(R,G,B)}{V} & V \neq 0 \\ 0 \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/(V-min(R,G,B)) & V = R \\ 120+60(B-R)/(V-min(R,G,B)) & V = G \\ 240+60(R-G)/(V-min(R,G,B)) & V = B \\ 0 & R = G = B \end{cases}$$

A HSV-s kép segítségével készíthető egy maszk, amely kezdetben egy a bemenet méretével megegyező nagyságú kép. Ennek küszöbértékeit határozza meg a 3-3 (minimális és maximális) *H,S,V* érték. Ezt felhasználva figyelmen kívül lehet hagyni a szükségtelen színkódú pixeleket.

3. Maszkolási eljárások

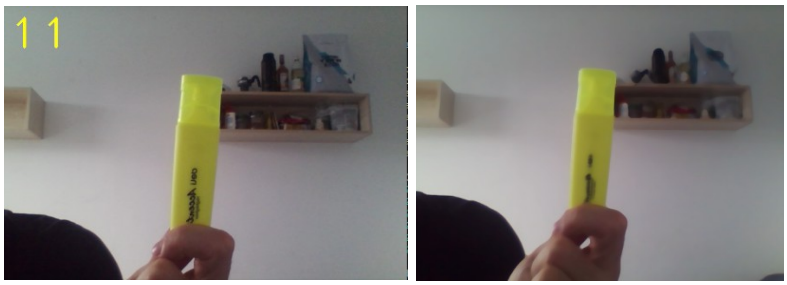
A következő fejezetben bemutatom a maszkolási eljárásokat, amelyeket alkalmaztam a szoftveremben. Ennek elkészítéséhez az alábbi forrásokat használtam fel:[4] [5] [6] [7]

3.1. Előfeldolgozás

A könnyebb szűrés és az esetlegesen felmerülő fals pozitív eredmények kiküszöbölése érdekében a bemeneti képet egy homályosító eljárásnak vetettem alá. Ehhez mediánszűrőt alkalmaztam, mely a következő módon működik:
Legyen *I(x,y)* a bemeneti kép, *J(x,y)* a rá vonatkozó értékkészlet transzformáció, *S(x,y)* pedig a vizsgált képrészlet. Ezek alapján legyen a vizsgált méretű képrészlet alapján képzett számsorozat mediánja az új intenzitás érték:

$$J(x,y) = medI(i,j)|I(i,j) \in S(x,y)$$

A medián szűrővel az a cél, hogy az összefüggő színeket egybemosva csökkenjen a háttérzaj mértéke a HSV határértékek beállítása alatt is már.



(a) (b)

4. ábra. A kép átalakítása a medián szűrő alkalmazásával.

3.2. Kontúrok

A kontúrok fontos szerepet töltenek be a projektem során. Egyrész segítenek meghatározni egy adott tárgy alkazatát, másfelől a kontúr által befoglalt terület nagyságát befolyásolva megadhatóak olyan paraméterek, amelyek segítenek a háttérben lévő zajok szűrésében.
Kontúrnak nevezzük azokat a helyeket, ahol az intenzitás értéke nagy mértékben hirtelen megváltozik. Ez a dolgozatomban segít meghatározni az objektum határvonalát.
Éldetektáláskor nincs nagy jelentősége a színeknek, úgyhogy a folyamathoz nyugodtan lehet használni a már korábban elkészített maszkot, amelynek küszöbértékei egyelőre csak a megfelelő HSV értékekhatárok alapján kerülnek meghatározásra.
Az élek felismerésére az Opencv *findContours* függvényét használtam fel a dolgozat során.

3.3. A befoglalt terület

A szoftver végigmegy valamennyi a maszkon fellelhető "folton" és megkeresi azok kontúrjait. Ezzel meghatározhatóvá válik az általuk befoglalt terület nagysága. Ennek jelentősége, hogy a képen egy bizonyos területnél kevesebbet, illetve többet foglaló testek ne jelenjenek meg zajként. A későbbi, középponti elmozdulással kapcsolatos fejezet egyik ábráján ennek a szűrőnek a működése jól látható.
A terület nálam a kísérletezés alapján *]3000,20000[* intervellumba került pixelnagyságban meghatározva.

3.4. Alakzatok

A fals pozitív értékek csökkentésének egy másik módjaként megkíséreltem figyelembevenni az vezérlőnek használt tárgy alakját. A kontúrok meghatározásával ennek megfigyelésére lehetőség is nyílik.

A detektált eszköz alakzatát belső pontjai alapján is le lehet írni. Ezt momentumok teszik lehetővé. Ezekből számos olyan mérték állítható elő, melyek kielégítően jellemzik az objektumot, és az objektumon végrehajtott lineáris transzformációkkal szemben invariánsak.
Az f(x, y) 2-dimenziós folytonos függvény (p, q)-adik momentuma a valószínűség elméletéből ismert definíció alapján:

$$m_{pq} = \int_{-\infty}^{\infty} x^p y^q I(x,y) dxdy$$

A centrális momentum meghatározása:

$$M_{pq} = sum_x \sum_y (x - \overline{x})^p (y - \overline{y})^q I(x,y)$$

ahol $\overline{x} = \frac{m_1 0}{m_0 0}$ és $\overline{y} = \frac{m_0 1}{m_0 0}$
Az OpenCV *moments()* nevű függvényét használtam fel a megtalált eszközöm középpontjának meghatározásához, ez fogja meghatározni az egér képernyőn lévő pozícióját, amely a fenti képleteket használja fel ehhez.

3.5. Maszkolás alkalmazása

Miután a megfelelő szűrési eljárásoknak alávetettem a szoftvert, alkalmazhatóvá válik a maszkolás. Látható, hogy a zajos képből hogyan lett kivehető egy egyszerű folt, amelynek a tulajdonságait figyelve kijelenthető, hogy az a keresett objektum. A maszkolás egy képösszeadás segítségével történik a következő módon:

$$J(x,y) = M(x,y)I_1 + (1 - M(x,y))I_2(x,y)$$

Ezt a műveletet az OpenCV *bitwise_and* függvényével hajtom végre, amelynek átadom az előfeldolgozott képet. A megfelelő pozíciók, - mivel a feldolgozott és az eredeti nyers képi bemenet egyforma méretűek - könnyen átadhatóak a másik ablaknak, ahol bekeretezésre kerül a használt eszköz. Átfogó ábra az eredményről a következő fejezetben látható.

4. Az egér vezérlése

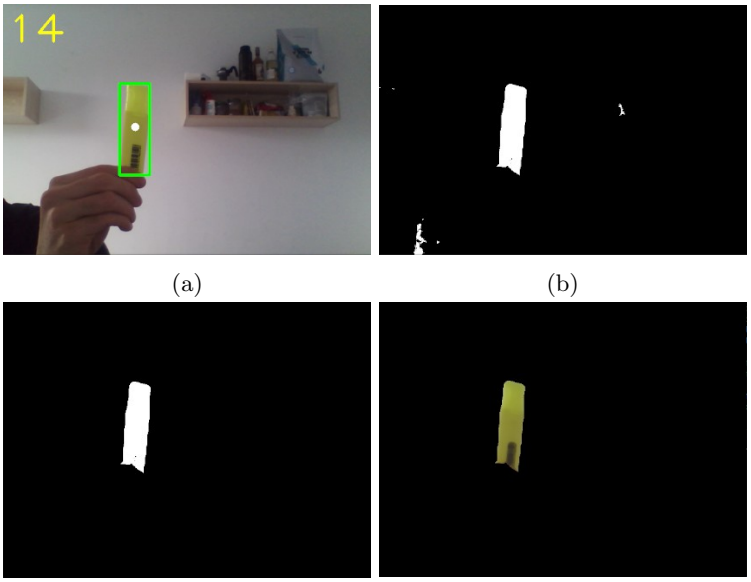
A következő fejezetben leírom az egér vezérlésével kapcsolatos megoldásaimat.

4.1. Követés középponti elmozdulás alapján

Ha megtaláltam az adott objektumot, amivel a feladatot végre szeretném hajtani, akkor annak középpontját veszem és a továbbiakban ennek a pontnak az elmozdulása fogja meghatározni az egér mozgását.

4.2. Az egér mozgása

A felismert eszköz középpontjának képi koordinátáit arányosítottam a képernyőm felbontásához. Ez egyszerű osztási műveletet követelt meg: Legyen a monitor felbontása wxh, ahol w a szélesség, h pedig a magasság pixelekben. Innen kiszámolható egy eltolási szorzó mind magasságra, mind szélességre a következő módon: $tX = w/cW$, illetve $tY = w/cH$, ahol cW és cH a bemeneti képet biztosító kamera szélessége és magassága.
A meghatározott centrális momentumok segítségével kiszámolható az eszköz középpontja, amely legyen a *C(cX, cY)* pont.
Ezek után meghatározható az egér pozíciója, ha vesszük, hogy $mX = cX \cdot tX$ és $mY = cY \cdot tY$, ahol mX és mY az egér koordinátái. Így az egér pozíciója az *M(mX,mY)* pont, amely a mozgás során folyamatosan változik, ezt pedig *autopy* csomagnak átadva könnyen kivitelezhetővé válik az egér mozgatása.



(a) (b)

(c) (d)

5. ábra. Itt látható a szoftver működése. Az (a) az objektum középpontját, a (b) a HSV szűrő segítségével létrehozott maszkot, a (c) a kontúr által befoglalt területi szűrőt, a (d) pedig a szoftver által látható képet ábrázolja.

5. Kiértékelés

5.1. Összegzés

A dolgozatban elkészített szoftver alkalmas rá, hogy az egér vezérlését át lehessen venni akármilyen tárggyal, amely megfelel a felhasználó által megadott

paramétereknek. Ezek a paraméterek a *HSV* értékhatárok, illetve a képernyő-, valamint a kamera felbontása. Az alakzatok meghatározásával lehetőség nyílik a középpont meghatározására, amelyet lekövetve az egér mozgathatóvá válik.

5.2. Korlátok

A fejlesztés során az alábbi korlátozó tényezők merültek fel:

- fontos a környezettől jól elkülönülő színű eszközt választani a feladatra
- az eszköz által határolt területet a kamerától való távolság könnyen befolyásolhatja
- Az FPS szám korlátossága darabossá teheti az egér mozgását
- az egér mozgása még viszonylag könnyen megoldható, egy egyszerű kattintás viszont már nehezen kivitelezhető

5.3. Lehetőségek

A feladat pontosabb megoldásához és a szoftver könnyebb felhasználásához érdemes lehet neurális hálókon alapuló megoldásokat alkalmazni. Ilyen lehet akár egy kézfelismerő rendszer fejlesztése, amelynek különböző gesztusait meg lehet figyelni. Például a *Google* által fejlesztett *mediapipe* is ilyen, illetve vannak olyan *state of the art* jellegű megoldások, amelyek segítségével már a felhasználó is betaníthat különböző neurális hálókat.

6. A program telepítése és működtetése

6.1. Futtatáshoz szükséges szoftveres követelmények

Amire szükség van a program futtatásához:

- Python 3.8.10
- OpenCV 4.5.2 (én a contrib verziót használtam a projekt elkészítése során)
- numpy csomag
- autopy csomag
- screeninfo csomag az OpenCV ablakainak elhelyezéséhez
- egy kamera, amiről a számítógép tudja olvasni a képi információkat

A szoftvert Ubuntu 20.04-es környezetben fejlesztettem, amelyen már alapból megtalálható a Python megfelelő verziója. A hozzá szükséges csomagokat a pip csomagkezelő segítségével lehet letölteni az alábbi parancsokkal:

```
python3 -m pip install numpy
python3 -m pip install opencv-contrib-python
python3 -m pip install autopy
python3 -m pip install screeninfo
```

Ezek után a program futtathatóvá válik a következő paranccsal terminálon keresztül:

```
python3 control.py
```

6.2. A program futtatása

A szoftvert alapvetően kétféle módban lehet futtatni. Az első a beállítási mód, amely lehetővé teszi a felhasználó számára, hogy a HSV kódolást saját magának beállítsa, lehetővé téve számára, hogy saját eszközt is kiválaszthasson a vezérlésre. A másik mód az, amelyikben a rendszer már a megfelelő HSV-s adatokat beolvasva és felhasználva élesben működik.

Irodalomjegyzék

[1] Berta Miklós, Farzan Ruszlán, Giczi Ferenc, Horváth András - Fizika mérnököknek, 298-300 oldal, 2006

[2] Richard Szeliski – Computer Vision: Algorithms and Applications, p90, 2010

[3] Levkowitz, H. Herman, G.T. - CVGIP: Graphical Models and Image Processing Volume 55, Issue 4, July 1993, Pages 271-285

[4] OpenCV Documentation - https://docs.opencv.org/4.5.2/index.html

[5] Werner Purgathofer - Einführung in Visual Computing – Kapitel 4, TU Wien, 2017

[6] Hollósi János - Gépi látás előadások, Széchenyi István Egyetem, Győr, 2021

[7] Fazekas Attila, Kormos János - Digitális képfeldolgozás matematikai alapjai, mobiDIÁK könyvtár, Debreceni Egyetem, 2004