

# TDK-dolgozat

Halász Péter

# Látássérülteknek szóló, mobiltelefonos asszisztív technológia egyéni színfölismerés és navigációs feladatokhoz

2024. 11. 19.

## **Intézmény:**

Pázmány Péter Katolikus Egyetem  
Információs Technológiai és Bionikai Kar

## **Szerző:**

Halász Péter  
mérnökinformatikus BSc, 9. félév

## **Témavezetők:**

Dr. Gelencsér-Horváth Anna, egyetemi adjunktus  
Dr. Karacs Kristóf, egyetemi docens

## **Kivonat**

Ez a dolgozat a LetSee vakok és látássérülteknek szóló asszisztív, mobiltelefonos applikációnak, az általam implementált ösvénykövető és egyéni színfölismerő funkcióit mutatja be. Ezen megoldások a Cybathlon 2024 verseny látássérülteknek szóló kategóriájára készültek, azzal a szándékkal, hogy később, a mindennapi fölhasználásra is ki legyenek fejlesztve. A színfölismerés megoldása, a kamera RGB képét használja bemenetként, amely alapján képes referencia színekkel összehasonlítást végezni, a külső fényhatások befolyását minimalizálva. Ezt a program egy fizikai kellekkel, annak képen való megkeresésével, valamint a vizsgált pontok klaszterezésével éri el. Az ösvénykövetés feladat megoldása, a mobiltelefon LIDAR és iránytű szenzoraira épül, és az egyik legkorszerűbb szegmentációs modell kimenetét használja föl, az út navigálásához szükséges visszajelzések előállítására. Az itt tárgyalt megoldásokat a LetSee applikáció prototípusába integráltuk, és egy vak fölhasználó segítségével, többször is teszteltük őket.

# Tartalomjegyzék

|                                                         |           |
|---------------------------------------------------------|-----------|
| <b>1. Bevezetés</b>                                     | <b>2</b>  |
| <b>2. Előismeretek</b>                                  | <b>3</b>  |
| 2.1. Színterek és színtávolság . . . . .                | 3         |
| 2.1.1. RGB . . . . .                                    | 3         |
| 2.1.2. CIE 1976 $L^*a^*b^*$ . . . . .                   | 3         |
| 2.1.3. Színtávolság . . . . .                           | 3         |
| 2.2. Fejlesztési környezet . . . . .                    | 4         |
| 2.3. Hierarchikus klaszterezés . . . . .                | 5         |
| 2.4. Gépi látás és OpenCV . . . . .                     | 5         |
| 2.5. ArUco jelölések . . . . .                          | 6         |
| 2.6. Homográfia . . . . .                               | 7         |
| 2.7. Szegmentáció . . . . .                             | 7         |
| <b>3. Színfölismerés feladata</b>                       | <b>8</b>  |
| 3.1. Fényhatások kezelése . . . . .                     | 8         |
| 3.2. Színkártya fölismerése . . . . .                   | 10        |
| 3.2.1. Színkártyák paraméterei . . . . .                | 12        |
| 3.2.2. A tesztel színkártyák . . . . .                  | 14        |
| 3.3. Színek összehasonlítása . . . . .                  | 17        |
| 3.4. Vizsgált szín klaszterezése . . . . .              | 18        |
| 3.5. Mérési eredmények . . . . .                        | 20        |
| <b>4. Ösvénykövetés feladata</b>                        | <b>21</b> |
| 4.1. Visszajelzés az eltolódásról . . . . .             | 22        |
| 4.2. Visszajelzés a kanyarodásról . . . . .             | 24        |
| 4.3. Visszajelzés az út végéről . . . . .               | 25        |
| 4.4. Visszajelzés az útelemek csatlakozásáról . . . . . | 27        |
| 4.5. Visszajelzés a megállásról . . . . .               | 27        |
| <b>5. Összefoglalás</b>                                 | <b>30</b> |

# 1. Bevezetés

A szem az ember egyik legfontosabb érzékszerve, és rengeteg információt a látásunkon keresztül fogadunk be. A látássérültek és vakok számára, a gépilátás területén elérhető technológiák nagy segítséget tudnak nyújtani. Azonban több olyan probléma is van, amelyre még nem készült robosztus megoldás, vagy azoknak a mindennapi használata illetve tömeges adoptálása, a jelenlegi álláspontok szerint nem lehetséges, például a szükséges számításai erő vagy az előállítási költségek miatt.

A LetSee applikáció egy, az EyeRider csapata által fejlesztett, vakoknak és látássérülteknek szóló asszisztív, mobiltelefonos alkalmazás. A jelenlegi publikus verziója képes, a pénznemek és bankkártyák fölismerésére, valamint a fény érzékelésre. A fejlesztésben fő szempont az elérhetőség, ugyanis egy mobiltelefonos applikáció széles körben elérhető, mert a legtöbbször már rendelkeznek a hozzá szükséges készülékkel, illetve annak beszerzése sem nehéz. A csapat jelentkezett a Cybathlon 2024. évében megrendezett versenyére, amelyben először indult látássérülteknek szóló kategória is. Ebben a kategóriában egy vak (vagy szemfedővel ellátott látássérült) versenyzőnek, a továbbiakban Pilótának, kell végig mennie egy 10 feladattal ellátott pályán, 8 perces időkorlát mellett. A remény az volt, hogy a verseny alkalmat ad az applikáció bemutatására, és annak további fejlesztésére.

A színfölismerésnél a cél az ismert referencia szín halmazból egy vizsgált színhez legközelebb álló szín azonosítása. A feladat kiindulási pontja egy már meglévő, Python nyelven írott prototípus volt, amely képes volt néhány szín fölismerésére. Nekem ennek a megoldásnak a kibővítése volt a feladatom, a Cybathlon színeire, majd a megoldás C++ nyelvre való átírása.

Az ösvénykövetés feladatában a Pilóta egy ösvényt navigál, csupán a telefon visszajelzéseit használva, anélkül, hogy arról lelépne. Itt egy alacsonyabb programrésztől kapott maszkot kell elemezni, és a visszajelzéseket abból előállítani. A színfölismeréssel szemben itt a Python prototípus már késznek volt titulálva, azonban az átírt C++ verzióba új funkciók is kerültek bele.

Mindkettő feladat esetében, a LetSee applikációban is futó háttér logikáját implementáltam, azonban annak integrációját már más végezte. A logikának azon részeit, amelyet mégis más implementált, a releváns fejezetekben fogom megemlíteni.

## 2. Előismeretek

### 2.1. Színterek és színtávolság

#### 2.1.1. RGB

A Színeket a számítástechnikában leggyakrabban az RGB (*Red, Green, Blue*) színértékekkel jellemezzük, és az ezen színértékek által leírható összes szín halmazát nevezzük az RGB színtérnek [1]. Itt a színérték egy vektort jelent, tipikusan 3 dimenzóval, amely egyes koordinátáit színcsatornáknak nevezzük. Ez a színtér azért elterjedt, mert a monitorok működését közvetlenül reprezentálja, ahol Vörös (*Red*), Zöld (*Green*) és Kék (*Blue*) fények összege képi, a megjeleníteni kívánt színt. Ezen három fény erőssége az egyes színcsatornákon van eltárolva. Amennyiben mind három színérték teljesen telített, fehér fényt kapunk, amennyiben teljesen telítetlenek, fekete szín ábrázolható. Az egyes színcsatornákon felvehető értékeket ennek megfelelően vehetjük a  $[0,1]$  valós intervallumként. Ennek megvalósításához, reprezentálnunk kell valahogy a fizikai memóriában ezt a  $[0,1]$  intervallumot. Egy specifikus reprezentációval együtt megadva, a színterekről színinformátumként beszélünk. Az RGB esetében a legelterjedtebb színinformátumokban a  $[0,1]$  intervallumot a 0 és 255 közötti egész számokkal ábrázoljuk, amely színcsatornánként 1 bájtnyi adatot jelent. Ezt nevezzük 8 bites színcsatornáknak. Habár könnyű a használata, hiszen könyvtárak által szinte univerzálisan támogatott, bizonyos feladatok megoldására jobb más színteret alkalmazni.

#### 2.1.2. CIE 1976 $L^*a^*b^*$

A CIE 1976  $L^*a^*b^*$  [2] egy, az International Commission on Illumination (CIE) által kifejlesztett színtér, ahol a színek közötti számbeli eltérés, az emberi színérzékelésben is hasonló eltérést próbál meg reprezentálni. Ez a tulajdonsága ideálissá teszi ezt a színteret arra, hogy a színek közötti hasonlóságot vizsgáljuk vele. A CIE 1976  $L^*a^*b^*$  színteret 3 csatorna írja le, egy megvilágítási csatorna  $L$ , ami a  $[0,100]$  valós intervallum értékeit veheti föl, illetve az  $a$  és  $b$  színcsatornák, amelyek a  $[-127,127]$  valós intervallum értékeit vehetik föl [3].

#### 2.1.3. Színtávolság

Színtávolság [4] alatt a színek közötti különbséget értjük, ahol megpróbáljuk számokkal ábrázolni azt, hogy két szín mennyire eltérő. A Színek közötti távolságot legegyszerűbb módon Euklideszi távolság [5] alapján lehet megmérni, ahol a színcsatornák vektor koordinátáknak felelnek meg, és az így alkotott vektorok között a Pitagorasz-tétel segítségével számítjuk, az azok által reprezentált színek közötti "térbeli" távolságot.

Az RGB színtérnél hasznosabb a CIE 1976  $L^*a^*b^*$  színtérben számolni a távolságot, amely úgy is alkottak meg, hogy maga az Euklideszi távolság is egy használható színtávolsági metrika legyen benne. A későbbiekben a CIE ezen metódust a CIEDE2000 [6] színtávolság metrikával múlta fölül. Ez a metrika a már korábbi CIE színtávolság szabványokra épül, és a formulája az 1. egyenleten látható. Ebben a formulában a  $\Delta L'$  a két szín közötti fény különbség,  $\Delta C'$  a telítettségi különbség,  $\Delta H'$  a színárnyalat különbség,  $k_L, k_C$  és  $k_H$  súlytényezők,  $S_L, S_C$  és  $S_H$  kompenzációs tényezők,  $R_T$  pedig egy szűrő tag, amely a kb.  $275^\circ$  kékes árnyalatain kívül elhanyagolhatóan kicsi.

$$\Delta E_{00}(L_1^*, a_1^*, b_1^*, L_2^*, a_2^*, b_2^*) = \sqrt{\left(\frac{\Delta L'}{k_L S_L}\right)^2 + \left(\frac{\Delta C'}{k_C S_C}\right)^2 + \left(\frac{\Delta H'}{k_H S_H}\right)^2 + R_T \left(\frac{\Delta C'}{k_C S_C}\right) \left(\frac{\Delta H'}{k_H S_H}\right)} \quad (1)$$

## 2.2. Fejlesztési környezet

A fejlesztések során, a prototípusok elkészítéséhez a Python szkriptnyelvet használtuk. Ez az egyszerű szintaxisa, hatékony tesztelhetősége, vizualizációs lehetőségei és a gépi tanulás és gépi látás területén elérhető könyvtárak miatt volt előnyös. A prototípust a telefon szenzoraitól valós időben közvetített adatok segítségével teszteltük, és a Pilóta visszajelzései alapján, több iterációban finomítottuk, és módosítottuk. A Pilótával tesztelt prototípusok alapján készítettünk C++ implementációt telefonra, amelyet a LetSee applikáció iPhone implementációjába, a Swift keretrendszerbe integráltunk. Ez fontos ahhoz, hogy az alkalmazás a telefonon, külső eszköz nélkül is működjön önmagában, és így könnyen elérhető legyen. A LetSee alkalmazás a telefon több szerzőjét használja. Ezek egyike a hátsólapos kamera, amely RGB képeket biztosít. A kameraképen kívül, használunk továbbá LiDAR szenzorból kapott mélységi adatot is. A LiDAR szenzor fény sugarakat használ, a környezetbeli távolságméréshez, ahol a fény konstans sebességét ismerve, a visszaverődés idejéből lemérhető, a sugár által megtett távolság [7]. Ez a szenzor, ezt a mérést rengeteg irányba elvégzi, és az így kapott eredményekből állít elő egy képet. Ez a mélységkép egy szűrkeskálás kép formájában van átadva a számításokat végző programrészeknek, a feladattól függően, valamilyen távolságra normalizálva. Továbbá az ösvénykövetés feladatában, használtuk a készülék iránytű szenzorát is, annak érdekében, hogy a Pilótának könnyebb legyen a pálya irányát megtartania.

### 2.3. Hierarchikus klaszterezés

A klaszterezés egy felügyelet nélküli gépi tanulási módszer, az adathalmazok elemeinek csoportosítására. A hierarchikus klaszterezés esetében, az egyes csoportokból egy hasonlósági metrika alapján képzünk új csoportokat iteratívan, ezzel egy hierarchia fát kialakítva [8]. A hierarchikus klaszterezésből megkülönböztethetünk fentről-lefele és lentől-felfele épülő implementációkat. A fentről-lefele épülő hierarchikus klaszterezésnél egy, az összes elemet tartalmazó klaszterből indulunk ki, amelyet folyamatosan szétbontjuk. A lentől-felfele épülő hierarchikus klaszterezésnél, az összes elemnek külön klasztert adunk, és azok iteratív összevonásával képzünk egy univerzális klasztert. A láncolási módszer adja meg azt a feltételt, amely alapján a klasztereket egybevonjuk vagy szétbontjuk. A legegyszerűbb ilyen feltétel a maximum és minimum láncolás, ahol a legkisebb vagy legnagyobb távolsággal rendelkező klaszter párost vonjuk egybe. A színfölismerés feladatában egy minimum láncolású, lentől-fölfelé épülő klaszterezési módszer lesz alkalmazva, azonban a klaszterek egybevonását csak egyszer végezzük el, és a hierarchikus fát nem mentjük el. Továbbá, a legkisebb távolságot mohó keresés alapján kerestük meg, ahol mindig a jelenleg elérhető, legjobb párt választottuk.

### 2.4. Gépi látás és OpenCV

A LetSee alkalmazás szinte az összes esetben a bemeneti kamera vagy a LIDAR szenzor alkotta képekkel dolgozik, így a feladatok megoldásai a gépi látás témakörébe tartoznak. Ezen belül is, képföldolgozási módszerek voltak nagyrészt alkalmazva. A feladat megoldásainak kontextusában, a kép egy kétdimenziósan elrendezett ponthalmaz, ahol az egyes képpontok egy valamilyen színtérbeli vektort tartalmaznak, legyen ez RGB (2.1.1. fejezet) színérték, vagy egyszerű skalár érték (szürkeskálás kép). Ezeknek a képeknek van egy megszabott szélessége és magassága, és az egyes képpontok koordinátái a  $[0, \text{magasság})$  és  $[0, \text{szélesség})$  tartományba esnek. Ezekben a tartományokon belül minden képponthoz tartozik egy szín. Ez sokszor a használt könyvtártól függ, de általánosságban a  $(0,0)$  koordináta a kép bal felső sarkát jelenti. Így például, a legalacsonyabb pontjai a képnek, a legnagyobb  $y$  koordinátákkal rendelkező képpontok, míg a legnagyobb  $x$  koordinátákkal rendelkező képpontok, a legjobboldalsó képpontok.

Mindkettő feladat tárgyalása során, de kifejezetten az ösvénykövetés feladatának esetében, fontos az OpenCV [9] könyvtár ismerete. Ezen könyvtár rengeteg olyan függvényt tartalmaz, amelyek a képföldolgozás építőelemeiként szolgálnak. A következő függvények a szürkeskálás képeken működnek, ahol minden képpontot, aminek az értéke nem 0, telítettnek tekintünk, a 0 értékű képpontokat pedig a háttér részének.

A kontúrok alatt a telített képpontok alkotta, összeérő alakzatokat értjük. Amennyiben

két alakzat között van akár egy képpontnyi összeérés, egy kontúrnak vesszük őket. Ezeket az alakzatokat az OpenCV, az őket alkotó pontok koordinátáinak listájaként menti el. A kontúr területe az azt alkotó képpontok száma.

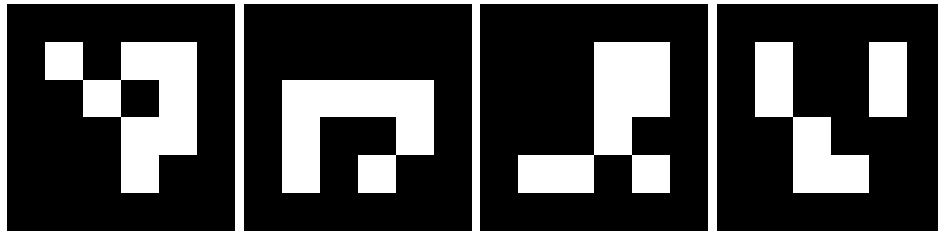
A bekerítő téglalap egy olyan téglalap alakú területet ír le, amely egy kontúr, egy szürkeskálás kép, vagy annak egy részének telített pontjai közül, az összeset bekeríti. Ez a téglalap, a kép dimenziójában vízszintes és függőleges oldalakkal rendelkezik. A téglalapot a bal felső sarokpontjával, valamint a szélességével és magasságával definiáljuk. A téglalap  $x$  és  $y$  koordinátája alatt a bal felső sarkának a koordinátáit értjük. A bekerítő téglalap tekinthető úgy, mint a legkisebb területű, a kép oldalaival párhuzamos téglalap, ami egy alakzat körül rajzolható. A téglalap minden oldala olyan közel van a téglalap középpontjához, mint az alakzatnak abban az irányban legtávolabb lévő pontja. Ebből adódóan, egy alakzat bekerítő téglalapjának minden oldalegyenese metszeni fogja az alakzatnak legalább egy pontját. Ezt a segédfüggvényt leggyakrabban a vizsgált terület leszűkítésére használjuk, ugyanis a rajta kívüli részekben ismert, hogy nincs telített pontja a vizsgált alakzatnak.

Az OpenCV biztosít függvényt, egy adott alakzat köré rajzolható legkisebb területű téglalap megtalálására. A bekerítő téglalappal szemben, a legkisebb területű téglalap oldalai nem feltétlenül párhuzamosak a kép oldalaival. Ez a segédfüggvény egy elforgatott téglalapot ad vissza, amelynek a sarokpontjai nincsenek megjelölve. Így például a legkisebb területű téglalap sarokpontjai közül nem ismert, hogy melyiknek a legkisebb az  $y$  koordinátája, és arra külön keresést kell végezni.

Az OpenCV továbbá lehetőséget ad tetszőleges alakzatú kontúrok középpontjának kiszámítására is, kép momentumok használatával [10].

## 2.5. ArUco jelölések

Az ArUco jelölések olyan fekete fehér minták, amelyek a kameraképen belüli fölismerésükre lettek kitalálva. Az ilyen jelölésekkel ellátott objektumokat, képesek vagyunk a kameraképen belül robotosan megtalálni. A színfölismerés feladatában az ArUco jelölések OpenCV implementációját használtuk föl [11]. Az ArUco jelölésekhez tartozik egy-egy azonosító is, amely a mintázatukból szűrhető le. Ezek a minták fekete alapon vannak, és azon belül egy négyzetes mátrixban elhelyezkedő, fehér képpontokból állnak össze. Az OpenCV-n belül több ArUco könyvtár is elérhető, amelyek közül a színfölismerésnél a 100 jelölést tartalmazó,  $4 \times 4$ -es mátrixokból álló könyvtár volt alkalmazva. Az 1. ábrán látható példa egy ebből a könyvtárból való jelölésre.



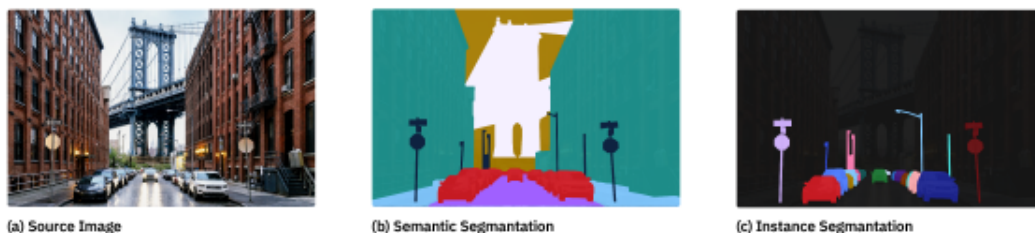
1. ábra. Az OpenCV 4x4 méretű, 100 jelölést tartalmazó ArUco könyvtárának a jelölései, balról jobbra a 0-tól a 3. azonosítóig.

## 2.6. Homográfia

A homográfia alatt két tetszőleges sík közötti relációt értünk, amelyet egy  $3 \times 3$ -as mátrixal tudunk jellemezni [12]. A homográfiának a színfelismerés feladatában van szerepe, ahol a kamera perspektíva eltávolítására fogunk előállítani egy transzformációs mátrixot, az OpenCV segítségével [13]. Ezzel a transzformációval fogunk előállítani négy, a kameraképen belüli adott sarok által leírt téglalapról egy téglalap alakú, perspektíva nélküli képet. Az ehhez a transzformációhoz használt négy sarokpont egy a téglalap alakú képen belüli, tetszőleges téglalap négy sarkának fog megfelelni. Érdekes megemlíteni, hogy a használt függvény képes tetszőleges négy sarokpontot tetszőleges négyszög alakzatra leképezni, azonban a dolgozatban tárgyalt megoldásoknál erre nem volt példa.

## 2.7. Szegmentáció

A szegmentációs eljárás a képet alkotó egyes képegyesekhez, például képpontokhoz, osztályokat rendel, amely osztályok a képen látható objektumoknak vagy azok csoportjának felelnek meg [14]. Két főbb változata a szemantikus szegmentáció, ahol az osztályok az összes egyező kategóriába tartozó objektumokat lefedik, illetve a példány szegmentáció (angolul "*instance segmentation*"), ahol minden objektumpéldány külön osztállyal rendelkezik. Például egy csoportkép szemantikus szegmentációján az emberek mind a „személy” osztályba tartoznának, példányszegmentációval viszont az egyes személyeket külön osztályba sorolnánk. A szegmentáció ezen két típusát a 2. ábra szemlélteti.



2. ábra. Példa [14], a szemantikus (b. kép) és példány (c. kép) szegmentációra.

A feladat nehézsége miatt, a legtöbb esetben gépi tanulási modellt alkalmaznak a szegmentáció elvégzésére. Az ösvénykövetés feladatának megoldásához is egy ilyen modellt alkalmazunk, az ultralytics könyvtár YOLO v8 szegmentációs modelljét [15]. Habár a YOLO v8 modellből előre tanított verziók is elérhetőek, a pontosabb eredményért, az egyes feladatokhoz specifikus modelleket tanítottunk be. A YOLO tipikusan RGB képeken van tanítva, azonban lehet szürkeskálás képeken is tanítani. Például az ösvénykövetés feladatában, a YOLO-t a telefon LIDAR szenzorából kiszámolt mélységképeken tanítottuk, amely mélységképek szürkeskálás képek formájában voltak ábrázolva.

### 3. Színfölismerés feladata

A színfölismerés feladatában, a Pilótának 6 ruhaanyagot kell szín szerint sorbarendezenie. A ruhaanyagok színei két szempont szerint csoportosíthatóak: az alapszínük szerint, ami lehet Piros, Zöld, Cián, Kék vagy Magenta, valamint a tónusuk szerint, ami lehet Világos, Közepes vagy Sötét. Ezek lesznek a továbbiakban a referencia színek.

Az alapszín szerint két csoportra kell osztani a ruhaanyagokat, majd pedig a csoportokon belül, sötétség szerint kell őket sorba rendezni. Az alapszínek lehetséges párosításai ismertek voltak. Ezek lehettek Piros és Zöld, Piros és Cián, Piros és Kék, Magenta és Zöld, Magenta és Cián vagy Zöld és Kék.

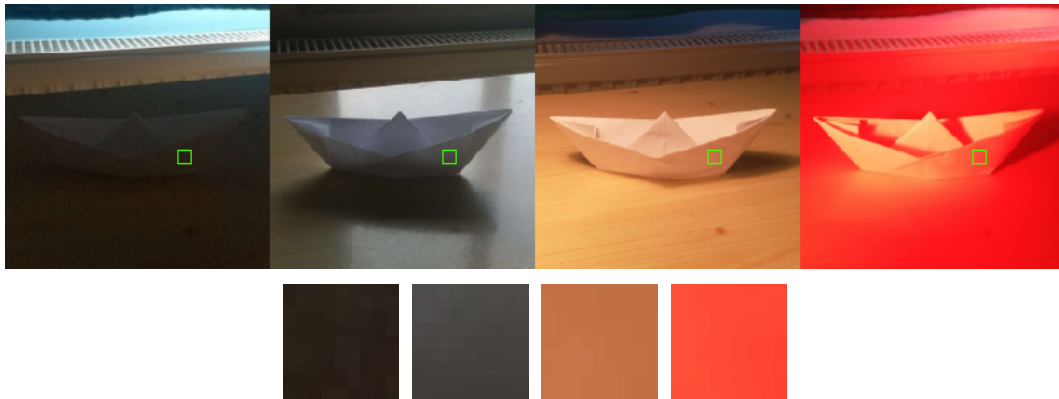
A jelenleg tárgyalt megoldás egyszerre egy tárgy színét képes, előre megállapodott referencia színek közül fölismereni. A továbbiakban a tárgy, amelynek színét akarjuk leolvasni, a vizsgált objektum lesz, annak színe pedig a vizsgált szín.

#### 3.1. Fényhatások kezelése

A feladat megoldásához elengedhetetlen, a telefon kamera RGB képének használata. Mivel az RGB és más színábrázolási értékek, önmagukban nehezen értelmezhetőek, így a vizsgált színt az előre megállapodott színekhez hasonlítjuk, és a legkisebb távolsággal rendelkező szín nevét adjuk vissza.

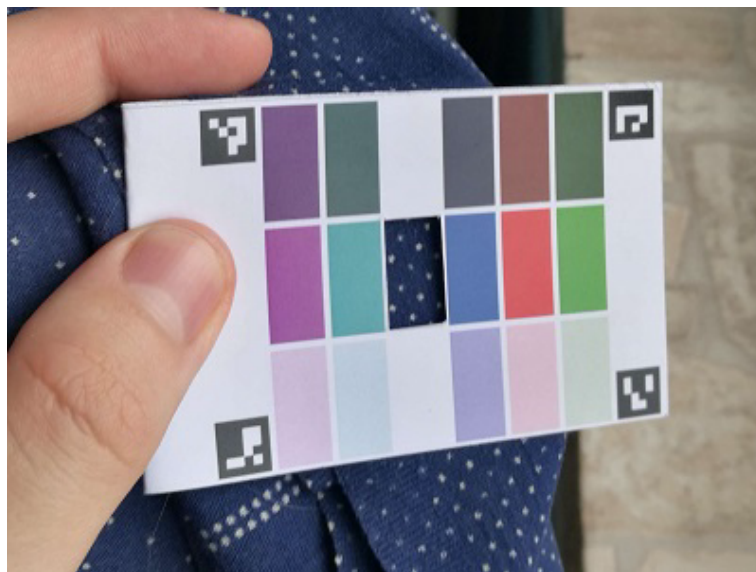
A kamera képén azonban, az egyes képpontok színértékeit, nem csak az azt ábrázoló objektumok színei, hanem a környezeti fényhatások is befolyásolják, torzítják. Az emberi látás egyik tulajdonsága a színkonstancia, amely lehetővé teszi, hogy különböző fényviszonyok mellett is állandónak érzékeljük a tárgyak színeit [16]. Azonban kameraképeken belül, csupán az RGB értékek alapján ez nagy kihívást jelent, amelyet a 3. ábra is demonstrál. Amennyiben a referencia színeket a programon belül tárolnánk, az összehasonlítás hibás lenne, ugyanis a vizsgált objektum színéhez hozzáadódnak a környezeti fényhatások, míg a referencia színekhez nem. Ha

az egyik referencia szín és a vizsgált szín a valóságban meg is egyezne, a vizsgált szín torzítottja a referencia szín torzítatlan értékével már nem egyezne meg.



3. ábra. Példa a színkonstanciára, és annak a hiányára. Az emberi szemnek, a fehér hajó színének fölismerése triviális, azonban az elkülönített színek (alsó ábra) nem hasonlítanak a fehérhez.

Erre a problémára az általunk használt megoldás az volt, hogy legyenek a referencia színek is a kameraképből beolvasva. Ideálisan a vizsgált objektumhoz fizikailag minél közelebb, hogy a színek között a fényhatások ne térjenek el túlságosan. Ez a megoldás azért működik, mert ha az egyik referencia szín, és a vizsgált szín a valóságban megegyezne, akkor az ugyanolyan fényhatások alatt vett torzítottjuk is megegyezne. Hogy a referencia színek a kameráról legyenek beolvasva, egy fizikai kelléket, referencia színkártyát használtunk, amely tartalmaz a színeken kívül ArUco jelöléseket is, a kártya releváns területeinek a kameraképen belüli meghatározásához. Az egyik ilyen színkártyáról a 4. ábrán látható kép.



4. ábra. A színfölismeréshez használt referencia színkártyák egyike.

### 3.2. Színkártya fölismerése

A színfölismerés első lépése a kameraképen belül a színkártya megkeresése. Ennek eléréséhez a kártyák négy sarkára, egy-egy, különböző azonosítójú ArUco jelölés van elhelyezve, amelyek pozícióját a kameraképen az OpenCV segítségével meg tudjuk találni. Az így kapott pontok leírják a kártya releváns területét, azonban a könnyebb kezelhetőség érdekében, ezen a területen homográfia transzformációt hajtunk végre, a kamera perspektívájának eltávolításához. Ennek a transzformációnak az eredménye egy olyan kép lesz, ahol a színkártya releváns területe téglalap alakú lesz. Ez a kép lesz a továbbiakban a téglalap alakú kép. A transzformációs mátrix előállításához négy-négy pont szükséges. Ezek közül az első négyes, a bemeneti képen belül, négy tetszőleges pont, a második négyes pedig, azoknak a kimeneti képen vett megfelelői. Az első négyes itt az ArUco jelölések egy-egy sarokpontja lesz, a kimeneti pontoknak pedig, egy általunk előállított képen belül, egy annak oldalaival párhuzamos oldalú téglalap sarokpontjait adjuk meg. Az, hogy melyik jelölésnek, melyik sarokpontját használjuk, a 3.2.2. fejezetben van részletezve. Példának, a 4. ábráról készült téglalap alakú kép, az 5. ábrán látható.



5. ábra. A 4. ábráról készült téglalap alakú kép.

A homográfia előállításához mind a négy ArUco jelölést fölhasználtuk, és akár egynek a hiányában, a jelenleg vizsgált képkockát átugortuk. A vak fölhasználók számára nem triviális az, hogy az ArUco jelölések ne lógjanak le a kameraképről, vagy hogy ne legyenek eltakarva valamelyik ujj által. Ezért implementáltunk egy visszajelzést arra, hogy ha nem sikerült minden jelölést megtalálni. Ezt a megoldást úgy implementáltam, hogy a hiányzó jelölések alapján más-más visszajelzést lehessen adni, amelyhez bitmaszkolást alkalmaztam. Minden sarkon egy-egy jelölés van, és az OpenCV vagy megtalál egy jelölést, vagy nem. Ez kettő lehetséges értéket

jelent minden saroknál, amelyet sarkanként egy biten tudunk ábrázolni. Így négy bitet használva egy ellenőrzőmaszkon azt pontosan tudjuk ábrázolni, hogy melyik jelöléseket sikerült megtalálni. Amennyiben ennek az ellenőrzőmaszknak az értéke értéke 0b1111, avagy 15, akkor minden jelölést sikerült megtalálni. A színfölsímerő modulban definiálva volt minden sarokra egy-egy, 0 és 3 érték közötti azonosító. Például a jobb alsó sarok azonosítója 2 volt. Az 1 értéket ezzel az azonosítóval balra eltolva, megkapjuk az ahhoz a sarokhoz tartozó bitmaszket. Az ellenőrző maszkon ezt a bitet tesszük igazzá, hogy ha az ahhoz a sarokhoz tartozó jelölést megtaláltuk. Az ellenőrzőmaszknak 16 lehetséges értéke van, az egész számok halmazának  $[0,15]$  tartományában. Itt minden érték egy-egy lehetséges fölállást kódol, és ismerve az azonosítókhoz tartozó sarkokat, egy tömbön belül minden lehetséges fölállásra írtam egy visszajelzést. Így ha az ellenőrzőmaszk értéke nem 15, akkor a megfelelő hibaüzenetet a tömbnek az ellenőrzőmaszkkal való indexelésével tudjuk megszerezni. A lehetséges indexekhez tartozó visszajelzések a legutóbbi verzióból a 1. táblázatban láthatóak. Ez a visszajelzési forma azonban a Pilóta számára összezavaró volt. Továbbá a visszajelzés feltételezi, hogy a jelölés a képen nem látszódik, azonban az OpenCV nem képes minden esetben tökéletes megtalálni a jelöléseket.

| Index | Visszajelzés                         | Bal alsó | Jobb alsó | Jobb felső | Bal felső |
|-------|--------------------------------------|----------|-----------|------------|-----------|
| 0     | Nincs jelölés                        | 0        | 0         | 0          | 0         |
| 1     | Mozdítsa a kártyát bal föntre        | 0        | 0         | 0          | 1         |
| 2     | Mozdítsa a kártyát jobb föntre       | 0        | 0         | 1          | 0         |
| 3     | Mozdítsa a kártyát fölfele           | 0        | 0         | 1          | 1         |
| 4     | Mozdítsa a kártyát jobb lentre       | 0        | 1         | 0          | 0         |
| 5     | Nincs jobb felső se bal alsó jelölés | 0        | 1         | 0          | 1         |
| 6     | Mozdítsa a kártyát jobbra            | 0        | 1         | 1          | 0         |
| 7     | Nincs bal alsó jelölés               | 0        | 1         | 1          | 1         |
| 8     | Mozdítsa a kártyát bal lentre        | 1        | 0         | 0          | 0         |
| 9     | Mozdítsa a kártyát balra             | 1        | 0         | 0          | 1         |
| 10    | Nincs jobb alsó se bal felső jelölés | 1        | 0         | 1          | 0         |
| 11    | Nincs jobb alsó jelölés              | 1        | 0         | 1          | 1         |
| 12    | Mozdítsa a kártyát lentre            | 1        | 1         | 0          | 0         |
| 13    | Nincs jobb felső jelölés             | 1        | 1         | 0          | 1         |
| 14    | Nincs bal felső jelölés              | 1        | 1         | 1          | 0         |
| 15    | OK                                   | 1        | 1         | 1          | 1         |

1. táblázat. Az ArUco jelölések hiányzásáról adott visszajelzések, az ellenőrző maszk alapján. A 0 a hiányzó sarkokat, az 1 pedig a megtalált sarkokat jelenti.

Fontos észrevétel, hogy a kimeneti képen belül, a kártya releváns területének elhelyezkedését mi adjuk meg. Például, hogy ha szeretnénk, a téglalap alakú képen belül, a színkártya jobb oldala melletti területet is meghagyni, akkor a transzformációs mátrix kiszámításához egy olyan téglalap sarokpontjait használjuk, amelynek a jobb oldala nem illeszkedik a kép oldalához, és ezzel a kártya jobb oldala melletti terület nem fog kilógni a képről. Az így meghagyott részek

lesznek a továbbiakban a margók. A téglalap, amelynek sarokpontjai az ArUco jelölések sarokpontjainak felelnek meg, megfeleltethető a kártya releváns területének. Ahogyan aránylik egy pont a fizikai színkártyán a releváns területhez, úgy fog aránylani az a pont a releváns területhez a téglalap alakú képen belül is. Ismerve a referencia színeknek a fizikai színkártyán való elhelyezkedését, képesek vagyunk előre, a program indításakor kiszámítani azt, hogy a téglalap alakú képen belül, hol vannak a referencia színeket tartalmazó képpontok. Ugyanígy tudjuk leolvasni a téglalap alakú képről a vizsgált színt is. A színek összehasonlításához a CIEDE2000 színtávolságot használjuk, amelyhez az RGB színeket a CIE 1976  $L^*a^*b^*$  színtérbe kell konvertálnunk. Az egyszerűség kedvéért ezt a konvertálást nem a színek összehasonlításakor végezzük el egyesével, hanem az egész téglalap alakú képet át konvertáljuk a CIE 1976  $L^*a^*b^*$  színtérbe.

### 3.2.1. Színkártyák paraméterei

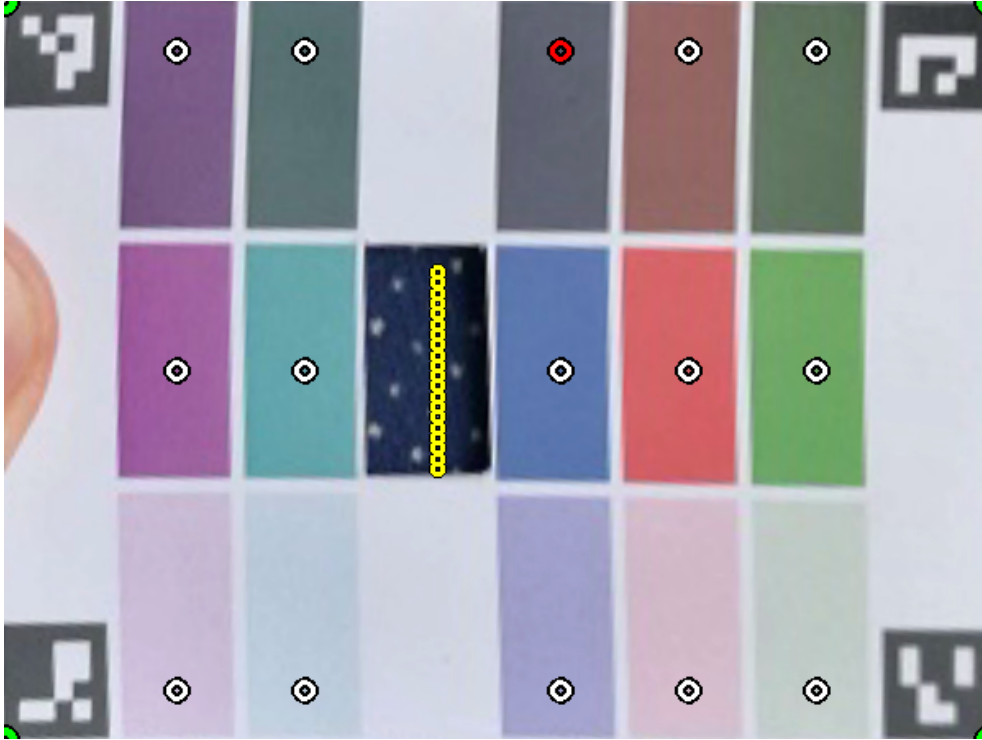
A feladat megoldására több referencia színkártyát is leteszteltünk. Ezek olyan paraméterekben különböztek, mint például a referencia színek leolvasási koordinátái a téglalap alakú képen belül. Még a program fejlesztésének elején azt a tervezési döntést hoztam, hogy mindegyik színkártyáknak adok egy azonosítót, amelyet a színfölismerést végző modul inicializálásakor lehet megadni, és az egyes színkártyáknak a paraméterei az azonosítók alapján lesznek beállítva. A színfölismerő modulnak, a telefonba való integrálásakor, a kártyaazonosító nem lett dinamikusan megadva, hanem csak a program újra fordításával lehetett megváltoztatni. Azonban a megjegyzéssé alakítással szemben könnyebben lehetett így a paramétereket változtatni, és szükség esetén az applikáción belüli beállítást is könnyen lehet rá készíteni.

Habár a program által támogatott összes színkártya között megegyeznek, az ArUco jelölések kiosztása is úgy van megadva, hogy szükség esetén a kártya azonosítója alapján meg lehessen változtatni. Külön állítható az, hogy melyik azonosítójú jelölés, melyik saroknak felel meg, és külön állítható, hogy az egyes sarkokban lévő jelöléseknek, melyik sarokpontja legyen fölhasználva, a transzformációs mátrix előállításában. A téglalap alakú kép felbontása, és az egyes oldalainak margói is állíthatóak voltak. Az utóbbira azért volt szükség, mert az első színkártyáknál, a vizsgált színt, a kártya jobb oldaláról olvastuk le.

A vizsgált színt a téglalap alakú képről szűrjük le, több pontot is mintavételezve. Ezek lesznek a továbbiakban a lekérdezési pontok, amelyeknek a szerepe a 3.4. fejezetben lesz részletezve. A lekérdezési pontok koordinátáin kívül más információt nem mentünk el róluk. A könnyű paraméterezés érdekében ezek a lekérdezési pontok egy olyan segédfüggvénnyel vannak létrehozva, amely egy szakaszon helyezi el őket. Az inicializáláskor a szakasz kezdő-, és végpontja van megadva, valamint az, hogy hány lekérdezési pont legyen rajta. A pontok a szakasz egyenesén

úgy helyezkednek el, hogy egymástól egyenlő távolságra legyenek, valamint a szakasz kezdő- és végpontjára is essen egy-egy lekérdezési pont. Az erre szolgáló segédfüggvény úgy van kialakítva, hogy többszöri meghívás esetén, a meglévő pontokat nem törli, hanem kiegészíti azokat. Amennyiben szükséges, lekérdezési pontokat a segéd függvény nélkül is megadhatunk.

A referencia színek esetében, a leolvasási koordinátákon kívül, az adott szín nevét is elmentjük. Itt minden referencia színt csak egy pontból olvasunk le, ugyanis azoknak elhelyezkedése a kártya releváns területén belül konzisztens. A feladat megoldása során, a Cybathlon verseny színeinek fölismerése volt a cél. Ezek a színek alapszínük és tónusuk szerint csoportosíthatóak, amely a színkártyán való kiosztásukban is reflektálva volt. A referencia színek a színkártyán oszloponként alapszínekre, soronként pedig tónusokra voltak osztva, ezért a téglalap alakú képen kiosztó segédfüggvény is így kategorizálta őket. A függvény két listát vár, az alapszínek és a tónusok listáját, majd pedig az egyes színeket ezek keresztszorzatával hozza létre, egy rácsos kiosztásban. A rácsnak a bal felső és jobb alsó sarokpontja volt megadva, a többi pont pedig úgy volt kiosztva, hogy közöttük tengelyenként egyenlő távolság legyen. A pontok kiosztásánál a tónus és alapszín listáján indexelünk végig, mint sor- és oszlopindex, és az adott pontban elmentett referencia szín nevének első szótagja lesz a tónus neve, míg a második pedig az alapszín neve. Amennyiben akár az alapszín, vagy tónus tag, egy üres string változó volt, nem lett új referencia szín elhelyezve. Ennek az volt az oka, hogy egyes kártyákon voltak üres oszlopok, és könnyebb megoldás volt a segédfüggvényen belül kezelni őket, mint hogy a segédfüggvényt többször meghívánk. Habár ez a segédfüggvény a Cybathlon színeinek kiosztására van tervezve, a mögötte lévő lista, tetszőleges nevű és koordinátájú referencia színt is képes eltárolni.

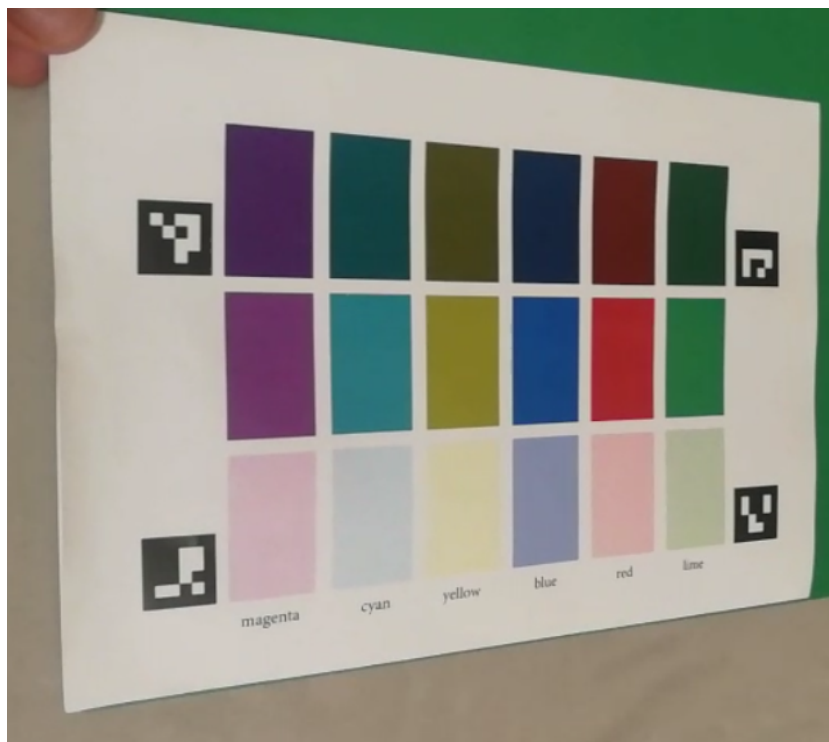


6. ábra. Az 5. ábrára téglalap alakú képén belül, a vizsgált paraméterek. A zöld pontok a releváns terület sarkait, a sárga pontok a lekérdezési pontokat, a fehér pontok pedig a referencia színek leolvasási pontjait jelölik.

### 3.2.2. A tesztel színkártyák

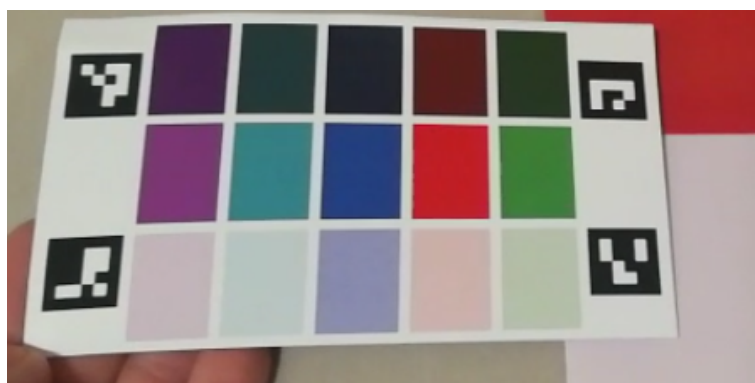
Az ArUco jelölések azonosítóinak kiosztása tetszőleges volt, és az összes színkártyánál megegyezett: A 0. azonosítójú ArUco jelölés került a bal felső, 1. jelölés a jobb felső, 2. a bal alsó, a 3. azonosítójú ArUco jelölés pedig a jobb alsó sarokban. Szintén megegyezett a színkártyák között az, hogy az ArUco jelölések külső sarokpontjait használták, a transzformációs mátrix kiszámításához. Például a bal felső sarokban az ott lévő jelölés bal felső sarkát használta, a jobb felső sarok a jobb felső sarkát használta, és így tovább. A színkártyák téglalap alakú képének a mérete tetszőleges volt, mivel az oldalhosszakhoz való arány volt vizsgálva, és még a hossz és magasság arányának sem kellett egyeznie a program működéshez.

A legelső, kezdetlegi színkártya egy A5-ös papírlapra volt nyomtatva. Ezen a színkártyán még szerepel a sárga szín, amely később a verseny színei közül el lett távolítva. Itt a kártyának a jobb oldalán van elhelyezve 20 darab lekérdezési pont, így a jobb margót megnöveltük, a kártya hosszának az  $\frac{1}{3}$ -ára. Ez a színkártya még az ötlet bizonyítására készült, és a nagy mérete, és hajlékony lapra való nyomtatása miatt nehezen volt kezelhető, azonban a kártya azonosító alapú paraméterezés miatt, a programmal még most is kompatibilis. Ez a színkártya a 7. ábrán látható.



7. ábra. Az A5-ös papír lapra nyomtatott, kezdeti színekártya.

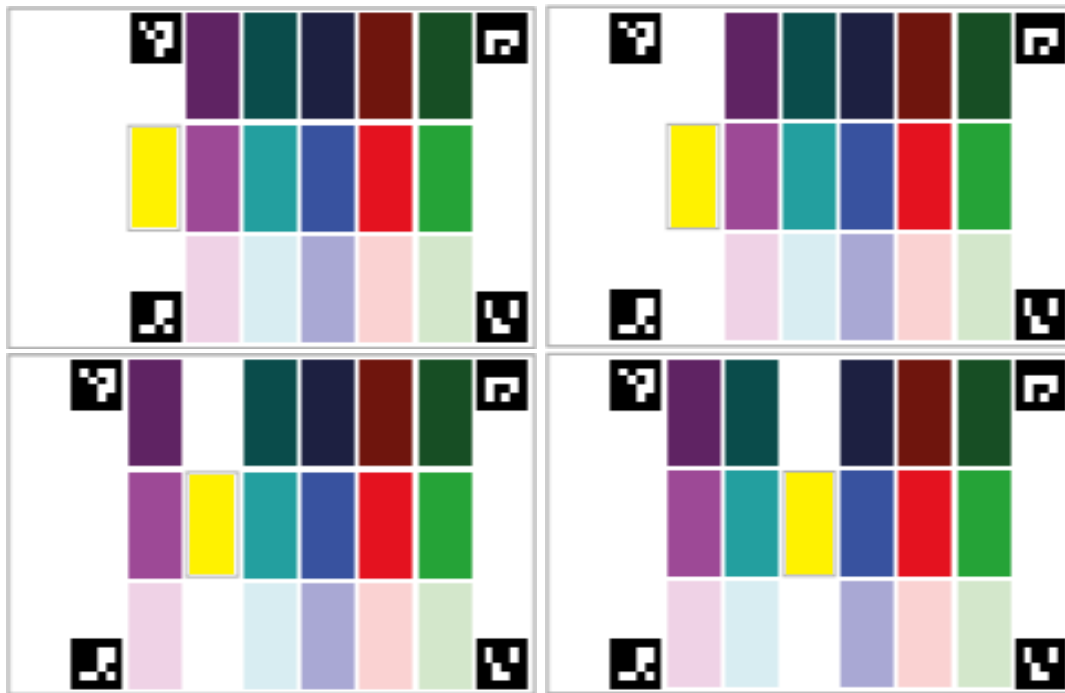
A második színekártya a telefonnal egyméretű papírkartonra volt rányomtatva. Ezt a színekártyát könnyebb volt kézben tartani, és a sárga színt már nem tartalmazta, de a 20 lekérdezési pont ugyanúgy a kártya jobb oldalán volt található. Ezzel a probléma az volt, hogy a vak felhasználónak nincs referenciája ahhoz, hogy a vizsgált objektumhoz képest, hol kell tartania a színekártyát. Ez a színekártya a 8. ábrán látható.



8. ábra. Kép a telefon méretű színekártyáról, felvétel közben.

A következő színekártya, a még könnyebb kezelhetőség érdekében bankkártya méretű volt, és négy verzió készült belőle. Ahhoz, hogy a főhasználó tudja, hol lesznek a lekérdezési pontok, egy rés lett elhelyezve a színekártyán, amelyen keresztül a vizsgált objektumot lehet látni. További előnye volt ennek az is, hogy így a vizsgált objektum, és a referencia színek mégközelebb voltak

egymáshoz, és a rajtuk eső fényhatások kevésbé különböztek. Ez a rés, a kártya releváns területén belülrre esett, így a téglalap alakú képen nem volt szükség margóra. Mind a négy verzióban, a rés mérete a középső sorban lévő referencia színek négyzetével egyezett meg, azonban annak az oszlopa különbözött. A nulladik oszlopú színkártya esetén, a rés a baloldali ArUco jelölések között volt, míg a többiben, egy külön oszlop lett neki beillesztve. A résen belül, továbbra is 20 lekérdezési pont volt elhelyezve. Ezek a színkártyák a 9. ábrán láthatóak.



9. ábra. Képek az egy réses színkártyákról. A sárgával jelölt rész a fizikai kártyákon ki van vágva.

A réssel ellátott színkártyákról, készültek olyan verziók, ahol a jobb ArUco jelölések között is van egy második rés. Az elképzelés az volt, hogy a vizsgált objektum színére pontosabb becslést lehet kapni, hogy ha két távolabbi pontról mintavételeznénk azt. Ezek a színkártyák minden más szempontból megegyeztek az egy réses változatokkal. A két rés lekérdezési pontjai nem voltak megkülönböztetve, azonban a bal oldali résnél 20 darab lekérdezi pont volt, míg a jobb oldalinál csak 12. Ezt azért implementáltam így, mert ezen a ponton a klaszterezés nem tudta volna kezelni azt, hogy ha a két oldalt más szín van, de egyenlő számban. Ezért feltételeztem, hogy így biztosabb lenne a fölismerés. A preferenciát a bal oldali rés kapta, mert a Pilóta a bal kezével tartotta a kártyát. Ez később meg lett változtatva, hogy mindkét oldalon 20 lekérdezi pont legyen, azonban ez a Pilótával nem lett tesztelve. A probléma ezzel a színkártyával az volt, hogy nehéznek bizonyult a kártyát úgy tartani, hogy mindkettő rés a vizsgált objektumhoz simuljon. Amennyiben a kártya túloldali rése, a vizsgált objektumtól túlságosan távol van, árnyékot vet rá, és így a lekérdezési pontokból leolvasott színértékek torzulni fognak. A két réses színkártyák

a 10. ábrán láthatóak.



10. ábra. Képek a két réses színkártyákról. A sárgával jelölt részek a fizikai kártyákon ki vannak vágva.

A legutóbbi színkártya továbbra is bankkártya méretű, és csak egy rést tartalmaz. Ez a rés, a baloldali ArUco jelölések között van elhelyezve, 20 lekérdezési ponttal. Az újdonság ebben a színkártyában az, hogy a releváns terület a kártya jobb oldalára van sűrítve, hogy a felhasználó a bal kezében könnyebben tudja tartani a kártyát. Mivel a Pilóta konzisztensen a bal kezében fogta a kártyát, ezért jobb kezes verzió nem készült. Ez a kompakt színkártya a 11. ábrán látható.



11. ábra. Minta a tömörített, kompakt színkártyáról. A sárgával jelölt rész a fizikai kártyán ki van vágva.

### 3.3. Színek összehasonlítása

A színek összehasonlítására a CIEDE2000 színtávolságot használtuk, amely a CIE 1976  $L^*a^*b^*$  színtérben van definiálva. Ehhez a téglalap alakú képet, még a klaszterezési eljárás (3.4. fejezet)

előtt, a CIE LAB színtérbe konvertáljuk át, az OpenCV segítségével. Habár az OpenCV képes CIE 1976  $L^*a^*b^*$  színtérbe váltani, a CIEDE2000 színtávolság függvényre nincs beépített megoldás, így ehhez Greg Fiumara [17], MIT licenz alatt kiadott implementációját használtuk. A CIEDE2000 használatának előnyei a 2.1.3. fejezetben vannak tárgyalva. A használt CIEDE2000 színtávolság függvény, a saját CIE 1976  $L^*a^*b^*$  adatstruktúráját használja, amely az  $L$ ,  $a$  és  $b$  csatornákhöz, egy-egy dupla pontosságú lebegőpontos számot használ, azonban az OpenCV a CIE 1976  $L^*a^*b^*$  színtér csatornáit 8 biten tárolja. Az OpenCV használta formában, az  $L$  csatorna  $[0,100]$  tartománya a  $[0,255]$  tartományra van normálva, míg az  $a$  és  $b$  csatornák értéke 128-al van megnövelve. A színtávolsághoz egy segédfüggvényt implementáltunk, amely az OpenCV színvektort átváltja, és a CIEDE2000 színtávolság függvénynek továbbítja, majd annak eredményét adja vissza.

A legutóbbi verzióban, a lekérdezési pontokból a klaszterezéssel több lehetséges színt is ki nyertünk, amelyek mindegyikét összehasonlítottuk a referencia színekkel. A továbbiakban ezek lesznek a lekérdezett színek. Az összehasonlítást egy egyszerű minimum kereséssel valósítottuk meg, ahol végig iterálunk a lekérdezett színeken, és azokon belül a referencia színek koordinátáin is. Az egyes koordinátákról leolvassuk a referencia színeket, és a jelenleg vizsgált lekérdezett színnel vesszük a CIEDE2000 színtávolságát. A keresés során a minimum távolságot, és az annak számításában használt referencia szín nevét mentjük el, amely adatok a fölhasználónak lesznek fölolvastva. A színtávolság azért van továbbítva, hogy a felhasználó tapasztalati úton érzékelhesse, hogy az egyezés mennyire pontos.

### 3.4. Vizsgált szín klaszterezése

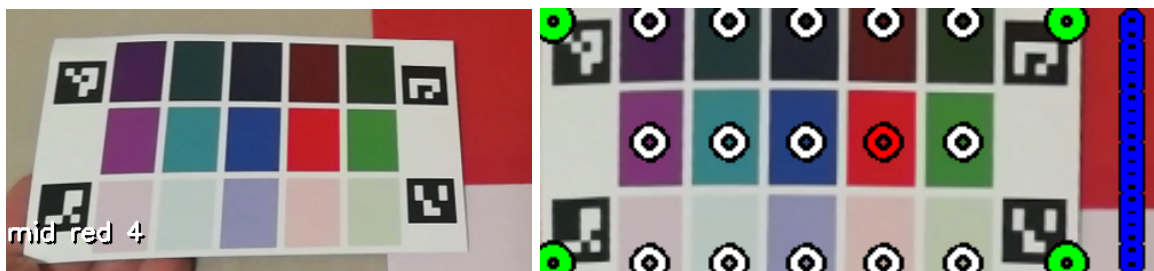
Mivel a vizsgált objektum nem a kártya része, ezért annak a kameraképen belüli elhelyezkedése, és a rá eső fényhatások inkonzisztensek tudnak lenni. Előfordulhat az, hogy a leolvasott pontra éppen árnyék vetődik, vagy hogy a vizsgált objektumot nem fedi le, és a háttér színét olvasnánk le. Emiatt érdemes a vizsgált színt, több pontból is leolvasni, hogy a figyelembe vett színértékek közül kevesebb legyen hibás. A lekérdezési pontok értékéből, a vizsgált színt legelőször számtani középpel képeztük le, azonban átlagolás esetében akár egy darab helytelen színtérték is képes elrontani az eredményt. Mivel a lekérdezési pontokban a színértékek nem voltak homogének, ezért klaszterezést alkalmaztunk. Az alkalmazott hierarchikus klaszterezési módszer, az összes lekérdezett színnek egy-egy külön klasztert ad, amelyeket a legkisebb távolság alapján von össze egymással. A továbbiakban, a klaszterek alatt a már összevont színpárokat értjük. A használt távolsági metrika a CIEDE2000 színtávolság volt.

A klaszterezés során, az összes lekérdezési ponton végig iterálunk, és leolvassuk azoknak

színértékeit. Az iteráción belül megkeressük a már meglévő klaszterek között azt, amely középpontjával a leolvasott szín a legkisebb távolságot képzi. A klaszterek középpontja, az abba belevont színek számtani közepe. Mivel csak a már meglévő klaszterek között végezzük a keresést, ezért ezt egy mohó keresésnek tekintjük. Hogy az egymástól túlságosan különböző színek ne kerüljenek egy klaszterbe, a keresésnek van egy tűréshatára, amely színtávolság felett, a leolvasott színt nem vonjuk be egy meglévő klaszterbe, hanem újat hozunk létre neki. Ez a tűréshatár a legutolsó verzióban 6.0 egység volt, amely a színfölsímerés átlagos értékénél (a 2. táblázat alapján) nagyobb. Az általános hierarchikus klaszterezési módszerekkel szemben, ezt a klaszterezési eljárást csak egy iterációban végezzük el, hiszen az összes klaszter egybevonása itt nem kívánt. Mivel a klaszterek középpontja számtani középpel van számolva, ezért elképzelhető olyan eset, ahol a lekérdezési pontok egy graduális átmenetre esnek, és nem hoznak létre új klasztereket. Ilyen esetben akár nagy távolsággal rendelkező színek is ugyanabba a klaszterbe kerülhetnének, ha annak középpontja a kettőhöz hasonló színt képez.

A következő lépésben megkeressük a legnagyobb klasztert, és lekérdezzük annak méretét, ami a klaszterbe belevont színek száma. Ez lesz az a klaszter, amely középpontja a legtöbb színhez hasonlított. A lekérdezett színek listájába, a legnagyobb klaszter számtani közepén kívül, az összes olyan klaszter középpontját is bele vesszük, amelyeknek a mérete a legnagyobb klaszterhez képest egy tűréshatáron belülre esik. A legutóbbi verzióban ez a tűréshatár 5 volt. Például, ha a 20 lekérdezési pontból 11, 8 és 1 méretű klaszterek jönnek létre, akkor a lekérdezett színek listája a 11 és 8 méretű klaszterek középpontjait fogja tartalmazni. Ezek a lekérdezett színek kerülnek összehasonlításra a referencia színekkel.

A klaszterezés eredménye az volt, hogy amennyiben a lekérdezési pontoknak csak egy kis része lóg le a vizsgált objektumról, a vizsgált szín leolvasása továbbra is működik, és a fölsimert referencia színnel vett távolság sem nő. Továbbá, a klaszterezés konzisztensebbé teszi, a mintázattal ellátott tárgyak színének fölsímerését, és az árnyékok okozta torzításokat is segít kiszűrni. A klaszterezés működésére a 12. ábrán látható példa.



12. ábra. Példa, a klaszterezés működésére, a színfölsímerés feladatában. A jobb oldali ábrán, a kékekkel jelölt lekérdezési pontok, a világos piros színre is rálógnak, azonban a klaszterezés miatt, a színfölsímerést nem befolyásolják, és a piros szín közel 4 egységnyi színtávolsággal lesz így fölsímerve.

### 3.5. Mérési eredmények

A tapasztalat az volt, hogy a Cybathlon színei közül, a legtöbbet sikeresen föl lehetett ismerni, és a színpárokat szinte minden esetben sorba lehetett rendezni. A programnak, a dolgozatban tárgyalt legutolsó verziójával, a Pilóta a verseny színpárosításait minden esetben képes volt átlagosan 1,5-2 perc alatt sorbarendezeni. A színfölismerés során sokkal nagyobb probléma volt, az ArUco jelölések fölismerésének megbízhatósága, mint a színek eltévesztése. A használt referencia színek közül, a világos vörös és világos magenta színek fölismerése inkonzisztens volt. A legtöbb esetben ezeket a színeket nem lehetett megkülönböztetni, mert a világos magenta sokszor világos pirosnak volt érzékelve. Ugyanakkor fontos megjegyezni, hogy ennek a két színnek a megkülönböztetése egészséges emberi látás mellett sem triviális, és így ez a két szín, a Cybathlon verseny párosításaiban nem szerepelt egyszerre.

Az A5-ös méretű színkártyával mesterséges megvilágítás alatt készültek teszt videók, a Cybathlon versenyen is használt színpárokkal. A színfölismerő program ezeken a teszt videókon mért eredményei a 2. táblázatban találhatóak. Az összes szín közül, csak három alkalommal volt tévesztés. A világos magentát mindkét alkalommal világos pirosként ismerte föl a program, azonban a kettő nem lehet egy párosításban, és ezeknek a színeknek a megkülönböztetése emberi szemmel is nehéz. Továbbá, a Magenta és Cián párosításban, a világos cián színt a program pár képkockán világos zöldként ismerte föl, azonban a többi cián színt sikerült fölismerni, így a versenyen a Pilóta képes lett volna ezt a hibát kiszűrni.

| Szín Párosítás  | Átlag távolság | Legjobb Távolság | Legrosszabb Távolság |
|-----------------|----------------|------------------|----------------------|
| Piros és Zöld   | 3.22186        | 0.955781         | 6.52444              |
| Piros és Cián   | 4.33821        | 0.430116         | 7.84429              |
| Piros és Kék    | 4.17062        | $\sim 0$         | 8.57589              |
| Magenta és Zöld | 3.41318        | 0.320281         | 5.31995              |
| Magenta és Cián | 4.48081        | 1.22726          | 8.01982              |
| Zöld és Kék     | 3.90733        | 0.783562         | 9.70732              |
| Összesen        | 3.922          | $\sim 0$         | 9.70732              |

2. táblázat. Eredmények a színfölismerésnek teszteléséből, mesterséges megvilágítás alatt. A világos magenta és világos cián színek nem voltak tökéletesen fölismerve.

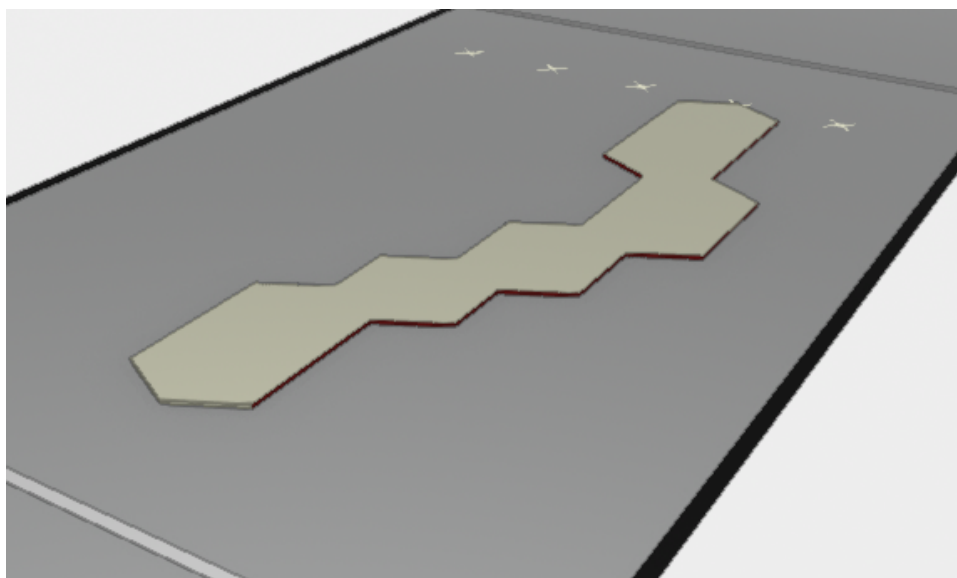
A helyes fölismerés távolságára volt extrém példa is, ugyanis a 13. ábrán látható fölismerés esetén, a helyes szín közel 19.0 egységnyi színtávolsággal van fölismerve, a kártya fénylő felülete miatt.



13. ábra. Példa a színfölismerés eredményére. A sárga pontok a lekérdezési pontokat, a piros a fölismert, míg a fehér pontok pedig a többi referencia szín leolvasási pontjait jelölik.

#### 4. Ösvénykövetés feladata

Az ösvénykövetés feladatában a Pilótának egy 1-2 centiméter magasságú fa lapokból kirakott ösvényen kell végigsétálnia, anélkül hogy arról lelépne. A lapok végei háromszög alakúak, így az ösvény minden illeszkedésnél balra, vagy jobbra kanyarodik, és két egymást követő falap között nem maradhat egyenes. Az egyes falapok kanyarodási irányáról a Pilótának nincs előismerete. A pályának egy lehetséges elrendezése a 14. ábrán látható. Az ösvény szélének tapintáson alapuló érzékelése sem lábbal, sem fehér bottal nem engedélyezett.



14. ábra. Az ösvénykövetés feladat fa lapokból kirakott ösvénye, a Cybathlon verseny bemutató videójából.

Ennél a feladatnál, a telefont lefele tartva használjuk, hogy a felvételen minél jobban látszódjon az ösvény és a felhasználó lábfejei. A még konzisztensebb látószög érdekében, a versenyen és a tanító anyagok felvételein, egy mellkasra fölvehető telefontartót is alkalmaztunk.

Ennél a feladatnál a telefonon futó alkalmazás az iPhone LIDAR szenzorát használja, és annak felvételét egy YOLO v8 modellnek adja tovább. Ez a modell, az úton való végigsétálás felvételein volt tanítva. A felvételek között a telefontartón kicsit állítottunk, hogy a modell ne legyen túltanítva a kamera látószögére. A LIDAR bemenet a színektől független, így az erre betanított modell képes kevesebb tanítási anyag mellett is, különféle körülmények között működni. A modell összesen négy objektum szegmentálására volt betanítva: az ösvényre, a bal és jobb lábfejekre (zokniban), és a talaj síkjára. A kép többi részének, mint például a felhasználó lábának, nem adtunk külön osztályt, hanem a háttér részeként kezeljük őket. A szegmentált maszkokat az útnavigáló modul egy három csatornás RGB kép formájában kapja meg. Itt a talaj, az ösvény és a lábfejek kombinált maszkja kapott, egy-egy külön csatornát. Ezek közül a talaj maszkot a modul nem használja föl, mert nem volt olyan hasznos információ, amely az ösvény maszkból nem nyerhető ki, de a talaj maszkból meg igen. Mivel a bal és jobb lábfejek egy kombinált maszkon vannak, így nem tudni azt, hogy a maszk melyik része melyik lábfejhez tartozik. Továbbá nem lehet eldönteni a kontúrok száma alapján azt, hogy a maszkon csak az egyik lábfej látszódik, vagy a két lábfej éppen összeér, és így nem alkotnak külön kontúrt.

Az út alakjából a releváns információk képfeldolgozási módszerekkel való leszűrése nem egyszerű. Ennek könnyítésére egy feltételezést tettünk, hogy a felhasználó mindig a menetirányba néz, és így a fölfele irány a maszkokon, mindig az előret jelenti. Azonban ez a feltételezés nem minden esetben valósult meg, mert egy vak személynek a menetirány pontos megtartása nem triviális. Az elfordulások elkerülésének érdekében, a telefon iránytű szenzorát is fölhasználtuk. Amennyiben a felhasználó túlságosan is elfordul a navigálás kezdeti irányától, visszajelzünk neki, hogy melyik irányba kell fordulnia, a menetirány helyreállításához. A navigálás kezdetét, a LetSee applikáción belül, az ösvénynavigáló funkció elindításától számítjuk.

A legtöbb változtatás után, az új verziót a Pilótával is teszteltük, a Cybathlon versenyen használt ösvénynek egy replikáján. A programot az ő visszajelzése és a mi megfigyeléseink alapján változtattuk. A programban voltak olyan számértékek, amelyeknek az értékét tetszőlegesen adjuk meg, és a tesztelések alapján finomhangoljuk őket. Ezek lesznek a továbbiakban a konstans paraméterek, amelyek a forrásfájl elején voltak elhelyezve, hogy könnyen átírhatóak legyenek.

#### **4.1. Visszajelzés az eltolódásról**

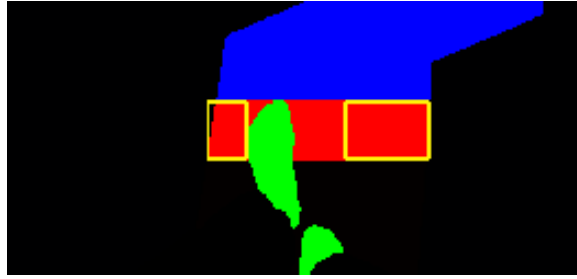
Amikor a fölhasználó az út valamelyik oldalsó széléhez túlságosan közel van, az alkalmazás visszajelez neki, hogy húzódjon a másik irányba. Ennek a visszajelzésnek egy feltétele az, hogy a lábfejek maszkján kettő kontúr látszódjon. Amennyiben ez nem teljesül, az eltolódásról nem adunk visszajelzést. Ez a visszajelzés csak azokra a képkockákra van tervezve, ahol a felhasználó egy

helyben áll. Azokon a képkockákon, ahol a felhasználó éppen lép, a térdét fölemeli, és ezzel a legtöbb esetben eltakarja az elől lévő lábfejét. Amennyiben csak egy lábfej látszódik a képen, biztos, hogy a felhasználó éppen egy lépés folyamatában van. Azokat az eseteket nem szűrjük le, ahol mind a két lábfej látszódik, de a kettő összeér, és így egy kontúrt alkot. Fölmerült problémaként, hogy a lábfejek maszkja tartalmaz kisebb, pár képpontból álló kontúrokat is, amelyek miatt a megtalált kontúrok száma nem egyezik meg, a képen látható lábfejek számával. Ennek megoldására, csak azokat a kontúrokat vesszük figyelembe, amelyeknek a területe egy tűréshatárnál nagyobb. Ez a tűréshatár a bemeneti kép területe alapján van kiszámítva, egy konstans paraméterrel való szorzással.

A következő lépésekben csak a felhasználó lábujjhegye alatti részét vizsgáljuk a képnek. Ennek a vizsgált résznek, a lábfejmaszk bekerítő téglalapjának  $y$  koordinátája lesz az alsó határa. Ez a koordináta a bekerítő téglalap tetejét határozza meg, amely a lábfej maszkon legfentebb lévő pontra illeszkedik. A vizsgált terület magassága, a bemeneti kép magasságának egy konstans paraméterrel megadott százaléka. Amennyiben ez a magasság kisebb, mint a kép lábujjhegy alatti részének magassága, az utóbbit használjuk. Az ösvénymaszkról készítünk egy, a vizsgált területre leszűkített másolatot, amelyen töröljük, a lábfejek bekerítő téglalapja alatti részt. Az így az kapott maszkon belül, csak a lábfejek oldalán lévő, vízszintesen szétválasztott részei maradnak meg az ösvénynek. A két oldalsó ösvény résznek megkeressük a kontúrjait.

A megtalált kontúrok közül, a kettő legnagyobbat választjuk ki. Ezeknek a kontúroknak vesszük a legkisebb területű téglalapját. Azt, hogy melyik téglalap melyik oldalhoz tartozik, egy-egy tetszőleges sarokpontjuknak  $x$  koordinátájából szűrjük le. Mivel a két kontúr vízszintesen szét van választva, így az összes pontjuknak az  $x$  koordinátája is szét van választva, és a tetszőleges sarokpont használata elfogadható. A legkisebb területű téglalapoknak a sarokpontjait rendezzük, az  $y$  koordináták szerint. A kettő legnagyobb  $y$  koordinátájú sarokpontból mindkét oldalt egy hosszúságot számolunk, ami a két pont  $x$  koordinátájának az abszolút értékben vett különbsége. Ez a két hossz a lábfejek két oldalán lévő ösvény részek szélességeinek felelnek meg.

A két oldal szélességét összevetjük, és ha valamelyik a másik oldal szélességénél többszörösen nagyobb, akkor visszajelzünk a felhasználónak. A visszajelzés csak a hosszabb szélességgel rendelkező oldal nevét közli, amely irányba való elhúzózkodással, a felhasználó újra az ösvény közepére kerül. A többszörösen nagyobb összehasonlítás alatt a másik oldal többszörösével való összehasonlítást értjük, ahol a használt szorzó egy konstans paraméter volt. A legutóbbi verzióban ez a konstans paraméter értéke 2 volt, így a felhasználónak „bal” visszajelzést például akkor adunk, hogy ha a bal oldali szélesség nagyobb, mint a jobb oldali szélesség kétszerese.



15. ábra. Teszt kirajzolás az ösvénykövetés feladatából, ahol a program a jobbra való elhúzózkodásra utasítja a fölhasználót. A sárga téglalapok a láb kétoldalán található útrészek bekerítő téglalapjai.

## 4.2. Visszajelzés a kanyarodásról

A feladat megoldásához elengedhetetlen az, hogy a felhasználó a kanyarodó falapokra tudjon lépni. Ehhez a felhasználónak visszajelzést adunk arról, hogy ha egy kanyar van előtte. A visszajelzésben megkülönböztetjük azokat az eseteket, ahol a kanyar közel, vagy távol van, azonban a logikának ezt a részét nem én implementáltam.

A kanyarodás kezelésénél, az ösvénynek csak a lábujjhegyek előtti részét vizsgáljuk. A vizsgált terület a kép tetejétől, a lábfej maszk bekerítő téglalapjának y koordinátájáig tart. Az ösvény maszkról készítünk egy másolatot, ami erre a vizsgált területre van leszűkítve. Ezen a másolaton kontúr keresést hajtunk végre, majd a maszkon minden képpontot nullázunk, és csak a legnagyobb kontúrt rajzoljuk újra. Amennyiben nem találtunk egy kontúrt sem, a kanyarodásról nem adunk visszajelzést, és a program tovább lép. A vizsgált terület maszkjának továbbá, töröljük az elejét, a kép teljes szélességében. A törölt rész magasságát a bemeneti magasságból úgy számoljuk, hogy a megtartott sorok száma, a kép magasságának egy konstans paraméterrel vett százaléka. Amennyiben a megtartott sorok száma 0, a kanyarodásról nem adunk visszajelzést. Az így kapott maszk, az ösvénynek a közvetlen felhasználó előtti részét tartalmazza.

Az ösvény kanyarodásának detekciójához, a felhasználó előtti ösvényt, vízszintes vonalakkal, egyenlő magasságú szeletekre osztjuk szét. Ezek az útszeletek a későbbi visszajelzésekben is használva lesznek. Az útszeleteknek továbbá a kontúrjait is megkeressük. Az ösvény irányának a kiszámításához, a kontúrok középpontjait mentjük el, amit az OpenCV-ben a kép momentumok használatával tudunk kiszámolni. A kanyar távolságának eldöntéséhez, az útszeletek bekerítő téglalapjait is fölhasználjuk. A Python prototípusban, a bekerítő téglalapok megkeresése előtt, az útszeletek kontúrapproximációval újra voltak rajzolva. Ez feltételezhetőleg zajszűrés miatt volt így, azonban a teszteléseim alapján, a bekerítő téglalapok nem változtak meg az újrarajzolás után, így ezt a lépést a C++ verzióban átugortam.

Az elmentett középpontok közül, a kettő legnagyobb y koordinátával rendelkező pontból irány

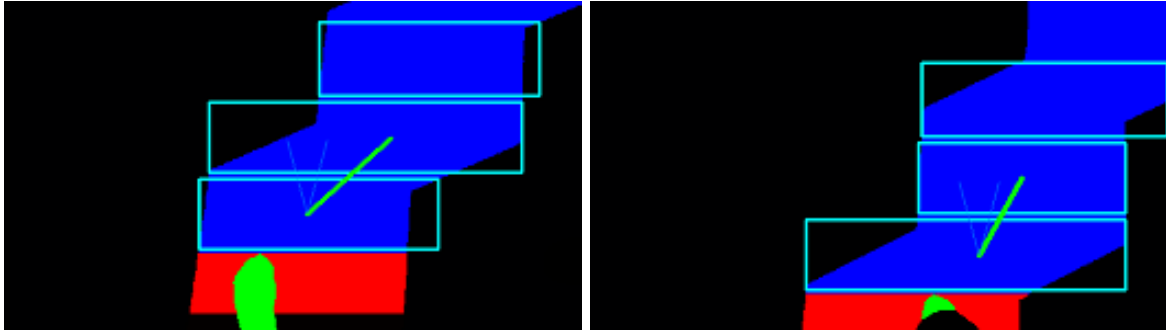
vektort képezzük, amelyből az út irányát tudjuk kiszámolni. Ezt az irányt úgy számítjuk, hogy a negatív szögek a balra kanyarodásnak, míg a pozitív szögek a jobbra kanyarodásnak feleljenek meg. Amennyiben a felhasználó pontosan az út irányába nézne, a kapott szög  $0^\circ$  lenne. A felhasználónak kanyarodásról akkor jelzünk vissza, hogy ha az irány szögének abszolút értéke nagyobb egy tűréshatárnál. Ezt a tűréshatárt egy konstans paraméterrel adtuk meg, ami a legutóbbi verzióban  $15^\circ$  volt.

Annak eldöntéséhez, hogy a kanyar rögtön, vagy kicsit távolabb van, a legnagyobb  $y$  koordinátájú útszeletek bekerítő téglalapját vizsgáljuk. A két téglalap között bal és jobb oldali különbséget számolunk. Itt, a bekerítő téglalapok  $x$  koordinátáinak különbsége tartozik a bal oldalhoz. A jobb oldali abszolút különbséghez, a bekerítő téglalapoknak  $x$  koordinátáihoz a szélességüket is hozzá adjuk, és azután számolunk belőlük különbséget. A a bal és jobb oldali különbségeknek az abszolút értékét vesszük. Ehhez a két különbséghez, az abszolútérték-vétel után, egy 1 képpontnyi epsilon érték is hozzá van adva, hogy az értékük ne legyen 0. Ennek jelentősége a kettő összehasonlításában van, amely osztással lett elvégezve. A felhasználónak akkor jelzünk vissza rögtön kanyarodásról, hogy ha a kanyarodás irányával megegyező oldal különbsége kisebb, mint a túloldal különbségének fele. Ennek az indoklására a 16. ábrán látható összeállítás lesz használva, ahol az ösvény jobbra kanyarodik. Amikor a kanyar távolabb van, akkor a legközelebbi útszelet, ami a legnagyobb  $y$  koordinátával rendelkezik, nem tartalmazza a kanyart, de az utána jövő útszelet igen. Ha a második útszelet jobbra kanyarodik, az utat abba az irányba eltolja, és a jobb oldali abszolút különbség megnő. Azonban a baloldalon, az útszeletek bekerítő téglalapjai továbbra is közel egyvonalban vannak, így a bal különbség közel 0 lesz. Ilyenkor, a kanyar irányának a különbsége, nem lesz a másik oldal különbségének a felénél kisebb, tehát távoli kanyarodásról jelzünk vissza. Azonban amikor a kanyarodást az első bekerítő téglalap tartalmazza, és az ösvény a második téglalapon belül egyenes, a két oldal különbsége megfordul. Ilyenkor a feltételek teljesülnek, és azonnali kanyarodásról jelzünk vissza a felhasználónak.

### 4.3. Visszajelzés az út végéről

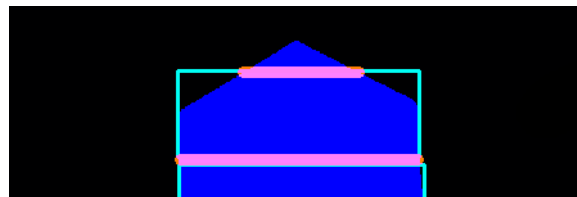
Az alkalmazásba bekerült egy ellenőrzés, amely az út végéről jelez vissza a felhasználónak. Ennek detekciójában azt használjuk ki, hogy az útelelemek végei háromszög alakúak. Amennyiben a felhasználó megtartja a menetirányt, az ösvény legtávolabbi része csakis akkor lehet a bemeneti képen háromszög alakú, hogy ha ott ér véget az út.

Ennél a visszajelzésnél a kanyarodás detektáláshoz kiszámolt útszeletek közül (4.2. fejezet), a képen legmagasabbat használjuk. Amennyiben az út vége kellően közel van a felhasználóhoz,



16. ábra. Teszt kirajzolások az ösvénykövetés feladatából, ahol a program a távoli (bal ábra), valamint az azonnali (jobb ábra) jobbra kanyarodásról értesíti a felhasználót. A cián téglalapok az útszeletek bekerítő téglalapjai, míg a zöld szakasz az ösvény irányvektora.

akkor az ebben az útszeletben lesz benne. Az útszeletnek vesszük a bekerítő téglalapját, és tesz-  
teljük, hogy az ösvény benne lévő része, fentről-lefele szűkül vagy sem. Ennek ellenőrzéséhez,  
kiszámítunk kettő téglalapot, amelyek szélessége az egész útszeletet átfedi, míg a magasságuk  
csak pár képpont. Az egyik ilyen téglalapot úgy adjuk meg, hogy annak felső oldala, a bekerítő  
téglalap felső oldalával egyezzen meg, míg a másikat úgy, hogy az alsó oldala, a bekerítő tég-  
lalap alsó oldalával egyezzen meg. A következő lépésben vesszük, az ösvény maszknak a felső  
téglalapra leszűkített részét, majd ennek a leszűkített résznek a bekerítő téglalapját vesszük. A  
bekerítő téglalap leszűkíti a vizsgált területet, a telített képpontokat tartalmazó részekre. Így, a  
kapott bekerítő téglalap felső oldala, az útszelet felső oldalának feleltethető meg. Ugyan ilyen  
módszerrel, kiszámoljuk az útszelet alsó oldalát is. Az így kapott szakaszokra a 17. ábrán látható  
példa.



17. ábra. Teszt kirajzolás az ösvénykövetés feladatából, ahol az útvégének detektálásában hasz-  
nált szakaszok láthatóak. A cián téglalapok az útszeletek bekerítő téglalapjai, míg a narancssárga  
szakaszok, az utolsó útszelet felső és alsó oldalai.

A felső és alsó oldalakból, bal és jobb oldali különbséget számolunk. A bal oldali különbség  
kiszámításánál, a felső oldalhoz tartozó téglalap x koordinátájából vonjuk ki, az alsó oldalhoz  
tartozó téglalap x koordinátáját. A jobb oldali különbség kiszámításánál, a téglalapok x koor-  
dinátáihoz a szélességüket is hozzáadjuk, és csak ezután vonjuk ki őket egymásból. Amennyiben  
az útszelet fentről-lefele szűkül, a bal oldali különbségnek pozitívnak, míg a jobb oldali külön-  
bségnek negatívnak kell lennie. Ezen felül, a két oldal különbség abszolút értékének összegét egy  
tűréshatárral vetjük össze, hogy annál nagyobb vagy sem. Erre azért van szükség, mert más-

különben 1 képpontnyi különbséget is az út végeként érzékelnénk. A tűréshatár a bemeneti kép szélességének egy százaléka, amelyet konstans paraméterrel adunk meg. Amennyiben ezek a feltételek teljesülnek, visszajelzünk a felhasználónak, hogy az út véget fog érni.

#### 4.4. Visszajelzés az útelelemek csatlakozásáról

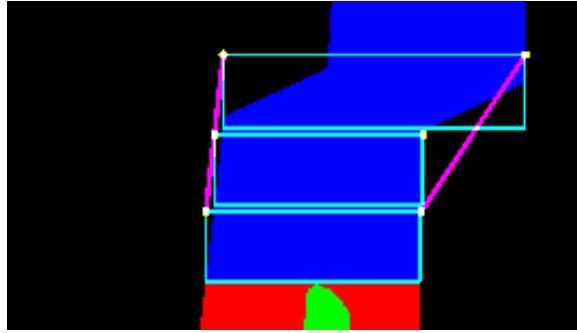
Fölmerült az, hogy a kanyarodásról szóló visszajelzés (4.2. fejezet) nem elég ahhoz, hogy a felhasználó ne lépjen le az ösvényről. Tesztelve lett egy visszajelzés azokra a helyzetekre, ahol a felhasználó közvetlen az útelelemek csatlakozása előtt van. A lelépés megakadályozására egy másik visszajelzést is implementáltunk, amely a 4.5. fejezetben lesz részletezve. Ez a visszajelzés hasznosabbnak bizonyosult, így az útelelemek csatlakozásának visszajelzését később kikapcsoltuk.

Az útcsatlakozás esetében, a kanyarodás detektálásakor kiszámolt útszeletek (4.2. fejezet) bekerítő téglalapjait vizsgáljuk. Az elképzelés az volt, hogy amikor az út kanyarodik, a bekerítő téglalapok oldalai nem esnek egy egyenesre. Azonban ez akkor is teljesülhet, hogy ha a kanyar, csak a felhasználótól legtávolabb lévő útszeletben látszódik. Ezért, az útelelemek csatlakozását csak akkor vizsgáljuk, hogy ha bármelyik útszelet bekerítő téglalapja szélesebb, mint a közvetlen fölötte lévő útszelet bekerítő téglalapja. Amennyiben ez nem teljesül, a detekció többi része nem fut le, és az útcsatlakozásról nem adunk visszajelzést. Ennek az ellenőrzésnek a hibája az, hogy akár 1 képpontnyi különbség is elég ahhoz, hogy igaz legyen, így utólagosan jobb lett volna egy tűréshatárt használni.

Az útszeletek bekerítő téglalapjainak a bal- és jobb felső sarkait, két külön listába helyezzük. Az egyvonalba eséshez írtunk egy segédfüggvényt, amelyet mind kettő listán meghívunk. Hogy ha az egyvonalba esés bármely oldalra nem teljesül, jelzünk a felhasználónak, hogy egy útcsatlakozás előtt áll. Mivel egy vonal fölírásához kettő pontra van szükség, így a függvényt csak akkor hívjuk meg, hogy ha három útszeletet sikerült találnunk. Ezt a segédfüggvényt a Python prototípus alapján implementáltam, és azt vizsgálja, hogy ha a megadott pontokat fölrajzoljuk, és egy adott vastagságú vonal alatt töröljük a képet, akkor marad-e telített képpont a képen. A vonalat mindig a listában legelső és legutolsó pontból húztuk, amely a legfelső és legalsó bekerítő téglalapok sarokpontjainak feleltek meg. A vonal vastagsága itt egy tűréshatár, ami a bemeneti kép szélességének egy százaléka, konstans paraméterrel megadva.

#### 4.5. Visszajelzés a megállásról

Az ösvénykövetés feladatát a Cybathlon versenyen a Pilóta akkor bukja el, hogy ha lelép az útról. Ezért implementálva lett egy visszajelzés arra, hogy ha a felhasználó a következő lépéssel letérne az ösvényről.



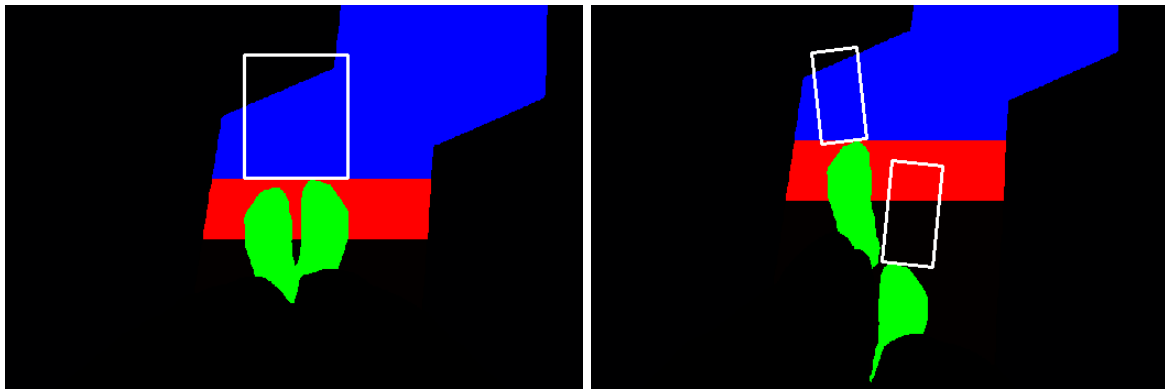
18. ábra. Teszt kirajzolás az ösvénykövetés feladatából, ahol az útsatlakozás látszódik. A cián téglalapok az útszeletek bekerítő téglalapjai, a sárga pontok azoknak felső sarkai, míg a magenta szakaszt a legelső és legutolsó sarokpontok alkotják.

Ehhez, a 4.1. fejezetben kiszámolt lábfej kontúrok listáján iterálunk végig. Minden egyes lábkontúr esetén, előre vetítünk egy téglalapot, és megvizsgáljuk azt, hogy annak területe hány százalékban lóg ki az ösvényről. Az ösvény maszkján, a lábfejek által eltakart rész nem feltétlenül telített, így ennél az ellenőrzésnél a lábfejeket is az ösvény részeként kezeljük. Az egyes lábfej kontúrokat attól függően máshogy kezeljük, hogy egy darab lábfejet, vagy két összeérő lábfejet ábrázolnak. Amennyiben kettő vagy több lábfej kontúr látható a képen, feltételezhető, hogy a felhasználó lábfejei nem érnek össze, és a két legnagyobb kontúrt vizsgáljuk, mint egy-egy lábfejet. Amikor csak egy lábfej kontúrt találtunk, megvizsgáljuk, hogy összetett lábfejeket tartalmaz vagy sem. Ehhez, a kontúr bekerítő téglalapjának a magasság-szélesség arányát vesszük. Hogy ha ez az arány egy tűréshatárnál kisebb, akkor a kontúrt túl szélesnek tekintjük, és összetett lábfejként kezeljük. Ez a tűréshatár egy konstans paraméterrel volt megadva, amely a legutóbbi verzióban 1.5 volt. Amennyiben a feltétel nem teljesül, feltételezzük, hogy a felhasználó másik lábfejet a térde eltakarja, és a megtalált kontúrt ugyan úgy kezeljük, mint egy egyedüli lábfejet.

Amennyiben egy egyéni lábfej kontúrt vizsgálunk, az előre vetített téglalap kiszámításához a kontúr legkisebb területű téglalapját használjuk föl, amely egy elforgatott téglalap. Ennek a téglalapnak ismerjük a sarokpontjainak koordinátáit, azonban nem tudni, hogy melyik koordináták melyik sarkokhoz tartoznak. Ennek szűréséhez, a téglalap legnagyobb  $y$  koordinátájával rendelkező pontját keressük meg, amely a képen a legalacsony sarokpont lesz. A többi sarokpont közül, a hozzá kettő legközelebbit is megkeressük, ugyanis tudni, hogy a tőle legtávolabbi sarokpont lesz a vele átlósan szemben lévő. Ismerve három egymástkövető pontot, a lábfej legkisebb területű téglalapjának két oldalvektora előállítható. Ezek közül a hosszabbik vektort tekintjük a menetirányba néző vektornak. Azonban ez a feltételezés nem mindig igaz, ugyanis van olyan eset, ahol a felhasználó térde a lábfejét csak részlegesen takarja el, és annak a menetiránnyal megegyező irányú oldala rövidebb lesz, mint a másik. Ilyen esetekben, a téglalapot valamilyen oldalirányba vetítenénk, nem előre. Ezért azokban az esetekben nem vetítünk előre, ahol

a hosszabbik oldalvektor  $y$  koordinátájának abszolút értéke kisebb vagy egyenlő, az  $x$  koordinátájának az abszolút értékénél. Ez annak felel meg, hogy a képen a tökéletesen fölfele iránytól, a vektor iránya legalább  $45^\circ$ -al tér el. Az előre vetített téglalap szintén egy elforgatott téglalap lesz, amelynek létrehozásához az OpenCV-nek három egymást követő sarokpontra van szüksége. A legalacsonyabb sarokpontból és a két oldalvektorból, a legkisebb területű téglalapnak a lábujjhegyekhez illeszkedő szakasza megkapható. A harmadik pont a menetirány vektorával kapható meg, amelynek hossza az előre vetített téglalap hosszát is befolyásolja. Emiatt az összeadás előtt, a hosszabbik oldalvektort normalizáljuk, és megszorozzuk a kívánt hosszal. Ez a hossz, a lábujjhegyre illeszkedő szakasz hosszának többszöröse. Ez azért lett így megválasztva, mert a tesztelések alapján, a lábfejek hossza konzisztens volt. Ezt a többszöröst egy konstans paraméterrel adjuk meg, amely a legutóbbi verzióban 2.0 volt. A visszajelzéshez az ezzel a három ponttal alkotott, elforgatott téglalapot vizsgáljuk, hogy a területe hány százalékban lóg ki az ösvényről. Erre az előre vetített téglalapra a 19. ábrán látható példa.

A tapasztalat az volt, hogy az összetett lábfejek kontúrján, a legkisebb területű téglalap iránya nem volt konzisztens. Emiatt, az ilyen kontúrok előre vetített téglalapjainak kiszámításához, bekerítő téglalapot használtunk, amelynek oldalai vízszintesek és függőlegesek. A kontúr bekerítő téglalapjának az  $y$  koordinátájából kivonjuk, a téglalap magasságát, ezzel megkapva egy ugyanolyan méretű, a képen fentebb lévő téglalapot. Erre az előre vetített téglalapra a 19. ábrán látható példa.



19. ábra. Teszt kirajzolás az ösvénykövetés feladatából, a megállás visszajelzéséhez használt előre vetített téglalapokról, összetett (bal ábra) és egyéni (jobb ábra) lábkontúrok esetén. Az előre vetített téglalapok fehér színűek, azokon belül van vizsgálva, hogy hány százalékuk van az ösvényen.

A lábkontúrok vizsgálatánál minden esetben egy olyan maszkot állítunk elő, amelyben az előre vetített téglalapon belül az ösvényről kilógó képpontok telítettek csak. Ennek előállításához az OpenCV bináris operátorait használjuk, a 2. egyenletnek megfelelően, ahol az  $S$  a lelógó részeket tartalmazó kép, az  $R$  az előre vetített téglalap képe, míg az  $M_p$  és  $M_f$  az ösvény és

lábfejek maszkjai.

$$S = \overline{M_p \cup M_f \cup R} \quad (2)$$

Az  $S$  képen megszámloljuk, a telített képpontokat, és kiszámoljuk, hogy az előrevetített téglalapnak hány százalékát teszik ki. Hogy ha ez a százalék meghaladja a konstans paraméterrel megadott tűréshatárt, amely a legutóbbi verzióban 10% volt, akkor úgy tekintjük, hogy a felhasználónak meg kell állnia. Ilyenkor, a vizsgált kontúr bekerítő téglalapjának az  $x$  koordinátáját egy listában tároljuk el. Amennyiben ez a feltétel nem teljesül, a bekerítő téglalap  $x$  koordinátáját egy összehasonlítási változóban tároljuk el.

Miután az összes lábkontúron végigiteráltunk, megvizsgáljuk az eltárolt  $x$  koordinátákat. Hogy ha két  $x$  koordináta lett eltárolva, visszajelzünk a felhasználónak, hogy mindkét lábával álljon meg. Amennyiben csak egy  $x$  koordinátát tároltunk el, de két kontúrt vizsgáltunk, az összehasonlító  $x$  koordináta alapján döntjük el, hogy a felhasználó a bal vagy jobb lábával álljon meg. Ha csak egy kontúrt vizsgáltunk, de annak az  $x$  koordinátáját eltároltuk, akkor visszajelzünk, hogy az egyik lábbal kell megállni. Az összetett és eltakart lábfejek között a visszajelzésben nem teszünk megkülönböztetést. Amennyiben az  $x$  koordináták listája üres, a felhasználót nem utasítjuk megállásra.

## 5. Összefoglalás

A dolgozat két olyan újszerű megoldást mutat be, amely a látássérültek számára mindennapi feladatok megoldását tehetik könnyebbé. A LetSee applikációba integrált színfölismerő és ösvénykövető modulok működését ismertettem a dolgozatomban.

A színfölismerés funkció előre megállapodott referencia színek, egyesével való fölismerésére ad megoldást, a telefon RGB kameráját használva. Ahhoz, hogy a külső fényhatások a színösszehasonlítás eredményét ne torzítsák, a referencia színeket egy fizikai színkártyáról olvassuk be, a vizsgált objektumhoz közel. Mivel mind a vizsgált színre és a referencia színekre, közel ugyanolyan fényhatások hatnak, a kettő közötti összehasonlítás nem lesz torzítva. A színkártyának a kameraképen belüli megtalálásához, a négy sarkára egy-egy ArUco jelölést helyeztünk el. Továbbá, hogy a kártya vizsgált területén könnyen lehessen dolgozni, az ArUco jelölések sarokpontjai alapján homográfia transzformációt hajtunk létre, és a kártya releváns területéről egy téglalap alakú képet alkotunk. A pontosabb összehasonlításért a CIEDE2000 színtávolságot használjuk, és ehhez a színek leolvasása előtt, a téglalap alakú képet a CIE 1976  $L^*a^*b^*$  színtérbe konvertáljuk át. Ezen a téglalap alakú képen olvassuk le a vizsgált objektum színét, több lekér-

dezési pontból, amelyekből hierarchikus klaszterezéssel állítjuk elő, a lekérdezett színeket. Ezek a lekérdezett színek aztán, az összes referencia színnel össze vannak hasonlítva, és a legkisebb színtávolsággal rendelkező pár referencia színét adjuk vissza a felhasználónak, a színtávolsággal együtt. Ez a megoldás nem igényel különleges hardvert, hanem egy hétköznapi okotelefonnal és egy, akár papírra nyomtatott színkártyával képes, a színek robosztus fölismerésére, és így a mindennapi fölhasználásra is könnyen adoptálható.

Az ösvény követés feladatában a telefon LIDAR szenzorából kapjuk a bemeneti képet, hogy minimalizáljuk a szín- és megvilágításkülönbségekből adódó varianciát. Ez a kép egy YOLO modellnek van átadva, amelytől az útnavigáló modul az ösvény, lábfej és talaj maszkokat kapja meg. Az alkalmazás feltételezi, a pálya menetirányának megtartását is, amelyet a telefon iránytű szenzora alapján ellenőrzünk is, és elfordulás esetén a felhasználót a megfelelő irányba igazítjuk. Az ösvény maszkja, a lábujjhegyek mentén vízszintesen szét van választva. A lábujjhegyek alatti részből az eltolódást vizsgáljuk, úgy, hogy azon a lábfejek bekerítő téglalapját kitöröljük, és a megmaradt két oldalsó rész szélességét vessük össze. Amennyiben valamelyik szélesség túlságosan nagyobb a másiknál, a felhasználót az út közepére igazítjuk. Az ösvény lábujjhegyek előtti részét három szeletre osztjuk szét, és azok közül a felhasználóhoz kettő legközelebbinek a középpontját összekötjük, hogy az ösvény kanyarodási irányát kiszámoljuk. Ezeknek az útszeleteknek a széléből azokat az eseteket is megkülönböztetjük, ahol az ösvény közvetlen közel, vagy még távolabb kanyarodik, és a felhasználót ennek megfelelően értesítjük. A legutolsó útszelet alakját vizsgálva, meg tudjuk állapítani azt, hogy az ösvénynek a vége látható, abból, hogy az útszelet háromszög alakú vagy sem. A lelépés elkerüléséhez, a felhasználó lábfejei elé, egy téglalapot vetítünk, amelyben megvizsgáljuk, hogy előre lépés esetén lelépne az útról, vagy sem. Az utóbbi esetekben a felhasználót megállásra utasítjuk.

A dolgozatban tárgyalt megoldások több szenzorra épülnek, azonban különleges hardver helyett, a LetSee applikáció, amelybe a modulok integrálva lettek, egy széleskörben elérhető és használt iPhone készüléken futott.

## Hivatkozások

- [1] Tanács Attila. *Színterek*. Magyar. URL: [https://www.inf.u-szeged.hu/%7Etanacs/pyocv/szntr\\_konverzik.html](https://www.inf.u-szeged.hu/%7Etanacs/pyocv/szntr_konverzik.html) (elérés dátuma 2024. 05. 27.).
- [2] 17-23-076 CIE 1976  $L^*a^*b^*$  colour space. Angol. URL: <https://cie.co.at/eilvterm/17-23-076> (elérés dátuma 2024. 05. 13.).
- [3] *Color conversions*. Angol. URL: [https://docs.opencv.org/4.x/de/d25/imgproc\\_color\\_conversions.html#color\\_convert\\_rgb\\_lab](https://docs.opencv.org/4.x/de/d25/imgproc_color_conversions.html#color_convert_rgb_lab) (elérés dátuma 2024. 05. 26.).
- [4] 17-23-015 colour difference, <psychophysical>. Angol. URL: <https://cie.co.at/eilvterm/17-23-015> (elérés dátuma 2024. 05. 13.).
- [5] Dr. Németh Gábor. *Távolság-metrikák*. Magyar. URL: [https://www.inf.u-szeged.hu/%7Egnemeth/kurzusok/igazsagugyi\\_kepelemzes/osztalyozas/tvolsgmetrikk.html](https://www.inf.u-szeged.hu/%7Egnemeth/kurzusok/igazsagugyi_kepelemzes/osztalyozas/tvolsgmetrikk.html) (elérés dátuma 2024. 05. 27.).
- [6] G. Sharma, W. Wu és E. N. Dalal. „The CIEDE2000 Color-Difference Formula: Implementation Notes, Supplementary Test Data, and Mathematical Observations,” angol. *Color Research and Application* 20.1 (2005. febr.), 21–30. old. URL: <https://hajim.rochester.edu/ece/sites/gsharma/ciede2000/> (elérés dátuma 2024. 05. 26.).
- [7] *What is LiDAR?* Angol. URL: <https://www.ibm.com/topics/lidar> (elérés dátuma 2024. 11. 13.).
- [8] Joshua Noble. *What is hierarchical clustering?* Angol. URL: <https://www.ibm.com/think/topics/hierarchical-clustering> (elérés dátuma 2024. 11. 15.).
- [9] *OpenCV - Open Computer Vision Library*. Angol. URL: <https://opencv.org/> (elérés dátuma 2024. 05. 26.).
- [10] *Structural Analysis and Shape Descriptors*. Angol. URL: [https://docs.opencv.org/4.x/d3/dc0/group\\_\\_imgproc\\_\\_shape.html](https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html) (elérés dátuma 2024. 11. 13.).
- [11] Sergio Garrido és Alexander Panov. *Detection of ArUco Markers*. Angol. URL: [https://docs.opencv.org/4.x/d5/dae/tutorial\\_aruco\\_detection.html](https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html) (elérés dátuma 2024. 05. 13.).
- [12] *Basic concepts of the homography explained with code*. Angol. URL: [https://docs.opencv.org/4.x/d9/dab/tutorial\\_homography.html](https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html) (elérés dátuma 2024. 05. 27.).
- [13] *Geometric Transformations of Images*. Angol. URL: [https://docs.opencv.org/4.x/da/d6e/tutorial\\_py\\_geometric\\_transformations.html#autotoc\\_md1375](https://docs.opencv.org/4.x/da/d6e/tutorial_py_geometric_transformations.html#autotoc_md1375) (elérés dátuma 2024. 05. 27.).

- [14] *What is image segmentation?* Angol. URL: <https://www.ibm.com/topics/image-segmentation> (elérés dátuma 2024. 05. 28.).
- [15] *Instance Segmentation.* Angol. URL: <https://docs.ultralytics.com/tasks/segment/> (elérés dátuma 2024. 05. 27.).
- [16] *Color Constancy - APA Dictionary of Psychology.* Angol. URL: <https://dictionary.apa.org/color-constancy> (elérés dátuma 2024. 11. 14.).
- [17] Greg Fiumara és Michael. *CIE Delta E 2000 (CIEDE2000).* Angol. URL: <https://github.com/gfiumara/CIEDE2000> (elérés dátuma 2024. 05. 22.).