



KÉPFELDOLGOZÁS

Közeledési tábla felismerés

Csanda Norbert

MWW7MI

(GKNB_INTM152)

2024/25/2

https://github.com/csanda7/road_sign_recognition

Tartalom

Projekt leírása.....	3
Program elindítása	3
Elméleti összefoglaló	4
Használt adathalmaz	4
Használt algoritmusok	4
HSV színtér	6
Fejlesztői dokumentáció	7
Rendszer Felépítése	7
Mappa- és fájlszerkezet.....	7
Működés leírása	7
Tesztek	10
További teszt eredmények	13
Források	14

Projekt leírása

A képfeldolgozási projektem témája közlekedési táblák típusának felismerése. A projektben 6 különböző típusú táblát különböztetek meg egymástól:

- Alárendelt utak kereszteződése
- 80km/h legnagyobb megengedett sebesség
- 100km/h legnagyobb megengedett sebesség
- Főútvonal
- STOP
- Körforgalom

A program Pythonban íródott, OpenCV, Numpy, OS és Random modulok használatával.

A program egy bemeneti kép (.jpg, .jpeg vagy .png kiterjesztések) alapján megvizsgálja a kép meghatározó pontjait, majd a tanításhoz használt példaképekkel összeveti. Ha van egyezés és felismerhető a program számára a jelzőtábla, akkor a konzolra kiírja a tábla nevét, egy értéket, hogy milyen konfidencia-szinttel ítéli ezt meg, illetve a bemeneti képet megjeleníti. Ennek a megvalósításához két feature matching algoritmust próbáltam ki a SIFT-et és az OpenCV-be alapértelmezettként megtalálható ORB-ot.

Program elindítása

1. A repository klónozása

```
git clone https://github.com/csanda7/road\_sign\_recognition
```

2. A függőségek telepítése

```
pip install requirements.txt
```

3. Kicsomagolni a DATA.zip, illetve a TEST_DATA.zip fájlt a főkönyvtárba
4. Elindítani a programot terminálból

```
cd .\src\  
python main.py
```

5. A teszteléshez a teszt programot indítsuk el

```
cd .\src\  
python test.py
```

Elméleti összefoglaló

Használt adathalmaz

A használt tanító adathalmaz a következőképp épül fel:

Tábla típusa	Tanításhoz használt képek száma
Alárendelt utak kereszteződése	17
100-as sebességkorlátozó tábla	25
80-as sebességkorlátozó tábla	15
Főútvonat	19
STOP tábla	19
Körforgalom	23

1. táblázat

A teszteléshez összesen 119 darab képet használtam. Ez nem feltétlen a jó felosztása a tanító, illetve a teszt képek száma szempontjából, de relatív jó eredményt ad vissza a program.

Használt algoritmusok

A projektet egyszerű megoldás alapján kezdtem el, ami csak szín és a tábla formája alapján állapított meg táblákat és próbálta besorolni bizonyos osztályokba, ami, ha jól lett volna kivitelezve, akkor lehetséges, hogy erre a pár tábla típusra jól működött volna, de ha bővíteni szeretném vagy több táblával szeretném tesztelni, akkor ez kevésbé megoldható, hogy jól működjön.

Ezt orvosolva kiválasztottam egy jellemző pont felismerő algoritmust a SIFT-et, amely ehhez a projekthez jól illik. Ezen algoritmus működése a következőképp működik:

Az eredeti képre többször különböző mértékekben Gauss-szűrőt alkalmaz, majd ezen képek különbségeit kiszámolja. Ezzel meghatároz olyan pontokat, amik különböző méreteken is jól észlelhetők.

A Gauss-szűrő futtatása után lokálisan keresi a kulcspontokat a képen, úgy, hogy ha talál lokális minimumot vagy maximumot az egy kulcspont lesz. Ezeket finomítani kell, ha zajos vagy instabil a pont. Az alacsony kontrasztú és él menti pontokat eltávolítja. Taylor-sorral közelítést alkalmaz, hogy a kulcspontokat pontosítsa.

Az így meghatározott kulcspontokhoz gradiens irányt és nagyságot számít. Meghatároz egy domináns orientációt, amiből, ha több van akkor többet határoz meg. Ezt azért teszi, hogy ha a objektum nem ugyanolyan szögben van, vagy el van forgatva, akkor is meg tudja határozni a kulcspontok alapján.

Létrehoz keypoint descriptorokat, amik a kulcspontok környezetét írják le. Ezek segítenek a detektálásban.[1]

Ezzel átlagban kedvező eredményeket értem el, íme néhány példa:

Tesztkép	Detektált tábla típus	Detektált domináns szín	Konfidencia szint	Helyes-e a detektálás?
00014_00015_00027.png	STOP	Piros	78.57%	igen
00005_00016_00025.png	80km/h	Piros	90.00%	igen
00011_00012_00027.png	100km/h	Piros	90.62%	nem
00040_00011_00018.png	Körforgalom	Kék	88.24%	igen
00007_00002_00024.png	100km/h	Piros	87.5%	igen
00014_00007_00026.png	STOP	Piros	94.23%	igen

2. táblázat – A SIFT algoritmus teszt eredményei

Nem hibátlan a működés, de nagy arányban helyesen ismeri fel a táblák típusát, illetve színét. A táblázatban megjelenő helytelen detektálásnál 100km/h táblának detektált egy igazából alárendelt utak kereszteződése táblatípust, nagyon magas konfidenciaszinttel. A tábla formája, illetve a táblán jellemző szimbólum sem hasonló a detektált táblához. Több tanítókép hozzáadásával valószínűleg elkerülhetők az ezen féle hibák.

A SIFT algoritmus mellett megvizsgáltam az ORB algoritmussal is, aminek a működése és az eredmények is különböznek a SIFT-től.

Először az ORB a FAST algoritmust használja, hogy gyorsan megtalálja a kulcspontokat. A FAST nagyon gyors, de nem ad információt arról, melyik pont mennyire "jó". Ezért az ORB minden pontnak kiszámolja a Harris-corner értékét, ami megmutatja, mennyire stabil és informatív a pont. A legjobb pontokat választja ki.

Ezután minden kulcspont környezetében meghatározza a domináns irányt, hogy az ORB forgatásfüggetlen legyen. Ezt egy gradiens-alapú módszerrel számítja ki.

A következő lépésben jönnek a descriptorok. Az ORB a BRIEF leírókat használja, amik nagyon gyorsak, mert bináris vektorokkal írják le a kulcspont környezetét. Viszont a BRIEF eredetileg nem forgatásfüggetlen, úgyhogy az ORB egy trükköt használ: elforgatja a BRIEF tesztpontjait a kulcspont iránya szerint.

Végül, amikor két képet hasonlít össze, az ORB a bináris leírók Hamming-távolságát nézi meg, ami nagyon gyors számítást tesz lehetővé.[2]

Érdekesekek voltak az eredmények ezzel az algoritmussal kapcsolatban, hiszen több képet egyáltalán nem ismer fel, de amit igen, azt rend szerint nagyobb konfidenciaszinttel, mint a SIFT.

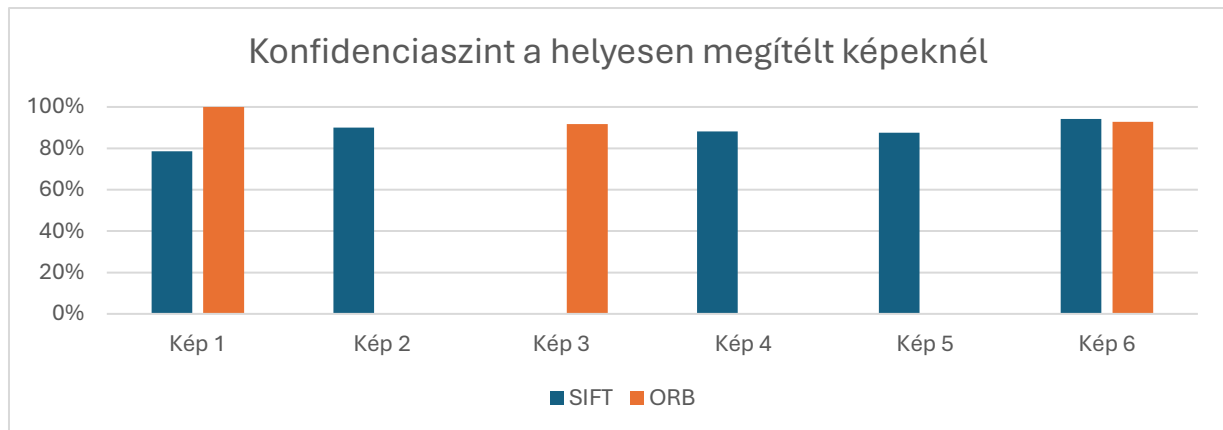
Íme az ORB teszt eredményei ugyanazokra a képekre, amit korábban a SIFT algoritmussal teszteltem:

Tesztkép	Detektált tábla típus	Detektált domináns szín	Konfidencia szint	Helyes-e a detektálás?
00014_00015_00027.png	STOP	Piros	100%	igen
00005_00016_00025.png	Nem azonosítható	Piros	0.00%	nem
00011_00012_00027.png	Alárendelt utak k.	Piros	91.67%	igen
00040_00011_00018.png	Nem azonosítható	Kék	0.00%	nem
00007_00002_00024.png	Nem azonosítható	Piros	0.00%	nem
00014_00007_00026.png	STOP	Piros	92.73%	igen

3. táblázat - Az ORB algoritmus teszt eredményei

Látható, hogy ez az algoritmus nem azonosított be tévesen táblát, viszont sokkal többet nem ismert fel. Azok a táblák, amiket felismert, mind 90% fölötti konfidenciaszinttel állapított meg.

Konkrétan összehasonlítva ezen 6 darab példa esetét:



1. ábra – A két teszt konfidenciaértékeinek összehasonlítása

Összegezve, az ORB algoritmus abból a szempontból megbízhatóbbnak bizonyult, hogy tévesen nem azonosított be táblát, viszont a SIFT algoritmus több tábla helyes detektálását volt képes elvégezni.

HSV színtér

A domináns szín meghatározása miatt átkonvertáltam a képet HSV színtérbe, mert a szín (hue) elválik a világossági értéktől és a telítettségtől. Az RGB színtérben a fényerő keveredik a színinformációval (árnyékban lehet valamit bordónak látszik, ami valójában piros). A HSV színtérbe az árnyék és a fényviszonyok csökkentése kevésbé torzítja a színt.

Mivel a közlekedési táblák az időjárás viszonyosságai miatt több fényviszonyban láthatók a képeken, ezért a HSV színtérben az előbb említett okok miatt könnyebb őket felismerni.



2. ábra – HSV színtérbe átkonvertált kép

A TQDM könyvtárat azért használtam, mert a tesztek futtatásakor egyszerűbben tudtam követni a folyamatot, hogy hol tart a tesztelés. Ez egy egyszerű progress bar illeszt be a konzolra és adatokat ír ki, hogy a képek hányad részén hajtott végre a detektálást. A terminál konzoljában ez a következőképp jelenik meg:

```
PS D:\Programs\Képfeldolgozás\road_sign_detection> cd .\src\
PS D:\Programs\Képfeldolgozás\road_sign_detection\src> python test.py
Képek vizsgálata: 100%|████████████████████████████████████████| 119/119 [00:27<00:00, 4.32img/s]
Vizsgálat befejeződött.
Eredmények mentve az output mappába.
PS D:\Programs\Képfeldolgozás\road_sign_detection\src> 
```

3. ábra – A progress bar a terminál konzoljában

Fejlesztői dokumentáció

Rendszer Felépítése

A program két fő Python-fájlból áll:

- `main.py`: A fő vezérlőszkript, amely végrehajtja az adatok betöltését, jellemzők kinyerését, illesztést és az eredmények kiértékelését.
- `utils.py`: A segédfüggvényeket tartalmazó modul, amely külön kezeli a SIFT és ORB jellemzők kinyerését és illesztését, valamint a színfelismerést és referenciaképek betöltését.

A programnak van egy teszteléshez használt szkriptje:

- `test.py`: A két használt algoritmus összehasonlításához készített szkript, ami grafikonon ábrázolja az átlagos konfidenciaszintet, illetve a felismert táblák darabszámát (az output mappába helyezi el a grafikonokat).

Mappa- és fájlszerkezet

```
ROAD_SIGN_DETECTION
- DATA (Tanítóképek táblatípusonként)[3]
    -80km/
    -100km/
    -...
    -stop/
- src
    -main.py (Főprogram)
    -utils.py (Függvények)
    -test.py (Tesztelő program)
- TEST_DATA (Tesztképeket tartalmazó mappa)
    -képl.png
    -...
- requirements.txt (Telepítendő függőségek listája)
```

Működés leírása

A főprogramban a legelső lépésként definiáltam, hogy melyik mappában milyen táblatípusok vannak. Ezt a `folder_sign_types` változóban tároltam el dictionaryként.

Key	Value
<code>alarendelt_utak</code>	Alarendelt utak kereszteződése
<code>100km</code>	100-as sebességhatároló tábla
<code>80km</code>	80-as sebességhatároló tábla
<code>fout</code>	Főútvonal
<code>stop</code>	STOP tábla
<code>korforgalom</code>	Körforgalom

4. Táblázat

A referenciaadatok és tesztadatok mappájának elérését megadtam a programnak, ahol az almappákban talált png, jpg és jpeg formátumú fájlokat keresi meg.

A program véletlenszerűen kiválaszt egy képet, amit betölt a programnak. Ha ez nem sikeres, akkor természetesen hibaüzenetet kap a felhasználó.

Következő lépésben a képet konvertáljuk HSV színtérbe, majd maszkolást hajtunk rajt végre. Az 5. ábrán látható, hogy a kép a HSV színtérbe konvertálás után, hogy néz ki.



4. ábra – Transzformáció előtti kép



5. ábra – HSV színtérbe konvertált kép

A `MIN_MATCH_COUNT = 10` és `RATIO_THRESH=0.85` értékekkel láttam el. Ezen hiperparaméterek szerepe a programban a következő:

`MIN_MATCH_COUNT = 10` azt határozza meg, hogy 10 darab jó egyezést kell a programnak találnia minimum a tesztkép és referenciakép között, hogy homográfiát számoljon. A 10 érték bizonyult a legjobban működőnek. Ha ennél alacsonyabb értékre állítjuk, akkor kisebb hasonlóság alapján is számítana homográfiát, ami nagyobb hiba értékkel dolgozna.

`RATIO_THRESH=0.85` feladata, hogy a kulcspontok egyezése közül melyiket tekintjük „jó” egyezésnek. Ez a Lowe-féle arányszűrő küszöbértéke. Ha `m.distance` kisebb, mint `0,85*n.distance` egy „m” és egy „n” vizsgált egyezés alapján, akkor az `m` elfogadható jó egyezésnek. Ezen érték csökkentése tehát csak nagyon "biztos" egyezéseket enged meg, így a találatok minősége nő, de kevesebb találat lesz.[4]

A SIFT alapú algoritmus lefutása úgy történik, hogy a vizsgált tesztképből kinyerjük a kulcspontokat és a hozzájuk tartozó leírókat (descriptors), amelyek jellemzik a kép egyedi pontjait. Ezek a descriptorok fogják képviselni a tesztképet a további egyezéskeresés során. Ezután beállítjuk azokat a változókat, amelyek a legjobb egyezés eredményeit fogják tárolni. A `best_match_type_sift` azt jelzi, hogy melyik táblatípus volt a legjobb egyezés, a `best_inliers_sift` azt tárolja, hogy hány "inlier" volt (azok az egyezések, amelyek jól illeszkednek egy geometriai modellhez), illetve a `best_good_matches_sift` a legjobb egyezések listája.

Ha a referencia és a tesztkép között elég sok egyezés található, akkor megpróbáljuk kiszámolni a két kép közötti homográfiát. A homográfia egy olyan geometriai transzformáció, amely megmutatja, hogyan lehet a tesztképet és a referencia képet egymásra illeszteni. Ez különösen hasznos, mert a táblákról készült képek különböző szögekből és távolságból készültek.

Ha sikerült homográfiát számítani, megnézzük, hány egyezés illeszkedik jól erre a geometriai modellre. A `mask` megmondja, hogy a jó egyezések közül melyek az inlierek. Azok a párosítások számítanak "jó illeszkedésnek", amelyek nemcsak hasonló pontokat találtak, hanem a képek közötti átalakításra is illeszkednek.

Ezután megnézzük, hogy az aktuális referencia kép több inliert adott-e, mint az eddigi legjobb. Ha igen, frissítjük a legjobb egyezést: eltávolítjuk a táblatípust, az inlierek számát és a jó egyezések listáját.

Miután minden referencia képet összehasonlítottunk, ellenőrizzük, sikerült-e bármelyik táblatípust felismerni. Ha igen, a `folder_sign_types` dictionary segítségével kiolvassuk az azonosító szám alapján a tábla nevét, és kiszámítjuk a konfidencia értékét. A konfidencia azt fejezi ki, hogy az inlierek aránya milyen magas az összes jó egyezéshez képest. Ha nem találtunk megfelelő egyezést, akkor "Nem azonosítható" üzenetet ad vissza, és a konfidencia 0.

```
test_keypoints_sift, test_descriptors_sift = extract_sift_features(image)
best_match_type_sift = None
best_inliers_sift = 0
best_good_matches_sift = []

for sign_type, ref_list in reference_data_sift.items():
    for ref in ref_list:
        ref_keypoints = ref["keypoints"]
        ref_descriptors = ref["descriptors"]
        good_matches = match_features(test_descriptors_sift,
ref_descriptors, ratio_thresh=RATIO_THRESH)
        if len(good_matches) >= MIN_MATCH_COUNT:
            H, mask = compute_homography(test_keypoints_sift,
ref_keypoints, good_matches)
            if H is not None and mask is not None:
                inlier_count = int(np.sum(mask))
                if inlier_count > best_inliers_sift:
                    best_inliers_sift = inlier_count
                    best_match_type_sift = sign_type
                    best_good_matches_sift = good_matches

    if best_match_type_sift:
        readable_sift = folder_sign_types.get(best_match_type_sift,
best_match_type_sift)
        confidence_sift = best_inliers_sift / len(best_good_matches_sift) if
best_good_matches_sift else 0.0
    else:
        readable_sift = "Nem azonosítható"
        confidence_sift = 0.0
```

A korábban leírtak kód formában így néz ki.

Majd ugyanezen műveleteket elvégezzük az ORB segítségével is. Ha ez megtörtént akkor egyszerűen kiírja a program a konzolra az eredményeket, és megjeleníti a tesztképet.


Tesztek

A tesztelési adathalmaz a következő bontásban volt elérhető:

Tábla típusa	Teszteléshez használt képek száma
Alárendelt utak kereszteződése	20
100-as sebességkorlátozó tábla	18
80-as sebességkorlátozó tábla	20
Főútvonal	21
STOP tábla	20
Körforgalom	20

5. táblázat


A tesztelést két módon csináltam. Természetesen a program írása közben folyamatosan futtattam a szkriptet, hogy a teszt képekre milyen eredményt ad ki. A main.py futtatásával egy random képet kiválaszt a program, majd a fent említett paramétereket kiírja a konzolban. Ezen gyakran észrevehető, hogy a program nem minden esetben működik megfelelően. Van mikor helytelenül ítéli meg a tábla típusát vagy a domináns színt a SIFT és ORB egyaránt, magas konfidenciaszinttel. A szín megítélésénél a képen szereplő háttér miatt a domináns szín nincs rosszul megítélve, csak nem a táblára vonatkozik.

Algoritmus	Domináns szín	Tábla típusa	Konfidencia	Tesztkép
SIFT	Kék	100-as sebességkorlátozó tábla	88.89%	
ORB	Kék	100-as sebességkorlátozó tábla	90.00%	

6. táblázat - A domináns szín rossz megítélése a háttér és fényviszonyok miatt

Láthatjuk, hogy a tábla típusát jól felismerte, de a domináns szín a háttér és a fényviszonyok miatt a kék lett.


A képen a fényviszonyok is rettentően befolyásolják a domináns szín megítélését, továbbá, mivel átkonvertáltuk HSV szintérbe, ez máshogy is fest. Egy melegebb / hidegebb színtartományban lévő kép megítélése is nehezen kivitelezhető.

Algoritmus	Domináns szín	Tábla típusa	Konfidencia	Tesztkép
SIFT	Piros	Főútvonal	61.54%	
ORB	Piros	Főútvonal	0.00%	

7. táblázat - A domináns szín rossz megítélése a melegebb színtartomány miatt

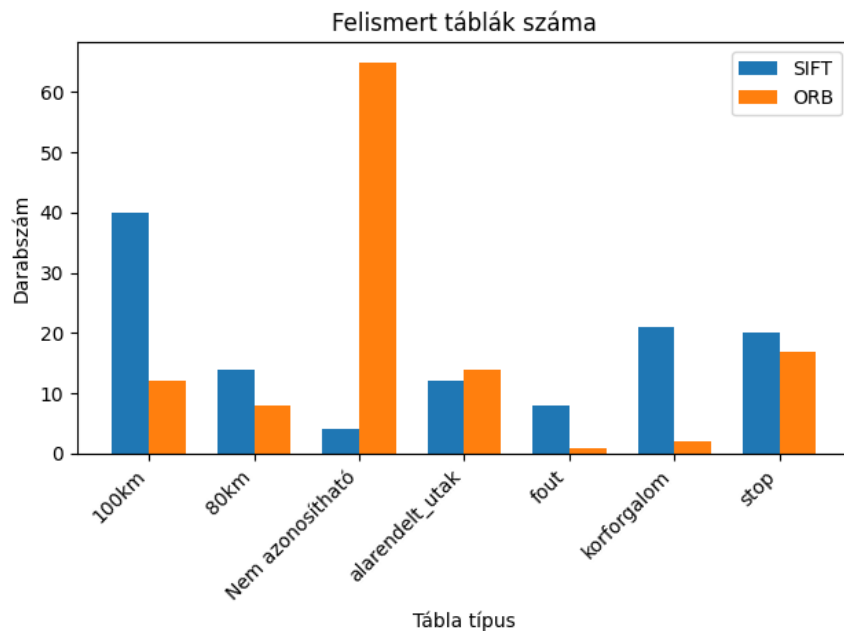
Itt még látható az is, hogy az egyik algoritmus viszonylag magasabb konfidenciaszinttel megállapította a tábla típusát, viszont a másik egyáltalán nem tudta beazonosítani.

Ez több kép esetén is mutatkozik, hogy ennél a feladatnál az ORB kevésbé hatékony a SIFT-el szemben. Nagyon sok képnél 0.00%-os konfidenciaszinttel, Nem azonosítható” eredménnyel kapjuk vissza a kimenetet.

Algoritmus	Domináns szín	Tábla típusa	Konfidencia	Tesztkép
SIFT	Piros	STOP tábla	82.14%	
ORB	Piros	STOP tábla	0.00%	

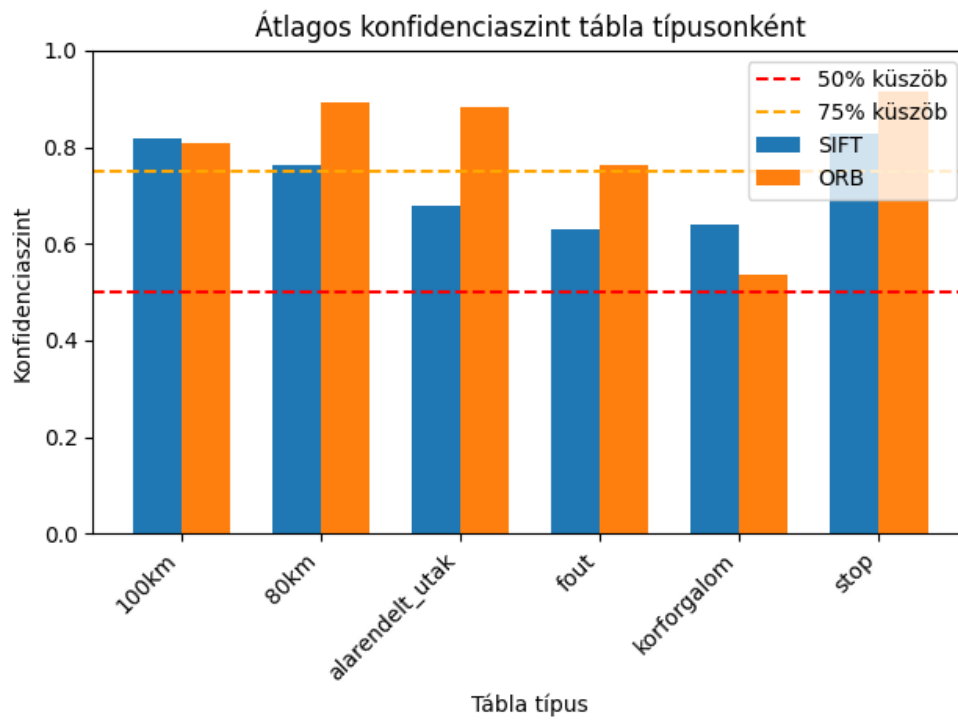
8. táblázat – Példa a két algoritmus hatékonyságára

A két algoritmus összehasonlításához végigiteráltam az összes teszt képen, eltároltam az eredményeket és készítettem belőlük egy ábrát, ami remekül szemlélteti, hogy a SIFT ebben az esetben sokkal hatékonyabb, mint az OpenCV-be alapból beépített ORB függvény ezen feladat esetén. A 9. ábrán látható, hogy mely algoritmus hány darab táblát ismert fel. Az ábrán nincsenek szűrve a helyes detektálások, ezen szimplán a felismerést számszerűsítve látjuk. A „Nem azonosítható” oszlop az ábrán azon képekre utal, hogy hány tesztképnél nem volt elegendő hasonló jellemző pont.




6. ábra – Felismert táblák száma algoritmusok szerint

A konfidenciaszintje az algoritmusoknak aszerint látható a 10. ábrán, hogy azok a táblák, amiket felismertek, azokat milyen biztosan tudják megítélni. Ezek sem feltétlenül helyesen megítélt táblák. Itt csupán az algoritmusok „biztosságát” szerettem volna szemléltetni.




7. ábra – Konfidenciaszint táblatípusonként


További teszt eredmények

Algoritmus	Domináns szín	Tábla típusa	Konfidencia	Tesztkép
SIFT	Piros	100-as sebességkorlátozó tábla	74.47%	
ORB	Piros	Alárendelt utak kereszteződése	100.00%	


9. táblázat

Algoritmus	Domináns szín	Tábla típusa	Konfidencia	Tesztkép
SIFT	Piros	STOP tábla	91.18%	
ORB	Piros	STOP tábla	100.00%	


10. táblázat

Algoritmus	Domináns szín	Tábla típusa	Konfidencia	Tesztkép
SIFT	Kék	Körforgalom	81.25%	
ORB	Kék	Nem azonosítható	0.00%	


11. táblázat

Algoritmus	Domináns szín	Tábla típusa	Konfidencia	Tesztkép
SIFT	Kék	100-as sebességkorlátozó tábla	82.35%	
ORB	Kék	100-as sebességkorlátozó tábla	100.00%	

12. táblázat

Algoritmus	Domináns szín	Tábla típusa	Konfidencia	Tesztkép
SIFT	Sárga	Főútvonal	71.43%	
ORB	Sárga	Nem azonosítható	0.00%	

13. táblázat

Algoritmus	Domináns szín	Tábla típusa	Konfidencia	Tesztkép
SIFT	Piros	100-as sebességkorlátozó tábla	78.57%	
ORB	Piros	80-as sebességkorlátozó tábla	93.85%	

14. táblázat

Források

- [1] H. Huang, W. Guo, és Y. Zhang, „Detection of Copy-Move Forgery in Digital Images Using SIFT Algorithm”, in *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, dec. 2008, o. 272–276. doi: 10.1109/PACIIA.2008.240.
- [2] „VPI - Vision Programming Interface: ORB feature detector”. Elérés: 2025. május 5. [Online]. Elérhető: https://docs.nvidia.com/vpi/algo_orb_feature_detector.html
- [3] „GTSRB - German Traffic Sign Recognition Benchmark”. Elérés: 2025. április 20. [Online]. Elérhető: <https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign>
- [4] E. G. Andrianova és L. A. Demidova, „An Approach to Image Matching Based on SIFT and ORB Algorithms”, in *2021 3rd International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA)*, nov. 2021, o. 534–539. doi: 10.1109/SUMMA53307.2021.9632214.