

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet



Autópályamatrica ellenőrző rendszer

OE-AMK

Hallgatók nevei: Szabó Attila (JW9PZT),
Kiss Bálint (BBZD5J), Zámbó Ádám
Zoltán (QGRVR8)

Tartalomjegyzék

1.	Bevezetés	1
2.	Követelmények és specifikációk	3
3.	Tervezési folyamat.....	4
3.1	Kivitelezéshez szükséges eszközök	4
3.2	Koncepció	5
4.	Kivitelezés	6
4.1	Képfeldolgozás fejlesztése.....	6
4.2	Videóformátum tesztelése.....	9
4.3	Adatbázis implementálása	12
4.4	Grafikus felület fejlesztése és tesztelése	14
4.5	Teljes szoftver összekovácsolása.....	19
4.6	A végleges szoftver tesztelése	20
5.	Statisztika.....	20
6.	Összefoglaló	22
7.	Summary	23
8.	Befejezés	24
9.	Irodalomjegyzék	24

1. BEVEZETÉS

Az autópályák jelentős szerepet töltenek be a közlekedési infrastruktúrában, amely lehetővé teszi a gyors és hatékony közlekedést. Az autópályák használatához vannak alapkötetelmények, amelyek között szerepel az autópályamatrica is, amellyel engedélyt adnak az autópályák használatához. A matricák érvényességének és a gépjárművek regisztrációjának ellenőrzése, kritikus fontosságú az autópályák hatékony működéséhez és a közúti biztonság oldaláról nézve.

Az „Autópályamatrica ellenőrző rendszer” című projektünk fő célja, egy olyan szoftver elkészítése, amely ezek követelményeknek az automatizálását hatékonyabbá teszi. E rendszer kifejlesztése során, a legújabb technológiai eszközöket és módszereket alkalmazzuk, beleérte az OpenCV (Open Source computer Vision Library)[3] és a Tesseract OCR (Optical Character Recognition)[4] szoftverkönyvtárakat, valamint az SQLite [6] adatbázis-kezelő rendszert. Szoftverünk két fontos területen kell helytállnia: az egyik terület, hogy a rendszámtáblákat felismerje, a másik fontos terület ezen rendszámtáblák alapján, matrica ellenőrzés végrehajtása. A rendszernek képesnek kell lennie az autópályamatrica érvényességének ellenőrzésére, valamint a járművek regisztrációjának és engedélyeztetésének ellenőrzésére az adatbázisban tárolt információk (rendszám, matrica vásárlásának ideje, matrica lejáratának ideje, rendszám tulajdonosának neve) alapján.

A szoftver alapmagját a magasszintű Python programozási nyelv fogja megadni, amiben az előbb említett szoftverkönyvtárakat (Tesseract és OpenCV) fogjuk egybehangolni a szoftvermag infrastruktúrájába. Szoftverünk képes beolvasni és elemezni a rendszámokat megadott videó alapján vagy kép és számítógép kameráján keresztül. Miután megtörtént a feldolgozási folyamat, a rendszerünk a feldolgozott rendszámot elraktározza és ezek után az elraktározott rendszámot egy adatbázisban lévő rendszámokkal összehasonlítja, hogy nyilvántartva van-e a rendszám vagy sem. Azután, hogy megtalálta a rendszámot az adatbázisunkban, lényeges információkat megtudhatunk belőle, mint például, hogy mikor vette meg a tulajdonos rá az autópálya matricát és hogy mikor fog lejárni a matricája. Ezt egy Python alapfüggvény segítségével fogjuk meghatározni, hogy az adott időpontban, amikor az ellenőrzés történt érvényes-e a matricája vagy sem.

Amennyiben a matrica érvényes, a rendszer zöld visszajelzővel illeti az autós rendszámot, és melléküzenetet hagyva kiírja, hogy „Érvényes matricával rendelkező jármű”. Ha viszont a járműnek nincs érvényes matricája, ebben az esetben egy piros visszajelzéssel illeti az autós rendszámot, amiről képet készít és ezt a képet elmenti egy másik adatbázisba és ki írja, hogy „Nem rendelkezik érvényes matricával”. Ennek a szoftvernek a fő prioritása, hogy ne csak a lehető leghatékonyabb megoldással lehessen autópályamatrica ellenőrzést elvégezni, hanem hogy felhasználói oldalról is felhasználóbarát megoldásra tegyünk szert.

Ezen része a dokumentációnak részletezi projektünk célját, fogalmát és értelmét, amelyben a lehető leghatékonyabb képfeldolgozási módszereket bevetve szeretnénk a lehető legjobb élményt nyújtani a felhasználók számára, valamint, hogy szoftverünk evolúciója hosszútávú támogatást kaphasson.

2. KÖVETELMÉNYEK ÉS SPECIFIKÁCIÓK

Projektünk követelményrendszerét igyekeztünk a lehető legkevesebb, viszont ezzel ellentében, a lehető legfontosabb funkcionális és nem funkcionális követelményekkel ellátni. A nem funkcionális követelmények közé lettek sorolva azok a funkciók, amelyek olyanokért felelnek, mint a megbízhatóság, mert a szoftvernek stabilnak kell lennie és a lehető legkevesebb hibával, a lehető legnagyobb stressztűrő képességgel kell működnie. Ezen felül rendszerünk teljesítménye prioritást élvez, melyben gyorsnak és hatékonynak kell lennie, hogy a felhasználók ne tapasztaljanak késedelmeket vagy időveszteséget az ellenőrzési folyamat közben, mivel elég nagy adatmennyiséggel dolgozik a szoftver. A szoftver biztonsága is olyan nem funkcionális követelmény, amelyben védenünk kell a felismert és egyidőben elraktározott rendszámtáblának információit.

A funkcionális követelmények között szerepel a felhasználói interfész, ahol a rendszernek a felhasználóbarát tulajdonságát tudja beteljesíteni, ezáltal sokkal érthetőbben és leegyszerűsítve átláthatóbb a program működésérvének a megértése. Mindemellett olyan funkciókat is implementálunk a szoftverbe, mint például, a rendszámtábla felismerése, ahol a rendszernek képesnek kell lennie, a járművek rendszámának felismerésére, valamint ennek sikeres feldolgozásában. A rendszernek képesnek kell lennie, az adatbázisban tárolt információkat ellenőrzésére, beleértve a járművek rendszámát, matrica érvényességének idejét, és ezen rendszám tulajdonosa nevét. Rendszerünk működéséhez elengedhetetlen a kamera. Mint specifikáció, a kamera pozíciója, ahhoz, hogy bemérjük az autókat és ezek rendszámát, nagyon fontos kritérium a távolság és az idő. A távolság fontossága azért lényeges, mert amikor kihelyezzük a kamerát az autópálya egy pontjára, akkor szoftverileg nekünk ehhez be kell állítanunk, a rendszámtábla felismeréséhez, a megfelelő paramétereket. Emellett, az idő fontossága is nagy szerepet játszik, mert rengeteg adatot kell neki rövid idő alatt feldolgozna, elemeznie és ezeket ki is kell a szoftvernek értékelnie. Ezzel ellentében a szoftver hatékonyságának és megbízhatóságának köszönhetően, a hardverspecifikációt nem kell durva feltételekhez kötni. Mind ehhez a feltételekhez, elegendő egy több magos processzor (minimum 4-6 magos), memoriából 8-16 GB kapacitás, háttértárból 40-50 GB kapacitás, grafikus vezérlőből/feldolgozóegységből viszont egy 10-12 magos videókártyát megkövetel a rendszer, a folyamatos működés elérése érdekében.

A szoftver rendszerint úgy van kialakítva, hogy annak evolúcióját ne befolyásolja, ebben a gyorsan fejlődő világban. Ez azt takarja, hogy rendszerünk úgy lett kifejlesztve, hogy rendszerint, új funkciók implementálása gördülékenyebben történjen meg és ez ne legyen hatással az alapvető funkciók működésére. A szoftverünket információdús dokumentációval ellátva adjuk át a felhasználó számára, amelyben minden hasznos és egyaránt fontos adatot megkaphat a szoftver működési elvéről és ennek üzemeltetéséről. Összeségében, egy olyan komplex képfeldolgozó szoftverről beszélünk, aminek jövője van az informatikában, legfőbbképpen orvostudományban és az autóiparban.

3. TERVEZÉSI FOLYAMAT

3.1 Kivitelezéshez szükséges eszközök

A fő tervünk mindig is az volt, hogy a lehető legegyszerűbb megoldásokhoz nyúljunk. Ahhoz, hogy a szoftverünket el tudjuk készíteni két fő elem közül kellett választanunk. Az első a programozási nyelv, míg a második a mely könyvtárakat szeretnénk használni, a képfeldolgozáshoz. Sok képfeldolgozó könyvtárat olvastunk át, hogy hogyan is működnének, mint például a YoloV8[5] vagy a Tesseract, esetleg az OpenCV. Mivel ezek a könyvtárak, attól függően, hogy melyik programozási nyelvben használjuk, elég hatékonyak tudnak lenni, viszont a lehető legkézenfekvőbb megoldásnál az OpenCV-t találtuk. Mindemellett persze projektünk során felhasználtuk a Tesseract könyvtárakat is, ami ahhoz adott nekünk segítséget, hogy a képen feldolgozott szöveget fel tudjuk ismerni. Programozási nyelvek közül, mind a C++, mind pedig a Python egy nagyon jó magasszintű programozási nyelvnek mondható és mind a két nyelvben nagyon hatékonyan tudnánk lefejleszteni és kezelní a szoftvert, de utóbbi nyelven, a Python mellett döntöttünk. A fejlesztői környezeten sokat variáltunk, melyik lenne a legideálisabb a fejlesztésre és sokkal kézenfekvőbb is lett volna a PyCharm IDE fejlesztő

környezete, azonban a Visual Studio Code mellett tettük le a voksunkat. Szerintünk a Studio Code egy nagyon felhasználó barát fejlesztői környezet és egyszerű.

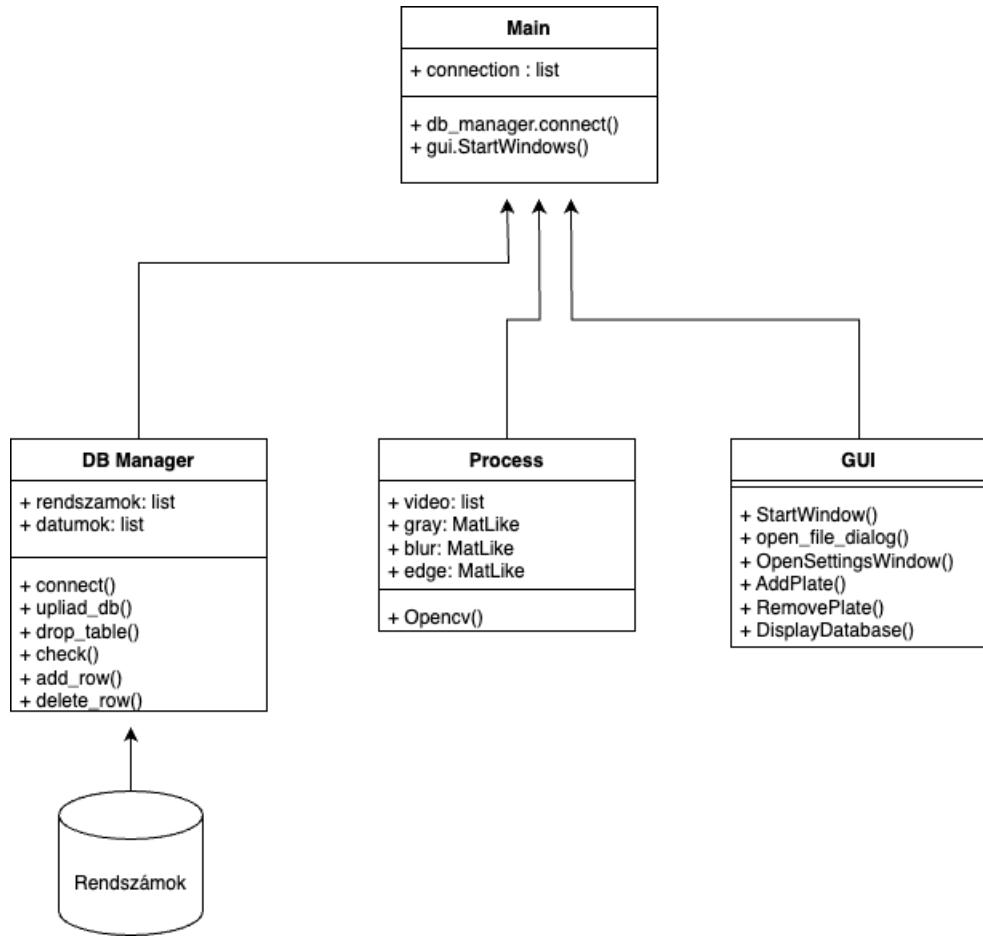
Mindemellett elmondható, hogy a Visual Studio Code, szerintünk nagyon is alkalmas szoftverfejlesztésre, hiába „code editor”, ettől függetlenül, nagyon jól bánik a compilerekkel és az erőforrásunkat sem veszi el.

Természetesen ahhoz, hogy tudjuk melyik rendszámnak ki a tulajdonosa és, hogy meddig is érvényes a matricája, gondoskodnunk kell egy adatbázisról is, ami segítségével ezeket az adatokat el tudjuk tárolni. Itt az SQLite adatbázist szeretnénk beleimplementálni a szoftverünkbe. Hasonló elven működik, mint az MSSQL localdb, amihez valójában nem kell nekünk SQL szervert futtatnunk, így több energiaforrást tudunk meghagyni a képfeldolgozásra. Evolúciós funkciója, hogy amennyiben ezt kibővíteni szeretnénk, vagy esetleg szerver-kliens oldali megoldás lenne preferált a programban, ezt ki lehet bővíteni.

3.2 Koncepció

Megvalósításhoz szükséges alapokat letéve, a koncepcióink a következők voltak. A legelején a szoftver magját tettük célunkká, hogy megtudjuk hogyan is működik a képfeldolgozás és ezáltal az OpenCV könyvtárakat használva, rájöjjünk a működés elvére. Mindezek után, miután megoldottuk, hogy egy darab képet, miképpen tudunk feldolgozni az alapképből egy olyan képpé, ahol a rendszámot a Tesseract-OCR szövegleolvasó algoritmusá segítségével, a karaktereket el tudjuk menteni egy változó értékeként, majd ezt az értéket összevetve az adatbázisunkból kiolvassuk, mely rendszámmal fog egyezni, amennyiben a rendszám regisztrálva van. Ha regisztrálva van, akkor a felismerés során, az adatbázisban megtalált rendszámnak a tulajdonosát megtudjuk, és a kép készítésének idejét összevetve megnézzük, hogy akkor érvényes matricája van-e a rendszámnak vagy sem.

Ahogy a szoftver alapelemeit létrehoztuk, azután ezt az egészet egy felhasználó barát grafikus felületre valósítjuk meg, hogy ezáltal se kelljen a felhasználónak a szoftver forráskódjához hozzányúlni.



3.2.1. ábra: Tervezett működés

4. KIVITELEZÉS

4.1 Képfeldolgozás fejlesztése

Magát a képfeldolgozó részt kezdtük el teljesen lefejleszteni, amivel a szoftverünk alapját szerettük volna elkészíteni. A fejlesztés előtt, meg kellett ismernünk, hogyan is működik ez az egész. A programozási nyelvet a Pythonot választottuk, amin gördülékenyebb forráskódot írhatunk és könnyen kezelhető könyvtárát hamar megismerhettük. Ahhoz, hogy programunk alapmagját megadjuk, szükségünk van a már korábban említett OpenCV és Tesseract-OCR könyvtárakra. Ezekre azért is van szükség, mivel két olyan könyvtárról beszélünk, amik létfontosságú prioritást élveznek a megfelelő képfeldolgozási procedúra miatt.

Ezek után úgy folytatódik a képfeldolgozási procedúránk, hogy fogunk egy adott képet jelen esetben egy nyers képet és mi ezt a képet folyamatosan formázzuk. A formázás menete, úgy megy végbe, hogy a nyers képet először egy szürke árnyalatos képpé konvertáljuk, hogy csak fekete és fehér színek kombináció legyenek. Ezek után mi ezt a képet tovább formáljuk bináris súlyozott képpé, ami annyit takar, hogy ezen a képen már csak natív fekete és fehér színek lehetnek. Ezek után pedig kontúrozzuk a képet, amivel pedig a karaktereket tudjuk leolvasni a rendszámtábláról.

A képfeldolgozás további lépése az, hogy az adott képen fogjuk és elemezzük, hogy merre is található a rendszámtábla a képen. Ezt úgy érhetjük el, hogy rendelkezésünkre állt egy xml fájl, amiben matematikailag olyan adatokkal rendelkezik, ami a kép mátrix elemei szerint tudja azonosítani, hogy melyik tartozik a rendszámtáblához. Ezeket az xml fajta fájlokat Cascade-nek [1] nevezzük.

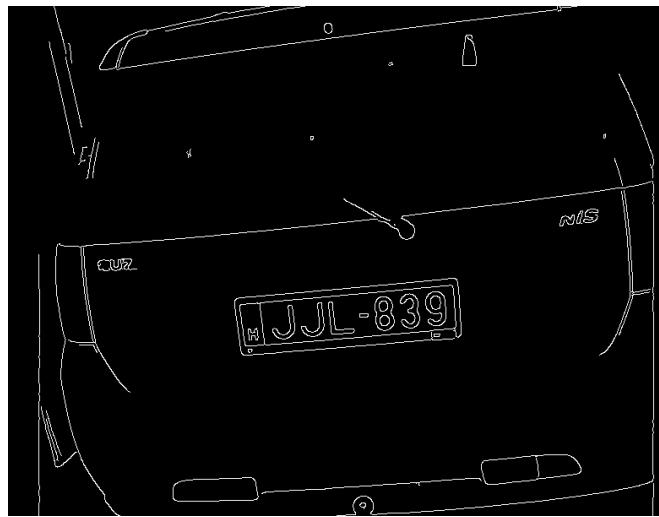
```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (5, 5), 0)
edges = cv2.Canny(blur, 100, 200)

# Kontúrok keresése
contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

4.1.1. ábra: Kép formázása több különböző formára



4.1.2.a ábra: Itt eredeti képből a szürke árnyalatig majd a blúrozást bevezetjük



4.1.2.b ábra: Edge formátumú, hogy kiolvasható legyen

A 4.1.1-es ábrán látható kódosorozat, minden szükséges ahhoz, hogy egy kirajzolható karaktereket kaphassunk a készített fényképről. Ezáltal pedig sikeresen egy szöveget megkapjunk, aminek segítségével már sikeresen tovább tudunk dolgozni az adattal.

A 4.1.2-es ábrákon pedig az látható, ahogyan azt az egy képet átalakítjuk először, szürke árnyalatos képpé az eredeti képet, majd ezek után átalakítjuk blúr szerűre, hogy leszedjük a fölösleges információkat a képről, amik nem lényegesek a karakterfelismeréshez. A legutolsó képen pedig edge filtert teszünk rá, ahogy azt a képen látható, hogy a karaktereket kiolvasható formában kapjuk meg.

Magát a karakterfelismerést a Tesseract-OCR-t használtuk, aminek a segítségével, leszűkítettük annak a körét is, hogy pontosabban milyen karaktereket tudjon felismerni, mivel alapesetben a karakterfelismerő algoritmus minden karaktert képes felismerni, viszont számunkra a lehető leghatékonyabb megoldást használtuk erre, azaz megadtuk, hogy pontosabban milyen karaktereket kell neki felismernie.

```
# Rendszám szövegének kinyerése Tesseract segítségével
text = pytesseract.image_to_string(roi, config='--psm 8 --oem 3 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-')
if len(text) >= 6:
    validation = db_manager.check(text)
    print("Felismert rendszám: ", text, " - ", validation)
    cv2.putText(frame, validation, (x, y), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow(validation, roi)
```

4.1.2. ábra: *image_to_string* metódus használata

Most már jelenleg minden adott ahhoz, hogy felismerjünk egy rendszámot, és ezt a felismert rendszámot eltudjuk raktározni későbbre, hogy aztán az adatbázisban lévő rendszámokkal összevetve tudjuk elemezni, hogy van-e autópálya matricája vagy sem.



4.1.3. ábra: A felismert szöveg kiírása

4.2 Videóformátum tesztelése

Mivel az alap ötlet az volt, hogy egy videó alapján kell felismernünk a rendszámokat és ezáltal kell a matricákat ellenőrizni, ezért már a forráskódunkat úgy írtok meg, hogy ezt már csak egy videó feldolgozó algoritmusba, azaz egy egyszerű hurokba ágyaztuk bele a képfeldolgozási funkciókat. Ez úgy készítettük el, hogy alapból a videót beimportáljuk és ezt egy egyfajta mátrixba menti el.

```

while video.isOpened():
    ret, frame = video.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blur, 100, 200)

    # Kontúrok keresése
    contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    for contour in contours:
        area = cv2.contourArea(contour)
        if area > 1000: # Azon kontúrok kiválasztása, melyek területe elég nagy
            x, y, w, h = cv2.boundingRect(contour)
            roi = frame[y:y+h, x:x+w]

            # Rendszám szövegének kinyerése Tesseract segítségével
            text = pytesseract.image_to_string(roi, config='--psm 8 --oem 3 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-')
            if len(text) > 6:
                validation = db_manager.check(text)
                print("Felismert rendszám: ", text, " - ", validation)
                cv2.putText(frame, validation, (x, y), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
                cv2.imshow(validation, roi)

            if text.strip() == "KHE-440":
                #Kép készítése a rendszámról
                cv2.imwrite("rendszer_khe440.jpg", roi)
                # Érvényes üzenet kiírás
                cv2.putText(frame, "Érvényes matrica", (x, y), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
                cv2.imshow('Érvényes matrica', roi)

            #cv2.putText(frame, text, (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

        # Kép megjelenítése
        cv2.imshow('Rendszam felismerese', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

```

4.2.1 ábra: A videófeldolgozás törzskódrészlete

Működési elve az, hogy a videót beimportálva framek-re, azaz képekre bontjuk a videót és ezt az egy képet alakítgatjuk át a nyers képből egy olyan képpé, amit kontúrozva a betűket le tudjuk olvasni.

Maga a tesztelés az úgy ment végbe, hogy van nekünk egy kiírató funkcióink, ami terminálba kiírja, hogy pontosabban, hogy milyen karaktereket is ismert fel. Ezzel, valójában nyomon követhetően tesztelhettük, hogy mennyire helyesen ismeri fel a karaktereket a képről és hogy milyen távolságból tudja a leg pontosabban értelmezni a szöveget. Ezt a következő 4.2.2 ábrán lehet majd látni.

```

- Nem található
Felismert rendszám: KNG-399
- Nem található
Felismert rendszám: BKNG-309
- Nem található
Felismert rendszám: KNG-395
- Nem található
Felismert rendszám: KNG-395
- Nem található
Felismert rendszám: KNG-399
- Nem található
Felismert rendszám: KNG-395
- Nem található
Felismert rendszám: KNG-395
- Nem található
Felismert rendszám: KNG-395
- Nem található
Felismert rendszám: KNG-305
- Nem található
Felismert rendszám: KNG-305
- Nem található
Felismert rendszám: KNG-3051
- Nem található
Felismert rendszám: KNG-3004
- Nem található
Felismert rendszám: KNG-3004
- Nem található
Felismert rendszám: SNG-3004
- Nem található
Felismert rendszám: NG-309
- Nem található

```

4.2.2 ábra: A videófeldolgozás törzskódrészlete

Maga a képfeldolgozás fejlesztése és tesztelése volt az első szempont, aminek működnie kell, hogy ezáltal tovább haladhassunk a fejlesztési folyamatokkal az adatbázis lefejlesztésével és grafikus felületbe való implementálásával. Mindemellett, hogy előkészítsük az adatbázis fejlesztését, tesztelésként egy if elágazással leteszteltük, hogy az elmentett rendszámot, ha felismerte és azonosak az összehasonlított rendszámmal, aminek érvényes matricája van, akkor ki kell írnia a 4.2.3. ábrán látható formátumban azt, hogy „Érvényes matrica”.



4.2.3 ábra: Matrica ellenőrzési teszt

4.3 Adatbázis implementálása

Az adatbázisunk létrehozásakor az SQLite3-at használtuk. A funkcionalitás elkülönítéséhez egy külön Python fájlt hoztunk létre, amely a db_manager.py nevet kapta. Ebben a fájlban több metódust definiáltunk az adatbázis-kezelés megkönnyítésére. Létrehoztunk egy connect() metódust, ami paraméterként az adatbázis nevét kapja meg esetünkben database.db. Ez a metódus segít leegyszerűsíteni az adatbázishoz való csatlakozást, mivel így csak az adatbázis nevét kell megadni. Van egy create_tables() metódusunk, ami egy SQL parancsot futtat, ami az adattábla létrehozásához szükséges. Tartalmaz egy egyedi azonosítót (PRIMARY KEY), egy rendszámot (TEXT), egy dátumot (DATE), és egy érvényességi dátumot (TEXT), ahogy a 4.3.1. ábrán is látható. Ezt a metódust általában csak egyszer kell futtatni, amikor az adatbázist létrehozzuk. A továbbiakban az adatbázis automatikusan megtartja a struktúráját, és nem szükséges újra létrehozni a táblát. Van még egy drop_table() metódusunk is, ezzel tudjuk kitörölni a táblákat, valamint egy upload_db() metódus, hogy feltöltsük pár teszt adattal az adatbázisunk. Utóbbi kettőt csak a fejlesztés során használtuk. Ezek segítenek az adatbázis inicializálásában és annak ellenőrzésében, hogy minden megfelelően működik.

```
def connect(name):
    return sqlite3.connect(name)

def create_tables(conn):
    # Kapcsolódás az adatbázishoz
    c = conn.cursor()

    c.execute("""CREATE TABLE IF NOT EXISTS tickets(
        id INTEGER PRIMARY KEY,
        plate TEXT NOT NULL,
        date DATE DEFAULT (date('now')),
        end_of_validity Text NOT NULL
    )""")
```

4.3.1. ábra: Kód részlet az SQL parancsról

Az egyik legfontosabb a check() metódus, ami paraméterként a pytesseract által leolvastott rendszámot kapja meg. Leszedi róla a felesleges szóközöket és keresést indít az adatbázisban. Ha megtalálja a rendszámot, akkor eltárolja az egész sort egy változóba. A következő lépés, hogy minden dátumot aktualizálja az operációs rendszertől és minden dátumot dátum formátumra konvertáljon. Az utolsó lépés, hogy a megkapott három dátumot logikailag megvizsgálja, hogy a mai dátum a kezdő és záró dátum közé

esik e. Ha igen, akkor az „Érvényes matrica” szöveget adja vissza, ellenkező esetben az „Érvénytelen matrica” szöveget. Azonban, ha az adatbázisban nem találta meg a keresett rendszámtáblát a „Nem található” szöveget adja vissza. Ezek a szövegek kiírásra kerülnek a képernyőre, amiről bővebben a 4.2 bekezdésben írunk. A check metódus kódja a 4.3.2 ábrán látható.

```
def check(plate2):
    plate = plate2.strip()
    connection = connect('database.db')
    c = connection.cursor()

    # Rendszám, dátum és lejárat dátumának lekérése az adatbázisból
    c.execute('SELECT date, end_of_validity FROM tickets WHERE plate = ?', (plate,))
    result = c.fetchone()

    if result:
        # Mai dátum lekérése
        today = datetime.datetime.now().date()

        # Dátumok lekérése az adatbázisból
        start_date = datetime.datetime.strptime(result[0], "%Y-%m-%d").date()
        end_date = datetime.datetime.strptime(result[1], "%Y-%m-%d").date()
        #test(start_date, today, end_date)

        # Ellenőrzés, hogy a mai dátum az adatbázisban található kezdő és befejező dátumok között van-e
        if start_date <= today <= end_date:
            return 'Ervenyes matrica'
        else:
            return 'Ervenytelen matrica'
    else:
        return 'Nem talalhato'
```

4.3.2. ábra: Check metódus kódja

A felhasználói felület fejlesztésekor szükségessé vált még kettő metódus implementálása, melyekkel képesek vagyunk adatot beszúrni, vagy törlni a tickets táblából. Ezek megkapják a szükséges adatokat paraméterként, majd felépítik a kapcsolatot az adatbázissal a műveletek végrehajtásához. Az add_row() a rendszámot kapja meg [2]és a lejáratit dátumot, a delete_row() pedig csak a rendszámot. Bár a felhasználói felületen már végzünk bizonyos szintű adatellenőrzést, az adatbázis műveletek kritikusak a program működésében, ezért a try-except blokkokat használjuk a hibakezelésre, hogy kezeljük az esetleges kivételeket. Az add_row metódus az SQL INSERT parancs segítségével beszúrja az adatokat a tickets táblába és menti a tranzakciót. Ha hiba történik, a try-except blokk kezeli a kivételeket, majd a kapcsolatot végül lezárja. A delete_row metódus is hasonlóan működik ez az SQL DELETE parancsot használja. A két metódus megvalósítása a 4.3.3. ábrán látható.

```
#GUI
def add_row(plate, end_of_validity):
    print("ezt kellene hozzáadni: "+plate+", "+end_of_validity)
    conn = connect("database.db")
    c = conn.cursor()
    try:
        # SQL INSERT parancs végrehajtása az adatokkal
        c.execute("INSERT INTO tickets (plate, end_of_validity) VALUES (?, ?)", (plate, end_of_validity))
        print("Sor hozzáadva az adatbázishoz: ", plate, ", ", end_of_validity)
        conn.commit() # Tranzakció mentése
    except sqlite3.Error as e:
        print("Hiba az adatok beszúrásakor:", e)
    finally:
        conn.close() # Adatbáziskapcsolat lezárása

def delete_row(plate):
    print("ezt kellene kitörölni:", plate)
    conn = connect("database.db")
    c = conn.cursor()
    try:
        # SQL DELETE parancs végrehajtása a megadott rendszámról
        c.execute("DELETE FROM tickets WHERE plate = ?", (plate,))
        print("Sor törlve az adatbázisból:", plate)
        conn.commit() # Tranzakció mentése
    except sqlite3.Error as e:
        print("Hiba a sor törlésekor:", e)
    finally:
        conn.close() # Adatbáziskapcsolat lezárása
```

4.3.3. ábra: Az add_row és a delete_row megvalósítása

4.4 Grafikus felület fejlesztése és tesztelése

A grafikus felület tervezésénél törekedtünk, az egyszerűségre és a könnyű kezelhetőségre, hogy egy átlátható felületet kapjunk. Projektünk készítésekor a Tkinter[2] nevezetű Graphical User Interface (GUI) mellett döntöttünk, mivel ez a Python beépített eszközökészlete, amelyet széles körben használnak könnyű, platformfüggetlen grafikus felületek létrehozására.

Felépítése nagyon egyszerű, egy, vagy több főablak is létrehozható és ebbe widgeteket szúrhatunk be. A widgetek lehetnek gombok (button), címkék (label), szövegdobozok (entry) és hasonlók. Választhatunk különböző elrendezési módszerek közül is, az előbb felsorolt elemeket tehetjük csomagokba (pack) és táblázatosan cellákba (grid) is.

A könnyű átláthatóság céljából a GUI felület metódusait egy külön fájlba tároltuk el, ezt gui.py-nak neveztük el. Létrehoztunk egy fő „StartWindow” nevezetű metódust, ez valósítja meg a Matrica ellenőrző rendszerünk kezdőképernyőjét, 4.4.1 ábrán látható. Ez az ablak fix méretű, nem átméretezhető, három „pack()” elem található benne, az első felső egy „Label” típusú, feliratot tartalmaz „Adjon hozzá egy videót!”, ezzel szólítjuk

meg a felhasználót belépéskor. Az alsó két elemben egy-egy gomb kapott helyet, baloldalon egy „Beállítások”, jobboldalon egy „Tallózás” gomb, amit a 4.4.2. ábra szemléltet.

```
def StartWindow():

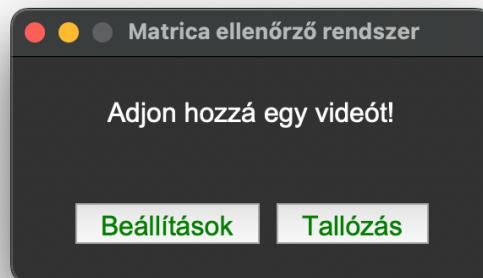
    start = tk.Tk() #Megcsinálom az ablakot
    start.resizable(False, False)

    start.title("Matrica ellenőrző rendszer")
    label = tk.Label(start, text="Adjon hozzá egy videót!", font=('Arial',16))
    label.pack(padx=20,pady=20)

    tallozas = tk.Button(start, text="Tallózás", font=('Arial',16), command=open_file_dialog, fg="green",
    settings = tk.Button(start, text="Beállítások", font=('Arial',16), command=OpenSettingsWindow, fg="green",
    settings.pack(side=tk.LEFT, padx=(35,2), pady=20)
    tallozas.pack(side=tk.RIGHT, padx=(2,35), pady=20)

    start.mainloop()
```

4.4.1. ábra: *StartWindow* metódus



4.4.2. ábra: *Alkalmasunk kezdőképernyője*

A Tallózás gomb megnyomásakor az `open_file_dialog()` függvény hívódik meg 4.4.3. ábra, amely egy fájlválasztó dialógust jelenít meg az operációs rendszer fájlkezelőjének segítségével. Ezzel a funkcióval a felhasználó kiválaszthat egy fájlt a rendszerében. Amikor ez megtörténik a fájl elérési útvonala mentésre kerül egy változóba, és ezt tovább adja `process.py` OpenCV metódusának feldolgozásra. Ha a felhasználó olyan fájlt jelöli ki, amit az OpenCV nem tud feldolgozni megszakítja a folyamatot és visszatér a kezdőképernyőre, hasonlósan, ha bezárja a dialógust, a fájl kiválasztása nélkül.

```
def open_file_dialog():
    file_path = filedialog.askopenfilename()
    if file_path:
        process.OpenCV(file_path)
```

4.4.3. ábra - *open_file_dialog* metódus

A Beállítás gomb megnyomásával elő jön egy újabb ablak, ez az adatbázis kezelésére szolgál. Szintén fix méretezésű, azonban eltérően a kezdő képernyőtől, itt „grid” elrendezést használunk. Ez az ablak összesen 8 darab label, 3 darab button és 3 entry elemet tartalmaz, a Tkinter megvalósítás ilyen sok elemnél már nagyon nehezen átlátható, 4.4.4. ábra.

A harmadik sorban tudunk hozzáadni rendszámot az adatbázishoz. Van itt két entry elem, egyiknél a rendszámot kell megadnunk a másiknál a lejáratú dátumot, majd a „Rendszám hozzáadása” gombbal eltárolni adatbázisunkban. Alatta lehetőségünk van rendszám törlésére is, hasonló módon, mint a hozzáadásnál, de itt elég csak a rendszámot megadnunk. Az ablak a 4.4.5. ábrán látható. Ezeknek a gomboknak vannak saját metódusaik melyeket meghívnak gombnyomásra, ezek dolgozzák fel az entry mezőkben megadott adatokat és használják a db_manager metódusait az adatok beszúrására, vagy törlésére.

```

def OpenSettingsWindow():
    settings = tk.Tk()
    settings.title("Beállítások - Matrica ellenőrző rendszer")
    settings.geometry("630x390")
    settings.resizable(False, False)

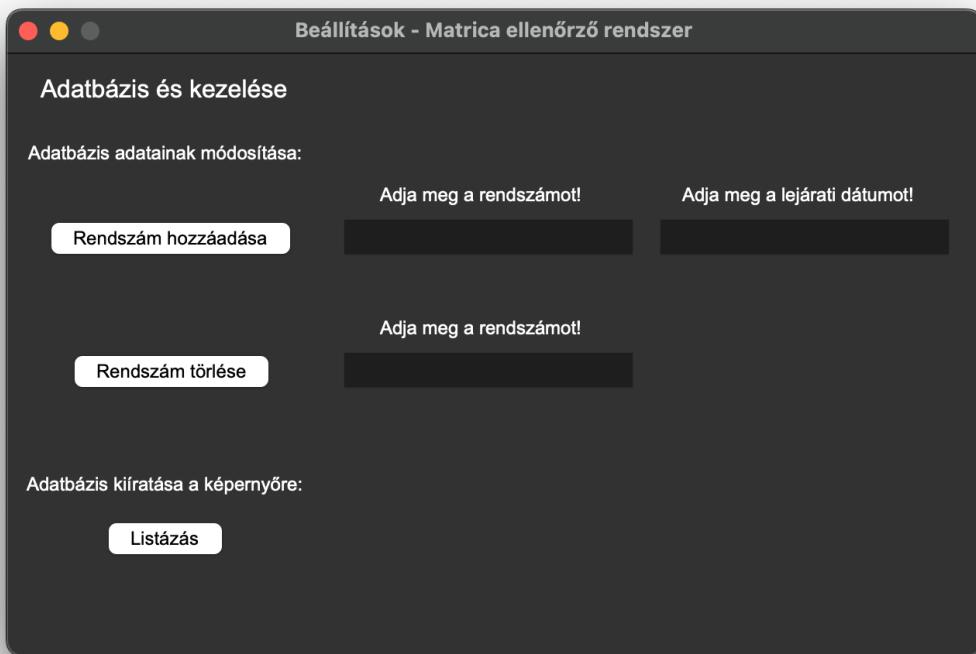
    title = tk.Label(settings, text="Adatbázis és kezelése", font=('Arial',16))
    title.grid(row=0, column=0, padx=10, pady=10)

    label1 = tk.Label(settings, text="Adatbázis adatainak módosítása:", font=('Arial',12))
    label1.grid(row=1, column=0, padx=10, pady=(10,2))
    bp_label = tk.Label(settings, text="Adja meg a rendszámot!", font=('Arial',12))
    bp_label.grid(row=2, column=1, padx=10, pady=(5,2))
    bd_label = tk.Label(settings, text="Adja meg a lejáratí dátumot!", font=('Arial',12))
    bd_label.grid(row=2, column=2, padx=10, pady=(5,2))
    beszuras = tk.Button(settings, text="Rendszám hozzáadása", command=lambda: AddPlate(bp_input.get(), bd_input.get(), b_error_label), font=('Arial',12))
    beszuras.grid(row=3, column=0, padx=(10,2),pady=2)
    bp_input = tk.Entry(settings)
    bp_input.grid(row=3, column=1, padx=(10,2),pady=2)
    bd_input = tk.Entry(settings)
    bd_input.grid(row=3, column=2, padx=(10,2),pady=2)
    b_error_label = tk.Label(settings, font=('Arial',12),fg="red")
    b_error_label.grid(row=4, column=0, padx=10, pady=(5,2), columnspan=2)

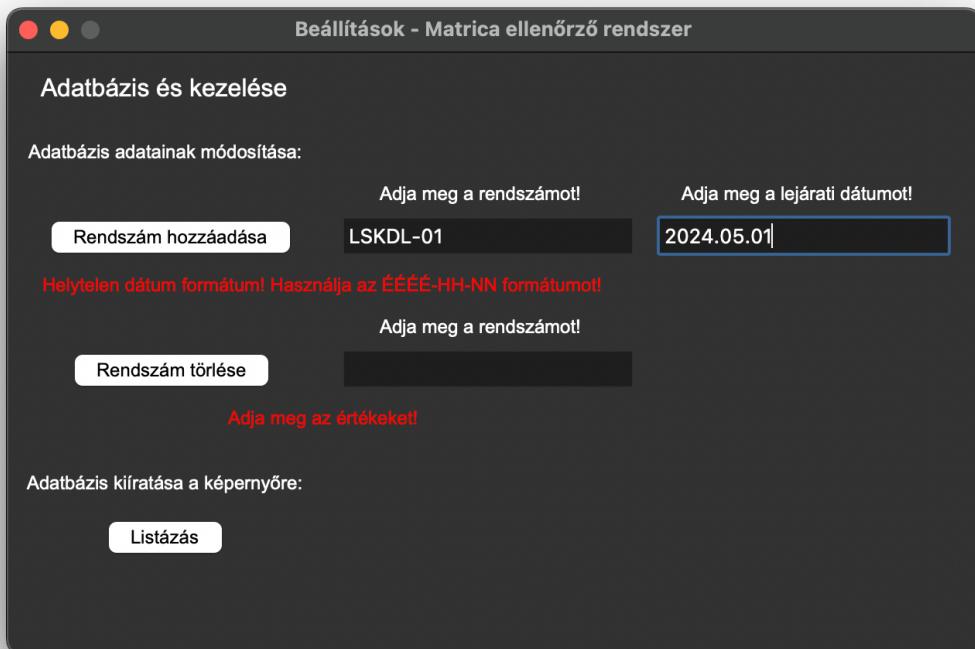
    t_label = tk.Label(settings, text="Adja meg a rendszámot!", font=('Arial',12))
    t_label.grid(row=5, column=1, padx=10, pady=(5,2))
    torles = tk.Button(settings, text="Rendszám törlése", command=lambda: RemovePlate(tp_input.get(), t_error_label), font=('Arial',12))
    torles.grid(row=6, column=0, padx=(10,2),pady=2)
    tp_input = tk.Entry(settings)
    tp_input.grid(row=6, column=1, padx=(10,2),pady=2)
    t_error_label = tk.Label(settings, font=('Arial',12),fg="red")
    t_error_label.grid(row=7, column=0, padx=10, pady=(5,2), columnspan=2)

    label2 = tk.Label(settings, text="Adatbázis kiíratása a képernyőre:", font=('Arial',12), justify="left")
    label2.grid(row=9, column=0, padx=10, pady=(20,2))
    listazas = tk.Button(settings, text="Listázás", font=('Arial',12), command=DisplayDatabase, justify="left")
    listazas.grid(row=10, column=0, padx=10,pady=10)

```

4.4.4. ábra: *OpenSettingsWindow* metódus4.4.5. ábra: *Beállítások ablak*

Hibaellenőrzés és adat validálás is be van építve a programba. Ha az entry mezőket nem megfelelően töltjük ki, vagy ha a dátum nem az elvárt formátumban van megadva, a 4.4.6. ábrán látható módon, a program piros színű szöveges üzenettel figyelmezteti a felhasználót. Ez a visszajelzés segíti a program helyes használatát, és biztosítja, hogy az adatbázisban csak érvényes és megfelelő formátumú adatok szerepeljenek.



4.4.6. ábra: Beállítások ablak hibajelzésekkel

A képernyő alján elhelyeztünk egy „Listázás” gombot, aminek a segítségével egy új ablakban megjeleníthetjük az adatbázis tartalmát. A gomb lenyomásakor a `DisplayDatabase()` metódus fut le, ami kapcsolatot épít ki az adatbázissal, majd lekérdezi a „tickets” tábla összes sorát és eltárolja egy listába. Megszünteti a kapcsolatot az adatbázissal és ha sikerült sorokat kiolvasni, akkor létrehozza az ablakot, ha nem, akkor azt az üzenetet kapjuk vissza, hogy üres az adatbázis. Ebben az ablakban egy úgy nevezett „Treeview widget” elrendezést használunk, ami egy táblázatszerű elrendezés, itt lehetőségünk van előre beállítani az oszlop címeket is. Ezután a Treeviewba könnyedén betölthetjük az adatokat a listából egy for ciklus segítségével. A kód megvalósítása a 4.4.7. ábrán látható, az ablak kinézete a 4.4.8. ábrán.

```

def DisplayDatabase():
    conn = db_manager.connect("database.db")
    c = conn.cursor()
    c.execute("SELECT * FROM tickets")
    rows = c.fetchall()
    conn.close()

    if rows:
        # Létrehozunk egy új ablakot a listázáshoz
        listazas_window = tk.Toplevel()
        listazas_window.title("Adatok listázása")

        # Létrehozunk egy Treeview widget-et a táblázathoz
        tree = ttk.Treeview(listazas_window)
        tree["columns"] = ("Plate", "Date", "End of Validity")
        tree.heading("#0", text="ID")
        tree.heading("Plate", text="Rendszám")
        tree.heading("Date", text="Kezdő dátum")
        tree.heading("End of Validity", text="Lejárati dátum")
        tree.pack(expand=True, fill="both")

        # Betöltjük az adatokat a Treeview-be
        for row in rows:
            tree.insert("", "end", text=row[0], values=(row[1], row[2], row[3]))

    else:
        print("Nincsenek adatok az adatbázisban.")

```

4.4.7. ábra: *DisplayDatabase* metódus

Adatok listázása				
ID	Rendszám	Kezdő dátum	Lejárati dátum	
1	ABC123	2024-03-24	2024-03-17	
2	DEF456	2024-03-24	2024-03-18	
3	GHI789	2024-03-24	2024-03-19	
4	AAA123	2024-03-24	2025-01-01	
5	NAI3NRU	2024-03-24	2025-01-01	
6	GXI5OGJ	2024-03-24	2025-01-01	
7	APO5JEO	2024-03-24	2025-01-01	
8	KHO5ZZK	2024-03-24	2024-12-21	
9	FJI4ZHY	2024-03-24	2024-11-11	
10	NAS4KGJ	2024-03-24	2024-10-10	

4.4.8. ábra: *Adatbázis adatok listázása ablakban*

4.5 Teljes szoftver összekovácsolása

Ahogy az előbbiekben is olvasható, ezekből az elemekből lett összekovácsolva a szoftver alkalmazásunk. Ezt úgy oldottuk meg, hogy ezeket a forráskódokat külön fájlokba mentettük el, mint például a *gui.py* vagy a *process.py* ami a képfeldolgozásért felel. Az egész úgy épül fel, hogy van nekünk egy *main* program fájlunk, amibe beimportáljuk a kisebb komponenseket, és így futtatjuk a szoftverünket.

Ezek után más dolgunk nincs, a felhasználó barát programunknak köszönhetően, egyszerűen tudunk videót beimportálni a szoftverbe, valamint rendszámot hozzáadni és törölni az adatbázisunkból.

4.6 A végleges szoftver tesztelése

Magát a szoftvertesztelés egy igazán érdekes szituációt vetett fel. Ugyan unit teszteket nem végeztünk rajta, valamint komponens teszteket sikeresek voltak, amit mi elvártunk azt a teljesítést hozta, viszont jól látható, hogy amikor a szoftverünket stressz teszt alá vetettük, akkor érezhető, hogy igen is számít milyen hardveren is futtatjuk a szoftvert. Egy általános számítógépen futtatva, aminek a specifikációi: 4 mag 8 szálas 8 GB rammal és 256 GiB tárhellyel rendelkezve stabilan lefutott a videó, viszont nagyon lassan. Ennek a konklúziója az, hogy ugyan hardver specifikus dolog a képfeldolgozás, viszont számít az is, hogy magában a videóban mennyi rendszámot is kell neki felismernie. Ha egy autó videójáról futtatjuk le, akkor nagyon is akadás nélkül lefut, viszont, ha már 5-6 autó van egy időben, akkor lelassul a feldolgozási folyamat.

A tesztelési folyamat során nem igen volt, Alfa és Béta tesztelés, egy nagyon egyszerű elfogadási tesztet végeztünk rajta, amin sikeresen átment a szoftverünk. A specifikáció, amik a mi igényeinknek felelt meg teljesen rendben találtuk hozzá az alkalmazásunk működését.

5. STATISZTIKA

A statisztikai részben arról lesz szó, hogy milyen sikerességgel tudtuk elérni, hogy a programunk felismerje a rendszámokat.

3 videóval próbálkoztunk alapból. Az egyik videó, ahol csak egy autó szerepel és annak a rendszámtábláját kellett felismernie. Ez lesz az a videó, ahol később még kitérek rá bővebben. A második videón, közepesen 5-10 db autó látható a videón, míg a harmadik videón, konkrétan egy autópályáról készített videót használtunk fel, amin több, mint 20 autó is látható.

Az utolsó két videónál jól látható, hogy itt a képfeldolgozó rendszerünk nagyon sokat gondolkozott azon, hogy pontosabban melyek a rendszámok, hol keresse azokat és hogy

ezeket az adatbázisunkkal összevetve megtudjuk a matrica érvényességüket. Mivel ez a két videó közel azonos távolságban készült el, így elmondható, hogy az esetek 70%-ban felismert karaktereket, ugyanakkor a teljes rendszám sikeres felismerése csak 10 esetből körülbelül 2-szer történt meg. Az utolsó két videó (közepes és sok autóból állóak) távolsága a kamerától körülbelül 100-150 méterre lettek felvéve az autók pozíciójához képest.

Az első videón, ahol csak autót teszteltünk, ott a lehető legnagyobb részletekbe is belementünk. A videó lényege, hogy az autóhoz sétálva 150 méterről indítva megnéztük, mikortól lesz az első és utolsó sikeres rendszámfelismerés, valamint mikortól van az a holdpont, amikor már egyáltalán nem sikerült neki felismerni a rendszámtáblát. Statisztikailag, a rendszámon lévő karaktereket 140 méterről kezdte felismerni és a legelső sikeres felismerés 110 méternél kezdődött. Itt a képkockák számához viszonyítva, a felismerések száma százalékban olyan 20%-os, ahol sikeres a rendszámtábla felismerése. Ezek után a 110 méter után az 50-es méter mérföldkőig erős 80%-os arányban felismerte sikeresen a rendszámot. A legutoljára felismert helyes rendszám kevesebb mint 3 méteren volt megfigyelhető, ezek után még karaktereket ismert fel, viszont sajnos ezekután olyan helyeken is érzékeltek karaktereket, ahol a rendszernek nem kellett volna. Természetesen a másik két videó alapján hozzátesz az is, hogy számít a jármű sebessége is, aminél azt a konklúziót vonhatjuk le, hogy a szoftverünk a 60 km/h-ás sebességen felül már nem képes olyan rövid idő alatt rendesen felismerni a rendszámot, szóval az ez alatti sebesség limitnél, ha nem is tökéletesen, de az esetek nagy részében sikeresen működik az autópálya matrica ellenőrző szoftverünk.

Összességében statisztikailag egy átfogó képet kapunk arról, hogy a képfeldolgozó szoftverünk az felhasználói igényekhez igazítva kell nekünk a lehető legjobban optimalizálni a szoftverünket. Ugyan akkor, a felhasználók erős 95%-ban, akik használni szeretnék, azoknak teljes mértékben ajánlani tudjuk.

6. ÖSSZEFOGLALÓ

Maga a rendszámfelismerés folyamata megkönnyebbíti az életünket. Felhasználóbaráttá teszi a funkciókat, ugyanis nagyon sok minden lehet egy ilyen rendszámfelismerő rendszerrel csinálni. Hozzátehetjük, hogy ez is egy egyfajta automatizálás, amivel kiválthatóak szintén, az ember által működtetett egykor folyamat, mint amikor meg kellett állítani az autókat ahhoz, hogy aztán az okmányok alapján eldönthető legyen, hogy van-e érvényes matricája vagy sem. A projektmunkánknak köszönhetően ebbe az egész műveletbe beletekinthettünk és mi magunk megtapasztalhattuk, hogyan is zajlik egy ilyen szoftver automatizáció lefejlesztése. Megismerkedhettünk azokkal a könyvtárakkal, amiket pont a képfeldolgozás működéséről nyújt betekintést és mellette továbbfejleszthettük programozási készségeinket Pythonban. Mindezek mellett projektünkhez létfontosságú alapeleme a probléma megoldó képesség volt, mivel ahhoz, hogy logikailag megértsük és rájöjjünk, hogy egyes dolgot hogyan és miért működnek így, ezért erre a problémamegoldó készségre nagyon szükségünk volt, hogy hatékonyak legyünk. Természetesen megismerkedhetünk mindenmellett, az adatbázis működésével is, ahol az SQLite-ot használva, maximalizálhattuk a potenciálját a szoftverünk. Enélkül az adatbázis nélkül, ugyan a képfeldolgozás működik, viszont a matricák érvényességét csak is ezzel a megoldással tudjuk elraktározni.

Az egész rendszer működéséhez nem kell elég nagy hardver igénynek megfelelnie és ráadásul még felhasználóbarát megoldásokkal vannak ellátva, így gondoskodtunk, hogy a szoftverünk a közeljövőben fejleszthetőbb legyen és jövőbiztos legyen. Természetesen, projektünk melegágyat adhat egy másfajta projekteknél is, amelyekkel nem csak rendszámot, hanem esetleg objektum felismerésre is lehet felhasználni, átalakítani.

Úgy gondoljuk, hogy az OpenCV és társai, mint például a Yolov8 és hasonló algoritmusokkal rendelkező képfeldolgozó eszközök, hozzájárulnak a világ fejlődéséhez, és egy biztonságos automatizált eszközöket bevetve, kényelmünk érdekében, fontos tagjává válik.

7. SUMMARY

License plate recognition streamlines our lives, making functionalities more user-friendly, as there is a plethora of tasks such a system can handle. It is also a form of automation, replacing manual processes, like stopping cars to verify, if they have a valid sticker based on documents. Thanks to our project work, we delved into this entire operation and experienced firsthand, how to develop such software automation. We acquainted ourselves with libraries, that offer insights into image processing and honed our programming skills in Python. Alongside, problem-solving was crucial for our project, as understanding the logic behind certain functions was essential for effectiveness. We also got familiar with database operations, using SQLite to maximize our software's potential, without it, while image processing works, storing sticker validity relies solely on this solution.

The system doesn't require high hardware demands and is equipped with user-friendly solutions, ensuring our software is future-proof and easily upgradable. Moreover, our project can serve as a foundation for other projects, potentially expanding beyond license plate recognition to object recognition.

We believe that tools, like OpenCV and its counterparts, such as Yolov8 and similar algorithms in image processing, contribute to global advancement. By deploying secure automated tools for our convenience, they become crucial components in our daily lives.

8. BEFEJEZÉS

A „Matrica ellenőrző rendszer” című projektünk eredményeként sikeresen létrehoztunk egy szoftvert, ami a rendszámfelismerés technológiát alkalmazva, eltudjuk dönten, hogy egy autónak érvényes matricája van-e vagy sem. Több fő előnnyel rendelkezik, mint például, hogy nem kell már megállítani ahol az autókat, hogy le tudjuk ellenőrizni, hogy van-e érvényes matricájuk vagy sem. A tervezési folyamatból a megvalósításig, majd onnan a tesztelésig járó úton nagyon sokat megtanulhattunk a projektünkben, hogy maga a képfeldolgozás hogyan is működik, milyen könyvtárakat kell használni és hogy a valóságban milyen realitása van egy ilyen projekt megvalósítása otthoni környezetben. Az elérte eredmények alapján, nagyon sok előnye van ennek az automatizálási módszernek, mivel felhasználó barát megoldásokkal vannak ellátva és természetesen a lehető leghatékonyabb megoldásokkal megoldott szoftvert, így kezünk a legjobb teljesítményt kihozni belőle.

Nagyon sok potenciál van a képfeldolgozásnak és maga a python programozói nyelvnek, hiszen az automatizációs programok alapja az esetek többségében ezekkel az eszközökkel valósul meg. Ezek a technológiák jövőállóak, és olyan módszereket alkalmaznak, mint a mesterséges intelligencia és mélytanulás. A jövő innovációi rengeteg lehetőségek kínálnak majd nekünk a projektünk továbbfejlesztésére.

9. IRODALOMJEGYZÉK

- [1] O. Team, „Cascade Classifier,” 2 januar 2006. [Online]. Available: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.
- [2] D. Amos, „Python GUI Tkinter,” 22 március 2022. [Online]. Available: <https://realpython.com/python-gui-tkinter/>.
- [3] O. Team, „OpenCV About,” [Online]. Available: <https://opencv.org/about/>.

- [4] A. Rosebrock, „OpenCV and Tesseract OCR working,” 17 Szeptember 2018. [Online]. Available: <https://pyimagesearch.com/2018/09/17/opencv-ocr-and-text-recognition-with-tesseract/>.
- [5] A. Ghos, „YOLOv8 Object Tracking and Counting with OpenCV,” 30 január 2024. [Online]. Available: <https://learnopencv.com/yolov8-object-tracking-and-counting-with-opencv/>.
- [6] R. A. S, „What is SQLite? Everything You Need to Know,” 16 február 2023. [Online]. Available: <https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sqlite>.