

DOKUMENTÁCIÓ

Intézmény: Neumann János Egyetem

Kar: GAMF Műszaki és Informatikai Kar

Tantárgy: Haladó programozás

Hallgató: Pataki Márton

Neptun kód: Q9HLSR

Dátum: 2024. 12. 03

FELADAT MEGFOGALMAZÁSA

FORGALOMSZÁMLÁLÓ RENDSZER MEGVALÓSÍTÁSA

Objektumdetektálás és követés YOLOv8 alapokon

A feladat célja egy olyan szoftveres megoldás létrehozása, amely képes egy rögzített videofelvétel alapján a közúti forgalom automatizált elemzésére. A rendszerrel szemben támasztott követelmények:

- Detektálás:** Különböző járműkategóriák (személyautó, teherautó, busz, motorkerékpár) valós idejű felismerése.
- Követés (Tracking):** A felismerésen túl az objektumok mozgásának lekövetése, egyedi azonosítóval (ID) való ellátása.
- Számlálás:** Egy virtuális ellenőrző vonalon való áthaladás érzékelése, és a forgalom irányának (bejövő/kimenő) megkülönböztetése.
- Statisztika:** Az eredmények vizuális megjelenítése a videón és egy végső összegző táblázatban.

Járműszámláló Rendszer – Dokumentáció

I. Fejlesztői Dokumentáció

Ez a szakasz a szoftver belső felépítését, az alkalmazott algoritmusokat és az adatstruktúrákat tárgyalja. A kód egyetlen, lineárisan futó szkript (`main` scope), amely a **YOLOv8** objektumfelismerőt és a **ByteTrack** követő algoritmust használja.

1. Architektúra Áttekintés

A program egy "csővezeték" (pipeline) elven működik, amely minden egyes videóképkockát (frame) azonos lépésekben vezet keresztül.

Adatfolyam:

- Input:** Videáfajl beolvasása.
- Preprocessing:** Képkocka átméretezése a feldolgozási sebesség optimalizálása érdekében.
- Inference (Következtetés):** YOLOv8 modell futtatása objektumdetektálásra és követésre (Tracking).
- Logic (Üzleti logika):** Objektumok szűrése osztály alapján, majd a vonalátlépés vizsgálata.
- Visualization:** Eredmények (dobozok, ID-k, számlálók) rárajzolása a képre.
- Output:** Valós idejű megjelenítés és végső összegzés.

2. Könyvtárak és Függőségek

- ultralytics (YOLO):** A neurális hálózat kezelése, objektumfelismerés és követés.
- cv2 (OpenCV):** Videó beolvasása, képmanipuláció (rajzolás, átméretezés), megjelenítés.
- numpy:** Mátrixműveletek (bár a kódban közvetlenül keveset használt, az OpenCV és YOLO erre épül).
- collections (deque, defaultdict):** Hatékony adatstruktúrák az előzmények (track history) tárolására.

3. Konfigurációs Változók (Konstansok)

Változó	Típus	Érték	Leírás
VIDEO_PATH	str	"video.mp4"	A feldolgozandó videó elérési útja.
TARGET_WIDTH	int	900	A videó átméretezési szélessége (px). A magasságot arányosan számolja.

Változó	Típus	Érték	Leírás
CONF_THRESHOLD	float	0.45	Konfidenzia küszöb. Csak az ezen érték feletti találatokat veszi figyelembe.
VEHICLES	list	["car", "truck", "bus", "motorbike"]	Szűrendő osztályok listája (pl. "car", "bus").
LINE_POS	float	0.60	A számlálóvonal relatív pozíciója (0.0 - 1.0) a kép tetejétől számítva.

4. Adatstruktúrák és Állapotváltozók

A program futása során a következő dinamikus változók tárolják az állapotot:

```
counted_ids = set()
track_history = defaultdict(lambda: deque(maxlen=20))
```

- **track_history (defaultdict(deque)):**
 - **Kulcs:** track_id (egyedi azonosító, int).
 - **Érték:** deque (maxlen=20). Koordináták (x, y) listája.
 - **Cél:** Az objektumok útvonalának kirajzolása és a mozgásirány meghatározása. A maxlen=20 biztosítja, hogy csak az utolsó 20 pontot tároljuk, megelőzve a memóriaszivárgást.
- **counted_ids (set):**
 - **Tartalom:** Azon track_id-k halmaza, amelyeket már megszámoltunk.
 - **Cél:** Megakadályozza ugyanazon jármű többszöri számlálását (pl. ha a vonalon "toporog").
- **active_ids (list):**
 - **Tartalom:** Az aktuális képkockán létható járművek ID-i.
 - **Cél:** "Garbage collection" – a track_history-ból töröljük azokat, akik már nincsenek a képen (active_ids-ben nem szerepelnek).

```
active_ids = []
```

5. Algoritmusok Részletes Magyarázata

5.1. Számlálási Logika (Line Crossing Logic)

Ez a kód legkritikusabb része. A rendszer nem területet figyel, hanem egy virtuális vonal átlépését.

```

# === SZÁMLÁLÁS LOGIKA ===
if track_id not in counted_ids:
    if len(track_history[track_id]) >= 2:
        prev_y = track_history[track_id][-2][1]

        # Átlépte a számlálóvonala?
        crossed = (prev_y < line_y and cy >= line_y) or \
                   (prev_y > line_y and cy <= line_y)

        if crossed:
            counted_ids.add(track_id)

            # Mozgás irányának meghatározása
            if cy > prev_y:
                count_in += 1      # lefelé
            else:
                count_out += 1     # felfelé

```

Lépések:

- Középpont számítás:** minden detektált dobozra kiszámoljuk a középpontot:

$$C_y = \frac{y_1 + y_2}{2}$$
- Előzmény vizsgálat:** Ha az objektumnak van legalább 2 rögzített pozíciója a track_history-ban.

```

x1, y1, x2, y2 = map(int, box)
cx, cy = int((x1 + x2) / 2), int((y1 + y2) / 2)

```

Átlépés detektálása:

- Legyen P_y az előző képkocka y koordinátája.
- Legyen C_y az aktuális képkocka y koordinátája.
- Legyen L_y a számlálóvonal y pozíciója.
- Feltétel:** ($P_y < L_y$ és $C_y \geq L_y$) VAGY ($P_y > L_y$ és $C_y \leq L_y$).
- Ez matematikailag azt jelenti, hogy a pont a vonal egyik oldaláról a másikra került két egymást követő pillanatban.

5.2. Iránymeghatározás

Ha az átlépés megtörtént:

- Ha $C_y > P_y$: Az Y koordináta nőtt. A képernyő koordináta-rendszerében (ahol a (0,0) a bal felső sarok) ez **lefelé** irányuló mozgást jelent (count_in).

- Ha $C_y < P_y$: Az Y koordináta csökkent. Ez **felfelé** irányuló mozgást jelent (count_out).

6. Kód Működése (Főciklus)

1. **Incializálás:** Modell betöltése (yolov8n.pt), videó megnyitása.
2. **Frame Loop:**
 - **Resize:** A cv2.resize biztosítja, hogy nagy felbontású videóknál is gyors legyen a feldolgozás.

```
# Kép átméretezése
h_o, w_o = frame.shape[:2]
new_w = TARGET_WIDTH
new_h = int(h_o * (new_w / w_o))
frame = cv2.resize(frame, (new_w, new_h))
```

- **YOLO Track:** model.track(..., persist=True). A persist=True kritikus: ez utasítja a modellt, hogy tartsa meg az ID-kat a képkockák között (ne új objektumként kezelje őket).

```
# YOLO + tracking
results = model.track(frame, persist=True, tracker="bytetrack.yaml",
| | | | | | conf=CONF_THRESHOLD, verbose=False)
```

- **Iteráció:** Végigiterálunk a detektált objektumokon (boxes, ids, cls).

```
for box, track_id, cls_id in zip(boxes, ids, classes):
    label = model.names[int(cls_id)]
```

- **Szűrés:** if label not in VEHICLES: Ha nem jármű, eldobjuk.

```
if label not in VEHICLES:
    continue
```

- **Track History Update:** Hozzáadjuk az aktuális középpontot a deque-hez.
- **Számítás:** (Lásd 5.1 pont). Ha számoltuk, hozzáadjuk a counted_ids-hez.
- **Rajzolás:**
 - Zöld doboz: Ha már meg lett számolva.
 - Fehér doboz: Ha még nem érte el a vonalat.
 - Nyomvonal (cv2.line): Az utolsó 20 pozíció összekötése.
- 3. **Cleanup:** A ciklus végén töröljük a track_history-ból azokat az ID-kat, amelyek nincsenek az active_ids listában (kimentek a képből), hogy ne fogyon el a memória.

```
# takarítás
for tid in list(track_history.keys()):
    if tid not in active_ids:
        del track_history[tid]
```

4. **Összegzés:** A while ciklus után egy statikus kép (summary_img) generálása az np.zeros segítségével.

```
# ----- ÖSSZEGZŐ KÉPKERET -----
summary_img = np.zeros((400, 700, 3), dtype="uint8")

total = count_in + count_out

cv2.putText(summary_img, "Vegeredmény", (30, 60),
           cv2.FONT_HERSHEY_SIMPLEX, 1.3, (255, 255, 255), 3)

cv2.putText(summary_img, f"Bejovo (Le): {count_in}", (30, 150),
           cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 255, 0), 2)

cv2.putText(summary_img, f"Kimenő (Fel): {count_out}", (30, 220),
           cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 0, 255), 2)

cv2.putText(summary_img, f"OSSZESEN: {total}", (30, 300),
           cv2.FONT_HERSHEY_SIMPLEX, 1.3, (255, 255, 0), 3)

cv2.imshow("Osszegzes", summary_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

7. Hibakezelés

- Fájl megnyitás:** if not cap.isOpened() ellenőrzi, hogy a videó elérhető-e.

```
if not cap.isOpened():
    print("Hiba: nem sikerült megnyitni a videót!")
    exit()
```

- Üres detektálás:** A kód ellenőrzi (if results[0].boxes.id is not None), hogy van-e egyáltalán objektum a képen, mielőtt iterálna, elkerülve a NoneType hibákat.

```
if results[0].boxes.id is not None:
    boxes = results[0].boxes.xyxy.cpu().numpy()
    ids = results[0].boxes.id.int().cpu().tolist()
    classes = results[0].boxes.cls.cpu().numpy()
```

- Hiányzó osztályok:** A VEHICLES lista szűri a nem releváns találatokat, így téves detektálás (pl. gyalogos) nem rontja a statisztikát.

```
VEHICLES = ["car", "truck", "bus", "motorbike"]
```

II. Felhasználói Dokumentáció

Ez az útmutató segít a program telepítésében, beállításában és futtatásában. A szoftver célja, hogy videofelvételeken automatikusan megszámolja az áthaladó járműveket irány szerint.

1. Telepítés és Előkészületek

A program futtatásához Python környezet szükséges.

Rendszerkövetelmények:

- Python 3.8 vagy újabb.
- Videókártya (GPU) ajánlott, de CPU-n is fut (lassabban).

Lépések:

Könyvtárak telepítése: Nyisson meg egy parancssort (Terminal/CMD) és futtassa az alábbi parancsot:

- pip install ultralytics opencv-python numpy vagy:
- pip install -r requirements.txt

A Program Indítása

- Nyissa meg a parancssort abban a mappában, ahol a szkript található.
- Indítsa el a programot:

```
python main.py
```

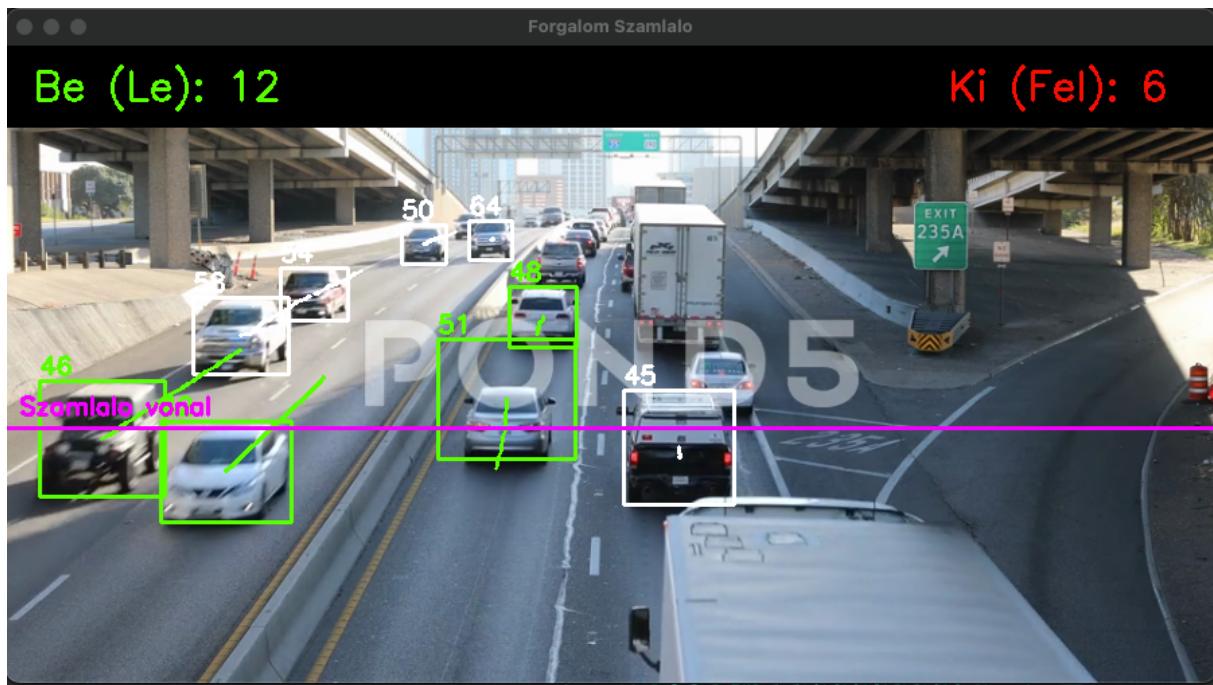
3. A Felhasználói Felület (UI) Értelmezése

A program futása közben egy ablakban látja a videót és a valós idejű adatokat.

Mit lát a képernyőn?

- Lila vonal:** Ez a virtuális "számláló kapu". Amikor egy jármű közepe áthalad ezen, a számláló növekszik.
- Jármű keretek:**

- **Fehér keret:** A járművet a rendszer követi, de még nem lépte át a vonalat.
- **Zöld keret:** A jármű áthaladt a vonalon és meg lett számolva.
- **Számok a jármű felett:** Egyedi azonosító (ID). minden jármű állandó számot kap, amíg a képernyőn van.
- **Felső panel (Fekete sáv):**
 - **Be (Le):** A fentről lefelé haladó, vonalat átlépő járművek száma (Zöld felirat).
 - **Ki (Fel):** A lentről felfelé haladó járművek száma (Piros felirat).

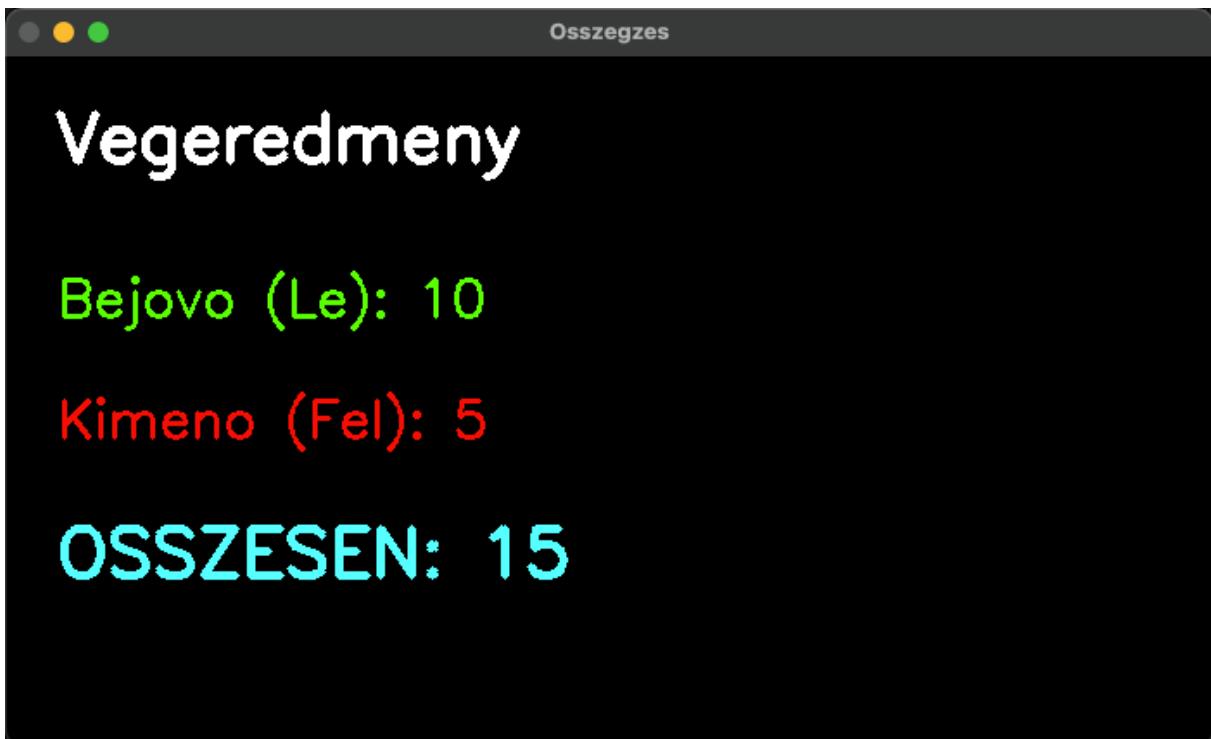


4. Végeredmény

Amikor a videó véget ér (vagy megnyomja az `ESC` billentyűt), a program bezárja a videót és megjelenít egy összegző táblát.

Az összegző képernyő tartalma:

- A "Bejövő" (Lefelé) forgalom összege.
- A "Kimenő" (Felfelé) forgalom összege.
- Az összesített forgalom.



A program bezárásához az összegző képernyőn nyomjon meg **bármilyen billentyűt**.

III. Mesterséges Intelligencia alkalmazása a projektben

A projekt szoftveres megvalósítása során a **Google Gemini** nyelvi modellt alkalmaztam fejlesztői segédeszközként. A munkafolyamat iteratív módszertannal zajlott: a specifikációk meghatározása, a generált kód validálása és a hibák detektálása az én feladatom volt, míg az AI a kódolási feladatok végrehajtásában segített.

A fejlesztés során az alábbi lépésekben keresztül alakult ki a végleges megoldás:

1. Kezdeti verzió és a felmerülő problémák Az elsődlegesen generált kód (V1) bár szintaktikailag helyes volt, a gyakorlati tesztelés során funkcionális hiányosságokat mutatott. A szoftver egy egyszerű pozíció-ellenőrzést végzett (azt vizsgálta, hogy a jármű középpontja éppen a vonalon tartózkodik-e), ami két fő problémát eredményezett:

- Hiányos irányérzékelés:** A rendszer nem tudta megkülönböztetni a mozgásirányt, így kizártlag a beérkező járműveket számolta, a kifelé haladó forgalmat figyelen kívül hagyta.
- Pontatlan számlálás:** Ha egy jármű gyorsan haladt át a képen, a detektálás kimaradt, mivel a program nem vizsgálta az előzményeket, csak az aktuális képkockát.

2. Hibajavítás és a végleges megoldás (V2) A hibák felismerése után specifikus utasításokkal (prompt engineering) kértem a kód módosítását. A fejlesztés irányára a **mozgásvektor alapú vizsgálat** és az **állapotmegőrzés** bevezetése volt. A végleges programkódba bekerült egy `deque` adatszerkezet, amely tárolja a járművek korábbi pozícióit (`track_history`). A továbbfejlesztett logika most már nem a vonalon való tartózkodást,

hanem a **vonal átlépését** vizsgálja (az előző pozíció még a vonal egyik, az aktuális már a másik oldalán van).

Ennek köszönhetően a szoftver képessé vált a **kétirányú forgalom** (be- és kilépő irány) stabil és pontos megkülönböztetésére. A végső kódot manuálisan teszteltem és validáltam a feladat követelményeinek megfelelően.

IV. Felhasznált Források és Eszközök

A projekt megvalósítása során felhasznált szoftveres könyvtárak, a fejlesztést támogató mesterséges intelligencia modell, valamint a teszteléshez alkalmazott médiaállományok listája:

Szoftveres környezet és könyvtárak:

- [1] **Ultralytics YOLOv8:** Jocher, G., Chaurasia, A., & Qiu, J. (2023). *Ultralytics YOLOv8 (Version 8.0.0)* [Computer software]. Elérhető: <https://github.com/ultralytics/ultralytics>
- [2] **OpenCV (Open Source Computer Vision Library):** OpenCV Team. (2024). *OpenCV-Python (Version 4.x)* [Computer software]. Elérhető: <https://opencv.org/>
- [3] **NumPy:** Harris, C. R., et al. (2020). *Array programming with NumPy*. Nature, 585, 357–362. Elérhető: <https://numpy.org/>

Mesterséges Intelligencia:

- [4] **Google Gemini:** Google. (2024). *Gemini Advanced* [Large Language Model]. Alkalmazva: kódoptimalizálás és hibakeresés támogatására. Elérhető: <https://gemini.google.com/>

Felhasznált médiaállományok (Tesztvideók):

A forgalomszámláló algoritmus teszteléséhez használt videóanyagok a Pond5 médiatár állományai.

- [5] **Pond5 Stock Footage:** Cím: Seattle Highway 520 Traffic Bridge Sunset (Item ID: 32089707). Elérhető: <https://www.pond5.com/stock-footage/item/32089707-seattle-highway-520-traffic-bridge-sunset>
- [6] **Pond5 Stock Footage:** Cím: View Of Busy Highway Traffic Climate Change Indicators (Item ID: 145996279). Elérhető: <https://www.pond5.com/stock-footage/item/145996279-view-busy-highway-traffic-climate-change-indicators-greenhou>