Járműszámláló Rendszer

Készítette:

Molnár Erik – FIA62J

Halmosi Martin – G5XG7K

**Tartalom**

[1 Bevezetés 3](#_Toc184273345)

[2 A program funkciói 3](#_Toc184273346)

[3 Fejlesztési kihívások és megoldásaik 3](#_Toc184273347)

[3.1 Adatkezelés optimalizálása 3](#_Toc184273348)

[3.2 Modell kiválasztása és integrációja 4](#_Toc184273349)

[3.3 Diagram frissítése 5](#_Toc184273350)

[3.4 Interaktív felület 5](#_Toc184273351)

[4 A program működésének áttekintése 6](#_Toc184273352)

[4.1 Videófájl kiválasztása és kezelése 6](#_Toc184273353)

[4.2 Járműfelismerés és számlálás 6](#_Toc184273354)

[4.3 Valós idejű vizualizáció 7](#_Toc184273355)

[5 Összegzés és jövőbeli fejlesztések 7](#_Toc184273356)

# Bevezetés

A járműszámláló rendszer célja, hogy valós időben képes legyen videók alapján járműveket felismerni, kategorizálni és a statisztikai adatokat vizuálisan megjeleníteni. Az alábbi dokumentáció bemutatja a program működését, a fejlesztés során felmerült kihívásokat és azok megoldását, valamint a tanulságokat. A kód kiegészítéseivel érthetőbbé tesszük a megoldásokat.

# A program funkciói

A program a következő funkciókat valósítja meg:

1. **Videófájl betöltése:** A felhasználó videófájlt tölthet be a rendszerbe.
2. **Járműfelismerés valós időben:** A YOLOv5 modellt használva képes járműveket felismerni és kategorizálni.
3. **Statisztikai adatok frissítése és megjelenítése:** A program diagramokon jeleníti meg a járművek típusonkénti számát.
4. **Felhasználói vezérlés:** Videólejátszás indítása, szüneteltetése, illetve a lejátszási sáv kezelésének lehetősége.

# Fejlesztési kihívások és megoldásaik

## Adatkezelés optimalizálása

**Probléma:**

* A nagy felbontású videók képkockáinak feldolgozása túl lassú volt a valós idejű megjelenítéshez.

**Megoldás:**

* Csökkentettük a feldolgozott képkockák számát. Minden második képkockát feldolgozunk az alábbi módon:

if self.current\_frame % 2 == 0:

# Csak minden második képkockát dolgozzuk fel

frame = cv2.resize(frame, (self.canvas.winfo\_width(), self.canvas.winfo\_height()))

results = self.model(frame)

**Tanulság:** A képfeldolgozási algoritmusok egyszerű optimalizációja jelentősen javíthatja a teljesítményt, anélkül hogy az eredményességet befolyásolná.

## Modell kiválasztása és integrációja

**Probléma:**

* Az eleinte kiválasztott modellek, mint például a Faster R-CNN, nem voltak elég gyorsak valós idejű alkalmazásokhoz.

**Megoldás:**

* A YOLOv5 „s” verzióját választottuk, amely gyorsabb és könnyebben integrálható. A modell előképzett változatát töltöttük be az alábbi módon:

self.model = torch.hub.load('ultralytics/yolov5', 'yolov5s')

Ez a sor egy könnyen alkalmazható, előre betanított modellt biztosít, amely támogatja a leggyakoribb járműtípusokat.

**Tanulság:** A gépi tanulási modell kiválasztásakor mindig figyelembe kell venni a feldolgozási sebesség és a pontosság közötti egyensúlyt.

## Diagram frissítése

**Probléma:**

* A valós idejű adatfrissítés késleltette a diagram megjelenítését.

**Megoldás:**

* A diagramot dinamikusan frissítettük, miközben megőriztük a feldolgozási sebességet. Az adatok frissítése minden feldolgozott képkocka után történik:

# Diagram frissítése

self.ax.clear()

self.ax.set\_facecolor("#1e272e")

vehicle\_types = ["Car", "Truck", "Motorcycle"]

counts = [cars\_count, trucks\_count, motorcycles\_count]

self.ax.bar(vehicle\_types, counts, color=['green', 'blue', 'yellow'])

self.bar\_chart.draw()

**Tanulság:** A Matplotlib diagramok dinamikus frissítése lehetséges megfelelő tervezéssel, anélkül hogy a rendszer teljesítménye jelentősen csökkenne.

## Interaktív felület

**Probléma:**

* A Tkinter reszponzív elrendezésének megvalósítása kihívást jelentett, különösen az eltérő képernyőméretek kezelése során.

**Megoldás:**

* A Tkinter rowconfigure és columnconfigure funkcióit használtuk az egyes keretek méretezhetőségének beállítására:

self.window.rowconfigure(0, weight=1)

self.window.columnconfigure(0, weight=2)

self.window.columnconfigure(1, weight=1)

**Tanulság:** A felhasználói élményt nagyban javítja a reszponzív elrendezés, amely alkalmazkodik a különböző képernyőméretekhez.

# A program működésének áttekintése

## Videófájl kiválasztása és kezelése

* A felhasználó videófájlt választhat a select\_video metódussal:

def select\_video(self):

self.video\_path = filedialog.askopenfilename(filetypes=[("Video files", "\*.mp4 \*.avi")])

if self.video\_path:

self.cap = cv2.VideoCapture(self.video\_path)

self.playing = False

## Járműfelismerés és számlálás

* A YOLOv5 modell az aktuális képkockán végzi az objektumok felismerését. Az eredményeket pandas DataFrame formában dolgozzuk fel:

results = self.model(frame)

detected\_objects = results.pandas().xyxy[0]

## Valós idejű vizualizáció

* A Matplotlib diagram és a Tkinter vászon frissítése lehetővé teszi az azonnali visszacsatolást a felhasználó számára.

# Összegzés és jövőbeli fejlesztések

A fejlesztési folyamat során megoldottuk a valós idejű feldolgozás és megjelenítés kihívásait, valamint modern, felhasználóbarát felületet alakítottunk ki.

**Jövőbeli fejlesztési lehetőségek:**

* Az objektumkövetés integrálása a pontosabb számlálás érdekében.
* Az előképzett modell továbbfinomítása specifikus igényekre.
* A Streamlit vagy más modern UI keretrendszer használata a Tkinter helyett.