
**Budapesti Műszaki és Gazdaságtudományi Egyetem**
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

# Intelligens képfeldolgozó rendszer integrálása CPS környezetbe

*Készítette:*
Garzó Benjamin

*Konzulens:*
Huszerl Gábor
*mestertanár*

2020

MŰEGYETEM 1782

SZAKDOLGOZAT FELADAT

**Garzó Benjamin János**
mérnökinformatikus hallgató részére

# Intelligens képfeldolgozó rendszer integrálása CPS környezetbe

Az informatikai rendszerek egy igen jelentős része ún. kiberfizikai rendszer (angolul Cyber-Physical System, CPS), amely általában egy elosztott, sokszor kifejezetten nagy méretű, összetett rendszer. Ezen rendszerek további jellemzője, hogy találkozik bennük az informatika (kiber-) és a fizikai világ, hogy képesek dinamikusan alkalmazkodni a változó környezethez, és képesek kihasználni a felhőszolgáltatások számítási lehetőségeit. A kiberfizikai rendszerek magukba foglalják például a manapság széleskörűen elterjedt (ipari) IoT rendszereket.

Ezen rendszerek megvalósításának egyik jellemző problémája a különböző integrációs kihívások kezelése. A rendszer különböző komponensei nagyon különböző platformokon különböző technológiákkal nagyon különböző feladatokat láthatnak el, mégis megbízhatóan, sokszor biztonság- és időkritikusan kell együttműködniük. Egy másik intergációs kihívás, hogy a különböző komponensek által ellátandó feladatok sokszínűsége miatt sokszor van szükség meglévő eszközöknek és technológiáknak a felhasználására, rendszerbe illesztésére.

Tanszékünkön tervezzük egy oktatási-kutatási célú CPS demonstrátor elkészítését, amely a fentiekben leírt sokféleséget és az ebből adódó tervezési nehézségeket is bemutatja. Ezen demonstrátor egy részében tervezzük intelligens képfeldolgozó eszközök (automatikus objektum osztályozás, felirat-leolvasás, sebességbecslés mozgóképeken) integrálását is, jelen szakdolgozat célja pedig ezen részfeladat közelebbi vizsgálata.

A hallgató feladatának a következőkre kell kiterjednie:

1. Tekintse át és hasonlítsa össze a fenti intelligens képfeldolgozási feladatokra használható meglévő technológiákat és eszközöket. A CPS környezetbe való integráláshoz tekintse át a DDS technológia nyújtotta lehetőségeket.
2. Válasszon ki a képfeldolgozási feladatokra képes és CPS környezetben használható eszközöket, és tervezze meg ezek DDS alapú integrálását egy nagyobb rendszerbe.
3. A megtervezett eszköz egy prototípusának elkészítésével demonstrálja a koncepció működőképességét.

**Tanszéki konzulens:** Huszerl Gábor, mestertanár

Budapest, 2020.10.15.

……..
Dr. Dabóczi Tamás
tanszékvezető egyetemi tanár, DSc

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

1117 Budapest, Magyar Tudósok krt. 2. I. ép. I.E.444.
Telefon: 463-2057, Fax: 463-4112
http://www.mit.bme.hu • e-mail: mitadm@mit.bme.hu

# HALLGATÓI NYILATKOZAT

Alulírott Garzó Benjamin, hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik.

Kelt: Budapest, 2020. 12. 07.

.....

Garzó Benjamin

# Kivonat

A 4. ipari forradalom egy fontos része az úgynevezett kiberfizikai rendszerek (CPS), amelyek elosztott, nagy méretű, összetett rendszerek. Szintén az elmúlt években (függetlenül a 4. ipari forradalomtól) lettek népszerűek az intelligens képfeldolgozással foglalkozó új technológiák. Szakdolgozatomban e kettő terület együttes használatába adok betekintést.

Feladatom, hogy egy intelligens képfeldolgozó alrendszert integráljak CPS környezetbe. A szakdolgozatomban részletezésre kerül, milyen rendszertervvel kell megközelíteni a feladatot, milyen technológiák szükségesek a megvalósításhoz, milyen képfeldolgozási módszertanokat lehet használni az alrendszerünkhöz, hogyan megy végbe ezen alrendszer különböző komponenseinek integrálása, illetve magát az alrendszert hogyan kell integrálni. Befejezésként pedig bemutatom és értékelem az általam elkészített prototípust.

# Abstract

One of the most important invention of the 4th industrial revolution are the cyber-physical systems (CPS). In recent years multiple intelligent image processing technologies became popular. In my thesis I'll demonstrate how to use these technologies together.

My object is to integrate an intelligent image processing subsystem into a CPS environment. I'm going to discuss what kind of system model does it take to represent my object, what technologies are required for my system, what image processing methodologies can be used for implementing my subsystem, how to integrate various components of our subsystem and finally how to integrate our subsystem into our final system. To finish it all off, I'm going to showcase my prototype and evaluate it.

# Tartalomjegyzék

# 1 Bevezetés

Ahogyan a szakdolgozatom címe ”Intelligens képfeldolgozó rendszer integrálása CPS környezetbe” leírja, a feladatom egy integrálási feladat. Egy ilyen feladat célja, hogy megismerkedjek a piacon általam elérhető eszközökkel, amik teljesítik a képfeldolgozó rendszerem feladatait és ezeket integráljam. Amennyiben nem elérhetőek megfelelő eszközök, két választásom van: Extra munkával létrehozok megfelelő projektet, vagy kutatásaim eredményét leírva azt a konklúziót vonom le, hogy adott eszköz hiányában bizonyos része az alrendszernek nem integrálható. Én az első opciót választottam. E választás eredménye, hogy szakdolgozatomban nem csak a technológiákat mutatom be és az integrálás folyamatát, hanem az elérhető módszertanokat is ismertetem. Ezek mellett egy rendszertervet is létrehoztam, illetve különböző kommunikációs követelményeket fogalmaztam meg a rendszeremmel kapcsolatban. Munkámat egy prototípus létrehozásával fejeztem be, amit be is mutatok és értékelek.

A szakdolgozatot a következő képpen közelítettem meg: A 2. fejezetben a képfeldolgozó és egyéb általam használt technológiákat mutatom be. Ezeket megismerve a 3. fejezetben a rendszerterv felvázolásával folytatom. Mivel a kutatómunkám alatt nem találtam megfelelő eszközt, minden képfeldolgozó feladat ellátáshoz. így a 4. fejezetben részletezem a munkámhoz releváns képfeldolgozó módszertanokat. Az 5. fejezetben az ismertetett módszertanok felhasználásával és a rendszerterv alapján elvégzett integrációt részletezem, majd a 6. fejezetben bemutatom és értékelem az elkészített prototípust.

# 2 Használt technológiák

Ebben a fejezetben a rendszeremhez használt technológiákat fogjuk megtekinteni. A felhasznált technológiákat két kategóriába osztom: képfeldolgozó és egyéb. Az általam ismertetett képfeldolgozó technológiákat később összehasonlítom. Először viszont szeretném ismertetni röviden, mik is ezek a technológiák.

## 2.1 Képfeldolgozó technológiák

Véleményem szerint megfelelő integrálást úgy lehet elvégezni, hogy előtte megismerjük a piacon meglévő elkészült projektek által használt technológiákat. A feladatomként adott integrálást is hasonlóan közelítettem meg, így mielőtt munkához láttam volna, megtekintettem a piacon elérhető képfeldolgozó technológiákat (Matlab, OpenCV, Tensorflow). Bizonyos esetekben különböző összehasonlításokat is elvégeztem.

### 2.1.1 Matlab

A Matlab [1] egy a MathWorks által fejlesztett programcsomag, a neve a Matrix Laboratory névből ered. Elsősorban mérnököknek fejlesztett programokat tartalmaz, de manapság különböző tudományterületeken is használják. Mondhatjuk, hogy mérnöki és matematikai problémák megoldására, szimulálására és grafikus adatmegjelenítésre lett létrehozva. A Matlab egy egyszerű felhasználói felületet szolgáltat iteratív analízis elvégzésére, illetve egy magasszintű programozási nyelvet, ami közvetlenül tud kifejezni mátrix- és halmazmatematikát. A Matlab segítségével egyszerűen tudunk létrehozni scripteket, amik kódot és kimenetet kombinálnak egy futtatható notebookban. A Matlabhoz fizetős license tartozik, bár a kiadott pénzért cserébe egy olyan terméket szolgáltatnak, amihez kimerítő dokumentáció és intuitív felhasználói felület tartozik.

### 2.1.2 OpenCV

Az OpenCV (*Open Source Computer Vision Library*) [2] egy ingyenesen használható, gépi látás megvalósításához létrehozott könyvtár. Python, C, C++ nyelveken lehet használni. Több ezer előre megírt algoritmussal rendelkezik, számtalan képmanipulációs eszközzel. Használata megkönnyíti képek és videók betöltését, illetve kezelését. Az OpenCV számítási hatékonyságra lett tervezve, mellette pedig nagy hangsúlyt fektettek a valós idejű használatra. C/C++ nyelven íródott, és képes multi-core feldolgozásra.

### 2.1.3 TensorFlow

Alapvetően a TensorFlow ("*Open Source Software Library for Machine Intelligence*") [3] egy gépi tanuláshoz létrehozott keretrendszer. Szolgáltat könyvtárat adatok feldolgozására, külön könyvtára, a Keras pedig különböző modellek (neurális hálók stb.) építésére és tanítására van kitalálva. A Python, JavaScript, Swift nyelveket támogatja, illetve minden nyelven rendelkezésre áll GPU általi gyorsítás is. A Tensorflow egy nyílt forráskódú könyvtár, ami adatfolyam gráfok által végzett numerikus számításokra van kitalálva. A gráfok csomópontjai matematikai műveleteket jelentenek, a gráf végpontjai pedig a többdimenziós adattömböket (ezeket hívják *tensor*nak). Rugalmas felépítése miatt lehetőségünk van egy számolást több CPU-n vagy GPU-n elvégezni.

### 2.1.4 Matlab és OpenCV összehasonlítása

Három alapvetően különböző összehasonlítást is találtam erről a két eszközről. Első általam említett egy Egyiptomi egyetem emberei által elkészített teljesítményalapú összehasonlítás. Ezután az interneten megtalálható felhasználói véleményeknek jártam utána (2.1.4.3 fejezet) és később rátaláltam ezen a véleményeket alátámasztó cikknek (2.1.4.2 fejezet).

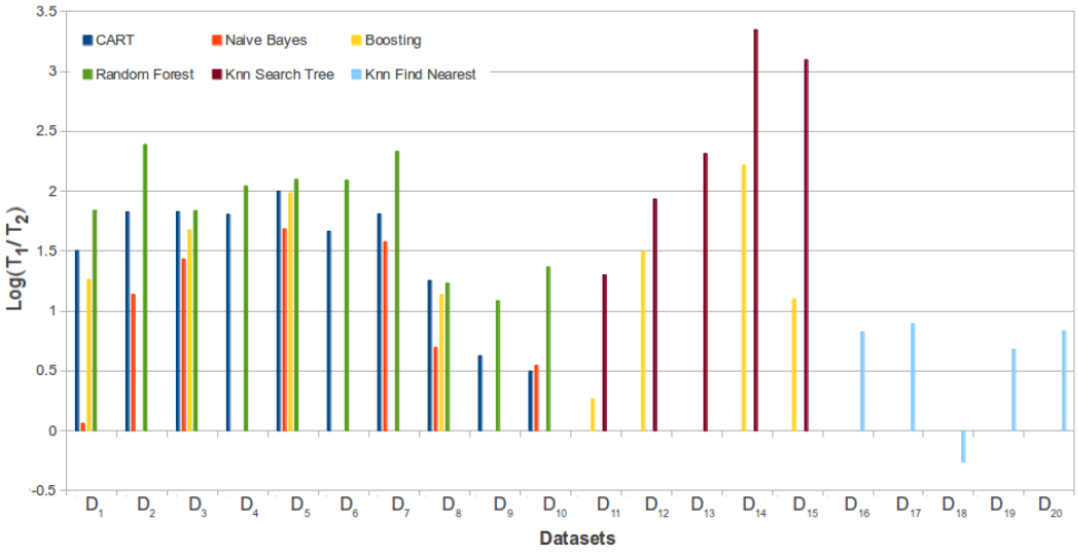
#### 2.1.4.1 Teljesítmény alapú összehasonlítás

Egy a Helwan egyetem által kiadott tanulmány [4] keretei között összehasonlították a Matlabot és az OpenCV-t teljesítményük alapján. Különböző algoritmusokat hasonlítottak össze, minden algoritmust különböző adathalmazokon 1000-szer futtattak le, és a lefutási idő átlagát vették. Voltak esetek, ahol az OpenCV akár 80-szor is gyorsabb volt, mint a Matlab.

| | Data sets | $n$ | $p$ | $C$ |
|---|---|---|---|---|
| $D_1$ | Connectionist Bench | 208 | 60 | 2 |
| $D_2$ | Breast Tissue | 106 | 9 | 4 |
| $D_3$ | Blood Transfusion | 748 | 4 | 2 |
| $D_4$ | Glass Identification | 214 | 9 | 6 |
| $D_5$ | Haberman's Survival | 306 | 3 | 2 |
| $D_6$ | Image segmentation | 210 | 19 | 7 |
| $D_7$ | Iris | 150 | 4 | 3 |
| $D_8$ | MAGIC Gamma | 19020 | 10 | 2 |
| $D_9$ | shuttle | 43500 | 9 | 7 |
| $D_{10}$ | Letter Recognition | 20000 | 17 | 26 |
| $D_{11}$ | Arcene | 100 | 10000 | 2 |
| $D_{12}$ | Madelon | 2000 | 500 | 2 |
| $D_{13}$ | Pen Based Recog. | 7494 | 16 | 9 |
| $D_{14}$ | SPECT Heart | 80 | 22 | 2 |
| $D_{15}$ | SPECTF Heart | 80 | 44 | 2 |
| $D_{16}$ | Arcene Test | 167 | 10000 | 2 |
| $D_{17}$ | Madelon Test | 1800 | 500 | 2 |
| $D_{18}$ | Pen Based Recog. Test | 3489 | 16 | 9 |
| $D_{19}$ | SPECT Heart Test | 187 | 22 | 2 |
| $D_{20}$ | SPECTTF Heart Test | 187 | 44 | 2 |

**1. ábra: Használt adathalmazok számozva**

Az 1. ábra bemutatja az adathalmazokat, ezek D-vel vannak jelölve. Az ábrán látható n, p és C értékek a következőek sorrendben: megfigyelések száma, jellemzők száma, osztályok.




**2. ábra: A végrehajtási idő logaritmusa Matlab és OpenCV között**

A 2. ábra vízszintes egyenese az adathalmazokat jelenti, amiket az 1. ábra is megmutatott nekünk. A függőleges egyenesen pedig a T1 az OpenCV teljesítménye, a T2 pedig a Matlab teljesítménye és a megjelenített érték pedig a T1/T2 természetes

logaritmusa. A természetes alapú logaritmus 1-nél 0. Tehát ahol az OpenCV teljesített jobban, ott 0-nál nagyobb számot látunk, ahol a Matlab teljesített jobban, ott pedig 0 alatti számot.

Az általam hivatkozott tanulmány továbbá alátámasztja, amit a 2.1.4.3 fejezetben különböző felhasználók tapasztalatai alapján elvégzett összehasonlításban fogok bemutatni, miszerint a Matlab egyszerűbb kezdő felhasználóknak, viszont a gyorsaságban és árban elhagyja az OpenCV.

#### 2.1.4.2 Az Analytics Insight által végzett összehasonlítás

Az Analytics Insight nevű cég is elvégzett egy Matlab, OpenCV összehasonlítást [5], aminek az eredményei továbbra is alátámasztják a felhasználók által tapasztaltakat. Cikkükben leírják mindkét szoftver előnyeit, illetve a végén egy konklúziót is hoznak, miszerint az OpenCV-t nehezebb kezdőként használni, de megéri ragaszkodni hozzá. Matlab viszont kezdőknek ajánlottabb, hiszen könnyebb debugolni a kódot, egyszerűbb benne kódot írni.

#### 2.1.4.3 Felhasználók tapasztalatai alapján összehasonlítás

Az fejezetben megtalálható összehasonlítás az interneten található felhasználói vélemények, fórumokon való felszólalásokból állítottam össze. Legtöbb segítséget a Quorán [6] adott válaszok nyújtottak.

**Matlab előnyök:**

- Egyszerű lineáris algebrát elvégezni a Matlab segítségével. Egyszerű és intuitív a használata, képes nagyméretű mátrixokat kiszámolni, azok inverzét stb.
- Magasabb szintű programozási nyelv, így egyszerűbb használni és kódot írni benne, kezdő felhasználóknak egyszerűbb.
- A beépített képfeldogozó könyvtárukkal könnyen lehet megjeleníteni az eredményeket, könnyen lehet képet importálni. A hozzátartozó integrált fejlesztői környezettel könnyebben lehet dolgozni.
- Több a szolgáltató által kiadott dokumentáció.

**Matlab Hátrányok:**

- A BME-nek van licence a Matlabhoz, viszont, ha a jövőre tervezünk, illetve minél elérhetőbbé akarjuk tenni a képfeldolgozó projektünket, nem érdemes használni a Matlabot, mivel nagyon drága egy licence.
- Alapvetően lassabb a Matlab, mint az ellenfelei, különösen valós idejű videó feldolgozásban, kisebb kapacitású eszközökön (pl. Raspberry Pi).

**OpenCV előnyök:**

- Gyorsabb, mint a Matlab, és ingyenes. Egy jól optimalizált kód, sokkal gyorsabb tud lenni, mint a Matlab, ingyenessége pedig sokkal jobban elérhetővé teszi.
- Az OpenCV közössége közel 40000 fejlesztőből áll, akik az interneten osztják meg az OpenCV-ben talált és saját maguk által elkövetett hibáikat, továbbá hozzájuk megoldásaikat. Ezzel sokkal könnyebb és gyorsabb fejlesztést eredményez az OpenCV.
- A forráskódja C/C++ -ban van megírva, így bármilyen eszköz, ami képes C/C++ kódot futtatni, képes futtatni az OpenCV-t is. Ezzel egy sokkal hordozhatóbb komponens tud lenni a képfeldolgozó része a rendszerünknek.

**OpenCV hátrányok:**

- Kezdőknek nehezebb, főleg olyan kezdőknek, akiknek nincsen tapasztalatuk C++-ban vagy Pythonban. Nehezebb kezelni az előbb felsorolt nyelveken a mátrixokat, mint Matlabban.
- Kevesebb a gyártó által szolgáltatott dokumentáció, mint a Matlabnak.

### 2.1.5 Tensorflow és OpenCV összehasonlítása

Az OpenCV az *„Image Processing and Management”* kategóriába tartozik, amíg a Tensorflow a *„Machine Learning Tools”* kategóriába. Innen is látszik, hogy a két szoftver összehasonlítása szinte alaptalan lenne. A Tensorflow egy sokkal általánosabb megoldást szolgál, míg az OpenCV kifejezetten képfeldolgozásra és gépi látásra van kifejlesztve. A Tensorflow is képes képfeldolgozást megvalósítani, de a szoftver általánossága miatt sokkal nehezebb ezt megtenni, és kevesebb a hozzá tartozó eszköz, ami segítené ezt. Ezzel ellentétben viszont, ha valaki tapasztalt a Tensorflow-ban, sok munkával képes lehet olyan képfeldolgozási feladatok elvégzésére, mint OpenCV-ben, cserébe az eredményül kapott rendszer nagyobb teljesítménnyel bírhat, illetve rugalmasabb esetleges változtatásokra. A kettő technológia között nincs még teljesítménybeli összehasonlítás, mivel ahogy említettem, a kettő nagyon eltér egymástól. Annyira eltérőek, hogy a legjobb megoldást sokszor a két technológia együttes használata adja meg.

## 2.2 Egyéb technológiák

Ebben a fejezetben azon technológiákat sorolom fel, amik nem fértek az előzőleg felsorolt képfeldolgozó technológiák közé. Első említett technológia a Point-cloud Library, ez a technológia is képfeldolgozáshoz használható, viszont alapjaiban más, mint

az általam felsorolt képfeldolgozó technológiák, így külön csoportosítottam. Szó lesz még a DDS-ről is, hiszen ez volt a feladatomban a kommunikációt megvalósító technológia.

### 2.2.1 Point-cloud Library (PCL)

A Point Cloud Library [7] egy nyílt forráskódú könyvtár, ami azért lett létrehozva, hogy point cloud modelleket (Point cloud modellekről később) dolgozzon fel, és 3D-s geometriai feladatokat végezzen el. Mindezt azért, hogy háromdimenziós gépi látást valósítsunk meg.

Mielőtt akármit is írnánk a PCL-ről, először meg kell említeni a technológiát, amin az egész könyvtár alapul, ez a Point-cloud modeling. A point-cloud modeling lényege, hogy háromdimenziós tárgyakat nem alapvető formák (háromszög) felépítésével vagy matematikai képletekkel modellezünk, hanem a háromdimenziós objektumunk felületén sűrűn elhelyezett pontokkal. Elég sűrűn elhelyezett pontokkal képesek vagyunk komplex felületek reprezentálására, például az emberi arcéra.

**Előnyei:**

- Point-cloud modelinggel képesek vagyunk véges számú ponttal pontosan reprezentálni relatívan komplex objektumokat.
- Point-cloud modellek lehetnek a leggyorsabban elkészített modellek, ha van hozzáférés 3D scanneléshez.

**Hátrányok:**

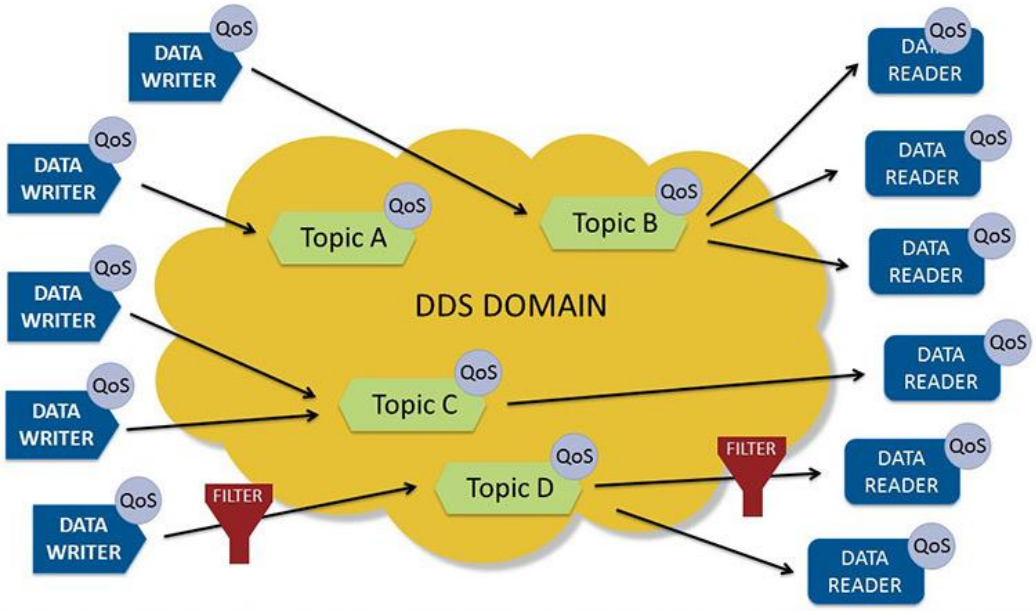
- A point-cloud modellek nem olyan pontosak, és nem képesek matematikailag tökéletes ívek leírására.
- Nem tartalmaznak információt a felületekről, így nem lehet natívan renderelésre vagy gyártásra használni.
- Nehéz lefordítani sokszögű modellekre.

A PCL egy könyvtár, ami segít point-cloud modellek létrehozásában és leírásában. Alapvetően a PCL csak az objektumok leírásában segít, így leghasznosabb más algoritmusokkal kiegészítve. Specifikus képfeldolgozó feladatokban, modellezésben hasznos lehet ez a technológia, egyéb helyeken viszont elmarad a társaitól.

### 2.2.2 Data Distribution Service

Data DistributionService (DDS) [8] egy az ObjectManagementGroup (OMG) [9] által kifejlesztett szabvány. Segítségével különböző eszközök között lehet valós időben

adatokat megosztani. További előnyei a teljesítmény és a skálázhatóság. Az adatok megosztása publisher/subscriber elven valósul meg.




**3. ábra: A DDS működése [10]**

A 3. ábra a DDS működési elvét ábrázolja. A publisher felelős az adatmegosztásért. Egy publisher többféle adattípust is publikálhat különböző témákban. Egyes adattípusok kezeléséért külön-külön data writerek felelnek. A subscriber felelős a publikált adatok fogadásáért. Képes többféle adattípust fogadni, többféle témából. Egyes adattípusok kezeléséért külön-külön data readerek felelnek. Ahogy az látható, megadhatunk különböző QoS (Quality of Service) beállításokat is.

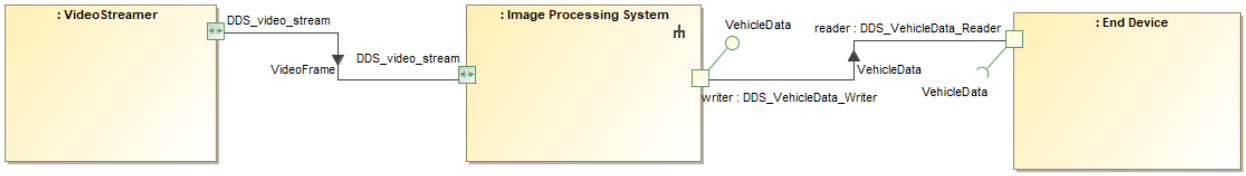
A prototípusimplementálás során fontos szempont, hogy az adatszerkezetek egységesen legyenek leképezve minden nyelvre, tehát szükségünk van egy közös leíró nyelvre. Ebből több is létezik, az implementáció során az XML alapú leíró nyelvét [11] használtam.

# 3 Rendszerterv

Ebben a fejezetben az általam elkészített rendszertervet fogom bemutatni, amit az intelligens képfeldolgozó rendszer integrálását megvalósító prototípusnál használtam fel. A tervezés alatt a kommunikációra vonatkozóan a következő követelményeket sikerült azonosítanom, amit teljesítenie kell a rendszeremnek. Ezen követelmények a következőek:

1. Folyamatos módon megszakítás nélkül és állandó gyorsasággal működjön a képkockák elküldése.
2. Képkocka vesztés aránya ne legyen túl magas, számszerűsíteni ezt az arányt nem igazán lehet.

A rendszerterv megvalósításáról a 5. fejezetben írok részletesen.




**4. ábra: Rendszerterv MagicDraw-ban elkészítve**

A 4. ábra bemutatja az általam elkészített rendszertervet, a továbbiakban ezt a tervet fogom részletezni.

## 3.1 Komponensek

Ebben a fejezetben a rendszertervem komponenseit fogom bemutatni.

### 3.1.1 VideoStreamer

A rendszerben ez a komponens látja el a videóküldés feladatát. Alapvetően egy flowport-on keresztül van reprezentálva a képkockák küldése, ez viszont a valóságban DDS-sel lesz megvalósítva. Ez a rendszerterven is fel van tüntetve a port nevével.

### 3.1.2 Image Processing Subsystem

A rendszerünk közepén helyezkedik el a képfeldolgozó alrendszer. A rendszertervemben blackbox-ként van bemutatva, tehát belső működését nem fogom

részletezni. Feladata, hogy a bejövő képkockákat feldolgozza, és DDS-sel továbbküldje a kiszűrt adatokat.

### 3.1.3 End Device

Ez a komponens reprezentálja azon eszközöket, amelyek fogadják a képfeldolgozó alrendszer eredményeit. Az eredmények fogadása után az eredményekből valamilyen konklúziót kell levonnia, és akár beavatkozást is elrendelhet, ezen részletek megvalósítása nem része a szakdolgozatomban megfogalmazott feladatnak.

## 3.2 Komponensek együttműködése

A 4. ábra nem csupán a rendszeremben szereplő komponenseket mutatja be, hanem a közöttük lévő kapcsolatokat is, az adatok típusát, illetve áramlását is. Először a VideoStreamer valamilyen formában videót biztosít (fogadja kívülről vagy belső kamerával felveszi), és ezt továbbküldi a képfeldolgozó alrendszernek. A képfeldolgozó alrendszer kiértékeli ezt a képkockát, és a számításait DDS-sel publikálja egy vagy több End Device-nak. Ezek a végfelhasználók eldöntik, mit tegyenek ezen adatokkal.

# 4 Képfeldolgozó alrendszerhez tartozó módszertanok

Ugyan ez a szakdolgozat nem képfeldolgozásról szól, viszont az is erős szerepet játszik benne. Úgy érzem, sikeres és hatékony integráláshoz elengedhetetlen, hogy ismerjük egy szintig az integrálni szükséges projektek, technológiák alapjait. Mellesleg hosszas kutatásom után nem sikerült a képfeldolgozó alrendszerem minden feladatát kielégítő eszközöket találnom, így magam kellett megalkotni az alrendszer alapját. Az imént felsorolt két ok miatt tartottam fontosnak a módszertanok ismertetését és megismerését.

Ebben a fejezetben azon módszertanokat szeretném ismertetni, amelyek a képfeldolgozó alrendszer feladatainak elvégzéséhez szükségesek. Ezen feladatok a következőek:

- Rendszámtábla-felismerés
- Járműdetektálás és -osztályozás
- Sebességbecslés

## 4.1 Rendszámtábla-felismerés

A rendszámtábla felismerés két különböző módját tartom relevánsnak megemlíteni. Segítségemre volt a 4.1.1 fejezetben tárgyalt módszertan bemutatásában egy a Mediumon közzétett cikk [12]. A 4.1.2 fejezetben bemutatott módszertanhoz is egy Mediumon közzétett cikk [13] vált segítségemre.

### 4.1.1 Rendszámtábla-felismerés kizárólag képfeldolgozó algoritmusok használatával

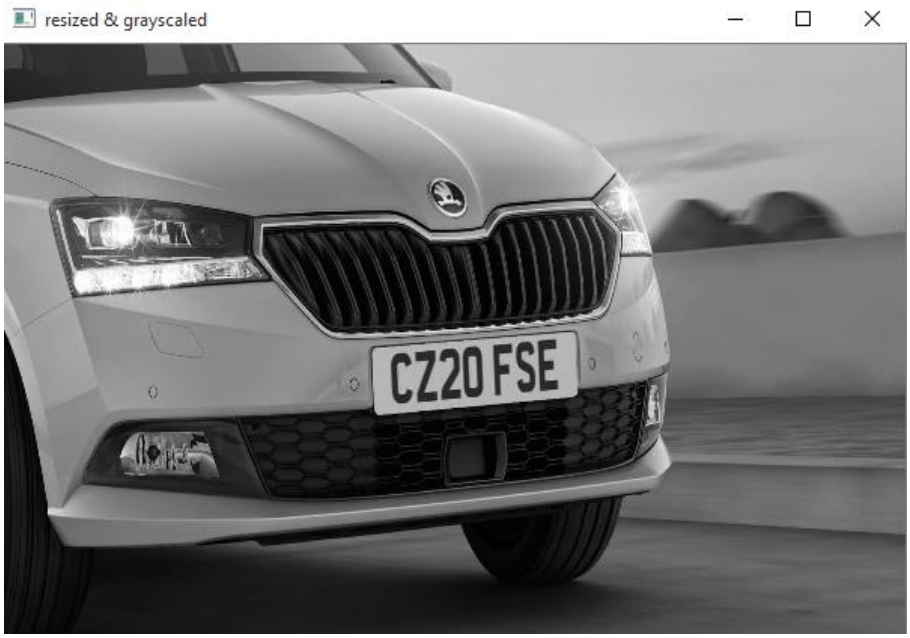
A rendszámtáblafelismerési feladatnak három lépése van:

1. Jármű felismerése
2. Rendszámtábla megtalálása
3. Rendszámtábla leolvasása

Az a módszertan, amit most fogok részletezni, az első kettő lépést képfeldolgozási technológiákkal hajtja végre.

Először betöltünk egy képet, amin látható egy rendszámtábla. A módszertan első lépése, hogy a megadott képet szürkévé alakítja (5. ábra), azaz eltünteti a piros, zöld és kék

csatornákat, preferencia alapján át is méretezhetjük a képet. Ez akkor előnyös, ha nagyon nagy felbontású a képünk.

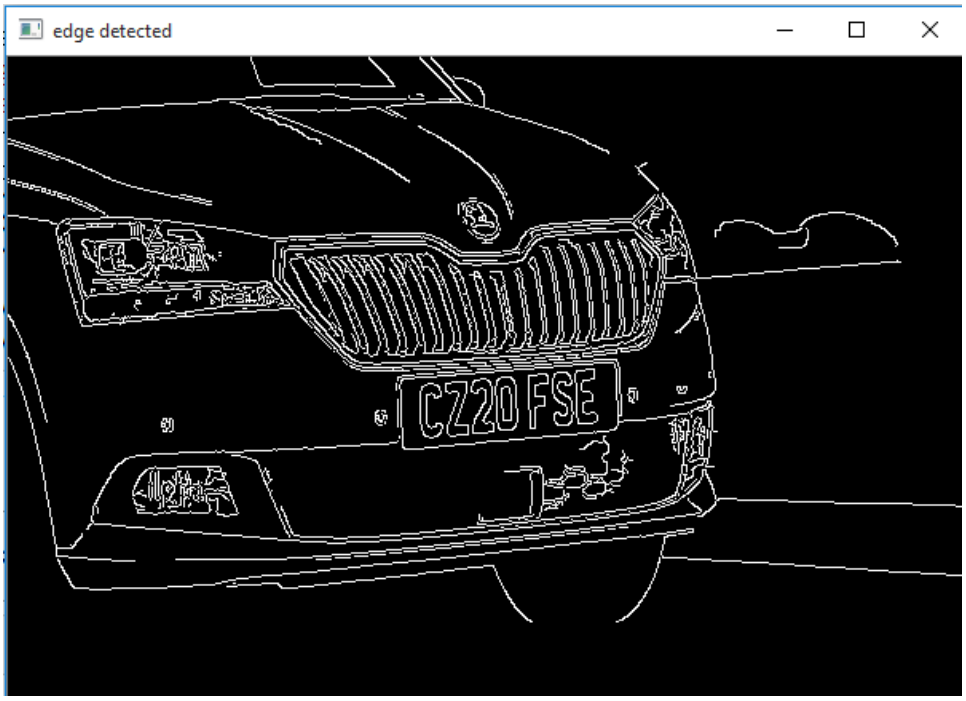



**5. ábra: Átméretezett és szürkített kép**

Ezután egy homályosítási módszerrel kiszűrjük a háttérzajt. A háttérzaj kiszűrése jobban észrevehető bonyolultabb képeken, de célja, hogy a háttér komplexitását csökkentsük, ezzel is növeljük a sikeres objektumfelismerés esélyét. Ezt a homályosítást bilaterális szűréssel [14] tesszük meg. A szűrt képet a 6. ábra mutatja be nekünk.


**6. ábra: Háttérzaj kiszűrése utáni kép**

A következő a küszöbértékek megjelenítése. A küszöbértékeket megjelenítő képen (7. ábra) megkeressük az összes kontúrt, és ezek közül különböző eljárásokkal megkeressük a rendszámtáblánkat.



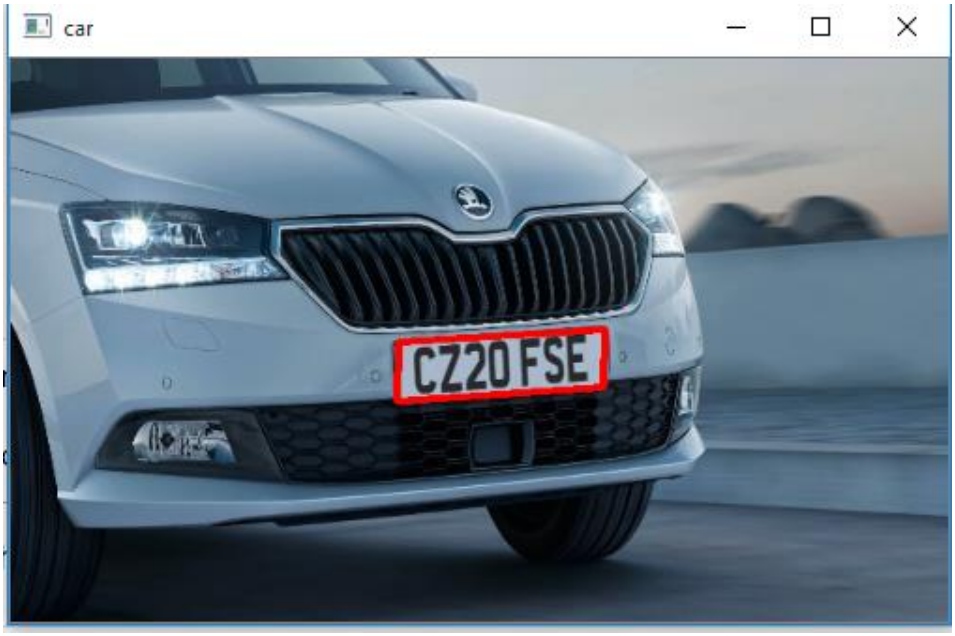

**7. ábra: Küszöbértékeket megjelenítő kép**

A küszöbértékeket a canny határfüggvénnyel [15] jelenítjük meg. A kontúrokat az OpenCV által szolgáltatott függvénnyel keressük meg. A talált kontúrok közül a rendszámtábla megtalálását az 1. kódrészlet valósítja meg.

```
for c in cnts:
    # approximate the contour
    peri = cv2.arcLength(c, True)
    approx = cv2.approxPolyDP(c, 0.018 * peri, True)
    # if our approximated contour has four points, then
    # we can assume that we have found our screen
    if len(approx) == 4:
        screenCnt = approx
        break
```

**1. kódrészlet: A kontúrok megtalálására elkészített kód**

Lényege, hogy az összes kontúron végigmegyünk, és megnézzük, melyik téglalap alakú, rendelkezik négy oldallal és zárt. A megtalált rendszámtáblát az OpenCV segítségével körberajzolhatjuk, ezt mutatja be a 8. ábra.




**8. ábra: Megtalált rendszámtábla**

A 3. lépéshez keresnünk kell egy karakterfelismerő technológiát. Minden karakterfelismerő technológia egy tanított neurális hálóval van megvalósítva. Az általam először tesztelt technológia a tesseract-ocr [16] volt. Ez a Google karakterfelismerő technológiája. Tapasztalataim alapján nem teljesített túl jól, mivel használata sokkal általánosabb, és ezért a rendszámtábla karaktereket nehezebben ismerte fel. Egy rendszámtábla karaktereken tanított karakterfelismerő neurális háló már jobban teljesít, annak ellenére, hogy kisebb adatbázison tanult.

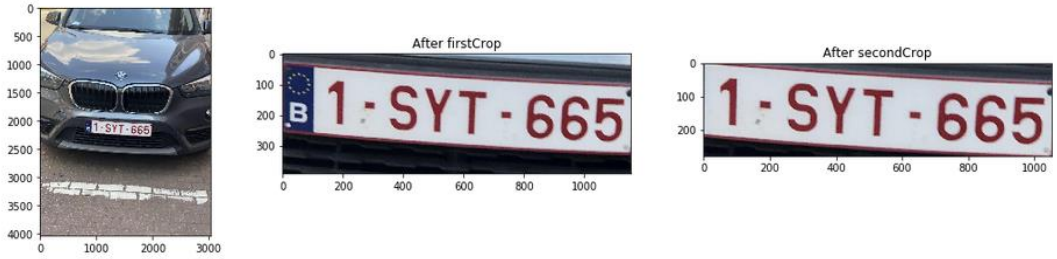

**9. ábra: Rendszámtábla körbevágva**

A tesseract-ocr működéséhez nem szükséges finomítanunk a képen, viszont ha egyéb neurális hálót használunk a karakterek felismeréséhez, célszerű az általam ismertetett

lépéseket (szürkítés, küszöbértékek, …) újra végrehajtani a körbevágott rendszámtábla képen (9. ábra) is.

### 4.1.2 Rendszámtábla felismerés elsősorban gépi tanulás alkalmazásával

Ez a módszertan ugyanazokon a lépéseken megy végig, mint az előzőleg tárgyalt. Alapvető különbségük, hogy ez a módszertan a rendszámtábla megtalálásához és a jármű felismeréséhez is tanított neurális hálókat használ. Először egy tanított neurális hálót használva megtaláljuk a rendszámtáblát (10. ábra), amit az OpenCV segítségével tovább körbevágunk.




**10. ábra: Rendszám tábla megtalálása neurális hálóval**

A következő lépésünk a rendszámtábla leolvasása. Először a karaktereket kell megtalálnunk, ezt megtehetjük képfeldolgozási technológiákkal, vagy egy tanított neurális hálóval.


**11. ábra: Képfeldolgozási technológiákkal a karakterek leolvasása**

Az utolsó lépésünk a karakterek felismerése, ezt megtehetjük a tesseract-ocr-el is, de jobb teljesítményt nyújt, ha tudunk keresni egy rendszámtábla karaktereken tanított karakterfelismerő neurális hálót. Az egész folyamatot reprezentálja a 11. ábra.

### 4.1.3 Konklúzió

Mindkét technológia működésképes, viszont tapasztalatom alapján az első módszertan csak jó minőségű képeken működik magas sikeraránnyal. A gépi tanulással megvalósított rendszámtábla megtalálás nagyobb arányban sikeres, akár rosszabb viszonyok között is. Egy neurális háló megtanítása viszont nagyon erőforrásigényes, szerencsére előre megtanított modelleket is lehet találni, így a tanítás nem a mi erőforrásainkat terheli. A legjobb megoldás a két technológia együttes használata, hiszen a neurális hálóval megvalósított rendszerünk sikeraránya is fokozható, például a képeink további finomításával, amit képfeldolgozó technológiákkal tudunk megvalósítani.

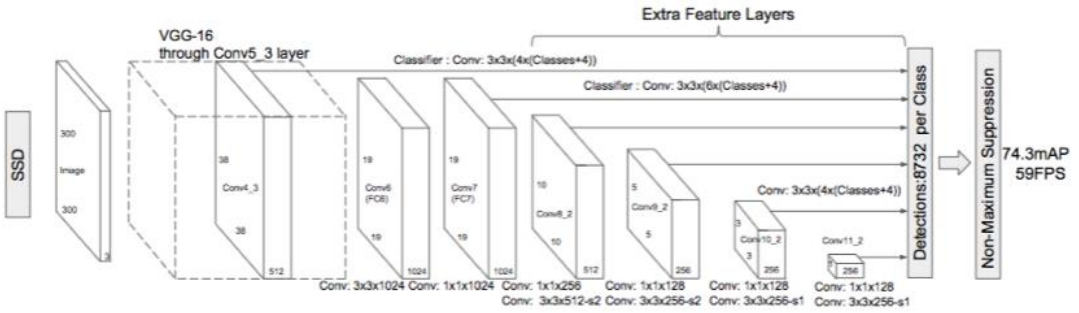
Az általam tervezett rendszerben a környezeti változók és a járművek változatossága miatt a gépi tanulással megvalósított rendszámtábla-felismerés megvalósítások használhatóak kizárólag, ha magas sikerarányt szeretnénk elérni. Épp ezért a rendszeremben is egy neurális hálóval működő rendszámtábla-felismerő rendszert használok, erről a 5.3.3 fejezetben beszélek bővebben.

## 4.2 Járműdetektálási módszertanok

Ebben a fejezetben bemutatom az általam megismert és kipróbált módszertanokat, amikkel járműdetektálást lehet megvalósítani. Mivel ezen módszertanok nagyon bonyolultak, nem fogom részletesen bemutatni őket. Központban a saját tapasztalataim, döntéseim, a módszertanoknak a rendszeremmel való összefüggésük és eredményességük lesz.

### 4.2.1 SSD (Single Shot Multibox Detector)

Az SSD [17] egy deep learningen alapuló objektumdetektálási és osztályozási megvalósítás. Létrejöttének motivációja az volt, hogy nem volt még olyan objektumdetektálást megvalósító neurális háló, ami valós időben képes lett volna ellátni feladatát. Az elmúlt években szerencsére sikerült a kutatóknak megvalósítani ezt, az SSD és YOLO [18] létrehozásával. Az SSD konvolúciós neurális hálóval van megvalósítva, és a sajátos architektúráját a 12. ábra mutatja be.

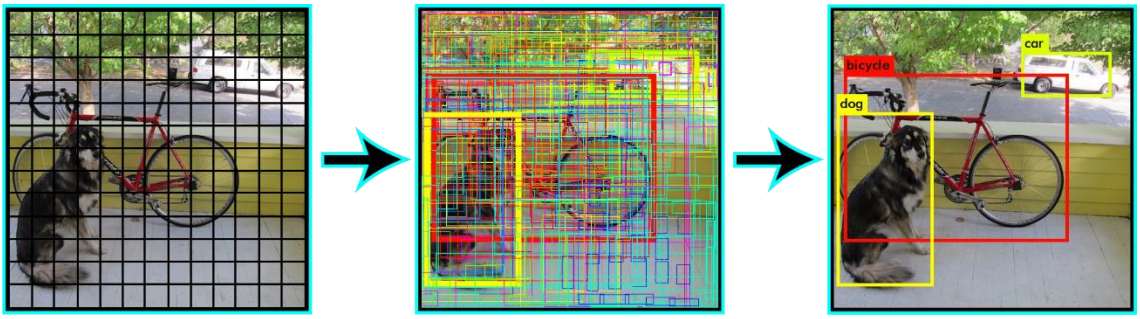



**12. ábra: Az SSD architektúrája**

Erőssége, hogy a detektálást és osztályozást a neurális hálón való egy darab átbocsájtással végzi el. További információ az SSD-ről az általam már hivatkozott hivatalos tanulmányban és egy általam talált cikkben [19] található. Különböző megvalósítások kipróbálása után azt vettem észre, hogy a referencia videókkal szemben az én videómon rosszabbul teljesítenek. A kulcstényezők, amiért nem teljesítenek jól a programomban, a használt videó tulajdonságaiból erednek. Az okok a következőek: A videót felvevő kamera egyedi dőlésszöge, pozíciója és minősége olyan képkockákat eredményez, amin nehezen ismer fel járműveket a COCO [20] adatbázison tanított neurális háló. Megoldás erre, hogy a videókból egy adatbázist kell létrehozni, és azon tanítani az SSD-t vagy YOLO-t.

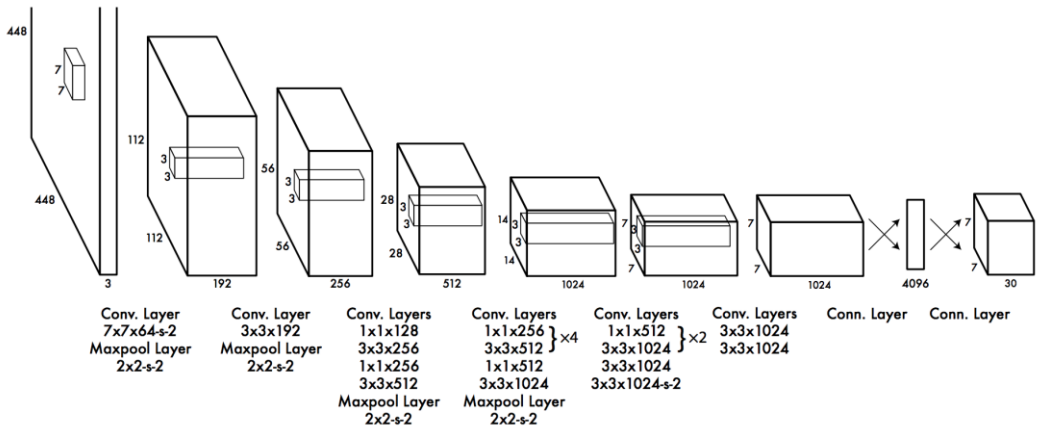
### 4.2.2 YOLO (You Only Look Once)

Hasonlóan az SSD-hez a YOLO is neurális hálón alapul. Alapvető különbségük, hogy a YOLO nem egy átbocsájtással detektálja és osztályozza az objektumokat. Először több predikciót képez le, majd a legnagyobb valószínűségűeket megtartja, és összevonja azok metszetét, így egy végső körbefoglaló négyzetet tud visszaadni. Ezt a módszert a 13. ábra mutatja be.




**13. ábra: Objektum detektálás YOLO-val**

A YOLO architektúrája is egy konvolúciós neurális hálón alapszik. Az architektúrát a 14. ábra mutatja be nekünk. Az évek során több verzióját kiadták a YOLO-nak. A mostani legújabb a YOLOv3, minden egyes verzió javítja a rendszer gyorsaságát és pontosságát. Akiknek a gyorsaság fontosabb, használhatják az új YOLO-LITE-ot [21], ami hordozható eszközökre lett fejlesztve. További információért ajánlom az általam már hivatkozott hivatalos tanulmányt és egy a Mediumon megosztott összefoglaló cikket [22].

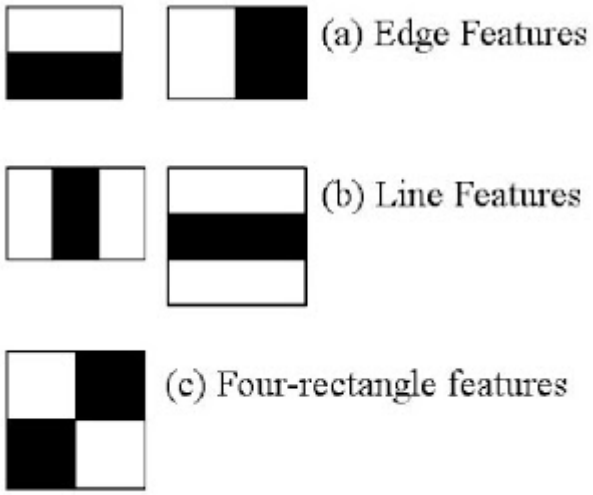



**14. ábra: A YOLO architektúrája**

### 4.2.3 Haar Cascade Classifier

Ezt a technológiát a rendszerem alapjaként vett program használta. Amikor először próbálkoztam a képfeldolgozó alrendszerem kialakításával, még Cascade

osztályozót használtam. Később ezt eldobtam, mivel azt találtam, hogy sok fals pozitív predikciót eredményez, és nem osztályozza a járműveket típusuk szerint, illetve nem képes többfajta jármű detektálására. A Haar tulajdonságalapú kaszkádosztályozó egy gépi tanulás alapú objektumdetektáló metódus. Alapjaként szolgáló kaszkádfüggvényt negatív és pozitív példákkal tanítjuk. Miután betanítottuk függvényünket, különböző úgynevezett Haar tulajdonságokat (15. ábra) kell kiemelnünk a képekből. További információért az OpenCV hivatalos oldalán található dokumentációt [23] ajánlom.




**15. ábra: Haar tulajdonságok**

### 4.2.4 Konklúzió

Az imént felsorolt három objektumdetektálási módszertanból az SSD-t választottam. Választásomat a 4.2.1-es fejezetben már részben kifejtettem. Alapvetően a választásom a YOLO és az SSD között volt. A Haar kaszkádosztályozó nem is volt számításban, hiszen nem képes több fajta objektum felismerésére. Illetve, ha képes is, az interneten nem áll rendelkezésre hozzá megfelelő betanított modell. Az SSD a YOLO fölött az erőforráshiány miatt győzött. A kettő közötti teljesítménybeli különbség nem volt olyan nagy, hogy indokolt legyen a YOLO-val érkező erőforrás ugrás tolerálása. A YOLO által okozott számítási overhead a videó feldolgozását még inkább akadozóvá tette. Az SSD is sok fals pozitívat jelez, illetve van néhány objektum, amit nem ismer fel. Ezeket javíthatnám, ha vagy olyan videót használnék, ahol messzebb vannak a járművek, illetve előnyösebb szögben látszanak, vagy olyan adatbázison való tanítással, ami az én videómban látható járművekkel azonos.

## 4.3 Sebességbecslő módszertanok

A sebességbecslés megvalósításával töltöttem el a legtöbb időt, és ezt volt a legnehezebb integrálni a prototípusba. Alapvetően minden sebességbecslő algoritmushoz szükséges tudnunk valamit a kameráról, illetve videóról, amin szeretnénk a járművek sebességét megbecsülni. Emiatt a tulajdonsága miatt nehéz integrálni egy új rendszerbe. Az interneten nagyrészt elérhető sebességbecslő algoritmusok mind használnak egy kódolt értéket, ami az adott videóhoz specifikus. Ez a pixel per meter (PPM), ami az egy métert reprezentáló pixelek számát jelenti. Ahhoz, hogy pontos becslést kapjunk, ismernünk kell ezt a mértékegységet. Ennek megszerzése nem egyszerű egy olyan videón, amit nem mi vettünk fel. A PPM megszerzéséhez használhatunk például útszélességet, vagy átlagos járműszélességet, rendszámtábla szélességet. Ezekhez viszont tudnunk kell a kamera pontos pozícióját. Sebességméréshez tudnunk kell két pozícióját a kocsinak pixelben és azokhoz tartozó képkockaszámot, videó fps-t, PPM-et. A formula a mindenki által ismert sebesség = megtett út / eltelt idő. Ahol az eltelt idő a két képkocka közötti különbség osztva az fps-sel, a megtett út méterben pedig a pixelben megtett távolság osztva a PPM-mel. Pontosabb eredményt lehet elérni kamera kalibrálással, illetve más algoritmusok használatával. Ilyen algoritmus még az OpticalFlow [24] használata a sebességbecsléshez. Ehhez a következő tanulmányt [25] ajánlom. Az általam ismertetett algoritmus részletesebb leírásához a Researchgate-en megtalálható cikket [26] ajánlom.

### 4.3.1 Konklúzió

Probléma más algoritmusok használatával, hogy minél bonyolultabb, annál jobban terheli az eddig is erőforrás korlátozott rendszerünket. Az én rendszeremben a PPM-et a detektált rendszámtábla hosszával számolom ki, ez sajnos a kamera tulajdonságai nélkül csak egy erős becslést tud adni. Továbbá én minden 10. képkocka feldolgozásánál nézem meg a detektált járművek sebességét. 10 képkocka alatt megtett távolságukból számolom ki a megtett távot, és illesztem be az általam ismertetett képletbe.

# 5 Integráció

Ebben a fejezetben a képfeldolgozó alrendszerem integráláshoz tartozó tervezői döntéseket, lépéseket és használt technológiákat, illetve az ezekhez tartozó megfigyeléseket, tapasztalatokat fogom részletezni.

## 5.1 Videó küldés DDS-sel

Ebben a fejezetben a fejezetcímként leírt probléma megoldásához tartozó gondolatmenetemet, progresszióimat szeretném leírni. Ezen funkció megvalósítása közben sok tapasztalatot szereztem, sok új tudásra tettem szert, ami segített a probléma megoldásában. Szeretném megemlíteni Samuel Raeburn-t, az RTI egyik alkalmazottját, aki az RTI közösségi fórumon [27] sok segítséget nyújtott nekem. Első próbálkozásként az RTI által kiadott példaprogramot [28] akartam alapul venni, majd annak átalakításával az én rendszeremhez illeszteni. Sajnálatos módon nem tartják karban eléggé ezeket a példakódokat, legalábbis ezt a verziót, és nagyon hosszú próbálkozás után sem sikerült működésre bírnom. A példaprogram alapvetően csak 32 bites platformon működik így a 32 bites RTI DDS Connext is kell hozzá. Tovább komplikálta volna a helyzetet, hogy a példaprogram C++-ban van megírva, az én rendszerem egésze pedig Python nyelven van.

### 5.1.1 Megvalósítás

Elméletben a videóküldés DDS-sel probléma nem bonyolult. Az alapvető gondolat mögötte, hogy képkockákként küldünk át egy videót. Az igazi nehézség nem is ebben állt, hanem a sok technikai problémában. A projektemet Python-ban írtam meg, és az RTI Connext DDS Python Connectort [29] használtam. Ezek a körülmények tovább nehezítették a dolgomat, mivel nincsen kész API Python-hoz (ezért használtam a connectort). A connector használata sok esetben csak csökkent egy RTI Connext DDS-t használó projekt bonyolultságán, viszont az én esetemben olyan alapvető funkcionalitások hiányoztak, amik egy teljes API-ban benne vannak. A számomra hiányzó funkcionalitások az adatok kezeléséhez tartoznak. A connectorban kisebb befolyásunk van a különböző adattípusok írására, számomra például a byte sorozatok írása hiányzott. Ezt a problémát egy byte sorozat utf-8-as kódolásával oldottam meg, hiszen a connector enged Python string típusút írni. A megvalósítás közben egy bug-ot is felfedeztem, ez a #CON-157–es belső hiba, ezt a hibát Samuel Raeburn segített

beazonosítani. Szerencsére egyetlen hatása, hogy egy hibaüzenetet nyomtat ki a konzolra, de nincs befolyása a program belső működésére. Ezt a hibaüzenetet a 2. kódrészlet mutatja be.

```
DDS_DynamicData2_get_string: Output buffer too small for member
(name = "frame", id = 1). Provided size (128), requires size (630).
```

**2. kódrészlet: A #CON-157-es belső hiba**

Az általam használt adatszerkezet egy ”data” és egy ”videoFps” adattaggal rendelkezik. A videoFps a küldött videó eredeti fps-ét jelenti, ez ahhoz kell, hogy pontosabban tudjam megbecsülni a járművek sebességét. A data adattag tartalmazza az elküldött képkockát stringbe kódolva. Fontos megemlíteni, hogy ahhoz, hogy képkockákat tudjunk átküldeni DDS-sel, a megfelelő QoS beállításokat kell használni. Szerencsére az RTI Connect DDS rendelkezik előre megírt QoS profilokkal, ezek közül nekünk a *StrictReliable.LargeData.FastFlow* [30] profil kell. A *StrictReliable* beállítás feladata, hogy minden adatminta publikálásra kerüljön. Ezzel a beállítással a minták nem lesznek felülírva, hanem eltárolásra kerülnek. Továbbá biztosítja, hogy a mintáink sorrendben legyenek elküldve és elveszett adatmintáink újra legyenek küldve. A *LargeData* rész engedélyezi nagy adatminták küldését, magában csak a minták memória használatát optimalizálja. Hogy az adatfolyamot is irányítsuk szükségünk van a *FastFlow* beállításra is. Ez már tartalmaz folyamirányítást is, ami 100 MB/sec-ig képes megbízható kommunikációt nyújtani. Ez a QoS profil elengedhetetlen a működéshez, mivel próbáltam egyéb beépített profilokkal, például a BestEffort és a StrictReliable [30] beállítással, és ezekkel nem működött a programom. Láthatjuk, hogy ez a beállítás teljesíti a 3. fejezetben ismertetett követelmények nagyrészét. A használt adatszerkezetet a 3. kódrészlet mutatja be.

```
<!-- types -->
    <types>
        <struct name="ImageType">
            <member name="data" type="string" stringMaxLength="-1"/>
            <member name="videoFps" type="float32"/>
        </struct>
    </types>
```

**3. kódrészlet: Az adatszerkezetem XML-ben leírva**

Az én megoldásom a következő lépésekből áll:

1. Videó beolvasása

2. Képkocka kódolása
3. Képkocka elküldése
4. Képkocka fogadása
5. Képkocka dekódolása

#### 5.1.1.1 Videóbeolvasás

A videóbeolvasást az OpenCV könyvtárával oldottam meg. Az ott használható VideoCapture függvényt használtam egy a hardveremen megtalálható videó beolvasására. Majd a VideoCapture osztály read függvényével tudok egy képkockát beolvasni. Ezekről a függvényekről és a videóbeolvasásról bővebben az OpenCV oldalán [31] olvashatunk. Azt, hogy a videóbeolvasás hogyan néz ki kódban, a 4. kódrészlet mutatja be nekünk.

```
cap = cv2.VideoCapture('./vehicle_counting_tensorflow_original/sample_v1.1
.mp4')
    while cap.isOpened():

        (ret, frame) = cap.read()

        if not ret:
            print ('end of the video file...')
            break
```

**4. kódrészlet: Az OpenCV által megvalósított videóbeolvasás példa**

#### 5.1.1.2 Képkocka kódolás és elküldése

Egy kódolt képkocka elküldéséhez először létre kell hoznunk egy connector objektumot. Ez az objektum végzi majd el a DDS által megvalósított adatküldést. A connectorból pedig ki tudjuk nyerni a get_output függvénnyel az output objektumunkat, amivel az adatírást valósítjuk meg. Ezt mutatja be a 5. kódrészlet.

```
vehicleDataConnector = rti.Connector("MyParticipantLibrary::MyParticipant"
,"Valami.xml")
    output = vehicleDataConnector.get_output("MyPublisher::MyWriter")
```

**5. kódrészlet: Egy connector és output objektum létrehozása**

Egy adott képkocka kódolásához base64-et [32] és OpenCV-t használtam. Elsőként magamtól próbáltam kitalálni a kódolás lépéseit, de kihagytam lépéseket, így az nem működött. Később egy cikk [33] segítségével sikerült megfelelően kódolnom a képkockáimat.

```
_, im_arr = cv2.imencode('.jpg', frame)
        im_bytes = im_arr.tostring()
        im_b64 = base64.b64encode(im_bytes)
        output.instance.set_string("data",im_b64.decode("utf-8"))
```

**6. kódrészlet: Egy képkocka kódolása**

Az 6. kódrészlet bemutatja hogyan oldottam meg a korábban hivatkozott cikk alapján egy képkocka kódolását. Először az OpenCV segítségével JPG formátumba alakítjuk, ez azért kell, mert alapvetően amikor egy képet beolvasunk OpenCV segítségével, az egy numpy tömbként lesz reprezentálva. Következő lépésként a *tostring()* függvénnyel egy bytesorozatot készítünk az előzőleg JPG formátumba alakított képünkből. Ezt a bytesorozatot végül b64-gyel kódoljuk, és hogy a *set_string()* függvénnyel el is tudjuk küldeni, utf-8-as dekódolást végzünk el rajta. Az output objektumról és az adatküldés további formáiról tanulni az RTI Connector for Python hivatalos dokumentációja [34] segített. Továbbá fontos megemlíteni, hogy a 3. fejezetben leírt követelmény szerint állandónak kell lennie a képkockák folyamának. Ezt a python kódban lévő korlátozással oldottam meg a time könyvtár segítségével. Folyamatosan figyelem az eltelt időt, és csak bizonyos időközönként engedem meg, hogy a DDS elküldje az adott képkockát, ezzel állandó lesz a sebessége a videónak.

#### 5.1.1.3 Képkocka fogadása és dekódolása

Képkocka fogadása és dekódolása a 5.1.1.2 fejezetben leírt folyamat fordítottja. Itt megszerezzük az adatot a connector segítségével, majd dekódoljuk hasonlóan, mint ahogyan azt kódoltuk. Ezt a folyamatot a 7. kódrészlet mutatja be kód formában.

```
input = connector.get_input("MySubscriber::MyImageReader")
                input.wait_for_publications()
                for i in range(1, 500):
                    input.wait()
                    input.take()

                    for sample in input.samples.valid_data_iter:
                        image = sample.get_string("data")
                        imageToByte = image.encode("utf-8")
                        decoded = base64.b64decode(imageToByte)
                        np_data = np.frombuffer(decoded,dtype=np.uint8)
                        frame = cv2.imdecode(np_data,flags=1)
```

**7. kódrészlet: Egy képkocka fogadása majd dekódolása**

A connector segítségével egy bizonyos subscriber-hez megszerezzük a hozzátartozó adatolvasó bemenetét, ez a *connector.get_input()*. A connector segítségével megvalósított adatolvasás megértéséhez az RTI hivatalos dokumentációja [35] segített. Miután a rendelkezésre álló beépített függvényekkel sikerült megszereznünk egy adatmintát, dekódolnunk kell azt. Először utf-8-cal kódoljuk, hogy a stringből bytesorozatot csináljunk, bár ez a lépés nem feltétlenül szükséges. Ezután a base64-es b64 dekódolást használjuk, az így kapott adatot a numpy *frombuffer()* [36] függvényével a bytesorozatot egy numpy tömbbé alakítjuk. Az utolsó lépésünk pedig az OpenCV által szolgáltatott *imdecode()* [37] függvény használata.

## 5.2 Képfeldolgozással szerzett adatok továbbküldése DDS-sel

A probléma megoldása egy olyan kód létrehozása ahol egyszerre fogadunk adatokat és küldjük tovább egy másik topic-ra. Ezt az Önálló Laboratórium keretei között már megoldottam (még Pristyák Dávid segítségével), az alapvető nehézség az volt, hogy az RTI Connext kódgenerálója sok külön fájlba generálja a kódot, emiatt sok munka kifésülni a szükséges kódot, hogy azt át tudjuk mozgatni az általa legenerált writer fájlból a reader fájlba. Annyiban más most a megoldás, hogy az RTI Connector for Python-t használva sokkal flexibilisebben lehet kezelni az adatíró és adatolvasó pozíciót, illetve az egyszerre író és olvasó fájlok is egyszerűbben megoldhatóak. Ennek oka, hogy a használt connector leegyszerűsít sok működést, de emiatt nem is használható olyan széles körben, mint egy teljesen kidolgozott API. A probléma megoldását ebben az esetben is egy XML fájl létrehozásával kezdtem el. Ebben definiáltam egy adattípust, ezt a 8. kódrészlet mutatja be.

```
<struct name="VehicleData" extensibility="extensible">
         <member name ="vehicleID" type="long"/>
         <member name ="licensePlate" stringMaxLength="128" type="string"
key="true"/>
         <member name="speed" type="long"/>
         <member name="type" stringMaxLength="128" type="string"/>
</struct>
```

**8. kódrészlet: A VehicleData struktúra**

A korábban már ismertetett módon egy connector és output objektum segítségével elvégzem az adatok továbbküldését. A kódban, mivel valós idejű adatfeldolgozás zajlik, nem akartam még nagyobb számítási overheadet okozni a képkockákkénti adattovábbítással, ezért csak minden 10. képkocka után küldöm el az eddig megfigyelt járművek adatait.

## 5.3 Integrálandó képfeldolgozó alrendszerek

Előző fejezetekben leírtam, hogyan kell az alrendszerünkbe DDS-sel videót küldeni, dekódolni a küldött képkockát, illetve a képkocka feldolgozása után szerzett információkat hogyan kell tovább küldeni. Ebben a fejezetben az általam kidolgozott, különböző projektekből, eszközökből összerakott alrendszert szeretném bemutatni. Szeretném leszögezni, hogy szakdolgozatom fő témája különböző meglévő technológiák integrálása. Az általam elkészített rendszer erőforrás és megfelelően integrálható technológiák hiányából kifolyólag nem használható valós rendszerként, de demonstrációként megfelelően működik. Alapjaként Luka Cosic projekjét [38] használtam. Eredetileg a projekt tartalmaz egy OpenCV-vel végrehajtott kaszkádosztályozót, amivel az autókat lokalizálja, illetve a lokalizált autókat a dlib [39] segítségével követte, és sebességüket 10 képkockánként kiszámolta. A következő fejezetekben leírom tapasztalataimat és eredményeimet az alrendszer különböző funkcióinak integrálásával kapcsolatban.

### 5.3.1 Sebességbecslés

Számomra ez volt a legnehezebb feladat. A 4.3 fejezetben már ismertetett okok közrejátszottak abban, amiért ezt tartottam a legnehezebb feladatnak. A sebességbecslés folyamatában sok ismeretlen változó miatt nehéz egy olyan technológiát találni, amit egyszerűbben lehetne integrálni egy meglévő rendszerben. Emiatt ez volt az első funkció, amit megvalósítottam, és a többi funkciót az így elkészített rendszerbe illesztettem bele, hiszen a többi már sokkal rugalmasabb. Problémát okozott még, hogy az alrendszeremnek

egyszerre kell elvégeznie rendszámtábla felismerést és sebességbecslést is. Így olyan videót kellett keresnem, amiben elég közel van a jármű, hogy felismerhető legyen a rendszámtábla, és elég távol, hogy képesek legyünk sebességbecslésre. Az interneten megtalálható sebességbecslést megvalósító projektek általában egy videóra vannak beállítva, és az esetek 100%-ában nem volt megfelelő a használt videó. Azon projektek, amik videófüggetlenül valósítottak meg sebességbecslést, vagy túl bonyolultak voltak, hogy fel tudjam használni, vagy túlságosan erőforrás-igényesek voltak. Ezért hoztam azt a kompromisszumot, hogy nem fogok teljesen pontos sebességbecslést végezni, viszont így az összes funkciót sikerül megvalósítani az alrendszeremben. Az alapként vett rendszerben a sebességbecslést érintően annyit változtattam, hogyan számolom a PPM-et.

### 5.3.2 Járműlokalizáció és -osztályozás

A 4.2 fejezetben leírt módszertanok úgy lettek kiválasztva, hogy miközben megfelelő projektet próbáltam találni, ezen megoldások voltak a leggyakoribbak. Az alapként vett projekt OpenCV által megvalósított kaszkádosztályozást használt. Azt találtam, hogy ez nem megfelelő az én alrendszeremhez, hiszen nem képes több típusú jármű lokalizálására és osztályozásukra. Miközben megfelelő projektet kerestem, az esetek közel 100%-ában vagy YOLO vagy SSD megvalósításokat találtam. A rendszeremben SSD modellt használtam, viszont integrálás szempontjából mindkét technológiát hasonlóan könnyű beilleszteni rendszerünkbe.

Egy lefagyasztott SSD modellt használtam, ez azt jelenti, hogy betanították a modellt majd különböző módszerekkel lementették. Így az én rendszeremben nem kell elvégezni a tanítás erőforrás igényes lépését. Az én SSD modellem a COCO adatbázison lett tanítva. Az SSD technológia kódban való felhasználását a Tensorflow-val valósítottam meg. A Tensorflow szerencsére szolgáltat egy API-t lefagyasztott modellek betöltésére és felhasználására. A kódban a következőképpen lehet felhasználni:

- Modell betöltése: A 9. kódrészlet ezt mutatja be.
- Kimeneti tensorok [40] betöltése: A 10. kódrészlet ezt mutatja be.
- Detekció elvégzése: A 11. kódrészlet ezt mutatja be.

```
detection_graph = tf.Graph()
    with detection_graph.as_default():
        od_graph_def = tf.GraphDef()
        with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
            serialized_graph = fid.read()
            od_graph_def.ParseFromString(serialized_graph)
            tf.import_graph_def(od_graph_def, name='')
```

**9. kódrészlet: Az SSD modell betöltése**

```
with detection_graph.as_default():
        with tf.Session(graph=detection_graph) as sess:
            image_tensor = detection_graph.get_tensor_by_name('image_tenso
r:0')
            detection_boxes = detection_graph.get_tensor_by_name('detectio
n_boxes:0')
            detection_scores = detection_graph.get_tensor_by_name('detecti
on_scores:0')
            detection_classes = detection_graph.get_tensor_by_name('detect
ion_classes:0')
            num_detections = detection_graph.get_tensor_by_name('num_detec
tions:0')
```

**10. kódrészlet: A kimeneti tensorsok betöltése**

```
(boxes, scores, classes, num) = \
                            sess.run([detection_boxes, detection_scores,
                            detection_classes, num_detections],
                            feed_dict={image_tensor: image_np_expanded})
                            boxes = np.squeeze(boxes)
                            scores = np.squeeze(scores)
                            classes = np.squeeze(classes)
```

**11. kódrészlet: A Detekció elvégzése egy képen**

Azért használtam a rendszeremben SSD-t, mivel nincsen a laptopomon sok erőforrásom, és az SSD egy sokkal kevésbé erőforrás-igényes megoldás, mint a társai, emelett viszont hasonló szintű pontossággal képes a detekcióra és osztályozásra. Próbáltam a YOLO-t is, mivel nem voltam megelégedve az SSD teljesítményével, viszont nem láttam szignifikáns különbséget a pontosságukban, de az erőforrás igénye erősen érződött a programomban.

Egyetlen nehézséget a fejezetben leírt funkció megvalósításában az okozta, hogy az eredetileg használt OpenCV megvalósítás és az általam választott Tensorflow használó SSD megoldás másként tért vissza az általuk kiszámított értékekkel. Az SSD a járműveket körbefogó négyzetekkel, osztályozási és lokalizációs valószínűséggel és osztályozás eredményével tért vissza. A kaszkádosztályozó egyedül a körbefogó négyzeteket tudta kiszámolni. Fontos különbség, hogy az OpenCV által megvalósított kaszkádosztályozó a négyzeteket abszolút koordinátákkal adja meg és a következő

sorrendben: x, y, width, height [41]. Ahol (x,y) a bal felső koordinátája a négyzetünknek, a width a szélessége pixelben, a height pedig a magassága pixelben. A Tensorflow által támogatott SSD pedig a négyzeteket a következőképpen adja vissza: top, left, bottom, right. A top változó a négyzetünk legfelső sarka és a képünk teteje közötti távolság. A left, a bottom és a right ehhez hasonlóan a négyzetünk bal, alsó és jobb sarkának a képünknek bal, alsó és jobb oldalától való távolsága. További információ a Tensorflow-val végzett objektumfelismerésről a hivatalos oldalon [42] található. Szerencsére a lokalizált objektumok követésével nem kellett már foglalkoznom, hiszen az eredeti projektem rendelkezett már ilyen funkcióval, és kompatibilis maradt, miután kicseréltem az objektumlokalizáló technológiát. Tovább változtattam a rendszert abban, hogy azon adatok, amiket tovább kell küldenem, le vannak mentve listákban. Ez azért kellett, hogy ne kelljen minden egyes detektálásnál elküldeni az adatokat.

Teljesítmény optimalizáció érdekében különböző modelleket kipróbáltam, ezeket a Tensorflow hivatalos modell gyűjteményéből [43] válogattam. A következő modelleket próbáltam ki:

| Modell név | Tapasztalat |
|---|---|
| ssd_mobilenet_v1_coco_2018_01_28 | Megfelelően gyors, és a predikciói is elég pontosak, az egyik legjobb modell teljesítmény/pontosság szerint. |
| ssd_mobilenet_v1_fpn_shared_box_predictor_640x640 | Pontos nagyon, viszont lassú magában is, mellette a rendszámtábla felismeréssel már elviselhetetlenül lassú. |
| ssd_mobilenet_v2_oid_v4_2018_12_12 | Hasonló eredményt ad, mint az első modell, viszont az osztályozása sokszor volt pontatlan az én videómon. |
| ssdlite_mobilenet_v2_coco_2018_05_09 | Hasonlóan jól működik, mint az első modellünk, viszont nincs teljesítményben akkora különbség, hogy ezt használjuk. |

### 5.3.3 Rendszámtábla-leolvasás

Először az OpenALPR-t próbáltam ki a rendszeremben, de sajnos nagyon rosszul teljesített, hiszen nem sikerült egyetlen rendszámtáblát sem lokalizálnia és leolvasnia. Második lehetőség a Nagy Simon József és Vajda Máté Levente által elkészített rendszámtábla felismerőrendszer. Projektjükből [44] a megfelelő részeket kiválasztva

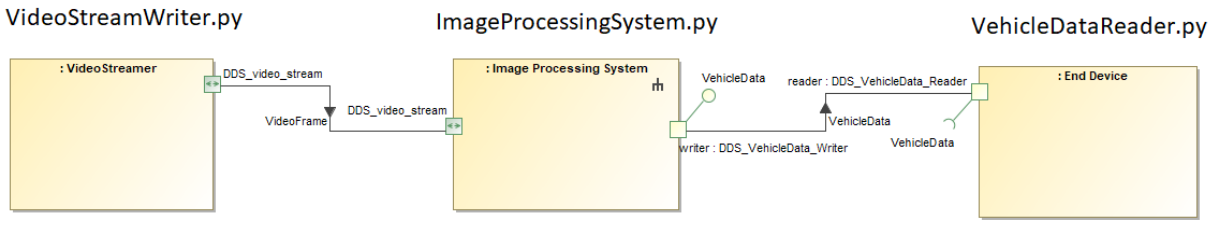
sikerült integrálnom a saját rendszerembe, viszont az én videómon nem teljesített jól az általuk tanított rendszámtáblát felismerő neurális háló. Az általuk elkészített alrendszert integrálva rájöttem, hogy a rendszámtábla felismerés ismét nagy erőforrású feladat, ezért olyan megoldást kell keresnem, ami nem erőforrás-igényes. Az ő alrendszerük SSD-vel ismeri fel a rendszámtáblákat, így én is ezen az úton folytattam kutatásomat, hiszen a 4.2 fejezetben leírtak bármilyen objektum felismerésére igazak. Így találtam meg Göksel Sahin projektjét [45], amit sikeresen integráltam a saját projektembe, és a tesseract-ocr-rel párosítva végeztem el a rendszámtábla leolvasását. Választásom azért esett a tesseract-ocr-re, mivel ezt találtam a legegyszerűbbnek felhasználni. Használata csupán egy könyvtár importálásával és egy függvény meghívásával megvalósítható, cserébe viszont nem a legpontosabb megoldás. Ennél a feladatnál a legnagyobb kihívást két hasonlóan elmentett modell importálása jelentette. A Tensorflow gráfokkal reprezentálja a betöltött neurális hálót, és egy gráfhoz egy ”Session” [46] tartozik. Sok próbálkozás után találtam egy cikket [47], amivel sikerült megoldani a modellek importálását és használatát. A megoldást az nyújtotta, hogy minden egyes alkalommal, amikor az adott modellt akarjuk használni, azt az alapméretezett folyamattá és alapméretezett gráffá kell tennünk. Ezt az általam már hivatkozott cikk részletesen bemutatja és megmagyarázza.

# 6 Prototípus

Az előző fejezetekben részleteztem, milyen projekteket mennyi változtatással sikerült integrálnom a rendszerembe. Most szeretném bemutatni a prototípusomat, illetve értékelném, milyen hibái vannak, hol lehetne rajta javítani. A prototípust sikeresnek találtam abban, hogy integráltam különböző technológiákat egy rendszerbe, és ezt sikerült összekötnöm a külvilággal DDS-sel. A prototípus képfeldolgozó alrendszere erőforrás és megfelelő technológiák hiányában pontatlan és lassú, viszont látom a lehetőséget, hogyan lehetne pontosabbá és gyorsabbá tenni. A prototípus tesztelése alatt bebizonyosodott, hogy a kommunikáció nem válik szűk keresztmetszetté, hiszen a 5.1 fejezetben leírt QoS profil magában foglalja azon követelményeket, amik az én rendszeremhez is szükségesek. Amennyiben a hálózat megfelelő átbocsájtási tulajdonsággal rendelkezik, az általam elkészített, DDS-sel megvalósított videótovábbítás nem fog akadozni. Szakdolgozatom fókusza azonban nem azon volt, hogy valós időben, ipari környezetbe bevethető képfeldolgozó rendszert építsek, hanem, hogy bemutassam, hogyan lehet különböző technológiákat felhasználni, hogy a címben leírt alrendszer integrálását megvalósítsam.

## 6.1 Prototípus bemutatás

Alapvetően három darab python script-ből áll a prototípus, ezek a következőek: *VideoStreamWriter.py, ImageProcessingSystem.py, VehicleDataReader.py* . Az első beolvassa a kiválasztott videót, és továbbküldi a másodiknak. A második script a főprogram, ebben van megvalósítva az intelligens képfeldolgozó alrendszer, ez folyamatosan küldi tovább az adatokat DDS-en keresztül. A harmadik script pedig csak olvassa a főprogram által kiküldött adatokat, és a konzolra kiírja őket. Legfőbb tulajdonsága a főprogramnak, hogy képes DDS-en keresztül küldött videót olvasni, és ezt feldolgozni. Az első és harmadik script csupán a tesztelés miatt volt szükséges.

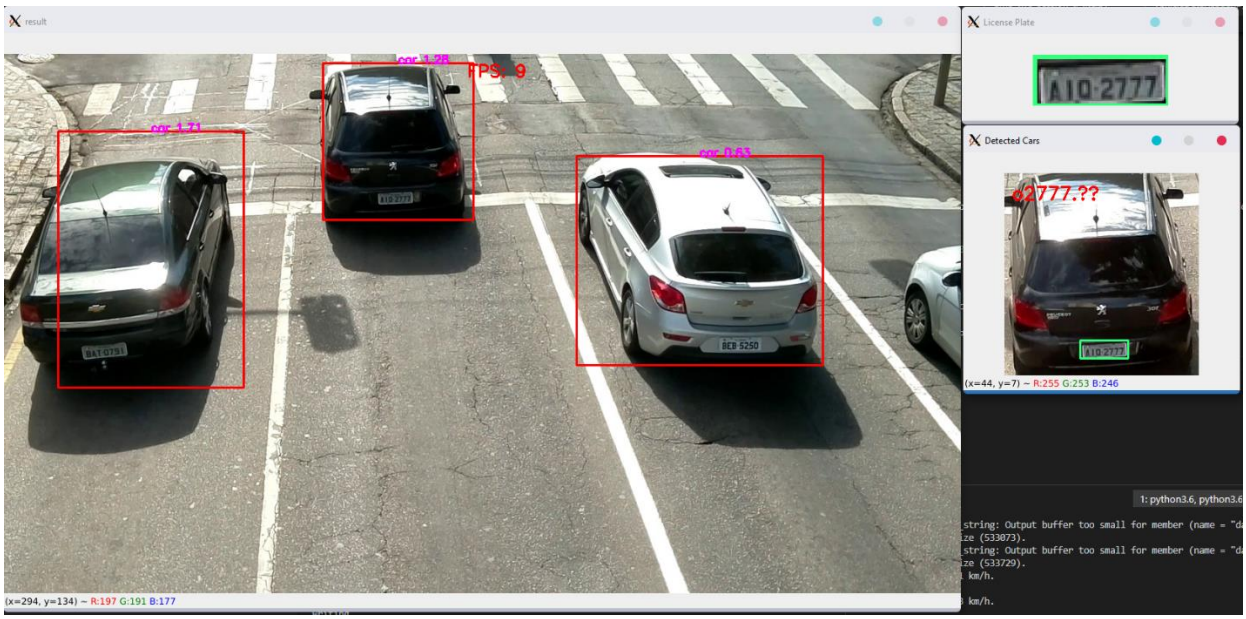



**16. ábra: Rendszerterv komponensei és python script megfelelői**

A 16. ábra mutatja be az elkészített rendszerterv komponenseit és a prototípusban megtalálható megfelelőit.

## 6.2 Prototípus értékelés

Ahogyan azt említettem, a prototípusom képfeldolgozó alrendszere lassú és pontatlan tud lenni. Bővebben kifejtve a rendszámtábla leolvasása pontatlan, illetve néhány esetben a különböző járművek osztályozása. A lassítást leginkább a rendszámtábla-lokalizáció és a rajta lévő karakterek felolvasása okozza.




**17. ábra: A prototípus működés közben**

A prototípus működését a 17. ábra mutatja be, a képfeldolgozó alrendszer végeredményét bemutatja, de a folyamatot, a felismerések sorrendjét nem képes bemutatni. A folyamat a járművek lokalizációjával kezdődik, ilyenkor ezeket a talált objektumokat követni is kezdi a program. Miután sikeresen lokalizált egy járművet, annak körbefogó négyzetét

felhasználva körbevágja azt, és ezen a képen elvégzi a rendszámtábla lokalizációt. Miután az is sikeresen megtörtént, annak körbefogó négyzetét felhasználva ismét körbevágjuk a képet, így már csak a rendszámtábla látszódik. A rendszámtáblaképen pedig elvégezzük a karakterfelismerést. Fontos még megemlíteni, hogy mikor történik meg a kiértékelt adatok továbbküldése. Az adatokat különböző változókban, listákban mentem el, és bizonyos időközönként küldöm tovább DDS-sel. Ez az időtartam változtatható a kódban, alapvetően minden 10. képkockánál küld tovább adatokat (ahogyan ezt már a 5.2-es fejezetben említettem), illetve minden alkalommal, ha új járművet sikerült felismerni.

Javítási lehetőség lehet, hogy többszálú programozást használunk, ennek megvalósítása jelentős átstrukturálással járna, de elvégzése lehetséges, és javítana a képfeldolgozó rendszer teljesítményén. A GPU bevonása is javítana az alrendszerünk gyorsaságán, ehhez használhatjuk a Tensorflow beépített GPU támogatását [48] is. Különböző megfigyelések pontosságának javítása különbözően hajtható végre. A sebességbecslés pontossága a gyorsaság növelésével is nőne, hiszen ha folyamatos a videó, és nem akadozik, pontosabban tud becsülni az algoritmusunk, egyéb javítást már a 4.3 fejezetben megfogalmaztam. Ami a járműfelismerést és rendszámtábla-leolvasást illeti, mindkettőhöz a 3. fejezetben hozzájuk tartozó alfejezetekben már ismertettem javítási lehetőségeket.

# 7 Összefoglalás

Kezdetként áttekintettem az intelligens képfeldolgozással foglalkozó technológiákat. Ezután létrehoztam egy rendszertervet, illetve megfogalmaztam kommunikációra vonatkozó követelményeket. Megismerkedtem a terv megvalósításához és a követelmények teljesítéséhez szükséges technológiákkal. Továbbá megismerkedtem az alrendszerem feladatait teljesítő képfeldolgozó módszertanokkal. Miután minden szükséges előzetes tudást összeszedtem, végrehajtottam az integrálást. Ehhez az integráláson túl saját magam meg kellett alkotnom az alrendszert ellátó feladatokat végrehajtó projektek egyikét. A prototípus egy élő bizonyíték arra, hogy a feladatomat sikeresen elvégeztem. A képfeldolgozó alrendszernek vannak hatékonysággal kapcsolatos hibái, de ezen hibák megoldása kívül esik a szakdolgozatom keretein.

Feladatom végére értem a szakdolgozatomban, viszont a projektet tovább lehet még fejleszteni. Utóéletében azt látom, hogy a MIT hibatűrő rendszerek kutatócsoportja által szervezett demonstráció részévé lehetne tenni. Ehhez viszont változtatásokon kell keresztül mennie a képfeldolgozó alrendszernek. Meg kell oldani a teljesítménnyel kapcsolatos hibáit, ezekre megoldás lehetne többszálú programozás használata, kisebb erőforrásigényes neurális hálók használata. Annak ellenére még értékes, hogy a jelenlegi formában nem lehet hasznosítani a projektemet. Értéke a nagyméretű tapasztalatban van, amit elkészítése közben szereztem, illetve bizonyos részei akár már most felhasználhatóak.

# 8 Hivatkozások


[1] MathWorks, „MathWorks - Matlab,” MathWorks, [Online]. Available: https://www.mathworks.com/products/matlab.html.

[2] OpenCV, „OpenCV,” [Online]. Available: https://opencv.org/.

[3] Tensorflow, „Tensorflow,” [Online]. Available: https://www.tensorflow.org/.

[4] W. A. Y. Ahmed A. Elsayed, „http://www.helwan.edu.eg/english/,” 14 Augusztus 2019. [Online]. Available: https://arxiv.org/pdf/1905.01213.pdf.

[5] S. Srivastava, „Analytics Insight,” 7 január 2020. [Online]. Available: https://www.analyticsinsight.net/opencv-vs-matlab-which-is-best-for-successful-computer-vision-project/.

[6] M. Khan, „Quora,” 20 September 2017. [Online]. Available: https://www.quora.com/What-are-the-pros-and-cons-of-using-Matlab-vs-Open-CV-for-image-processing.

[7] P. C. Library, „Pointclouds,” [Online]. Available: https://pointclouds.org/.

[8] OMG, 2015. [Online]. Available: https://www.omg.org/spec/DDS/1.4/PDF.

[9] Object Management Group, „About Us: OMG,” [Online]. Available: https://www.omg.org/about/index.htm.

[10] Object Management Group, Inc., „dds-foundation.org,” [Online]. Available: https://www.dds-foundation.org/what-is-dds-3/.

[11] OMG, December 2018. [Online]. Available: https://www.omg.org/spec/DDS-XML/1.0/PDF.

[12] Medium, „Medium,” 7 Július 2020. [Online]. Available: https://medium.com/programming-fever/license-plate-recognition-using-opencv-python-7611f85cdd6c.

[13] Theophilebuyssens, „Medium,” 7 Augusztus 2019. [Online]. Available: https://medium.com/@theophilebuyssens/license-plate-recognition-using-opencv-yolo-and-keras-f5bfe03afc65.

[14] The University of Edinburgh School of Informatics, [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html.

[15] OpenCV, „OpenCV.org,” [Online]. Available: https://docs.opencv.org/master/da/d22/tutorial_py_canny.html.

[16] Google, „github.com,” Google, [Online]. Available: https://github.com/tesseract-ocr/tesseract.

[17] D. A. D. E. C. S. S. R. C.-Y. F. A. C. B. Wei Liu, „Cornell University,” 8. December 2015. [Online]. Available: https://arxiv.org/abs/1512.02325.

[18] A. F. Joseph Redmon, „pjreddie.com,” University of Washington, 2018. [Online]. Available: https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[19] E. Forson, „Towards Data Science,” 18. November 2017. [Online]. Available: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab.

[20] M. M. S. B. L. B. R. G. J. H. P. P. D. R. C. L. Z. P. D. Tsung-Yi Lin, „Common Objects in Context,” [Online]. Available: https://cocodataset.org/#home.

[21] R. H. Jonathan Pedoeem, „Cornell University,” 14. November 2018. [Online]. Available: https://arxiv.org/abs/1811.05588.

[22] J. Hui, „Medium,” 18. Március 2018. [Online]. Available: https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088.

[23] OpenCV, „docs.opencv.org,” [Online]. Available: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.

[24] Nanonets, „nanonets.com,” [Online]. Available: https://nanonets.com/blog/optical-flow/.

[25] M. G. Indu Sreedevi, „researchgate.net," [Online]. Available: https://www.researchgate.net/publication/50392061_Vehicle_Tracking_and_Speed_Estimation_using_Optical_Flow_Method.

[26] D. R. S. Budi Setiyono, „researchgate.net," December 2019. [Online]. Available: https://www.researchgate.net/publication/342732516_MULTI_VEHICLE_SPEED_DETECTION_USING_EUCLIDEAN_DISTANCE_BASED_ON_VIDEO_PROCESSING.

[27] RTI, „RTI Community Forum," [Online]. Available: https://community.rti.com/.

[28] RTI, „rti.com," [Online]. Available: https://www.rti.com/developers/case-code/streaming-video.

[29] RTI, „github.com," [Online]. Available: https://github.com/rticommunity/rticonnextdds-connector-py.

[30] RTI, „RTI Community," [Online]. Available: https://community.rti.com/kb/configuring-qos-built-profiles.

[31] OpenCV, „opencv-python-tutorials," [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html.

[32] Python Software Foundation, „docs.python.org," [Online]. Available: https://docs.python.org/3/library/base64.html.

[33] Jdhao, „Jdhao's blog," [Online]. Available: https://jdhao.github.io/2020/03/17/base64_opencv_pil_image_conversion/.

[34] RTI, „community.rti.com," [Online]. Available: https://community.rti.com/static/documentation/connector/current/api/python/output.html.

[35] RTI, „community.rti.com," [Online]. Available: https://community.rti.com/static/documentation/connector/current/api/python/input.html.

[36] Numpy, „numpy.org,” [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.frombuffer.html.

[37] dcole, „programmer.group,” [Online]. Available: https://programmer.group/opencv-python-cv2.imdecode-and-cv2.imencode-picture-decoding-and-coding.html.

[38] L. Cosic, „github.com,” [Online]. Available: https://github.com/lcosicnus/CMS.

[39] dlib, „dlib.net,” [Online]. Available: http://dlib.net/.

[40] Tensorflow, [Online]. Available: https://www.tensorflow.org/guide/tensor.

[41] OpenCV, „opencv.org,” [Online]. Available: https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html.

[42] Tensorflow, „tensorflow.org,” [Online]. Available: https://www.tensorflow.org/lite/models/object_detection/overview.

[43] Tensorflow, „github.com,” [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md.

[44] V. M. L. Nagy Simon József, „github.com,” [Online]. Available: https://github.com/simon5521/TMIT_IoT_verseny_2020.

[45] Göksel Sahin, „github.com,” [Online]. Available: https://github.com/gkslsahin/ssd-plate_detection.

[46] Tensorflow, „tensorflow.org,” [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/compat/v1/Session.

[47] ProgrammerSought, „programmersought.com,” [Online]. Available: https://www.programmersought.com/article/25451750460/.

[48] Tensorflow, „tensorflow.org,” [Online]. Available: https://www.tensorflow.org/guide/gpu.

[49] K. Nahtkasztlija, „Az idegen szavak toldalékolása,” június 2009. [Online]. Available: http://www.pcguru.hu/blog/kredenc/az-idegen-szavak-toldalekolasa/5062.

[50] P. Koopman, „How to Write an Abstract,” október 1997. [Online]. Available: https://users.ece.cmu.edu/~koopman/essays/abstract.html. [Hozzáférés dátuma: 20 október 2015].

[51] W3C, „HTML, The Web’s Core Language,” [Online]. Available: http://www.w3.org/html/. [Hozzáférés dátuma: 20 október 2015].

[52] OpenALPR, „openalpr.com,” OpenALPR, [Online]. Available: https://github.com/openalpr/openalpr.

[53] M. Harmouch, „Medium,” Medium, 19 Június 2020. [Online]. Available: https://medium.com/swlh/local-binary-pattern-algorithm-the-math-behind-it-%EF%B8%8F-edf7b0e1c8b3.

[54] Dassault Systemes, „Spatial Blog,” 17 december 2019. [Online]. Available: http://blog.spatial.com/the-main-benefits-and-disadvantages-of-point-cloud-modeling.

[55] J. Delmerico, „University at Buffalo,” 11 Február 2013. [Online]. Available: https://pdf4pro.com/view/pcl-tutorial-the-point-cloud-library-by-example-5ac562.html.

[56] D. W. Tibor, „Óbudai Egyetem,” [Online]. Available: http://www.uni-obuda.hu/users/feketez/Hiradastechnika_I/Bevezetes_a_Matlab_hasznalataba.pdf.