Közúti szabálytalanságok felismerése Deep Learning tanuló folyamatokkal[1]

Max Gyula

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és alkalmazott informatikai tanszék
(e-mail: max@aut.bme.hu)

Absztrakt: A cikk forgalmi helyzetek felismerésének kialakítását mutatja be Deep Learning alapú rendszer felhasználása mellett. Ez a cikk egy olyan rendszert mutat be, amely képes bizonyos egyszerű forgalmi helyzetek értelmezésére, szétválasztva ezzel a szabályosan vagy szabálytalanul haladó járműveket. Maga a rendszer a forgalmi képek osztályozásán és a közlekedési jelek és táblák felismerésén alapuló tudásra épít. A Deep Learning alapú tudásra létrehozásának az volt a célja, hogy ezzel a technikával növeljük közúti közlekedésbiztonságot. A rendszernek képesnek kell lennie arra, hogy több különböző, a mindennapi közlekedésben szereplő jelet, táblát, stb értelmezni tudjon és ezeket megfelelően osztályozni is tudja. Az így létrehozott tudást felhasználva adjuk meg a forgalom változásából következő forgalmi viszonyok értelmezését, amely jelen esetben a közlekedési szabályok értelmezésre vonatkozik. A valós forgalmi helyzetet vizsgálva meg tudjuk adni a forgalomban résztvevő járművek általunk meghatározott mélységben meghatározottan a közlekedési szabályoknak megfelelően közlekedik-e.

1. BEVEZETÉS

Közúton, de főleg a városi környezetben a biztonságos közúti forgalom általános és megfelelő közlekedési szabályokat követel meg, beleértve a járművezetőknek vagy a forgalmat megfigyelő rendszereknek eljuttatott különféle információkat is. A leggyakrabban használt jelek, jelzések:

- közlekedési jelzőtáblák (forgalomirányítás az áramlási és térbeli szabályozáshoz),
- közlekedési jelek (pl. útburkolati jelzések)
- közlekedési lámpák (közlekedésbiztonsághoz)

Ezeket az eszközöket arra használjuk, hogy segítsük a járművek vezetőit, de egyben tájékoztassuk is őket a helyi közlekedési szabályokról. A tervezett intelligens városok egyik várható szolgáltatása az intelligens mobilitás. Már ma is több, mint egymilliárd gépjármű mozog a világ utjain, amelyek száma 20-25 éven belül megkétszereződhet. A biztonságos közlekedés fenntartása a kor univerzális kihívásainak egyike [Kun, (2009)]. A célok elérésének érdekében kulcsfontosságú a pontos és megbízható információ megszerzése a jelenlegi és prediktív forgalmi adatok alapján [Gupte, (2002)], melynek fontos eleme a közlekedési szabálytalanságok kiszűrése a biztonságos közlekedés megvalósítása érdekében. Az elmúlt években jelentősen megnőtt az utak mellett elhelyezett megfigyelőeszközök száma. Ezek az eszközök időegységenként igen jelentős mennyiségű adatot képesek különböző típusú érzékelőik segítségével begyűjteni. Már itthon is rengeteg kamera működik, amelyek adatokat továbbítanak a megfigyelt területről, az ott megfigyelhető objektumokról, eseményekről [Max, 2012]. Egy közúti megfigyelés során, az ott mozgó járművekről folyamatosan különféle adatok – pozíció, sebesség, orientáció, méret, szín, felhasznált üzemanyag mennyiség, stb. –generálódnak, amelyek igen jelentős adatfogalmat jelentenek a megfigyelőhely és a központ között [Max, (2014)], [Simic (2014)]. Az már egy másik [Max, (2007)], külön döntéshozatali mechanizmus, hogy ezeket az adatokat kik számára tesszük elérhetővé, mert az itt nyert adatok döntéshozatali pozícióba állíthatják a rendszer kezelőit, akár human, akár automatikus felismerési rendszereket alkalmazunk [Uke, (2012)]. A cikkben bemutatott rendszer célja, hogy kommunikáljon a rendszer működtetőjével akár real time, akár offline üzemmódban is. A rendszer képes tájékoztatást adni az üzemeltetőknek a potenciálisan veszélyes esetekről, mint pl. a közlekedési jelzőtáblák értelmezéséről vagy az egyes forgalmi sávokban mozgó járművek fizikai paramétereiről. A közlekedési táblákkal vagy lámpákkal kapcsolatos információk értelmezése jelentősen csökkentheti a veszélyt a közlekedési szabályok véletlen megsértése közben, amit pl. azokban az esetekben alakulnak ki, amikor a jármű vezetője ismeretlen területen halad át. Rendszerünket elsősorban a közlekedési szabálytalanságok észleléséhez fejlesztettük ki. A rendszer valós idejű képet készít kamerájával, melyek feldolgozása közben a végrehajtott algoritmusok részben a közlekedési táblák, jelek, lámpák felismerésére koncentrálnak, miközben már néhány alapvető közlekedési szabályt (sebesség, irányváltoztatás, jelzőlámpák) is monitoroznak.

A felszíni közlekedési rendszerek esetén kritikus fontosságú a gyors és hatékony reagálás a különböző emberi eredetű katasztrófákra. Az intelligens közlekedési rendszerek

1. This work was partially supported by the TÁMOP-4.2.1.D-15/1/KONV-2015-0008 project.

részeként internetkapcsolat és kamerák jelentek meg mindenhol az útak mentén. Bár a közlekedési szabálytalanságok felismerésének technikai kivitelezése ma még jelentős műszaki problémának minősül, több irányból kísérleteznek a megvalósításáért. Jelenleg két nagy irányzat létezik. Az egyik a jármű környezetének megfigyeléséből von le következtetéseket, míg a másik a vezető nélküli közlekedés kérdéseit vizsgálja. [Wenjie, (205)] magas szintű algoritmusokból álló rendszert hozott létre a közlekedési lámpák ellenőrzésére. [Zang, (1997)] egy párhuzamos rendszert használ a sebességgel kapcsolatos információk kinyerésére, a forgalom mozgásirányának meghatározására. Ennek a rendszer az volt a hátránya, hogy nem szolgáltatott adatokat a jármű méretéről, alakjáról és a járművek számáról sem. A közlekedési rendszerek tanuló algoritmusait [Max, (2017)] iamertette. Önjáró autókról naponta hallhatunk, bár a legtöbb esetben a hírekbe azok a jelentések kerülnek be, amelyek valamilyen hibáról vagy kárról tudósítanak.
A cikk további részeiben a következőkről olvashatunk. A 2. fejezetben a javasolt rendszer technikai hátteréről szólunk, bemutatva azokat a technikákat, tanuló algoritmusokat és feltételes megközelítéseket, amelyek a várható események kombinációit tartalmazzák. A 3. fejezet a rendszer elkészítése során felhasznált OpenCV bemutatásával és installálásával foglalkozik. A kísérleti eredményekről az 5. fejezetben olvashatunk, míg a 6. fejezet a következtetéseket tartalmazza.

2. OBJEKTUMDETEKTÁLÁS PONTÉRZÉKELÉSSEL

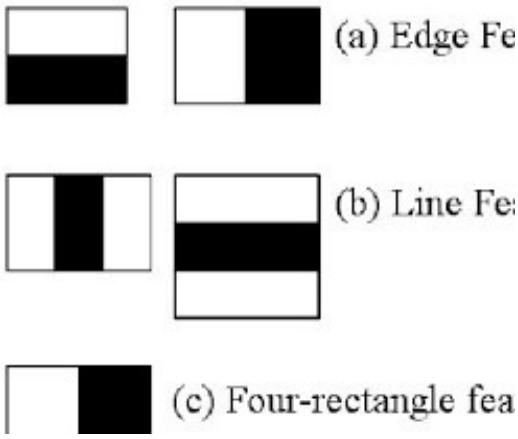
2.1 Viola-Jones kaszkád felismerő

A Viola-Jones objektum felismerő keretrendszert 2001-ben mutatta be Paul Viola és Michael Jones. Ez volt az első olyan tárgyfelismerő keretrendszer, ami képes volt valós időben érdemi detektálást produkálni [Viola, (2001)]. A keretrendszert igaz elsősorban arc-felismerésre találták ki, de jól alkalmazható bármilyen objektum detektálására is.

Ez a megközelítés négy lépésből épül fel:

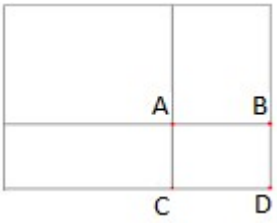
1. Haar szűrők kiválasztása
2. Integrált kép létrehozása
3. Adaboost gépi tanulási módszer
4. Kaszkád osztályozó, amely hatékonyan kombinálja a jellemzőket

Az algoritmus kezdetben átkonvertálja az összes képet fekete-fehér képekké, ekkor a pixelek értékei szürkeskála szerint 0-tól 255-ig vehetnek fel értékeket. A Haar szűrőket felhasználva, amelyek ahogy azt a 2.1 ábra mutatja nem mások, mint fehér és fekete téglalapok régiói, meghatározzuk a régiók közti különbséget, még pedig úgy, hogy kivonjuk fekete régiókból a fehér régiók összegét. A régiók alatti pixel értékek összegének a kiszámítása során, alkalmazzuk a 2.2 ábrán bemutatott integrál képet. Ez jelentősen meggyorsítja a számolást, hiszen egy 24x24-es ablak is több mint 160 ezer szűrőt eredményez [OpenCV Face, (2017)].



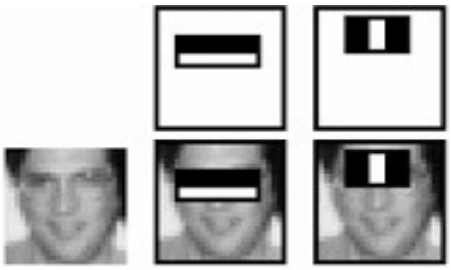
2.1 ábra: Haar szűrők fajtái

Az integrál kép lényege, hogy a téglalapok által lefedett régiók pontonkénti összegzése helyett, a téglalap négy sarok pontjából számítja ki az összeget.



2.2 ábra: A fenti kép pixeleinek összege az integrál kép alapján: D – B – C + A

Miután megkaptuk a régiók különbségét, összehasonlítjuk egy adott küszöb értékkel és ennek eredménye dönti el, hogy az adott jellemző (lásd 2.3 ábra) megfelelt-e vagy nem. Ahogy haladunk a szintekkel, úgy növekszik ezeknek a jellemzőknek a bonyolultsága, illetve egyre szigorúbb küszöbértékeknek kell megfelelniük.



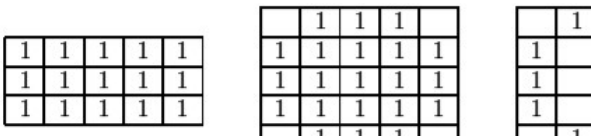
2.3 ábra: Haar jellemzők különböző szinteken

Viszont, a legtöbb jellemző, amit kiszámolunk azok számunkra irrelevánsak. Ha a felső kép első ábráját tekintjük, akkor azt látjuk, hogy az arc azon jellemzőjét vizsgáljuk, hogy a szemek, illetve a szemközti rész sötétebb, mint az alatta, illetve a felette lévő részek. Azonban, ha ezt a kép többi részén vizsgálnánk pár esetett eltekintve aligha kapnánk releváns választ. Annak érdekében, hogy ezeket az érdektelen területeket kiszűrjük, az Adaboost kerül alkalmazásra. Először minden képhez hozzá rendeljük az összes Haar jellemzőt. Az Adaboost, amely egy gépi tanuló algoritmus,

minden jellemzőhöz megkeresi a legjobb küszöbértéket, amivel eldönthetjük, hogy az adott tárgy a keresett tárgynak megfelel, vagy nem. Ezzel az algoritmussal nagyban növeljük a hatékonyságot, így csökkentve a számítási időt.
Végül egy összegző osztályozó összegzi az előbb említett úgynevezett „gyenge" osztályozókat. A megnevezés onnan ered, hogy önmagukban ezek az osztályozók nem képesek osztályozni a képet, viszont együttvéve egy igen effektív osztályozót alkotnak. A Viola-Jones publikáció szerint akár 200 jellemző is képes 95%-os detektálást biztosítani.
A Viola-Jones objektum felismerő keretrendszer alkalmazásához, a fentiek alapján először egy tréninget kell lefuttatnunk. Ehhez az OpenCV biztosít egy alkalmazást opencv_traincascade néven [OpenCV Cascade, (2017)]. Itt szeretném kiemelni, hogy a tréning az elérhető számolási teljesítmény függvényében, illetve a tréning paramétereitől függően, órákat esetleg napokat is igénybe vehet. A tréning végén egy XML fájlt kapunk, amely tartalmazza a Haar jellemzőket.

2.2 Matematikai morfológia

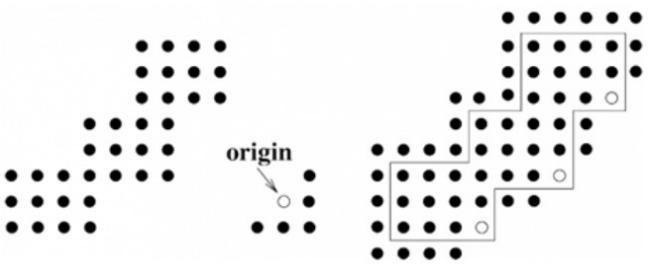
A matematikai morfológia [Csetverikov, (2014)] jól használható bináris képeken, ilyen képeket kapunk a háttérleválasztás során is. A képeken található tárgyak szegmentációjára, illetve kiemelésére alkalmas műveletek közé tartozik a dilatáció, erózió, nyitás és a zárás. Ezen operátorok két bemenettel rendelkeznek az előbb említett bináris képpel és egy, a 2.4 ábrán látható strukturáló elemmel. A strukturáló elem, más néven kernel, maga is egy bináris kép. Ezek az elemek elérhetőek különböző alakban, nagyságban.



2.4 ábra: példák strukturáló elemekre: doboz (3,5), diszk (5), gyűrű (5)

2.2.1 Dilatáció

A dilatáció során az előbb említett strukturáló elemet középpontjánál fogva pixelenként illesztjük a bináris képhez. Azokon a képpontokon ahol az illesztett maszk alatt háttér van, ott az illesztés után előtér képződik.

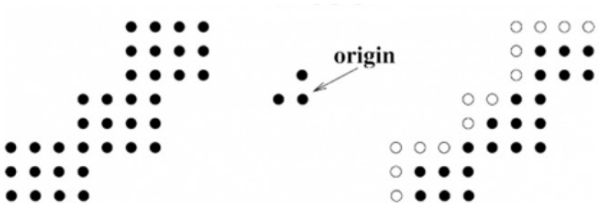


2.5 ábra: Példa dilatációra

Ennek következtében a határok, ahogy azt a 2.5 ábra is mutatja, vastagodnak az esetleges kisebb méretű lyukak akár el is tűnnek.

2.2.2 Erózió

Az erózió során is strukturáló elemet használjuk fel, annyi különbséggel, hogy az illesztés után, ha a maszk alatt bármely pixel háttér, akkor a teljes maszk alatti rész is, a 2.6 ábra szerint, háttérré alakul. Az erózió használható zajszűrésre, illetve képek szegmentációjára.



2.6 ábra: Példa erózióra

2.2.3 Nyitás

A 2.7 ábrán bemutatott nyitás az előző két művelet kombinációja, egy erózió majd egy dilatáció elvégzésével kapjuk meg. A nyitás után a kisebb zajok eltűnnek, illetve az objektum mérete is csökken valamint a régiók vastagsága is. A nyitás újbóli elvégzése nem változtat a képen.

2.7 ábra: Példa nyitásra

2.2.4 Zárás

A zárás a nyitáshoz hasonlóan a dilatáció és erózió kombinációja, azonban először a dilatációt végezzük el és ezt követően az eróziót. A 2.8 ábrán látható, hogy a dilatációhoz, az előtér mérete növekszik, azonban nyitáshoz képes ez a transzformáció kevésbé romboló. A zárás, a nyitáshoz hasonlóan cask egyszer kell elvégezni.

2.8 ábra: Példa zárásra

3. OpenCV

Az OpenCV egy nyílt forráskódú számítógépes látást megvalósító függvénykönyvtár, ami Android, Linux, Windows, illetve Mac OS operációs rendszerekkel is kompatibilis. Az elsődleges interfésze C++, viszont tartalmaz C, Python, Java és MATLAB interfészeket is. A több mint 2500 algoritmus közül, amit az OpenCV tartalmaz a legtöbb algoritmus C++-ban implementálták. Az implementált algoritmusok sokszínűségéből és minősége miatt nem csak magánemberek körében terjedt el ez a függvénykönyvtár, hanem cégek, kutató csoportok, illetve kormányzati szervek is előszeretettel használják [OpenCV, (2017)].Az OpenCV támogatja a többmagos processzorhasználatot [Uke, (2007)], ami kulcsfontosságú volt a Haar algoritmus betaníttatása miatt.

3.1 Telepítés és a fejlesztő környezet konfigurálása

A rendszer kialakításához az OpenCV 3.3.0 függvénykönyvtárát használtuk. Az OpenCV hivatalos oldaláról a megfelelő platformra tudjuk letölteni a kívánt verziót. Miután letöltöttük a programot fontos, hogy adminisztrátor módban futassuk a telepítőt, ugyanis csak ekkor érhető el a többmagos processzorhasználat. Miután feltelepítettük a programot, az OpenCV „bin” könyvtárának az elérési útvonalához hozzáadunk egy rendszerváltozót. Ki kell emelni, hogy a fejlesztői környezet nagyban befolyásolja, hogy milyen verziójú OpenCV-t használhatunk. A rendszer megvalósításához a Microsoft Visual Studio 2015-öt választottam, amihez a vc14-es verziójú OpenCV ajánlott.

A fentebb leírtak elvégzése után hozzunk létre egy új, üres C++ nyelvű projektet. A projekt elkészítése után szükségünk van arra, hogy az OpenCV könyvtárakat belinkeljük a projekt számára, hogy azokat használni tudjuk. A következő útvonalakat release és debug módban is szükséges megadnunk:

- Configuration Properties/C-C++/General/Additional Include Directorieshez az OpenCV „include” mappájának elérési útvonala,
- Configuration Properties/Linker/General/Additional Library Directorieshez az OpenCV „lib” mappájának elérési útvonala.

Release módban Configuration/ Properties/ Linker/ Input/ Additional Dependecies-hez hozzáadjuk a „lib” kiterjesztésű fájlok közül az opencv_world330.lib-et, debug módban az opencv_world330d.lib-et. Ez a két fájl az általunk választott verzió függvényében változhat.

4. MEGVALÓSÍTÁS

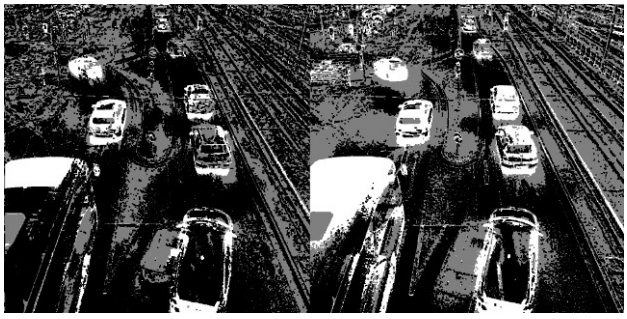
4.1 Járművek felismerése

A járműfelismeréshez először a MOG2 algoritmus felhasználásával próbálkoztunk. A BackgroundSubtractorMOG osztályt a createBackgroundSubtractorMOG függvénnyel hozhatjuk létre, majd paramétereit a megfelelő setter függvényekkel állíthatjuk be. Az aktuális képkockához tartozó előtér maszk lehívásához meg kell hívnunk az osztály apply(InputArray image, OutputArray fgmask, double learningRate) metódusát. Az apply függvény learningRate paraméterének lehetséges értékei 0 és 1 között választhatóak, annak függvényében, hogy a háttérmodell milyen gyorsan tanul. Abban az esetben, ha nullát választunk a háttérmodell egyáltalán nem fog frissülni, ha egynek választjuk a learningRate-t akkor teljes mértékben az utolsó kockából fog újrainicializálódni a háttérmodell. A KNN osztály hasonló felépítéssel rendelkezik, mint a MOG2 osztály.

Több videó elemzése után egyértelművé vált, hogy a két algoritmus közül a KNN jelentősen jobban teljesít, abban az esetben, ha az előtér pixeleinek száma alacsony. Azonban a háttér hirtelen megváltozásánál, például egy világos nagyméretű jármű megjelenésekor jelentősen megváltozik a kontraszt arány és ilyenkor a MOG2 láthatóan jobban teljesít. Mindkét algoritmust szürkeárnyalatos kép konverzió után futott, ugyanis a színes képek feldolgozása jelentősen több számítási erőforrást igényel, ami viszont a végeredményen alig észrevehető. A különbséget a 4.1 és a 4.2 ábrák mutatják be.

4.1 ábra balra MOG2, jobbra a KNN (learningRate = 0.008)

A döntés végül a KNN háttérleválasztó algoritmusra esett, ugyanis az objektum követő algoritmus implementálása után, a hirtelen fényviszony változások által okozott zajok hatása elhanyagolhatóvá vált. Itt kell megjegyezni, hogy a legtöbb problémát azok a járművek okozták, amelyek szín intenzitása, közel hasonló az aszfaltéval.

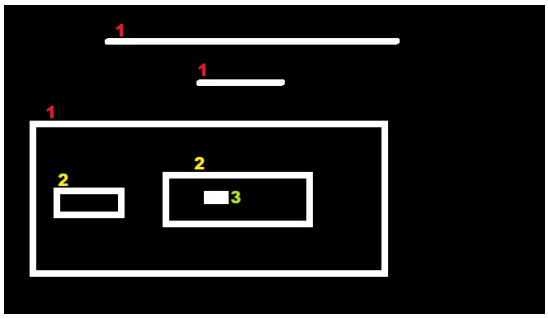


4.2 ábra balra MOG2, jobbra a KNN (learningRate = 0.008)

Annak érdekében, hogy ezeket az autókat is detektálhatóvá tegyük, egy új osztály kialakításával további képfeldolgozási eszközöket kellett alkalmazni.

4.2 Kontúrok

Ahhoz, hogy a járműveket érzékelni lehessen, fontos volt, hogy egybefüggő ponthalmazok alkossák ezeket a mozgó objektumokat. A 2.2.4-es bekezdésben részletezett zárás műveletét felhasználva, sikerült a régiók vastagságának megnövelni, ami elősegítette a pontok összekapcsolását, miközben a zajok sem növekednek jelentős mértékben. Ezek után a findCounturs(InputOutPutArray image, OutputArrayOfArrays contours, int mode, int method, Point offset=Point()) függvény segítségével kinyertük a képből az összefüggő ponthalmazok határait. A függvény első paramétere a bemeneti kép, a második a képen található kontúrokat tartalmazza. A harmadik paraméter a kontúrok egymáshoz viszonyított hierarchiájával kapcsolatos. Ez azt jelenti, ahogy azt a 4.3 ábra is mutatja, hogy kapcsolatot teremtünk a kontúrok egymáshoz viszonyított helyzetéhez, még pedig jelen esetben úgy, hogy ha az egyik kontúron belül található egy másik kontúr, akkor az előbbi leszármazottja lesz az utóbbi. Négy módja van: RETR_LIST, RETR_EXTERNAL, RETR_CCOMP, RETR_TREE. Rendszerünkben a RETR_EXTERNAL mód jelentette a célszerű megoldást ugyanis, ez a mód csak a legfelső rangú kontúrokat adja vissza.



4.3 ábra: Kontúrok hierarchiájára példa

A negyedik paraméterrel meghatározzuk a visszaadott kontúrok pontjait. Jelentős mennyiségű memóriát spórolhatunk meg, ha a teljes határpontok halmaza helyet, csak a sarok pontokat adjuk meg. Ehhez a CHAIN_APPROX_SIMPLE változót kell megadnunk. Az utolsó paraméter az úgynevezett ROI-k (region of interest) használata mellett érdemes használni, ugyanis ezzel egy ofszet értéket adhatunk a kontúrok pontjaihoz. A drawContours utasítással ki is rajzolhatjuk a 4.4 ábra szerinti kontúrokat.

4.4 ábra: Kontúrok megjelenítése

4.3 Blob osztály

A Blob osztály az előző pontban megkapott kontúrokra épül. A blobokat az aktuális képen jelen lévő kontúrokat felhasználva hozzuk létre. A boundingRect függvény segítségével megkapjuk a kontúrokat körbehatároló téglalapokat, először a bal felső sarok pontját, majd a szélességét és a magasságát. Ezekből a paraméterekből könnyen kiszámolható a kontúr átmérője, oldalainak aránya, illetve középpontja, amit a Blob osztályban külön vektorban tárolunk. A Blob oztály további paraméterei a jármű helyzetéről ad információt, illetve egy-egy paramétert rendeltünk hozzá minden szabálytalansághoz.

A Blob osztály egyik alapfüggvénye a predictedPos(void), ez a függvény kulcsszerepet játszik a már felismert járművek nyomon követésében. A felismert járművek összes pozícióját egy vektorban tároljuk. Ezen függvény segítségével, egy becslést adunk az előző középpontok helyzete alapján. Annak érdekében, hogy ez a becslés releváns legyen, legfeljebb az előző 5 helyzetet vettük számításba.

Az isCarEntered(Point entrypoint1, Point entrypoint2) függvény egy vonal két végpontját veszi át, majd ez alapján vizsgálja, hogy az adott blob belépett-e a vizsgált zónába vagy sem. Mivel az elmozdulás képkocáról képkockára nagyobb is lehet egy pixelnél, így a vonalat egy 20 pixeles sávvá alakítottuk, hogy az adott blobot biztosan érzékelje. Ez a függvény a carEntered és beingTracked bool értékeket állítja igazra, amint a blob a sávon belülre ér.

Az isCarExited(Point exitpoint1, Point exitpoint2) függvény hasonlóan az előző függvényhez, egy vonal két végpontját veszi át, azonban itt a kilépést vizsgáljuk. Ha a blob középpontja ezen a sávon belül helyezkedik el, carExited értéke igazzá válik, a beingTracked értéke hamis értékre változik. Ez függvény tartalmazza a measureTheSpeed(void) metódust is amit, abban az esetben hív meg, ha a blob frameOfSpeeding változója nagyobb, mint nulla, azaz a sebességméréshez szükséges képkockák száma elérhető.

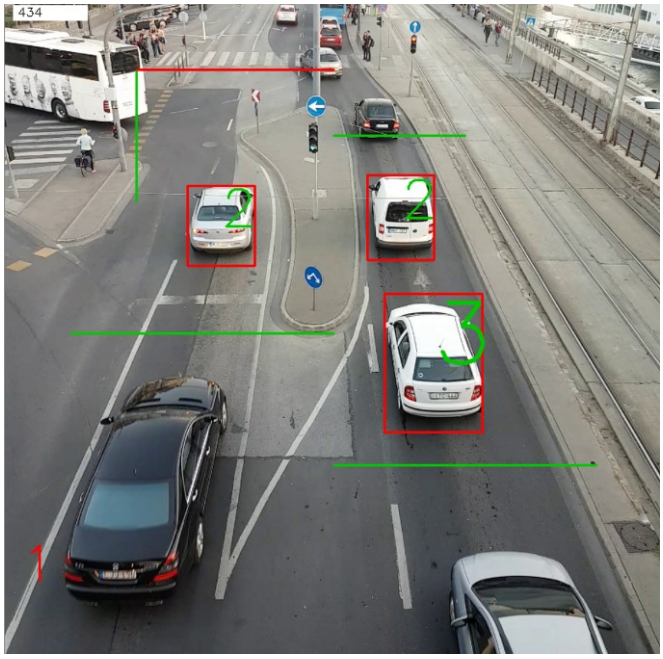
A measureTheSpeed(void) függvény a speedOFvehicle változóba menti el a blob sebességét, amit a frameOfSpeeding változó felhasználásával számít ki a következőképpen:

```
void Blob::measureTheSpeed(void) {
    speedOfvehicle = (20 / (frameOfSpeeding / 25))*3.6;
}
```

A sebességet a megtett táv és az eltöltött idő hányadosaként kapjuk meg. Az idő kiszámításához a blob belépése óta eltelt képkockák számát osztjuk el az átlag képkocka számmal. Ahhoz, hogy km/h legyen, a mértékegység szükség van egy 3.6 szorzóra, ugyanis ennek hiányában m/s-ban kapnánk meg az eredményt.Az isCarExitedandViolated(Point violationpoint1, Point violationpoint2) metódus szintén a kilépést ellenőrzi, azonban itt a carExitedandViolated változó értéke lesz igaz, illetve a beingTracked értéke hamis.

4.4 Járművek identifikálása és nyomon követése

A járművek identifikálása képkockáról, képkockára történik. Minden képkockán, egy nyomkövető algoritmus segítségével azonosítjuk a járműveket. Az ezt megvalósító függvény a matchTheBlobs(vector<Blob> &existingBlobs, vector<Blob> &currentFrameBlobs, Point entryLine[], Point exitingLine[], Point violationLine[], bool redLamp) és a matchTheBlobs(vector<Blob> &existingBlobs, vector<Blob> &currentFrameBlobs, Point entryLine[], Point exitingLine[]). A két függvény a két különböző sávban haladó gépjárművek mozgását rögzíti, illetve a Blob osztály függvényei segítségével rögzíti az esetlegesen elkövetett szabálytalanságokat. A különböző sávokon haladó járművek megkülönböztetése az alapján történik, hogy a kép mely felén találhatóak az elhaladó járművek. Mindkét esetben első lépésben – lásd 4.5 ábra – az eddig rögzített járművek CurrentMatchFoundOrNewBlob változóját hamis értékre állítjuk, illetve kiszámítjuk a predictedPos() függvény segítségével a várható pozícióját. A következő lépésben összehasonlítjuk az aktuális képkockán lévő járművek távolságát az eddig nyomon követett járművekével.



4.5 ábra: Pillanatkép a program működéséről

Azon jármű, amely a legkisebb távolságra van az eddig felismert járművektől, ha ez a távolság az adott jármű átlójának a 20%-nál kisebb, az addBlobToExistingBlobs(Blob &currentFrameBlob, vector<Blob> &existingBlobs, int &intIndex) függvény segítségével felülírja a jármű addigi tulajdonságait, pozícióját, illetve igaz értéket ad a CurrentMatchFoundOrNewBlob változónak, ezzel jelezve, hogy a jármű az adott képkockán azonosítva lett. Abban az esetben, ha az aktuális képkockán található járműre nem teljesül az előbb említett feltétel, illetve csak a belépési vonalon haladt keresztül az addNewBlob(Blob &currentFrameBlob, vector<Blob> &existingBlobs) függvényen keresztül hozzáadjuk a már megfigyelt járművekhez. Attól függően melyik sávban haladt a jármű ellenőrizzük, hogy áthajtott-e piros lámpán, ha igen a redRun változónak igaz értéket adunk. Végül az elmentett járműveket ellenőrizzük, hogy áthajtottak valamely kilépési ponton, illetve ha a CurrentMatchFoundOrNewBlob paraméterük értéke hamis növeljük a NumOfConsecutiveFramesWithoutAMatch paraméter értékét eggyel. Ennek a paraméternek a szerepe, ahogy a nevéből is kiderül az, hogy számlálja hány képkocka óta vesztettük el egy jármű helyzetét, ha ennek a száma meghaladja a 15 képkockát, akkor a jármű követését leállítjuk, ugyanis ekkor már minimálisra csökken annak az esélye, hogy újra a megfelelő járműt találjuk meg. A paraméter létezésének az indoka nem más, mint a háttérleválasztó algoritmus érzékelési pontatlanságának a javítása például hirtelen fényváltozás hatására hirtelen megnőhet a járművek kiterjedése a leválasztott előtéren.

A járműveket és az általuk elkövetett szabálytalanságokat a void drawBlobInfoOnImage(vector<Blob> &blobs, Mat &FrameCopy) függvény jeleníti meg.

4.5 Közlekedési táblák felismerése

A közlekedési táblák felismeréséhez a 2.1 bekezdésben részletezett Viola-Jones kaszkád felismerő algoritmust használtuk fel. Az objektum detektálás megvalósításának első lépése egy pozitív képekből álló gyűjtemény létre hozása. Jelen esetben ezek a pozitív képek a kereszteződésben számomra releváns két tábla az egyenesen haladást jelző, illetve a balra haladás jelző táblák. Ehhez a [Stallkamp, (2011)] publikációhoz tartozó adatbázist (lásd 4.6 ábra) használtuk fel. A képek különböző fényviszonyok mellett lettek elkészítve, illetve kisebb árnyékokat tartalmaznak, ami segít abban, hogy a betanított osztályozó robusztus képfelismerésre legyen képes. A következő lépés az olyan képekből álló gyűjtemény létre hozása volt, amelyek nem tartalmazzák a felismerni kívánt tárgyat, ugyanakkor célszerű hasonló környezetben készíteni ezeket a képeket, mint amiben a detektálni kívánt kép helyezkedik el, ugyanis így jelentősen csökkenthetjük az álpozitívok számát. Mindkét tábla betanításánál saját készítésű negatív képeket használtunk fel. A pozitív képek adatbázisában csak az előre haladást jelző tábla szerepelt. Így a balra fordulást jelző táblát ennek elforgatásával kaptuk meg.

4.6 ábra: Egyenes irányban való továbbhaladást jelző table

Ahogy a fenti ábrán is látható a képek tartalmazzák a hátteret is. Az általam felhasznált adatbázis tartalmazza ezeknek a képeknek azon téglalap koordinátáit, ami körbe határolja a táblát. Abban az esetben, ha nem tartalmazta volna, manuálisan kell kijelölni ezt a régiót az OpenCV által is

biztosított opencv_annotation programmal. Ez a program az algoritmusnak megfelelő formátumban menti ezen régiókat, amit később egy szintén OpenCV által előre megírt, opencv_createsamples programot felhasználva egy vec fájlt készítünk. A vec fájl tartalmazza, a pozitív képek egy általunk előre meghatározott szélességre és magasságra konvertált változatát.

4.5.1 Az algoritmus betanítása

A Haar kaszkád osztályozó számára meg kell adnunk az előző pontban megadott vec file elérési útvonalát és a negatív képek elérési útvonalát tartalmazó szöveg fájl elérési útvonalát, illetve meg kell adni a kaszkád tréning 4.1 táblázat szerinti paramétereit.

Változók	Paraméterek
-numPos	1180
-numNeg	3002
-numStages	17
-minHitRate	0.999
-maxFalseAlarmRate	0.5
-weightTrimRate	0.999
-height	15
-width	15

4.1 tábla: kaszkád tréning paraméterei

A tréning befejezte után, minden szint osztályozóját, illetve a végső osztályozót XML fájlokként kapjuk meg, mely tartalmazza az összes olyan információt, ami szükséges a felismeréshez. A 4.1-es táblázatban található paramétereket az internetes ajánlások és sok próbálkozás eredményeként lehet cask pontosan beállítani. Az ezekhez szükséges betanítás idő a kis szélesség és magasság választás miatt csak 2 órát vett igénybe az általunk használt Intel Core i7-4700MQ @ 3.4GHz processzorral és 8GB RAM felhasználásával.

4.5.2 Felismerés a programban

A detektálást a CascadeClassifier osztály detectMultiScale(InputArray image, vector<Rect> & objects,double scaleFactor = 1.1,int minNeighbors = 3,int flags = 0,Size minSize = Size(),Size maxSize = Size()) függvénye végzi. Ezt C++-ban megírt detektort használtuk fel, a searchObject(char* CascadeName, Mat &frame, int width, int height) függvényben. Ez a függvény egy bool értékkel tér vissza, ha talált objektumot a képen és egy ellipszist rajzol a talált objektum köré.

4.6 Piros lámpa felismerése

A piros lámpákat hasonlóan a közlekedési táblákhoz, a Viola-Jones algoritmus felhasználásával valósítottuk meg. Azonban itt nem volt már elérhető adatbázis, így a pozitív képekből álló adatbázist is önállóan hoztuk létre. Ideálisan 1000-1500 kép szükséges egy érdemi detektáláshoz, azonban idő hiánya miatt nekem csak 650 képet sikerült készült. A felismerés hatékonyságát jelentősen csökkentette a nem pontos objektum kijelölés a pozitív képeken. Legtöbbször a piros lámpán kívül még a háttér egy kis részlete is szerepelt a képen. A betanításhoz használt jellemzők száma ugyanakkor jelentősebb nagyobb, ugyanis a közlekedési tábláknál felhasznált képekkel ellentétben itt a legkisebb kép is jelentősen nagyobb volt 15x15 pixelnél.

Változók	Paraméterek
-numPos	650
-numNeg	3002
-numStages	14
-minHitRate	0.999
-maxFalseAlarmRate	0.5
-weightTrimRate	0.999
-height	17
-width	40

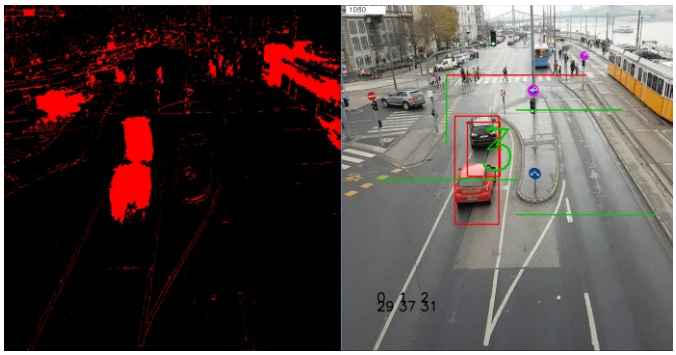
4.2 tábla: kaszkád tréning paraméterei

4.7 Szabálytalanságok megjelenítése

A szabálytalanságok megjelenítését a drawBlobInfoOnImage(vector<Blob> &blobs, Mat &FrameCopy) függvény valósítja meg. Ez a függvény minden képkockára meghívásra kerül és az eddig identifikált járművek jellemzői alapján jeleníti meg az elkövetett szabálytalanságokat. Pirossal jeleníti meg azon kocsik azonosító számát, amelyek áthajtottak a piroson, kékkel azokét, amelyek egyenesen haladtak tovább a balra tovább haladást jelző táblánál, illetve feketével a baloldali sávban haladó járművek azonosítóját és sebességét.

4.8 Rendszer konfigurálása és tesztelése

Az elkészített rendszert két különböző időpontban és két különböző időjárási körülmény között teszteltük. Az első esetben délután felhős, száraz időben, a második esetben ugyancsak délután, azonban esőben. A megfigyelési zónákat jelölő vonalakat mindkét esetben nekünk kellett előre konfigurálnunk.
A járművek felismerését nagyban befolyásoló tényező az árnyékok nagysága, ugyanis a háttérleválasztás során az árnyékokat a jármű részeként tekinti a felhasznált algoritmus, így, ha két jármű egymáshoz közel halad a két járművet egyként érzékeli. A járművek belépése és kilépése a megfigyelési zónákba mindkét esetben hibamentesen történik, nincsenek álpozitív felismerések. Azonban a megfigyelési zónán belül az esős időben a megfigyelt járművek, ahogy azt a 4.7 ábra is mutatja, a kis követési távolság miatt egymásba érnek.



4.7 ábra: példa hibás detektálásra

5. ÖSSZEZÉS

A rendszer kialakítása során rá kellett ébrednünk arra, hogy egy adott képfeldolgozási feladatot a sokféleképpen lehet megközelíteni. Azonban ahhoz, hogy egy működő rendszert felállítsunk rengeteg idő és tapasztalat szükséges, ugyanis a optimalizálásához először meg kell értenünk a felhasznált algoritmusok. Az így megszerzett tudás azonban sokszor nem elég a feladat megoldásához, mert rengeteg olyan akadályba ütközhetünk, amikre előzetesen nem is gondoltunk. Rendszerünk tökéletesen demonstrálja a képfeldolgozás által nyújtott rengeteg lehetőséget, azonban a rendszer közel sem optimális. További fejlesztési lehetőségként elsőként a piros lámpa felismerést emelnénk ki, ugyanis itt jelentős mértékű javulás érhető el a felhasznált pozitív képek adatbázisának növelésével, illetve a Hough kör transzformáció felhasználásával a már felismert objektumokra, ezzel is csökkentve az álpozitív találatok számát. Egy másik jelentős fejlesztési terület a rendszer valósidejű működésének a megvalósítása. Ennek elérése érdekében célszerű lenne leszűkíteni a vizsgált területek nagyságát. Erre egy lehetséges megoldás lehetne a képen változó területek összegzése majd azok vizsgálata. Annak érdekében, hogy valósidőben fusson az alkalmazás a tárterület nagyságát is figyelembe kell vennünk, amihez célszerű egy olyan rendszer algoritmus megvalósítása, ami rögzíti minden vizsgált jármű be és kilépési idejét, majd ennek alapján menti el a szabálytalanságokat, illetve törli azokat az időközöket, amikor nem történt szabálysértés.

IRODALOMJEGYZÉK

Max, G., (2007). *Képfeldolgozó eljárások a közlekedésben*, Alkalmazott Informatikai Kongresszus, Kaposvár

Max, G., (2012). *Közlekedési szabálytalanságok*, IFFK 2012, Budapest, ISBN 978-963-88875-2-8

Zhang X.U., Forshaw M. R. B. (1997), *A Parallel Algorithim To ExtractInformation About The Motion Of Road Traffic Using Image Analysis,* Centre for Transport Studies, University College London, UK, Elsevier Science Ltd.

Max, G. (2014). *Gépjárműtípus felismerés beépített SW segítségével*, IFFK 2014, Budapest, ISBN 978-963-88875-2-8

Max, G. (2017). *Forgalom előrejelzés Deep Learning tanuló folyamatokkal*, IFFK 2017, Budapest, ISBN 978-963-88875-2-8

Gupte, S., Masoud, O., Martin, R.F., Papanikolopoulos, N.P.,(2002). *Detection and Classification of Vehicles*, in Proc. IEEE Transactions On Intelligent Transportation Systems, Vol. 3, NO. 1.

Kun, A.J., Vamossy, Z., (2009). *Traffic monitoring with computer vision,"* Proc. 7th Int. Symp. Applied Machine Intelligence and Informatics (SAMI 2009)

Simic M., Krerngkamjornkit R., (2014). Multi object detection and tracking from video file, in Modern Tendencies in Engineering Sciences, vol. 533 of Applied Mechanics and Materials, pp. 218–225, Trans Tech Publications, 5.

Uke, N.J., Thool, R. (2012), *Moving Vehicle Detection for Measuring Traffic Count Using OpenCV,* Proceedings of 4th International Conference on Conference on Electronics Computer Technology (ICECT)

Viola P., Jones M.. (2001) Rapid object detection using a boosted cascade of simple features, Conference on computer vision and pattern recognition

OpenCV Face Detection using HaarCascades https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html *(2017)*

OpenCV Cascade Classifier Training, https://docs.opencv.org/3.3.0/dc/d88/tutorial_traincascade.html *(2017)*

OpenCV, https://opencv.org/about.html *(2017)*

Stallkamp J., Schlipsing M., Salmen J., Igel C.. (2011) The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In Proceedings of the IEEE International Joint Conference on Neural Networks, pages 1453–1460.

Wenjie, C., Lifeng, C., Zhanglong, C., Shiliang, T. (2005) *A realtime dynamic traffic control system based on wireless sensor network.* International Conference Workshops on Parallel Processing, pp. 258-264

Csetverikov D., (2014) Digitális képelemzés alapvető algoritmusai TÁMOP-4.1.2 A1 és a TÁMOP-4.1.2 A2 könyvei