Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Lipták Levente

TÖBBHIPOTÉZISES FORGALOMSZÁMLÁLÁS

KONZULENS

Dr. Csorba Kristóf

BUDAPEST, 2016

Tartalomjegyzék

Összefoglaló **3**
Abstract **4**
1 Bevezetés **5**
2 A nyers információk kinyerése **6**
2.1 Háttér eltávolítók 6
2.2 Sarokpont keresők 7
2.3 Alakok 9
3 A rendszer részei **10**
3.1 Kiegészítő rendszerkomponensek 12
3.2 GMM Detector 13
3.3 FP Detector 13
3.4 MSER Detector 14
3.5 GMM Tracker 14
3.6 FP Tracker 16
3.7 MSER Tracker 16
3.8 Assigner 17
3.8.1 Új hipotézisláncok létrehozása és bezárása 17
3.8.2 A meglévő láncok folytatása – a PeakAssigner algoritmus 18
3.9 A valószínűtlen hipotézisek szűrése és az útvonal meghatározása 21
3.10 A hipotézis szűrő 24
3.10.1 Az adatszerkezet felépítése 24
3.10.2 A duplikátumok szűrése 25
3.10.3 A bizonytalan intervallumok meghatározása 26
3.11 A megálló járművek kezelése 27
4 Konklúzió **29**
Irodalomjegyzék **30**

Összefoglaló

Az úthálózat terheltségének mérése megkerülhetetlen feladat a forgalmi lámpák beállításához, és a jövőbeli útfejlesztések megtervezéséhez. Ezt a munkát a mai napig emberi forgalomszámlálók, úgynevezett enumerátorok végzik. Az elkészült rendszer célja a munkájuk megkönnyítése a forgalomba kihelyezett kamerák képeinek feldolgozásával OpenCV (Open Computer Vision Library) segítségével, majd az azokon megtalált járművekből statisztika készítésével.

A kültéri videók miatt a rendszernek tolerálnia kell a változó időjárási viszonyokat, a különböző útkereszteződéseket. A képet több módszerrel dolgozzuk fel, melyek különböző helyzetekben teljesítenek jól, a keletkező adatok azonban eltérő típusúak, összekapcsolásukhoz külön rendszerre volt szükség.

Ahhoz, hogy a nehéz helyzetekben is az elvárt pontosságot biztosíthassuk, több hipotézist hozunk létre az adatokból, melyek így nagyobb valószínűséggel tartalmazzák a helyeset. Az egymást takaró járművek esetén pedig több is helyesnek bizonyulhat. Így ritkán vesztünk el járműveket, viszont téves találatok is bekerülnek az eredmények közé, melyeket később, több információ birtokában meg kell próbálnunk kiszűrni. Dolgozatom során a duplikátumok lehető legpontosabb szűrésének módját is kerestem.

Abstract

The measurement of the traffic load on the road network is essential for calibrating the traffic lights and designing future infrastructural developments. This work is done by so-called enumerators, ie. human traffic counters. The system was created in order to ease their work by processing the pictures of cameras placed on street furniture with OpenCV (Open Computer Vision Library), and make statistics of the vehicles found.

Due to the outdoor environment the software has to tolerate varying weather conditions, and diverse junctions. The image is processed using multiple methods which perform well in different traffic situations, but result in dissimilar data structures which have to be fused by a dedicated subsystem.

To provide the expected accuracy even in complex situations, multiple hypotheses are created from the input data, which include the right one with higher probability. In the case of overlapping vehicles, more of them might be viable. In this way vehicles get rarely lost, but false positives also occur among the results, which have to be filtered out in possession of more information. In my paper, I also aimed at filtering the duplicates with the highest accuracy.

1 Bevezetés

Már az 1860-as években felismerte Hieronymi Károly a magyarországi utak forgalomszámlálásának fontosságát [1]. Bár a számolás módja kissé változott, hisz ma már nem érdemes igavonó állatokat számolni, a mai napig használt módszer lényegében azóta változatlan. Mindkét számolás többszöri mintavételezésen alapszik, melynek célja, hogy ismerjék a forgalom időbeli ingadozásának törvényszerűségeit. Ezt a hosszadalmas műveletet mai napig többségében emberi forgalomszámlálók végzik ezért költséges. A probléma megoldására már léteznek számlálórendszerek[2][3], ezek azonban költségesek, mert nyomásérzékelőket, vagy indukciós hurkokat használnak, melyeket a legtöbb esetben az aszfaltba kell építeni. Célunk egy olyan rendszer létrehozása volt, mely mobilis és kis eszközigényű (egy egyszerű kamera), ezáltal használata költségkímélő. Mindemellett fontos kritérium volt a nagy pontosság is. A jelenleg használatos rendszerek ugyanis 95% feletti hatásfokot ígérnek, minél jobban maradunk el ettől az értéktől, annál kevésbé jelent alternatívát a megoldásunk.

A rendszernek azonban nem csak laboratóriumi körülmények között kell működnie, tűrnie kell a változatos időjárást, illetve könnyen javíthatónak, vagy olcsón cserélhetőnek kell lennie. Mivel a kamerák nem igényelnek emberi felügyeletet, ezért csak a kihelyezésük, és begyűjtésük jár költséggel. Ahhoz hogy ezt minimalizáljuk, érdemes a lehető leghosszabb (és így a statisztikai ingadozásoktól leginkább mentes) időtartamokat rögzíteni, ahelyett hogy több rövid időtartammal [1] próbálnánk garantálni a pontosságot. Még hosszabb üzemidőt érhetünk el, ha kis energiaigényű felvevő rendszert választunk, beszédes adat, hogy az iparban mai napig általános VHS kazettákra való a rögzítés. A fent leírt kritériumok (időjárás tűrése, ár, üzemidő) között nem csak nem szerepel a képminőség, hanem általában még kompromisszumokra is kényszerítenek. Ennek eredményeként a feldolgozandó videók rossz minőségűek, zajosak, melynek tűréséhez különösen robusztus rendszer szükséges.

2 A nyers információk kinyerése

Első lépésben el kellett döntenünk, hogy a rendszer milyen módon dolgozza fel a videókat, vagyis milyen algoritmus hozza létre a videóból a nyers információkat. A különböző algoritmusok ugyanis más helyzetekben teljesítenek jól, más típusú információkat vonnak ki a videóból, amelyek így eltérő zajjal terheltek, ezáltal a rendszer architektúráját is meghatározzák. A célunk rossz képminőség tűrése volt, amihez lehető legtöbb információt kellett kinyernünk a videóból.

2.1 Háttér eltávolítók

Az objektumkövetésben általános módszer a háttér eltávolítók használata[5], melyek a kép minden pontjának tárolják a szokásos színét, és az ettől jelentősen eltérő pixeleket jelzik előtérként. A háttér eltávolítók általában stabilan felfedezik a mozgó objektumokat, azonban a lassan mozgó, vagy esetleg megálló (pl. elsőbbséget adó) járművek képe beolvadhat a háttérbe, ezáltal eltűnhetnek. Az algoritmus tanulási idejének hosszabbra állításával ezt némiképpen korrigálni lehet, de ez nem teljes értékű megoldás, hisz ezzel folyamatosan rontjuk a rendszer adaptálódási képességét a megváltozott háttérhez. Ahhoz, hogy egy forgalmi lámpa miatt hosszabb időre megálló autót se veszítsünk el, több percnyi videóra kell visszatekinteni, viszont egy ennyire lassan adaptálódó háttér eltávolító több másodpercre megvakul, ha a fényerő megváltozik, pl. kibújik a nap egy felhő mögül. A megálló objektumok környékén le is lehet tiltani a háttér adaptációját (a Selective Updating Of GMM algoritmus pl. egy ilyen megoldás)[6], azonban ez sem tökéletes megoldás, mert ezáltal a rendszerbe egy pozitív visszacsatolás kerül, ami természeténél fogva gerjedést okozhat. Esetünkben egy leparkoló autó így örökké a kép előterébe éghet, illetve egy erős fényerőváltozás miatt tévesen detektált folt alatt a háttér beragadhat, ami az okozott problémát a sokszorosára növelheti.

Mivel a keletkező foltok két dimenziósak, az egymást takaró objektumok képei összeolvadnak, így számítanunk kell az összeolvadó objektumok problémájára is használatukkal.

A legszélesebb körben használt háttér eltávolító a GMM (Gaussian Mixture Model), mely háttérnél is jól teljesít[7], ami jól használhatóvá teszi kültéri

rendszerekben, ráadásul OpenCV-ben megtalálható az implementációja. Az általunk használt implementáció neve MOG2, mely egy olyan továbbfejlesztett verzió, amely árnyékokat is tartalmaz[8][9].

Az árnyékok keresése azon alapszik, hogy egy árnyékban lévő objektum színének árnyalata nem változik, csak az intenzitása csökken. Ez az elképzelés igaz is, viszont a közeli objektumokról egymásra verődő fény eltorzítja a színüket. A mi esetünkben további problémát jelent az is, hogy a járművek több mint 70%-ának nincs tényleges színárnyalata, színe a fekete-fehér skálán található[10]. Ezen járművek gyakran az aszfalt színétől sem ütnek el, nagyon könnyű őket árnyékos vagy anélküli aszfaltnak tekinteni, ráadásul saját magukra is árnyékot vethetnek. Vagyis az árnyékok nem tekinthetők megbízható információnak, érdemes szándékosan engedékenyre vagy szigorúra választani az értékét attól függően, hogy a rendszer a hamis pozitívokat, vagy hamis negatívokat képes jobban tolerálni.

Az előtér foltok hibáik ellenére is nagy segítséget nyújtanak minden olyan rendszerben, ahol többnyire mozgó objektumokat szeretnénk követni. A párosításuk pedig könnyű, a keletkező foltok között egyszerűen lehet távolságot, vagy átfedést számolni.

2.2 Sarokpont keresők

Objektumkövetésben gyakori módszer a jellegzetes pontok (feature point) használata is[11]. Ez azt jelenti, hogy olyan pontokat keresünk a képen, melyek jellegzetesek, matematikailag könnyen leírhatóak, ezért később nagy eséllyel újra azonosíthatóak. Minél nagyobb különbség van a pont környékén lévő színek között, annál könnyebben találja meg egy ilyen detektáló, ezért gyakran szokás őket sarokpontoknak is hívni.

Bár nem igaz, hogy egyáltalán nincs kiterjedésük, hisz különböző leírókkal jellemzik a környezetüket, mégis tekinthetőek pontszerűnek. Ennek oka, hogy egy részben takart pont környezete vagy még felismerhető, vagy már egyáltalán nem, ellentétben az előtér foltokkal, ahol a beazonosítás általában pozícióhoz kötött, így egymást átfedő objektumoknál kétséges esetek állnak elő.

Önmagában való használatuk azonban nehézkes több objektum esetén, mert nehéz meghatározni a két objektum közti határvonalat, amennyiben egymáshoz közel

mozognak, esetleg hasonló mozgáspályán. Ez az eset pedig gyakran fenn áll, hisz általános jelenség, hogy a járművek egymás után haladnak egy sávon belül. Amíg a háttér eltávolítók készen szolgáltatják a kép mozgó részeit, és így a mozgó objektumokat követik jól, addig a sarokpontok az állandósult képrészek megtalálásában jeleskednek. Minél gyorsabban haladnak a járművek a képen, annál több sarokpont környezete fog felismerhetetlenné torzulni, így annál kevesebbet tudunk majd egy későbbi képkockán újra megtalálni. Fontos megjegyezni, előtér foltokkal ellentétben a sarokpontok csak ritkán válnak teljesen használhatatlanná, a nem ideális környezetben is szolgáltatnak használható adatot.

Az általunk választott algoritmus a GFTT (Good Features To Track)[12], amely ahogy a neve is mutatja, kifejezetten objektumok követésére lett optimalizálva. Előnye hogy nem csak sok sarokpontot tud találni, de ezeket különböző fényviszonyok között nagy arányban újra meg tudja találni[13], így jól használható kültéri objektumkövetéshez. Az implementációja pedig szintén megtalálható az OpenCV függvény könyvtárban így könnyedén hozzáférhető.

2.3 Alakok

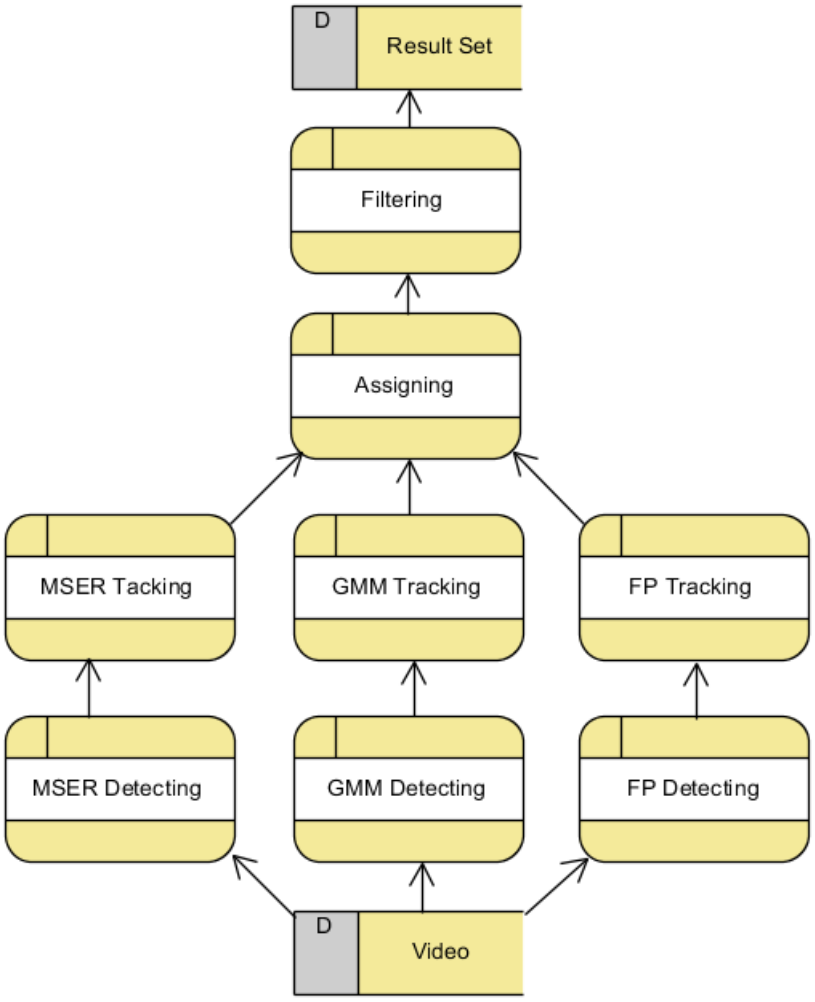
Bár ez előző két detektáló kiválasztásával már elvben tudnánk kezelni mind a lassú, mind a gyors járműveket, mégis érdemes tovább keresnünk, hogy a lehető legtöbb információt nyerjük ki a videóból. A fenti detektálók egy fontos területet még nem fedtek le, ez pedig a lassan mozgó, vagy megálló objektumok esetében jól működő 2 dimenziós kimenetet létrehozó detektáló, ilyenkor ugyanis a háttér eltávolítók kimenete haszontalan.

A megoldást egy olyan tulajdonság figyelése szolgáltatja, amelyet a háttér eltávolítók teljesen figyelmen kívül hagynak, ez a képen található körvonalak figyelése. Az alakok bár 2 dimenziósak, mégis közelebbi rokonságban állnak a sarokpontokkal, hisz magát a képet elemezik, és azon próbálnak újra megtalálható elemeket keresni, nem pedig a képkockák közti változásokat figyelik. Ennek megfelelően akkor hatékonyak, ha a követendő objektum alig mozdul, vagy mozdulatlan, hisz az különböző képkockák alakzatai között ilyenkor van a legnagyobb hasonlóság.

A projektben való használatra az MSER detectort (Maximally Stable Extremal Region) választottuk, mely olyan alakzatokat keres a képen, amely színe nagymértékben eltért a környezete színétől (a határai különböző szinteken történő binarizálás során se változnak sokat). Az általa talált alakok nevükhöz méltóan stabilak, a megvilágítás változása, vagy más zaj esetén nehezen torzulnak, ezért nagy eséllyel újra megtalálhatóak más képkockákon is[14]. Az általunk használt konkrét implementációja a MSCR (Maximaly Stable Colored Region), mely az eredeti algoritmussal ellentétben nem csak monokróm képeket képes feldolgozni. Több információból pedig, igaz nagyobb futásidő árán, de pontosabb detekciókat tud szolgáltatni[16]. „Szerencsére” az OpenCV ezen algoritmust is készen szolgáltatja.

3 A rendszer részei

Azáltal, hogy nem akartunk az információforrások között választani, nem kellett fájdalmas döntést hoznunk, hogy milyen körülmények között működünk rossz hatásfokkal, a problémákat egyszerűen kikerültük a nehézkes megoldásuk helyett. Fontos megérteni azonban, hogy mint minden tervezői döntés, ez sem csak előnyökkel jár, hisz a rendszerünknek képesnek kell az eltérő típusú detekciókat együtt kezelni, egymással összehasonlítani, közöttük konfliktust feloldani. Ez pedig nem triviális feladat, hisz teljesen más típusú adatot tartalmaznak, pl. egy sarokpontnak értelmetlen az alakjával való bármilyen számítás. A megoldást egy olyan architektúra jelenti, melyben külön válik a különböző képkockákról származó detekciók párosítása, illetve a párok lánccá való összefűzése.



3.3.1. ábra. A rendszer egyszerűsített működése adatfolyam diagramon.

A fenti diagram nem teljes, célja az, hogy a videóból származó információ útját le lehessen vele követni a videótól a kész eredményig, így nem szerepel rajta többek között a konfigurációs fájlok olvasása, UI kezelés, a verifikáció.

A diagramon látható, hogy 4 különböző szinten történik a videó feldolgozása. A legalsó szinten csak az OpenCV-ből származó algoritmusok konfigurálása, ütemezése, bemeneti adattal ellátása, illetve a kimeneti adatok zajoktól való szűrése történik. Mivel a munka nagy részét a detektáló algoritmusok végzik, az őket burkoló osztályokat is detectoroknak, a szűrt adatokból előálló eredményt pedig detekciónak nevezzük. Minden detektáló más-más adatokat jegyez fel, így a 3 féle detektálóhoz 3 detekció típus tartozik. Ezen a szinten az egyetlen cél a lehető legtöbb információ kinyerése, ezért nem történik semmilyen más művelet. Ezt az elvet követve a szint független az időtől, nem történik semmilyen képkockák közti párosítás, valójában az aktuális képkocka sorszámát sem lehet megmondani. A detektálók egymás létezéséről sem tudnak, teljesen függetlenül cserélhetőek, vagy ki-be kapcsolhatóak.

A második szinten történik az egymást követő képről származó detekciók párosítása. Ezek a párosítások alapján tudjuk követni a kép konkrét elemeinek elmozdulását, ezért az ezen a szinten dolgozó algoritmusokat trackereknek, vagy követőknek nevezzük. További célja ennek a szintnek, hogy az előálló detekció párok már egységesek legyenek, és ezáltal a különböző detektálók eredményei összefűzhetőek. A detekció párokban már csak olyan adatok hozzáférhetőek, amelyeket minden detekciópárból leszűrhetőek, pl. a párosítás valószínűsége, az elmozdulás mértéke. A párok mindig tartalmaznak egy detekciót a jelenlegi képkockához, de nem feltétlen tartalmaznak az előzőhöz. Ennek az az oka, hogy azokat a detekciókat is tovább kell tudnunk adni, amelyeknek nem találtunk párt (pl. egy új autó felbukkanásakor).

A harmadik szinten történik az egy autóhoz tartozó információcsomagok képkockánként való összegyűjtése, illetve az esetleges inkonzisztenciák feloldása. Ezen csomagok egymás után fűzve (innen a feldolgozóegység neve, Assigner) hozzák létre a járműveknek észlelésének láncolatait. Az általunk fejlesztett rendszer nem mohón követi a legvalószínűbb esetet, hanem ha az egymással inkonzisztens véleményekből több is valószínűnek tűnik, akkor mindkét esetet továbbépíti, és csak később a teljes lánc ismeretében hozza meg a döntést. A koncepció következtében a már említett láncok valójában elágazhatnak, de időrendben történő építés miatt kör nem alakulhat ki,

így tekinthetőek fa gráfoknak. Bizonytalanságuk miatt a fákból kivágható láncokat hipotézisnek neveztük el, a láncszemeket pedig a gráfokra történő asszociáció miatt node, vagy csomópont néven említem majd a későbbiekben.

A hipotézisek lusta kiértékelése miatt egy-egy pillanatnyi zavar esetén sem választunk a több közepesen valószínű eset közül, hanem mindegyiket párhuzamosan továbbépítjük. Ezzel a módszerrel sokkal kisebb az esélye a helyes hipotézisek elvesztésének, cserébe viszont hamis pozitívok tömegeit kell kiszűrni. Rendszerint a legyártott hipotézisek száma az eredményhalmaz méretének a többszöröse. A rendszer sikere tehát nagy részben azon múlik, hogy az utólagos szűrés mennyire pontos. Ezt a munkát végzi el a hipotézis szűrő, amely után a beigazolódottnak tekintett hipotézisek bekerülnek az eredményhalmazba.

Ahhoz hogy a későbbiekben kényelmesen tudjuk kezelni a hipotézis láncokat, a láncok végéhez egy az egész láncra vonatkozó információkat tároló osztályt csatolunk. Ezt a feladatot a hipotézis osztály tölti be. Itt tárolódik többek között a hipotézis típusa, vagy az, hogy a jármű által bejárt útvonal alapján a kereszteződésen milyen irányban halat át a jármű. Amikor tehát a hipotézisekről írok, egyszerre értem alatta a láncokat, és az ezekhez a láncokhoz hozzátartozó hipotézis osztályt is.

3.1 Kiegészítő rendszerkomponensek

A rendszerhez logikailag csak lazán kötődik néhány olyan komponens, amelyek azonban az eredményünket nagymértékben javítják. A rendszerben igyekeztünk SI mértékegységekkel számolni, amihez a másodpercben eltelt időt az fps-ből, a pozíciókat egy lyukkamera modellből nyertük ki. A modellt a videót felvevő kamera lencséjének katalógusadataiból, illetve a magasságából, és a kamera irányultságából építettük fel, melyek közül az utóbbi két értéket minden videóra külön kell meghatározni. Ezen modell lehetőséget ad térbeli pozíciók, méretek, távolságok becslésére.

Ahhoz hogy pontos becslést tudjunk készíteni a járművek útvonaláról, sebességéről, vagy méretéről, egy Kalman filterrel[15] simítjuk azok értékét. A filter lehetőséget ad előrebecslésre is, így pl. meg tudjuk becsülni, hogy a detekcióknak hol keressük a párját a következő képen.

3.2 GMM Detector

A folyamatosan változó környezet miatt a MOG2 algoritmust rövid, 30 másodperces időablakkal használom (mely a videó fps-éből számolódik).

A VHS felvevőknél szokásos problémát jelent, hogy a képen egy alsó és felső sávban nagyon zajos a kép a kazetták gyűrődése miatt. Ezért, és mert némelyik kamera mindenképpen beleégeti a képbe a felvétel idejét, szükséges a folyamatosan változó régiók szűrése. Ezért ha egy pixel folyamatos mozgást jelez, akkor azt lemaszkoljuk a háttér eltávolító kimenetén. Természetesen előfordulhat az is, hogy a szükséges részeket vágunk így ki a képből, de ez a szűrés csak nagyon kevés részt távolít el, amelyet a későbbi morfológiai műveletek ki tudnak javítani. A célja az, hogy egy, az érzékelés határán lévő folt, ne egy folyton változó zóna miatt kerüljön be a találatok közé.

A szürkés árnyalatú járművek, árnyékok és kocsik szétválogatása miatt, az átlagosnál agresszívabb árnyékszűrést állítottam be, így az árnyékok nagy részét ki tudja az algoritmus szűrni, annak árán, hogy a járművek némely darabja is árnyéknak van osztályozva. Ennek korrigálására nem csak a szokásos nyitás, aztán zárás műveleteket végzem el a foltok kivonatolása előtt, hanem az előtér részeket az árnyékokon belül is zárom, így a hibásan árnyéknak minősített részek előtér részekké válhatnak, ha mindkét oldalukon előtér részek állnak. Ezzel nem kerülnek vissza tényleges árnyékok a képre, mert az árnyékok a jármű mellett vannak, így nem áll mindkét oldalukon előtér.

Fontos megemlíteni, hogy a képből kivont előtér foltokból csak akkor készül detekció, ha egy mérettartományon belül tartózkodik nem csak pixelben, hanem négyzetméterben mérve (ez utóbbi értéket a folt súlypontjának pixel/méter értékből számoltuk ki a kameramodell segítségével).

3.3 FP Detector

A sarok pontok csak minimális konfigurációt igényeltek, beállításaink a GFTT algoritmushoz alapbeállításhoz közeliek. Érdemes kiemelni, hogy a rendszer fejlesztése során végig arra törekedtünk, hogy lehetőleg magasan tartsuk a megtalált sarokpontok számát, akár annak ellenére is, hogy így rosszabb találatok bekerülhetnek. Ennek oka, hogy a későbbiekben tárgyalt Peakassigner (amely az Assigner-t megvalósító algoritmus), jól tűri a hibás párosításokat, amíg azok valószínűsége alacsony. A GFTT algoritmus monokróm képet vár, így ezt is itt állítottuk elő.

3.4 MSER Detector

A stabil alakzatokat gyakran szokás közelíteni ellipszisekkel, amelyekkel ugyan jelentős sebességnövekedés érhető el, azonban információt veszítünk [16]. Mivel a rendszer célja a legnagyobb pontosság, ezeket nem használtuk (nem is volt rá szükség, hisz a teszt videókat a felvétel hosszánál kevesebb időt alatt dolgoztuk fel).

Bár a detektálók egymástól függetlenül működnek, az MSER detector nem a teljes képen futtatom, hanem csak azoknak a helyeknek a környékén, ahova járművet várunk. A járművek mozgásából ugyanis megbecsülhető a következő pozíciójuk (melyet mi egy Kalman filterrel teszünk). Ha a detektáló a kép ettől távoli részein is fut, az a tapasztalatim szerint csak zajt hoz a rendszerbe. Ahhoz hogy ezzel a visszacsatolás a lehető legkevesebb hibát vigye a detektáló bemenetére, a maszkban van némi ráhagyás, így lényeges részeket nem takar el. Az MSCR által szolgáltatott alakok meg fogják találni a maszk határát is, így egy egyfajta buborékot fognak rajzolni a jármű köré, ez azonban szűrhető, mert a járműnek a legtöbbször kontrasztosak a határai. Az általam használt kritérium egyszerű, csak azt vizsgálom, hogy a régió határos a maszk szélével. Ahogy a lenti képen is látható, még egy rossz minőségű elmosódott képen is kirajzolódnak a jármű részei (a kép az eredeti minőségben lett kivágva egyik tesztvideóból).

3.2. ábra. Az eredeti kép, az MSCR kimenete, és az ebből kiválasztott alakzatok, sorrendben. A világosabb foltok több átfedő alakzatot jelölnek (az MSCR által szolgáltatott alakzatok hierarchikusan tartalmazhatnak további alakzatokat).

3.5 GMM Tracker

A GMM típusú detekciók összehasonlításának több lehetséges módja van, annak ellenére is, hogy az alakját nem szerettem volna felhasználni, mert az a morfológiai

operátorok miatt (szándékosan) torzított. Bár a párok képzését gyakran párosítás néven említem, itt egy több-több típusú hozzárendelésről van szó. A módszerekben az is közös, hogy a friss detekciókhoz már meglévő párokat mindig a már meglévő hipotézisek legfrissebb csomópontjaiból keresek, így minden létrejövő pár egy folytatási lehetőséget jelöl.

A leggyakrabban használt módszer a foltok párosítására a legnagyobb átfedés keresése. Én ezt annyiban finomítottam, hogy a keresés előtt eltolom az előző képkockán megtalált foltot oda, ahova a már említett Kálmán filter szerint kerülnie kellene, illetve a méretét is arányosan változtatom. A párosítás valószínűségét pedig a két terület metszetének és uniójának aránya adja (ami értelemszerűen egy 0 és 1 közé esőszám lesz). Előfordulhat, hogy ugyanabból a két detekcióból több pár is létrejön, mert egy detekció több hasonló hipotézisben is szerepelhet, amik eltérő helyre becsülik a pozíciójukat. Ilyenkor a valószínűségek maximumát vesszük, mert a rendszer a kevésbé jól sikerült detekciókat is továbbépíti, így számolnunk kell a rosszabbul sikerült becslések jelenlétével.

Működőképes az a módszer is, ha nem vizsgálunk a foltokkal átfedést, hanem csak az aktuális folt pozícióját, és az előző képről származó folt becsült helyét hasonlítjuk össze (pl. a foltok középpontját). Ilyenkor meg kell határozni egy maximum távolságot, amely fölött a párosítás valószínűsége 0, innen pedig (nem feltétlen) egyenesen arányosan nő egészen 1-ig, ha a két pozíció ugyanoda esik. Ez a számítás pixelben és valós mértékegységben is elvégezhető. Bár ezzel a módszerrel több párosítást lehet elérni, mint az átfedést vizsgálóval (hisz amelyik két folt átfedésben van az közel is), mégsem használom, mert a pozícióbecslésünk annyira pontos, hogy az átfedést vizsgáló módszerrel gyakran 90% fölötti átfedést érünk el. Ha egy folt a keresett hely mellett találunk meg, az általában más foltnak a párja, így az egyébként is kis esélyű párosítás a legtöbbször hibás is lenne.

Érdemes még szót ejteni a színhisztogramokkal való összehasonlításról. A színhisztogramok használata kézenfekvő, és mivel készítését és összehasonlítását elvégzi helyettünk az OpenCV, egyszerű is. Viszont tudván hogy a járművek elsöprő többsége a szürke valamilyen árnyalatában fog látszani[10], már kevésbé ételképes elképzelés. A használatát az is rontja, hogy az aszfalt is szürke, illetve a morfológiai operátorok és a videokazetta rossz színminősége miatt minden hisztogram szürke, csak kis különbségek fedezhetőek fel. Ráadásul az összetartozás valószínűsége is nehezen

indokolható matematikailag, helyette egy önkényes határt kellene húzni, ami videónként különbözne.

3.6 FP Tracker

A sarokpontok párosítására az OpenCV készen szállít több megoldást is[17]. A FLANN (Fast Approximate Nearest Neighbor) hatékonyabb nagy adathalmazokra, cserébe csak az optimálishoz eredmény ad, mert csak becsüli a legközelebbi pontokat. A futását úgy is gyorsítja, hogy egy K-dimenziós fát épít (ez egyfajta bináris fa, amely minden elágazásában ketté osztja a teret), és abban keres. Így alacsonyabb lépés számmal rendelkezik, mint az egyszerű négyzetes algoritmus, a BF matcher (Brute Force), amely minden lehetséges pontpárt megvizsgál. A bináris fa építése azonban időbe telik, ráadásul az ehhez felhasznált pointereken való lépkedést a processzorok gyorsítótára nem támogatja, ezért a 10^3 nagyságrendben generálódó sarokpontjainkhoz a BF matcher még sebesség szempontjából is előnyösebb. Természetesen ez élesebb, nagyobb felbontású videóknál változhat, de mivel mindkét implementáció sok sarok pont típust támogat, könnyedén cserélhetőek.

Sarokpontokat az egész képen kerestünk, ezért általában a többségük nem járművekhez tartozik. Ahhoz hogy pontosabb legyen a párosítás, nem csak a hipotézisekhez tartozó sarokpontokkal vetjük őket össze, hanem egy tárolóban gyűjtjük az elmúlt néhány képkocka felesleges sarokpontjait. Így az algoritmus párokat talál köztük is, ahelyett, hogy létrehozna kis esélyű és hibás párokat is.

3.7 MSER Tracker

Az alakzatok párosítására a legáltalánosabb módja a Hausdorf távolság[18], ráadásul az OpenCV részét is képezi[19]. Ez a távolságmérték a két alakzat minden pontjából megkeresi a másik alakzat legközelebbi pontját, és az így keletkező távolságok közül a legnagyobbat választja. A távolságmérték függ az alakok pozíciójától, aminek a megválasztása nem egyértelmű, ezért ehelyett két másik távolságmértékkel is kísérleteztem.

Az én alakzatpárosító algoritmusom a Hausdorf távolság egyfajta területekkel való továbbgondolása. Úgy helyezem el a két alakzatot, hogy a súlypontjuk egybe essen. Ezután kizáró vagy segítségével veszem a különbségeket, majd azt számolom meg, hogy ezeket a területeket hány lépésben kell erodálnom ahhoz, hogy semmi se

maradjon. A hasonlóságuk azt jelenti, hogy a lépések mekkora arányát spórolom meg így ahhoz képest, mint ha az alakzatok unióját akarnám eltüntetni. Látható, hogy ez a szám is 0 és 1 közé esik, mint a többi esetben, és felfogható annak valószínűségeként, hogy az alakzatok ugyanannak az objektumnak a különböző képkockán való leképezései. Ez a távolságmérték azt adja, hogy a különbségek területeinek a pontjai legfeljebb milyen távolságra vannak az alakzat szélétől. Az új távolságmértékkel nincs pozícionálási probléma, hiszen belátható, hogy ha elkezdem távolítani a súlypontjaikat, akkor szinte mindig csökken a hasonlóságuk (szándékosan gyártható olyan eset is, ahol növekszik, pl. villáskulcsra emlékeztető alakzatokkal). Ráadásul az ember hasonlóságról alkotott fogalmához is közelebb van, pl. a Hausdorf távolsággal nézve egy cseresznye alakja drasztikusan megváltozik, ha a szárát letépjük, míg az új távolsággal nézve hasonló marad. Látni kell azonban, hogy az erodálási lépések nagyon lassúvá teszik az algoritmust nagyméretű alakzatok esetén, így nagyobb felbontású videókhoz csak nehezen lenne használható. Az algoritmus sebessége azonban a későbbiekben optimalizálható azáltal, ha az erodálási lépések helyett a legnagyobb belerajzolható kört keressük meg, amire már létezik hatékony algoritmus[20].

3.8 Assigner

A trackerek által szolgáltatott eredményben közös tulajdonság, hogy mindegyiknél mérhető a valószínűsége, és maguk a detekciók pozíciója. Ezekre az adatokra támaszkodva kell tehát csoporttokba fogni a párokat, amikből kiolvasva az későbbi detekciókat megalkothatóak a hipotézisláncok új node-jai. Az hipotézisláncok építését 3 fő részre lehet bontani, az új láncok létrehozására, a már meglévőek tovább építésére, és a bezárásukra.

3.8.1 Új hipotézisláncok létrehozása és bezárása

Azokból a GMM foltokból (pontosabban az azt tartalmazó párokból), amelyeknek nincs párja, automatikusan hipotézis generálódik, ugyanis ez egy erős jelzés arra vonatkozóan, hogy valami új dolog tűnt fel a képen. Az ilyen hipotézisekhez hozzáadjuk az összes olyan sarok pontot is, ami rajta található (az alakzatoknál nem számíthatunk erre).

A hipotéziseket akkor zárom be (akkor kerül ki az továbbépíthető hipotézisek közül), ha már sokadik képkockán nem található pár a detekcióihoz, így nincs mivel folytatni. Ez azt jelzi ugyanis, hogy az autó már kiért a képből.

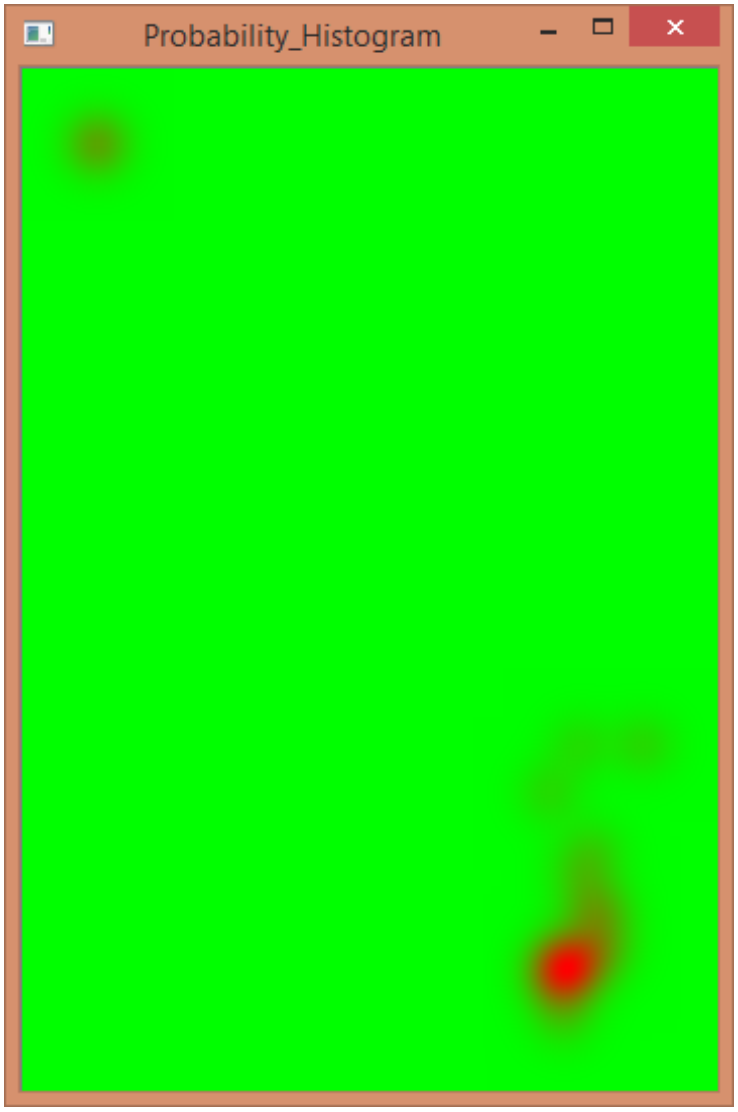
3.8.2 A meglévő láncok folytatása – a PeakAssigner algoritmus

A láncok folyatásához első lépésben összegyűjtöm az összes olyan párt, amely információval szolgálhat az aktuálisan folytatni kívánt hipotézis következő csomópontjához. Ez onnan állapítható meg, hogy az előző képkockáról származó detekciója hozzá van rendelve az éppen folytatni kívánt hipotézishez (minden hipotézist, egymástól függetlenül próbál az algoritmus továbbépíteni a meglévő adathalmazra támaszkodva).

Második lépésben a kiválasztott párok elmozdulásait egy felülnézeti térképen jelölöm (bár teljesen vízszintes kereszteződés nincs, a kocsik függőleges elmozdulása elhanyagolható). A jelölés egy szám, mely az elmozdulás valószínűségének és a pár típusához tartozó súlyozásnak a szorzatából áll elő. Amennyiben néhány pár elmozdulása is megegyezik, a hozzájuk tartozó súlyozott valószínűségek összegét jelöljük az ábrán. A térkép közepén található az origó, mely a helyben maradást jelöli.

Az elmozdulásokat világkoordinátában jelölöm, így a felülnézeti térképet megfelelő nagyításban ténylegesen fel lehetne rajzolni az úttestre. A rajta jelölt számokat pedig ezután különböző erősségű bizonyítékoknak tekintem.

Miután minden pár információja felkerült a térképre, elvégzek rajta egy simítást Gauss szűrővel, ez ugyanis matematikailag megfelel annak, hogy az párokhoz tartozó elmozdulásokat normál eloszlásúnak feltételezem, és a helyhez tartozó sűrűségfüggvényeket jelenítem meg. Az így elkészülő térképen jól látszódik minden egyes elmozduláshoz tartozó valószínűség, az egymás közelében található tippek (hasonló pár elmozdulások) valószínűsége pedig aggregálódik. A lenti képen egy látványosabb esetet figyelhetünk meg, ahol sok tipp érkezik egy pozícióra (ami a jobb alsó sarokban látszódik), de érkezik egy ettől jelentősen eltérő tipp is (bal felső sarok).



3.3. ábra. A jobb láthatóság érdekébe a 0 valószínűségű elmozdulásokat zöld, a nagyobb valószínűségű elmozduláshoz tartozó pontokat pirosabb színnel jelöltem.

A célom az, hogy a legnagyobb valószínűségű független csúcsokat válasszam ki (innen az algoritmus neve PeakAssigner). Ehhez nem kell mást tennem, csak egy hegymászó keresést indítanom a térkép minden pontjából. Ahhoz hogy ne találjanak meg a hegymászók minden szomszédos csúcsot, a keresésben nem csak a szomszédos helyek közül választanak, hanem bizonyos körön belül (mintha a hegymászók egy bizonyos távolságig ellátva keresnék a legmagasabb pontot). Mivel a térkép SI mértékegységben rajzolódott, könnyedén megszabhatom, hogy mekkora szórást engedélyezek az autó elmozdulására kapott bizonyítékokban.

Érdemes megemlíteni, hogy a tényleges implementációban nem indítok a térkép minden pontjából helymászót csak olyan távolságonként, amennyit képesek ellátni, hisz így is leellenőrzik az összes pontot. Amennyiben pedig a simításhoz használt Gauss

kernel sugara nem kisebb a hegymászók látótávolságától (amire szándékosan figyeltem a beállításakor), akkor a 0 értékű pontokban felesleges hegymászót létrehozni, mert még ha látótávolságban is van hozzá egy hely, azon biztosan van már egy másik. Valójában az utóbbi optimalizálás mindenképp megtehető, ha a simítás ablakának sugara és a hegymászók látótávolsága közül a kisebbel egyenlő távolságra teszek le hegymászókat. A két optimalizálás segítségével az algoritmus 4 nagyságrenddel gyorsul, amivel nem csak használható sebességű lett, de kifejezetten gyors is (igaz, ez jelentősen függ, a térkép méretétől, a valószínűség-hegyek által lefedett terület méretétől, és a hegymászók látótávolságától, de mindenképp drámai).

A nagy valószínűségű pontok kiválasztásával egy csapásra megoldódik a hipotézis elágaztatás, illetve az inkonzisztencia kérdése is, mert minden detekció a hozzá legközelebbi ponthoz tartozik a legnagyobb valószínűséggel (és minden detekció tartozik egy csúcshoz, hisz indult róla hegymászó). Az új csomópont pozíciójának kiszámolása is adott, hisz a csúcshoz tartozik egy elmozdulás. Azt kell tehát csak eldöntenünk, hogy melyik csúcsokat tekintünk elegendően nagy valószínűségűnek ahhoz, hogy arra hipotézist építsünk (különben minden hibásan összekötött és emiatt kis valószínűségű csúcs elágaztatna egy új, haszontalan hipotézist). Csak akkor szeretnénk elágaztatni egy hipotézist, ha mindkét lehetőség valószínűnek tűnik. Ezt úgy a legegyszerűbb megfogalmazni, hogy a legmagasabb csúcs magasságának egy bizonyos százalékáig hajlandó az assigner további csomópontokat létrehozni. Az elágazás úgy jön létre, hogy egyszerűen az összes új csomópont a saját őseként hivatkozik az előzőre (pointerekkel).

A gyengébb csúcsok eldobása egy erős, folyamatos szűrést valósít meg, így a node-ok elkészülte után bátran hozzáfűzhető minden új detekció, ahogyan azt a hipotézisláncok esetében is tettem. A hibásan rákötött detekciók ugyan generálnak párokat, de egyrészt ezek aránya alacsony, másrészt ezek elmozdulása feltételezhetően egyenletes eloszlást követ, így csak nagyon kis esélye van annak, hogy valós alternatívát állítsanak az egy irányba mutató helyes párok sokaságának.

3.4. ábra. A PeakAssigner működés közben. A kép jobboldalán látható, hogy a tábla takarása miatt két külön hipotézisre (zöld téglalapok) esik szét a kereszteződést épp elhagyó jármű.

3.9 A valószínűtlen hipotézisek szűrése és az útvonal meghatározása

Miután a hipotézisekhez már sok képkocka óta nem érkezett új node, kikerülnek az építendők közül. Ezeknek a hipotéziseknek már nem fog bővülni a lánca így elvégezhető rajtuk a teljes láncra vonatkozó ellenőrzések.

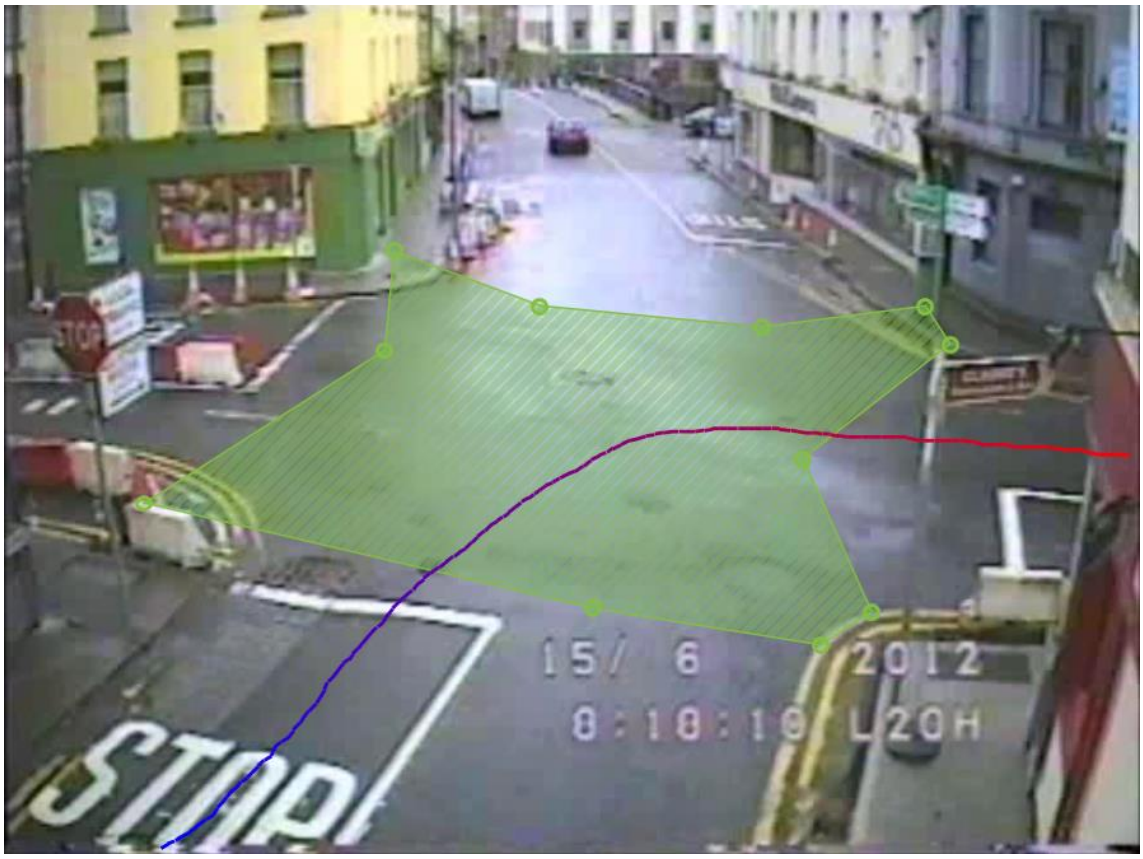
A valószínűtlen, vagy más néven életképtelen hipotézisek az alapvető fizikai paramétereik alapján kapják meg ezt a minősítést, pl. túl nagy sebesség, túl nagy gyorsulás, túl kevés időt töltött a képernyőn (ami félbehagyott hipotézisre utal) stb. Ezek ellenőrzése egyszerű feladat a kamera modell birtokában.

Meg kell viszont határozni azt is, hogy a kereszteződésen milyen irányban haladt át a jármű. Ahhoz, hogy ezt megtehessük, ismernünk kell a kereszteződésen átvezető valid útvonalakat, amelyet egy erre kialakított felhasználó felületen adhat meg a felhasználó.

Görbék összehasonlítására a legáltalánosabb módszer a Fréchet távolság[21]. Ez annak a póráznak a minimum hosszát méri, amely ahhoz kell, hogy egy kutya, és a gazdája végig tudjon a két görbén sétálni. Számunkra a diszkrét Frechet távolság fontos,

ami annyiban tér el, hogy a görbéknek csak megadott pontpárjaira kell az állításnak teljesülnie (így ugyan csak közelítő eredményt ad, cserébe nagyon gyors és egyszerű ellenőrizni).

A Fréchet távolság ugyan kiinduló alapnak jó, azonban nem pontosan erre van szükségünk. Egyrészt a felhasználók várhatóan sietve, kissé pontatlanul fogják berajzolni a minta útvonalakat, másrészt a járművek sem mindig a sávjukban közlekednek. Rendszeresen előfordulnak előzések, kikerülések, ami miatt a teljes sávjukat elhagyják, ezért nem elég a görbék legnagyobb eltávolodását büntetni.



3.5. ábra. A kereszteződés belseje (zöld terület), illetve egy lentről jobbra kanyarodó járműhez tartozó minta útvonal.

Ahogy a képen látszik az útvonal meghatározás algoritmusának két fő információra van szüksége. Az egyik a minta útvonalak csoportja, a másik a kereszteződés közepe, melyet olyan módon kell berajzolni, hogy a járműveknek mindenképen át kelljen rajta haladniuk. Az útvonal választó algoritmusnak szüksége van arra, hogy a kereszteződés belső területén kívül tűnjön fel az autó, illetve ott is hagyja el a képet, ezért a rosszul belátható utcáknál ezt akár benyúlásokkal is érdemes biztosítani (ahogy a kép jobb oldalán látszik).

Az algoritmus első lépésében a hipotézis minden node-jának pozíciójához megkeresem a minta útvonal legközelebbi pontját, az így előálló párokból két féle információt nyerek ki.

Az egyik az, hogy melyik az első és utolsó olyan láncszeme a hipotézisnek, amelyik már nem a kereszteződés belsejében van. Amennyiben nem lóg ki a kereszteződésből a hipotézis mindkét vége, akkor az életképtelennek tekintendő, viszont ha sikeresen megtaláltuk a két pontot, akkor onnantól kezdve átlagolhatom a kezdő és végpontjainak helyét. A létrejövő belépőoldali és kilépő oldali átlaghelyeket összevetve, a minta útvonalak két végén található helyekkel jól megkülönböztethetőek a különböző útirányok. Ennek ellenére a képzett pozíciópárokból a másik kinyerhető információra is szükségem van a végleges döntéshez, mert az első információval nehéz megkülönböztetni az éppen egymással szemben haladó útvonalakat.

A másik információ a párok inverziószáma, vagyis hogy a vizsgált autó pozícióinak párjai mennyire vannak szintén időrendi sorrendben. Ez az információ éppen a szemben haladó irányokat különbözteti meg a legélesebben. Önmagában ugyan ez sem lenne alkalmas a döntésre, mert pl. két egymásra merőleges egyenes útvonal inverziószáma 0 lenne, vagyis ugyanolyannak látná. „Szerencsére” azonban, a távolságalapú döntés épp az ilyen helyzetekben a legerősebb. A két döntés átlagát stabil mérőszámot kapok arra, hogy melyik két útvonal mennyire hasonlít.

3.6. ábra. Látható, hogy bár nem teljesen illeszkedik a két görbe, de a nem is távolodik el egymástól túlságosan, és a párosításban sincsenek inverziók.

Innentől csak annyi a feladat, hogy minden hipotézisre ki kell választani a legnagyobb egyezést, figyelve arra, hogy ez lényegesen jobb legyen a 2. legjobbtól.

Ennek az ellenőrzésnek az az oka, hogy belvárosban is készült tesztvideó, így előfordulnak az úttesten átsétáló gyalogosok, akiket nem tudunk méret alapján kiszűrni, mert akkor a biciklistákat is elveszítenénk.

3.10 A hipotézis szűrő

A hipotézis szűrőnek két fontos feladata van. Az egyik a végső döntés meghozatala, vagyis hogy melyik hipotézist tekinti igazoltnak, mi kerül végül a forgalmi statisztikába. A másik hogy mérje a rendszer bizonytalanságát, ugyanis a egyenlőre a program célja az emberi számlálás felgyorsítása, nem a teljes leváltása, ezért a nagyon problémás részeknél segítséget kérhet a felhasználótól.

A bizonytalan intervallumok becslésénél az életképtelen hipotézisekre támaszkodtam, mert ezek jól jelzik, ha a rendszerben valami nem a tervnek megfelelően alakult (ez lehet külső körülmény is, pl. szabálytalan közlekedés, a szél miatt rázkódó kamera). Az emberi felhasználók kiinduló irányonként számolnak, ezért segítséget is csak úgy tudunk kérni. Emiatt az útvonallal nem rendelkező hipotézisekre, nincs szüksége az algoritmusnak, mert azok nem kerülhetnek be a forgalmi statisztikába, de az esetlegesen jelzett bizonytalan intervallum se használható kiinduló útvonal nélkül.

Fontos megemlíteni, hogy a hipotézis szűrés több annál, hogy minden fa legjobb ágát kiválasztom. Gyakran előfordul olyan eset, ahol a járművek egymás takarásából lépnek ki, így egy fából több értékes hipotézist is meg kell menteni. Ez ráadásul fordított esetben is előfordulhat, vagyis hogy a járművek egymás mögé sorolnak be, így a különböző fák egyes ágai érnek össze (ez azonban a struktúrában nincs jelölve, mert hipotézisláncok építése párhuzamos).

Azért, hogy a hipotézis szűrő algoritmusai lehető legegyszerűbbek maradjanak, külön választottam a szerkezet felépítését, az életképes hipotézisek leszámoltakra, és duplikátumokra osztását, illetve a bizonytalan zónák meghatározását.

3.10.1 Az adatszerkezet felépítése

A kapott hipotézisek egy átláthatatlan gráf erdőt alkotnak, ezért a felhasználható hipotéziseket más adatstruktúrába kell rendeznem mielőtt a szűrést elvégezhetném. A pointerekkel felépített fák helyett a csomópontok asszociatív tömbbe kerülnek. Az asszociatív tömbökön belül a csomópontok csoportosítva vannak tárolva, egy csoportba a hasonló csomópontok kerülnek.

Ahhoz, hogy össze lehessen hasonlítani csomópontokat, szükséges bevezetni a hasonlóság fogalmát. A hasonlóságot úgy definiálom, hogy egy képkockán, és egy bizonyos mértékben (mely állítható) fedi–e egymást a képük. Azért alkottam a hasonlóság definícióját ilyenre, mert az azonos pillanatképeket akartam így megtalálni, a csomópontoknak viszont sok tulajdonsága eltér, pl. a sebesség, a szülők, a becsült pozíció. Egy időben egy helyen viszont pontosan egy jármű lehet (és bár a kamera vetítési miatt takarhatják egymást, arra számítok, hogy ez nem fog rendszeresen fennállni). Fontos hogy egy ilyen csoportban életképes és életképtelen hipotézisek is lehetnek. Az asszociatív tömb segítségével hatékonyan megkereshető egy csomópontot csoportja.

A két gyorstárat hipotézisenként építem fel. Minden hipotézis láncán végighaladva megnézem, hogy az adott csomópont betehető-e az ő képkockáján valamelyik csoportba. Az, hogy az ő képkockáján keressük a csoportját, csak a gyorsabb elérést segíti. Ha nem találunk hozzá csoportot, akkor új csoportot hozunk vele létre.

3.10.2 A duplikátumok szűrése

Miután már megvan a megfelelő szerkezetünk, egyszerűen elérjük a hasonlóakat, így kezelhető problémává változik duplikátumszűrés. Azt használom ki, hogy ugyanazt a járművet követve a hipotézisek is hasonló pályát járnak be, rendszeresen nagyon hasonló, vagy akár azonos tartalmú láncszemeken keresztül.

A szerkezetben lévő hipotézisek mindegyikének végigmegyek a láncán, és a csoportjától elkérve összegyűjtöm a csoporttársainak hipotéziseit, illetve feljegyzem, hogy ezek hányszor ismétlődtek. Amelyikkel csak néhány képkocka erejéig találkozott a hipotézisünk, azzal csak kis időre takarták egymást. Az ilyen járművek valószínűleg csak elhaladtak egymás előtt, viszont amelyikkel az idő nagy részében takarás állt fenn, az jó eséllyel ugyanerről a járműről készült.

Amennyiben egy hipotézissel a csomópontok több mint fele azonos (akár melyik hipotézis csomópontjainak számához képest), akkor többet láttuk őket együtt, mint külön. Ilyenkor a kisebb esélyűt elvetem. Ha a kisebb esélyű ugyanazon az útvonalon haladt, akkor duplikátum, egyébként egy a követés átragadására tippelünk (vagyis, hogy a hipotézis egy ponttól a saját járműve helyett egy másikat követett). Az ilyen esetek

hibát jeleznek, hisz itt épp a hipotézis útvonalával van probléma, így nem kijelenthető, hogy a kereszteződés melyik bejáratára kell emberi számlálást kérnünk.

Az összes olyan hipotézis, amelyiket nem tudtam megcáfolni, vagyis a duplikátum szűrésen nem bukott el, bekerül a statisztikába. Ehhez leszámolt státuszúra állítva töröljük őket az adatszerkezetből.

3.10.3 A bizonytalan intervallumok meghatározása

A bizonytalan intervallumok meghatározása akkor sikeres, ha a hibás hipotézisek aránya az ilyen intervallumokon belül jóval több, mint azokon kívül. A feladat így paradox jellegű, hisz amennyiben pontosan be tudnánk azonosítani, hogy melyik esetekben tévedünk, egyszerűen javíthatnánk is azokat.

A feladat első lépése a hibáink kiértékelése volt. Először az tűnt fel, hogy a járműkövető rendszer érzékenysége miatt valamelyest minden mozgást látunk a képen, még azt is, amikor csak a fák ágait mozgatja a szél. Tehát ha semmilyen hipotézist sem látunk a képen, akkor soha sem kell bizonytalannak jelölnünk a videó azon részét.

Az egyszerű zavaroknál csak nagyon ritkán alakul ki tényleges mozgáspálya. Hihető mozgáspálya nélkül pedig egyébként sem dönthető el a haladási irány. Ezért a hipotézis szűrőbe beépített életképtelen hipotézisek nagy része is járműveket követ, bár hibásan (időközben elveszti azt, esetleg átugrik egy másik járműre). Így ezek kezdőirányából, és a videó képén töltött idejéből egyszerűen generálhatóak bizonytalan intervallumok.

Mégsem vagyunk kész a feladattal, mert gyakran előfordul, hogy a rosszul követett autót egy másik hipotézis jól követi (pontosan emiatt terveztük a járműkövetőt többhipotézisesre). A megoldás az életképes hipotézisek által nagyobbrészt fedett hipotézisek szűrése, a duplikátumszűréshez hasonlóan. Amiket kiszűrünk, azokat nem érdemes felhasználni, mert leszámolt hipotézisek elhibázott verziói, tekinthetjük őket kijavított hibának. A többi esetében viszont úgy látunk valami hihető útvonalon haladó dolgot, hogy nincs rá életképes hipotézisünk. Vagyis nem megállapítható mi van ott, de valami van. Érezhető az erős bizonytalanság, amit minden esetben jelez is az algoritmus (bizonytalan intervallumként).

Ezek az intervallumok között lehet átfedés, ezért később összegezni kell őket, illetve sorrendbe rendezni. Ahhoz, hogy az emberi forgalomszámlálók kényelmesen

nyújthassanak segítséget, az intervallumokra néhány tizedmásodpercet ráhagyunk az elején. A nagyon közeli intervallumok közötti szűk időtartamra pedig nem vesszük vissza a vezérlést, mert abban valószínűleg úgy sem tud elkészülni egy hipotézis, ellenben a felhasználókat a gyakori ugrás jobban fárasztaná.

3.11 A megálló járművek kezelése

A rendszer alapvető működéséhez ugyan nem tartozik szorosan hozzá, mégis fontos kiegészítés a megálló járművek kezelése, mert koncepció szerint gyakran történik számolás kereszteződésekben (ahol előfordulhat pl. elsőbbségadás). A megálló járművek esetén a legfontosabb detekció típust, az előtér foltokat veszítjük el, hisz azokat a háttér eltávolító megtanulja, ezáltal azok beolvadnak a háttérbe.

A probléma megoldását két részre bontottam. Az első lépés, hogy detektáljam a lassú, vagy megállni készülő járműveket, így felkészülten várjam, hogy a hozzá tartozó GMM detekció esetlegesen elkezd összeomlani. Szerencsére a jármű sebességét könnyen és pontosan meg tudjuk becsülni a Kalman szűrőnk segítségével.

A megoldás másik része a hibás foltok megtalálása, és cseréje. Ehhez a GMM detector és a GMM tracker közé egy új eszközre van szükségünk, melyet nevezzünk correctornak. Ez a programkód megkeresi a nagyon lassan haladó vagy álló járműveket jelző hipotéziseket, és megnézi, hogy milyen GMM detekció van azon a helyen, ahova a következő node-jának GMM detekcióját becsüljük.

Ha az itt lévő előtér foltot szinte teljesen le tudja takarni a csomópontból származó eltolt (és méretben korrigált) előtér folt, az azt jelenti, hogy a másik folt elkezdett összeomlani, vagyis érdemes lecserélni. Amennyiben egyáltalán nem találok foltot, azt feltételezem, hogy teljesen eltűnt, és létrehozok egyet. Ezt azért tehetem meg, mert annyira pontos a pozícióbecslésünk, hogy nem kell számítani arra, hogy a folt nem ott van, ahova becsültük, hanem mellette (ezért is vetettem el azt, hogy a foltok középpontjainak távolságát vizsgáljuk a GMM trackerben). Ráadásul lassú mozgás esetén még pontosabban tudunk előre becsülni, mint gyors esetén.

Fontos azonban hogy ezeket a mű detekciókat ne használjuk fel ugyanúgy, mintha mért adatok lennének, különben a rendszerbe egy pozitív visszacsatolás kerülhetne, és előfordulhatna, hogy egy tökéletesen becsült megálló folt saját magát mindig ugyanoda másolva örökké a képbe égne. Ennek megakadályozására a

PeakAssignerben 0 súllyal rajzoljuk az ilyen mű előtér foltot tartalmazó detekciópár elmozdulását a térképre. Ezért egyrészt nem szavazhat a hipotézis pozíciójáról, másrészt nem fordulhat elő olyan, hogy emiatt az egy detekció miatt építünk tovább egy hipotézisláncot. A mű előtérfoltokból új hipotézisláncot sem szabad kezdeni, ez azonban automatikusan teljesül, mert ezeknek a detekcióknak mindig lesz legalább egy párja, hiszen tökéletes egyezést talál majd a GMM tracker az őt létrehozó hipotézis GMM detekciójával. Új hipotézisek pedig csak új, pár nélküli előtér foltokból jönnek létre.

4 Konklúzió

A rendszer két fő újdonsága, hogy a hipotézisek építését, és kiértékelését különválasztjuk, illetve, hogy három különböző detektort egyszerre használunk, hasznosnak bizonyult. A rendszerrel a rosszabb minőségű, kopott VHS szalag által okozott hibákkal és konvertálási hibákkal (képkockák kimaradnak, a kép az alján és tetején „csúszik”, illetve a színek nem valósak) terhelt videókon is el tudtunk érni 90%-os statisztikai pontosságot. Vagyis az egyes útvonalakon áthaladó járművek száma átlagosan 10%-kal tér el az általunk számolttól. Amennyiben a rendszer a kért emberi segítséget megkapja (ami a videó 30-50%-át fedi) a rendszer pontossága 98-99% közé emelhető.

Nagyon jó jelnek tekinthető, hogy modern kamerával, illetve SD kártyára rögzített videó esetén emberi segítség nélkül is 97% fölötti a pontosságunk, annak ellenére, hogy a rendszer nem erre az esetre lett tervezve. Így a VHS-re rögzítő kamerák visszaszorulásával párhuzamosan folyamatosan nő a program használhatósága.

Fontos azonban megemlíteni, hogy az általunk használt videókon közepes, vagy ritka forgalom található, így az autók 30-40%-a kerül takarásba, vagy takar ki egy másik autót. Azonban nehezítő körülmény, hogy mi kereszteződésben számolunk forgalmat, így a takarás alatt a járművek irányt is változtathatnak, az elsőbbségadások következtében pedig a takart vagy a takaró jármű is lehet egyhelyben álló. A hibákat elemezve világosan látszik, hogy a takarásból adódó nehéz helyzetek időnként még a videó minőségétől is nagyobb kihívást jelentenek. Ennek megoldása további kihívást állít elénk.

Irodalomjegyzék

[1] Magyar Közút Nonprofit Zrt.: *Az országos közutak 2015. évre vonatkozó keresztmetszeti forgalma*, 2016

[2] TDC Systems: *High-Speed Traffic Weigh-in-Motion & Classification System*, http://tdcsystems.co.kr/filestore/HI-TRAC%20100%20Brochure%20(Rev.2%2001-06).pdf (2016. október)

[3] Peek Traffic Corporation: *ADR 2000 Plus – Portable Traffic Counter/Clasifier*, http://www.midsigncyprus.com/wp-content/uploads/2015/04/ADR2000-Plus.pdf (2016. október)

[4] US Department of Transportation: *Traffic Detector Handbook*, Third Edition, Volume I, 2006 május, http://www.fhwa.dot.gov/publications/research/operations/its/06108/02.cfm (2016, október)

[5] Norbert Buch, Sergio A. Velastin, James Orwell: *A Review of Computer Vision Techniques for the Analysis of Urban Traffic*, IEEE Transactions on Intelligent Transportation Systems, Volume 12 , Issue 3, 2011, pp. 920 – 939

[6] Jinhui LanEmail, Yaoliang Jiang, Guoliang Fan, Dongyang Yu, Qi Zhang: *Real-Time Automatic Obstacle Detection method for Traffic Surveillance in Urban Traffic*, 2012

[7] Yannick Benezeth, Pierre-Marc Jodoin, Bruno Emile, H´el`ene Laurent, Christophe Rosenberger: *Comparative study of background subtraction algorithms*, 2012

[8] Zoran Zivkovic: *Improved adaptive Gaussian mixture model for background subtraction*, 2004

[9] Zoran Zivkovic, Ferdinand van de Heiden: *Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction*, 2006

[10] Axalta Coating Systems: *Global Automotive 2015 Color Popularity Report*, http://www.axaltacs.com/content/dam/New%20Axalta%20Corporate%20Website/Documents/Publications/Axalta-2015-Global-Color-Popularity-Report.pdf (2016. október)

[11] Kiran Mantripragada, Flavio Celso Trigo, Flavius P. Ribas Martins, A.T. Fleury: *Vehicle Tracking Using Feature Matching and Kalman Filtering*, 2013

[12] Jianbo Shi and Carlo Tomasi: *Good features to track*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1994

[13] Khairulmuzzammil Saipullah, Nurul Atiqah Ismail, Ammar Anuar, Nurishah Sarimin: *Comparison of feature extractors for realtime object detection on android smartphone*, 2013

[14] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, L. Van Gool: *A Comparison of Affine Region Detectors*, 2006

[15] R. E. Kalman: *A new approach to linear filtering and prediction problems*, 1960

[16] Per-Erik Forssen: *Maximally Stable Colour Regions for Recognition and Matching*, 2007

[17] OpenCV: *Feature Matching*, http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html (2016. október)

[18] Normand Grégoire, Mikael Bouillot: *Hausdorff distance between convex polygons*, http://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/98/normand/main.html (2016. október)

[19] OpenCV: *HausdorffDistanceExtractor*, http://docs.opencv.org/trunk/d0/de1/classcv_1_1HausdorffDistanceExtractor.html (2016. október)

[20] Daniel Garcia-Castellanos, Umberto Lombardo: *Poles of Inaccessibility: A Calculation Algorithm for the Remotest Places on Earth*, 2007

[21] Thomas Eiter, Heikki Mannila: *Computing Discrete Fréchet Distance*, 1994