Szegedi Tudományegyetem

Informatikai Intézet

SZAKDOLGOZAT

Sári Gábor

2020

Szegedi Tudományegyetem

Informatikai Intézet

**Objektum követése videó szekvenciákon**

Szakdolgozat

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Készítette: |  | Témavezető: |  |
|  | Sári Gábor |  | Dr. Nagy Antal Sándor |  |
|  | programtervező informatikus szakos hallgató |  | egyetemi docens |  |

Szeged

2020

Feladatkiírás

A hallgató feladata egy olyan képfeldolgozó rendszer kidolgozása, amely egy videó felvételen objektumokat tud követni nagy pontossággal. A feladat során meg kell oldania a hallgatónak az előfeldolgozási és utófeldolgozási lépéseket is. A dolgozatban a hallgatónak az elkészült algoritmust ki kell értékelnie.

Tartalmi összefoglaló

* A téma megnevezése:

Objektum követése videó szekvenciákon

* A megadott feladat megfogalmazása:

A hallgató feladata egy olyan képfeldolgozó rendszer kidolgozása, amely egy videó felvételen objektumokat tud követni nagy pontossággal. A feladat során meg kell oldania a hallgatónak az előfeldolgozási és utófeldolgozási lépéseket is. A dolgozatban a hallgatónak az elkészült algoritmust ki kell értékelnie.

* A megoldási mód:

Videó felvételeken mozgás alapú objektumok szegmentálást követően, hasonlósági számítások segítségével az objektum követése és végig menedzselése a felvételen.

* Alkalmazott eszközök, módszerek:

A megvalósítás Python nyelven történt.

Felhasznált eszközök, külső könyvtárak:

* Opencv
* Numpy
* Matplotlib
* Bgslibrary
* YOLO
* Elért eredmények:

Objektumok detektálása és követése felvételeken.

* Kulcsszavak:

képfeldolgozás, detektálás, szegmentálás, követés, OpenCV

Tartalomjegyzék

[Feladatkiírás 2](#_Toc40116039)

[Tartalmi összefoglaló 3](#_Toc40116040)

[Tartalomjegyzék 4](#_Toc40116041)

[1. BEVEZETÉS 6](#_Toc40116042)

[2. Előfeldolgozás 8](#_Toc40116043)

[2.1. Beolvasás 8](#_Toc40116044)

[2.2. Előkészítés 9](#_Toc40116045)

[2.3. Háttérkivonás 10](#_Toc40116046)

[2.4. Morfológia 12](#_Toc40116047)

[2.5. Szegmentálás 13](#_Toc40116048)

[3. OBJEKTUM KÖVETÉSI MÓDSZEREK 17](#_Toc40116049)

[3.1. Objektum életút 17](#_Toc40116050)

[3.2. Osztályozás 17](#_Toc40116051)

[3.3. Módszerek 19](#_Toc40116052)

[3.4. Predikció és korrekció 19](#_Toc40116053)

[4. IMPLEMENTÁCIÓ 21](#_Toc40116054)

[4.1. Fejlesztési nyelv és felhasznált könyvtárak 21](#_Toc40116055)

[4.2. Struktúra 22](#_Toc40116056)

[4.3. Paraméterek 22](#_Toc40116057)

[4.4. Előfeldolgozás 24](#_Toc40116058)

[4.5. Szegmentálás 25](#_Toc40116059)

[4.6. Kontúrok és árnyékok 26](#_Toc40116060)

[4.7. Osztályozás 28](#_Toc40116061)

[4.8. Követési módszerek 30](#_Toc40116062)

[4.9. Összegzés 32](#_Toc40116063)

[5. Eredmények 33](#_Toc40116064)

[5.1. Teljesítmény 37](#_Toc40116065)

[5.2. Más módszerek 37](#_Toc40116066)

[6. Irodalomjegyzék 39](#_Toc40116067)

[7. Nyilatkozat 40](#_Toc40116068)

[8. Köszönetnyilvánítás 41](#_Toc40116069)

# BEVEZETÉS

Az objektumok, járművek optikai detektálása és nyomon követése egy régóta kutatott terület. Az algoritmusok folyamatosan fejlődnek, de nem létezik egy általános eszköz a feladat megoldására. Fontos a statisztikai adatok elemzése, optimalizálás, ezek alapján a helyes döntések óriási előnyt jelenthetnek, mind a szállító forgalomban, várostervezésben, mind a kritikus esetek, balesetek felderítésénél lehetőséget adva gyors és hatékony reagálásra.

Népszerű felhasználása a megjelenő új okos, önvezető autók szoftvereben, ahol a sok száz érzékelő által adott bemenet es a mesterséges intelligencia segítségével képes a jármú megállni a gyalogátkelő elött, szabad utat biztosítva a gyalogos szamara, vagy jelezni a vezetőnek, ha érzékeli, hogy a jármú elhagyja a kijelölt sávot, vagy a kihelyezett sebességkorlátozó tábláról képes leolvasni a maximális sebességet, és ezt nem átlépve biztonságosan közlekedni. (lásd 1. ábra)

Az objektumok detektálást követését általánosan két oldalról lehet megközelíteni.

A manapság igen népszerű és kimagasló teljesítményű kutatási terület, a mesterséges intelligenciával, továbbá a nyers képfeldolgozó módszerekkel.

Az mesterséges intelligenciával általánosan nagy pontosságú eredményeket lehet elérni, ugyanakkor problémát jelent a működésében, hogy az általa felépített rendszerek néhol fekete dobozként működnek, az olyan esetekben, ahol nem várt eredményt kapunk, nem pontosan lekérdezhetők, elemezhető, módosíthatók az egyes részfeladatok elemei.

A képfeldolgozó módszerek esetben a részfeladatok jobban definiálhatóak, szükség esetén elemezhető és a legutolsó paraméterig módosíthatóak.

Egyik módszernél sem elérhető tökéletes eredmény es pontosság, különösen igaz ez a képfeldolgozásra.

A mai igényeknek megfelelően, az azonnali reakció lehetőségét megteremtve, a rendszernek élő képet és azonnali feldolgozást kell biztosítania. Az iparban található rendszerek online, felhőalapú feldolgozást preferálják, ám ez aggodalmat vethet fel olyan videók felhasználásánál, amiben szenzitív adatok szerepelhetnek.

A dolgozat alapvetően a hagyományos képfeldolgozást tartalmazza, de ezek kombinálhatóik a mesterséges intelligenciával a pontosabb eredmények elérésében.

További elvárt követelmény, hogy az újabb technológiák megjelenésével, az egyes modulban szereplő algoritmusok gyors cserélhetősége, kiváltása, és a felhasználásnak megfelelően a specifikálása.

Ennek megfelelően igyekeztem különböző modulokra bontva megoldani a részproblémákat és választ találni a felmerülő kérdésekre.

A választott technológiák megfelelnek a kor követelményinek, a program könnyen olvasható, a modulok lineáris csővezetékben helyezkednek el.

A program által használt keretrendszer C++ nyelven íródott, erre építették rá a különböző programnyelvekhez szükséges külső rétegeket, interfészek. A keretrendszer széleskörben elterjedt, nagy fejlesztői közösséggel és részletes dokumentációval rendelkezik.

Ez a C++ mag lehetővé teszi a hordozhatóságát, előnye a skálázhatóság és az alacsony, rendszer közeli futás melynek eredményeképpen ideális lehet kritikus gyorsaságú rendszerek esetében is.

A Python nyelvet választottam az alkalmazás megvalósításához, hiszen nagyon sok belső és külső könyvtárat tartalmaz tudományos, képfeldolgozási munkákra, emellett gyorsan és könnyen tanulható nyelv.

A képen zöld, fű, színes, épület látható

Automatikusan generált leírás

1. ábra Okos város (NVIDIA IVA)

# Előfeldolgozás

A detektálás és követés szétválasztandó fogalom, így a folyamat során fontos az előfeldolgozás, hiszen az objektumok egyértelmű azonosításához és követéséhez elengedhetetlen a lehető legjobb nyersanyag kinyerése a képmátrixból. Az előfeldolgozási lépések határozzák meg a végeredmény minőségét, egy hipotetikusan tökéletes előfeldolgozással szinte nullára lehet csökkenteni a csővezetékben később előforduló hibák számát.

A program alapvetően előzetes terület, határvonalak specifikálása nélkül működik, felhasználói interakció nélkül, általánosságban. Automatikusan próbálja meghatározni a működéshez szükséges adatokat, ebben van nagy szerepe az előfeldolgozásnak.



2. ábra Csővezeték vázlata

## Beolvasás

Az input képek származhatnak videófájlból, mappából, ha a fájlok neve meghatározzák a beolvasás sorrendjét, vagy stream szolgáltatásból, esetleg videókamerából. A program és a dolgozat is alapvetően a megszokott RGB színtérrel rendelkező 0-255 tartományba eső értékű képmátrixokkal dolgozik. Ugyanakkor az alkalmazott keretrendszer a színcsatornákat a megszokottól eltérően BGR sorrendben tárolva olvassa be. A támogatott kép és videóformátumok listája operációs rendszerenként eltérő lehet, de a legtöbb elterjedt kép és videó formátumot támogatja, kezdve a Windows bittérképektől (BMP), az általánosan használt JPEG és PNG formátumokon át, a nagy dinamikatartományú (HDR) képekig. Ezeket részleteiben a használt keretrendszer dokumentációja tartalmazza pontosan.

A szekvencia beolvasás sebessége függ a teljesítménytől vagy az inputtól. Alapértelmezetten a lehető leggyorsabban olvassa be a képmátrixokat egymás után.

A feldolgozással minden esetben képkockánként halad, egyszerre egyet olvasva be szekvenciálisan, ezeket aztán beküldi a csővezeték alapú feldolgozó rendszerbe.

## Előkészítés

Először a képmátrixot átméretezi a parancssori paramétereknek megfelelően, arányosan százalékként, vagy a megadott méreteknek megfelelően.

A maximális méret, alapértelmezetten, 2^30 pixel/kép, a feldolgozás sebességének ez a nagy méret lehet az első buktatója, ezért érdemes minimalizálni a méretet mindaddig, ameddig a szükséges adatok kinyerhetőek.

Az általános RGB színtérben lévő képmátrixot, átkell konvertálni, ez az emberi szemnek megfelelő és esztétikusan reprezentálja az értékeket, de más színterek sokkal kevésbé érzékenyek a zajra.

A két szóba kerülő színtér így a HSV színtérből a Value csatorna, és a LAB színtérből a Light csatorna. A konvertálás általában konstans számokkal való szorzásból és összeadásból áll. [1] [2]

A program a LAB színtérré konvertálja a meglévő BGR színtérből, és az L csatornát használja a további munkához, amíg szükségessé nem válik az eredeti kép értelmezése is, ilyenkor visszatér az eredeti, de már átméretezett képhez.

A következő lépésként Gauss-simítását végeztem, hogy a jellemzően kis zajoknak, elmozdulásoknak ne legyen jelentősége a későbbiek folyamán, egy távolban lévő vagy egy kis objektum bemozdulása ne idézzen elő láncreakció szerű folyamatot, ha semmi praktikus szükségessége nincsen, nem lényeges, sem feltűnő még emberi szem számára sem. Illetve különböző további metódusoknak más és más módszerekkel kezelik a bemeneti képet, így fontos, hogy ezen lépéseket ne duplázzuk meg, ha már ők maguknak kezelik ezeket.

A simításhoz használt konvolúciós mátrix méretének aránya a képmátrixhoz képest állandó, ehhez viszonyítva néhol kisebb, nagyobb méretet használnak az algoritmusok, de ez függ az alkalmazási területtől, hogy mekkora zaj értékeket szeretnénk elsimítani.

A számítás módja a képterületének és az arány szorzatának négyzetgyöke lefelé kerekítése páratlan számra, erre azért van szükség, hogy megtartsam a kernel egyértelmű középpontos tulajdonságát. Így egy HD kép esetén 9-es méretet, míg egy kisebb 640\*480 kép esetén 5-t kapok.

## Háttérkivonás

Ahhoz, hogy megtudjuk keresni az objektumokat elkell választani a mozgó tárgyakat a háttértől.

Ennek a legegyszerűbb módja, ha az adott képmátrix és egy statikus kép különbségét számoljuk ki, a mátrixok képponti szerint abszolút különbségének a segítségével. A program 41 féle különböző algoritmust tud futtatni a háttérkivonás elvégzésre. A könyvtár szerzőjének egy másik projektjében [3] több mint 100 algoritmus van implementálva MATLAB-ban, ezek könnyen átültethetök az adott technológiába.

Ezek közül néhány [4], a szerző és a teljesítmény alapú válogatás során generált maszk egy teszt felvétel alapján:

|  |  |
| --- | --- |
| *A képen fekete, ülő, sötét, tűzijáték látható  Automatikusan generált leírás*  *MOG* [5] | *A képen étel látható  Automatikusan generált leírás*  *PixelBased Adaptive Segmenter* |

1. táblázat Különböző algoritmusok eredménye azonos képről

|  |  |
| --- | --- |
| *A képen kültéri, fénykép, nagyon nagy, szökőkút látható  Automatikusan generált leírás*  *KDE* | *A képen kültéri, síelés, fénykép, hó látható  Automatikusan generált leírás*  *Static Frame Difference* |

Ezt a választást paraméterrel lehet szabályozni, az algoritmus nevének megadásával, részletesen az implementáció fejezetben olvasható.

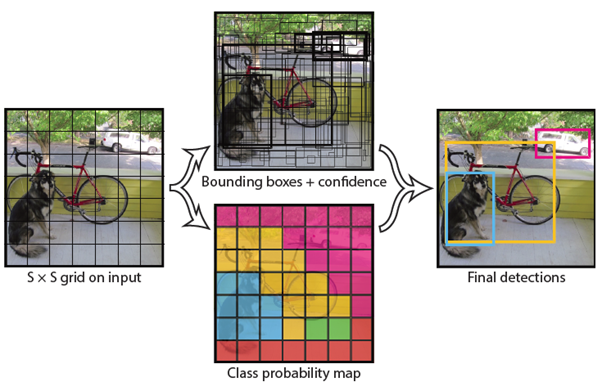
Ebben a lépésben válnak a képekből fehér-fekete maszkok, melyek a tárgyakat jelölik. Az OpenCV által biztosított algoritmusok nagy részében, a megadott napló szám befolyásolja a tanulási rátát, ha az egyébként nincsen megadva, ez által pedig az intenzitás változások súlyát. A megadott háttérarány szám, pedig azt, hogy ha ezek az intenzitás változások súlyozott összege egy határérték alatt van, akkor az terület háttérként funkcionál, ha túllépi a határértéket akkor előtérként.

Általánosságban elmondható, hogy a mozgás alapú háttérkivonásban sok tényező nehezíti a munkát.

* A modern kamerák például rendelkeznek például autófókusszal, automatikus hisztogram kiegyenlítéssel, ezek hasznos és esztétikus dolgok, de itt fontos lehet az eredeti kép megtartása.
* A háttérben lévő olyan objektumok mozgása, ami attól még háttérnek kell kezelni.
* Ha egy korábban az előtérhez tartozó objektum nem mozog, akkor az beleolvad a háttérbe, ugyanakkor, ha egy korábban a háttérhez tartozó objektum mozogni kezd akkor a mögötte, helyette keletkező háttér és új lesz, így szellemképek jöhetnek létre.
* Az eleve nem stabil ponton rögzített kamerák, állandó zajforrások lehetnek.
* A túlzott dinamikájú háttér, ami már az algoritmus nem tud hibahatáron belül tartani.

Így felmerül egy másik modern lehetőség, hogyha nem a háttérkivonásos algoritmusokat választjuk, hanem a tárgyak detektációját, egy előre elkészített neurális hálózatra bízzuk.

A YOLO [6] egy valós idejű nyílt forráskódú objektum detektáló rendszer, ami egy regressziós megközelítést használ, lényege, hogy a meghatározott befoglaló téglalapokra kiszámolja az osztályok valószínűségét, mindezt egyetlen hálón való átfutással, innen a név.



3. ábra YOLO működése [7]

A program a megadott alapértelmezett, előre betanított súlyokat használja, amit a Microsoft COCO adathalmazon tanítottak be.

Ebben az általános adathalmaz minden hétköznapi tárgy szerepel, így a specifikusan betanított modellek lényegesen jobb eredményeket érhetnek el, ehhez hozzákapcsolódik az az előny, hogy ha rendelkezünk adatbázissal és ismerjük az alkalmazás várható körülményeit, nehézségeit, akkor ezeknek a kiküszöbölésére is betaníthatjuk a rendszert.

A betanítást, és az algoritmus futtatását, nem muszáj lokálisan megtenni, léteznek Cloud GPU szolgáltatások, ahonnan kölcsönözhetünk GPU-t, így gyorsítva az általában órákig, napokig tartó folyamatot.

## Morfológia

Amennyiben nem a YOLO által végrehatott objektum detektálást választjuk, úgy a háttérkivont képből magunk kell kinyernünk a tárgyak kontúrjait.

A legtöbbször a mozgó objektumok belső területében képződnek vak foltok, ezeket könnyedén javíthatjuk a floodfill algoritmussal.

|  |  |
| --- | --- |
| A képen játék látható  Automatikusan generált leírás | A képen rajz látható  Automatikusan generált leírás |

4. ábra Floodfill algoritmus

A zajok eltüntetéséhez egyszerű morfológiai operátorokat használunk, dinamikus mérettel. Megfelelő alkalmazva, a tárgyakat kiemelve könnyítik a következő lépéseket.

|  |  |
| --- | --- |
| A képen fekete, fénykép, fehér, víz látható  Automatikusan generált leírás | A képen étel látható  Automatikusan generált leírás |

5. ábra Morfológia eredménye

|  |  |
| --- | --- |
|  |  |

## Szegmentálás

A meglévő maszkokból ki kell nyerni a tárgyak pozíciót.

Ehhez elégséges meghatározni a körül határoló téglalapot, a hierarchia megtartásával. Ha ezt a hierarchiát egy faként fogjuk fel, akkor ebből a hierarchiából mindig a csak a szülő nélkülieket kell venni, így az éppen lokális maximális méretű kontúr kerül kiválasztásra.

Ha neurális hálózatot használunk, már rendelkezésekre állnak ezek a koordináta adatok, csak a szükséges konvertálást kell elvégezni, mivel a visszaadott adatok hálózatbeli pontjukkal vannak megadva.

A csővezetékbe így is bekerülhetnek olyan adatok, zajok, és tárgyak, amiket nem szükséges objektumként kezelni, így itt szükséges lehet a terület nagysága szerinti szűrés. Ezek a beállítások is paraméterezhetőek, a felhasználásnak és az inputnak megfelelően.

|  |  |
| --- | --- |
| A képen világos látható  Automatikusan generált leírás |  |

6. ábra A detektált kontúrokból generált hőtérkép

Hogy tovább csökkentsük a lehetséges zajokat, és növeljük a praktikusságot, figyelni kell, hogy milyen releváns egy mozgás helye, milyen időben történt.

Ha nagyon régen történt az időben, akkor nem biztos, hogy szeretnénk megfigyelés alatt tartani azt a területet, ahol az esemény történt, ugyanígy, ha kevés, nem releváns mozgás történt azon a terület.

Ezt egy N\*M\*2 méretű mátrixban tárolom, ahol az első réteg tárolja gyakoriságot, a második pedig az utolsó módosítás idejét, vagyis azt a kép sorszámot, ahol utoljára történt módosítás. A maximális időtartam pedig a megadott, paraméterezhető megfigyelési ablak 1,2-szerese, a minimális előfordulási gyakoriság pedig az ebben az ablakban történt nem nulla előfordulások átlagának 0,8-szorosa. A gyakoriságok, így a megfigyelési ablak letelte után nullázódik, ebben a részben, van egy másik hasonló felépítésű tömb, ami nem törlődik, igy onnan a teljes képet kaphatunk, az összes detektált objektum maszkról.

Tipikus példa, a háttértárgyak bemozdulása a képen, ilyenkor ez az időben, régen történt, egyszeri jelzés nem releváns a kép frekventáltabb lényegi részeihez képest.

Ezekből az adatokból megadhatunk egy olyan részét a képnek, egy hőtérképet, ami érdekes a megfigyelés szempontjából, egy referenciaként szolgálhat a későbbi algoritmusokban.

Ezen kívül előfordulhat olyan eset, hogy a bár nem szorosan az objektumhoz tartozó árnyék vagy egyes részek összeérthetnek más tárgyakkal, így a két tárgy, egy nagyként kerül detektálásra, ez a hőtérkép szempontjából egyáltalán nem szignifikáns, de a további lépésekben igen.

Ilyenkor távolságtérkép segítségével redukálom a két objektum nagyságát, ilyenkor csak a középpontban maradnak releváns értékek, majd ezeket használva próbálom szétválasztani a két tárgyat. Fontos, hogy a legrosszabb esetben az eredmény akár lehet az is, hogy az egyik a tárgy maga, a másik pedig az árnyék, és ezeket válnak szét két külön egyedre, ami helytelen, ilyenkor érdemes a túl sötét tárgyakat, legyen az árnyék vagy tényleges tárgy, figyelmen kívül hagyni.

|  |  |  |  |
| --- | --- | --- | --- |
| A képen zöld, kicsi, utca, busz látható  Automatikusan generált leírás |  | A képen zöld, ülő, kicsi, torta látható  Automatikusan generált leírás |  |

7. ábra Távolság térkép segítségével szeparált tárgyak

A legtöbb képen van árnyék, és a napszak, évszak, kamera pozíciótól függően, ezek problémát okozhatnak, hiszen elvonnak a tényleges terület nagyságának arányából, ezek pedig értékes információk. Az ilyen hibák, pedig rontja az azonosíthatóságot, a pontosságot, és csökkeneti a hasznos információt, a tárgyról alkotott képben.

Az ilyen esetekben próbáljuk kiszűrni azokat az objektumról készült képrészleteket, ahol biztosan vannak árnyékok, levágni ezeket az árnyékokat, majd a tapasztalt arányokat gyűjteni és felhasználni a továbbiakban. Természetesen ezeket a tapasztalatokat folyamatosan gyűjtjük.

Létezik egy határérték, ami megszabja mennyire kell, hogy az árnyéknak kitolódnia a szélére, ez alapértelmezetten 0,75, tehát a kontúr által kiemelt képben minimum a kép utolsó 25 százalékában kell legyen az árnyék, az adott iránynak megfelelően. Például, ha az alsó részt vesszük figyelembe, akkor a képrész felső maximum 75 százalékában, nem árnyék kell, hogy szerepeljen, ha ennél nagyobb arányban szerepel, akkor a vágás nem történik meg, ezen az oldalon. Emellett van egy szorzó tag, ami befolyásolja az arányokat, így lehet szigorúbb, illetve lazább beállításokat megadni. 1-es együttható esetén pontosan tartja az arányokat, >1 esetén pedig, a számláló növelésével éri el a módosítás mértéknek csökkentését. Illetve paraméterezhetően ki lehet kapcsolni, az árnyékszűrést, és be lehet állítani irányokra.

A képen jelenet, út, kültéri, zöld látható

Automatikusan generált leírás

8. ábra Árnyékok levágása balról, alpha ≈ 0.6

9. ábra Árnyékok levágása jobbról alpha ≈ 0.7, lent alpha ≈ 0.8

A képen út, jelenet, fű, követés látható

Automatikusan generált leírás

# OBJEKTUM KÖVETÉSI MÓDSZEREK

Még maga a követési algoritmus előtt nehézséget okoz a megfigyelt tárgyak életútjának definiálása. Mikor, hogyan kerül egy alakzat be, illetve ki a nyilvántartott rendszerből. és listából. Az előző fejezetben látottak segítségével tudunk ezekről a határokról képet alkotni.

## Objektum életút

A képen térkép látható

Automatikusan generált leírás

10. ábra Objektum vázlatos életútja

A csővezetéknek köszönhetően, már egy árnyékoktól körülvágott, megfelelő alakzatot kapunk, a kontúr által vágott tárgy képét, és annak határoló koordinátáit. Ezeket az adatokat felvisszük egy adatszerkezetbe, majd listába fűzve tároljuk. Ez a lista reprezentálja a tárgyakat.

A szegmentált detektált alakzatokkal dolgozunk, ezekről az osztályozás során megtudjuk, hogy már szerepel-e a rendszerben megfigyelés alatt, vagy sem. Ha már szerepel akkor frissítjük a rá vonatkozó adatokat, ellenben bevesszük a listába.

A hőtérkép segítségével eltudjuk dönteni, hogy hol legyenek azok a határok, ameddig szeretnénk megfigyelni az objektumokat, csak ez az által közrefogott területen vesszük figyelembe a tárgyakat, ezen kívül valószínűleg nem releváns a tárgyak mozgása, ezt a hőtérkép kiépítése biztosítja.

## Osztályozás

A bejövő alakzatokról elkell dönteni, hogy újként detektáljuk-e, vagy már meglévőhöz rendeljük hozzá.

Végig iterálunk a már követés alatt lévő, megfigyelt tárgyakon egy listában, megkeressük a hozzá legközelebbit, a tárolt információk alapján. Megfelelő, állandó képfrekvencia esetén, a következő képkockán is viszonylag közel kell lennie az adott tárgynak, az előző pozícióhoz képest, esetleg az új tárgy középpontja benne van egy korábban detektált tárgy területében, körülzárja az.

Ebbe az összehasonlításba belevesszük azokat a pontokat, amik a határokon szerepelnek.

Ha olyan mozgásról van szó, ami egyirányú figyelembe kell venni, az elmozdulás irányát, tehát nem haladhat visszafele a tárgy, az előző pozícióhoz képest. Ezen kívül a detektált terület nagysága is szigorúan monoton csökken vagy nő.

Ha egy tárgy benne van egy előző képkockán detektált tárgy területében, akkor valószínűleg ugyanaz a tárgy, megfelelő képfrekvencia esetén. Amennyiben a detektált tárgy a hasznos területen kívülre esik bármely dimenziójában, automatikusan figyelmen kívül hagyjuk az eredményt.

Ezentúl még három módszer által generált más tényezőt veszek figyelembe, hogy megbizonyosodjak, hogy ténylegesen ugyanaz a tárgy, mint ami a legközelebbi.

* Hisztogram
* Minta illesztés (Template matching)
* Kulcspont párok (Keypoint and descriptors)

Hisztogram összehasonlításkor mind a három csatornáról tárolt adatokat hasonlítom össze, korreláció tekintetében vagy cosine hasonlósággal, ezek szintén opcionálisan módszerek, ezeken kívül biztosít más módszereket is a keretrendszer.

Minta illesztés alapvetése, hogy kicsi mintát próbáljunk illeszteni, de az esetek többségében az egyezést jó eredménnyel detektálja, nagy méretekben is.



11. ábra Sikeresen azonosított tárgy  
a hisztogram egyezés értéke 0.91, template matching maximuma 0.73

Az implementált programban van erre lehetőség, hogy megtudjuk miért és mi az eredménye, tehát, hogy miért került elutasításra az objektum, és került be új elemként a rendszerbe.

## Módszerek

A program 9 féle módszert biztosít, amely módszerek paraméterként beállíthatóak.

* CSRT
* KFC
* BOOSTING
* MIL
* TLD
* MEDIANFLOW
* MOSSE
* POLY
* SIMPLE

Az első 7 algoritmust az OpenCV könyvtár biztosítja [8], melyeknek inicializáláskor megkell adni a követendő tárgy képét, amit a kontúr alapján vágunk ki, majd minden további frame-et, és az algoritmus visszatér az új frame-n belüli tárgy koordinátáival, ha sikeres volt a követés, ekkor frissítjük a rendszerben a hozzá tartozó adatokat.

Vizsgáljuk azt is, hogy ha sokáig nem következik be változás a tárgy pozíciójában, vagy ha az említett módszerek sokáig sikertelen követési eredménnyel zárulnak.

Ha sikeresen azonosított, párosított egy tárgyat a bejövő detektált alakzattal, akkor ezek az új adatok is természetesen bekerülnek a tárgyról alkotott adatszerkezetbe, mint az új koordináták, a kép a tárgyról, hisztogram, ellenkező esetben pedig új tárgyként kerül be a listába.

Ezen a beépített módszereken kívül még további kettő lehetőséget biztosit a program, lásd 3.4 fejezet.

## Predikció és korrekció

Az dolgozatban implementált egyik, poly, módszer szerint, ha az adott frameből nem nyerhető ki tárgy képe, vagy egyáltalán nem vizsgáljuk az adott frame-t, csak minden *N*-ediket, akkor a meglévő adatok alapján *P*-ed fokú polinomot próbál illeszteni a pontokra, és így megbecsülni a következő y koordinátát, az előző *M* x koordináta átlagos eltérése szerinti x koordinátára. Ezek a becsült koordináták, ugyanúgy bekerülnek az tárgy útját tároló adatszerkezetbe.

A becsült koordinátákat, a tényleges, párosított, biztos találatok korrigálják, abban az esetben, ha van párosítás, akkor a predikció rész átugorjuk, nem szükséges duplán ugyanazt, vagy közel ugyanazt a pozíciót bevenni.

Ez a módszer egyenes vonalú egyenletes elmozdulások esetében működik jól.

A másik lehetőség, ha nem végzünk predikciót, és csak a párosított, azonosított tárgyakra támaszkodunk, vagyis ebben az esetben csak ezeket a frameket vesszük figyelembe. Ez természetesen jóval kevesebb számítást igényel, mert nem kell minden frame-ra predikciót végezni, de ha az eredeti képből nem tudunk információt kiszűrni, akkor az növeli a tárgyak inaktivitási számát.

Minden módszer esetében figyeljük azt, hogy ha egy tárgy adatai sokáig nem frissülnek, esetleg kívül esik a hőtérkép által határolt területen, vagy inaktív, akkor a tárgy lezárásra kerül, ami azt jelenti, hogy figyelmen kívül hagyom az összehasonlításoknál, és más listázásokkor.

A képen objektum, óra látható

Automatikusan generált leírás

12. ábra Az nyilak iránya az objektumok haladási irányát reprezentálják

A képen ülő, víz, férfi, fekete látható

Automatikusan generált leírás

13. ábra Pattogó kosárlabda követése

# IMPLEMENTÁCIÓ

A fejlesztés időpontjában is annyira gyorsan változtak az egyes modulok és könyvtárak, hogy nem feltétlen voltak kompatibilisek, például a legújabb YOLO verzió és az OpenCV, így a program nem feltétlen a legfrissebb, de kompatibilis modulokat tartalmazza. Célom volt továbbá, hogy a lehető legszükségesebb fájlokat tartalmazza csak a csomag, így, alapértelmezetten, nem kell külön más projektet lefordítani, elég a beépített Pip csomagkezelőt használni.

Az implementációban egy forrásfájlon keresztül látható a csővezeték, ez foglalja össze, és továbbítja a modulok eredményeit. A részeredmények külön megjeleníthetőek.

## Fejlesztési nyelv és felhasznált könyvtárak

Python

A szakdolgozat megvalósításának fő nyelve a Python, mely egy magas szintű, általános célú, interpretált nyelv. Guido van Rossum fejlesztette és az első kiadása 1991-ben jelent meg. A nyelv legfőképpen a gyors, rugalmas fejlesztésre és az olvashatóságra törekszik, ezeket előtérbe helyezi a futási sebességgel szemben; egyszerű szintaxisával kezdő programozók is könnyen használatba vehetik. Támogatja a funkcionális, az objektumorientált, az imperatív és a procedurális programozási paradigmákat. Rendkívül széles körben használhatóvá vált, mert a Python értelmezőt számos operációs rendszerre elkészítették (Windows, Unix, Mac, Linux) és ráadásul nagyon sok kiegészítő csomag és könyvtár tölthető le hozzá.

OpenCV

Sok száz képfeldolgozási algoritmust megvalósító és megjelenítő nyílt forráskódú elterjedt könyvtár. Python mellett más nyelvű kötéssel is rendelkezik, például Java, C# és C++.

NumPy

Főleg tudományos munkára használt, Python alapú, nyílt forráskódú matematikai csomag, erősen kapcsolódik az OpenCV-hez, a fejlett tömb kezelés miatt.

Matplotlib

Pythonban íródott adat megjelenítéshez használt eszköz.

YOLO

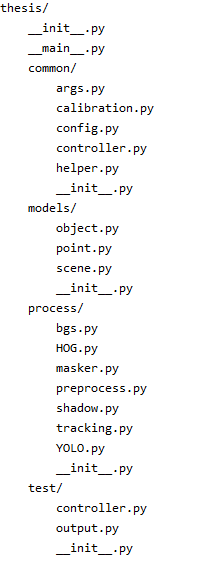
You Only Look Once, egy modern, valós idejű, objektum felismerő konvolúciós neurális hálózat, előre betanított súlyokkal.

BGSLibrary

C++ alapú előtér, háttér elkülönítő algoritmusokat tartalmazó csomag.

## Struktúra

A forráskódhoz nem telepíthető könyvtárak külön mappában helyezkednek el, elkülönítve szintén, a generált, és teszteléshez használt részeredmények, illetve a virtuális környezet mappája, melynek segítségével a projekthez lehet rendelni a telepített Python csomagokat.



14. ábra A projekt szerkezete

A BGSLibrary C++ alapú könyvtárat egyedileg, külön kellett lefordítani a megfelelő paraméterekkel, a megfelelő OpenCV verzióval.

## Paraméterek

A projektet lehet modulként és szkriptszerűen is használni.

Parancsorból py -m thesis paranccsal indítható

Szkriptként pedig importálás után a start paranccsal.

A 20 féle paraméterből szinte végtelen mennyiségű kombináció összeállítható, a pontos felhasználásnak megfelelően. A paraméterek feldolgozást külön szerveztem ki, egy jól kezelhető modul segítségével, mely szöveges leírást is támogat, a paraméterekhez.

A paraméterek és fontosabb beállítások külön tárolódnak, így gyorsan áttekinthetőek és módosíthatóak, akár az alapértelmezett értékek is.

|  |  |
| --- | --- |
| Kapcsoló | Leírás |
| in, input | Input megadása |
| i, interactive | Interaktív mód bekapcsolása, rész eredmények vizuális megjelenítése a futtatás során. |
| pp, preprocess | Előfeldolgozás bekapcsolása |
| delay | Képkockánkénti szünet érték |
| size | Átméretezési érték, százalékban, két érték esetén SZ\*M |
| kr | Kernel aránya, alapértelmezett kernel méreténél játszik szerepet |
| obs,observation | Megfigyelési ablak idő tartama, másodpercben |
| bgs | A válaszható háttérkivonási algoritmusok közül egy |
| yolo | YOLO algoritmus futtatása során használt háló, ablak mérete |
| analyse | frameN mod érték, ezen sorszámú képkockák kerülnek elemzésre |
| fst | Teljes képes követés, nem veszi figyelembe a hőtérképet |
| tracker | A válaszható követési algoritmusok közül egy. (BGSLibrary alapján, OpenCV alapján, poly, simple) |
| maxLostIndex | fps\*érték képkocka után tétlenség után az objektum életútja lezárul |
| polydeg | Poly követési mód esetén az illesztendő polinom fokszáma |
| shadow | Árnyék levágás végrehajtása |
| shadowDir | Árnyék levágási irányok megadása,  0 1 2 3, fent, jobbra, lent, balra |
| area | Minimum maximum kontúr méret |
| linear | A követési rendszerben figyelembe veszi, hogy a hasonló tárgy, nem mozdulhat visszafelé. |
| fd | Floodfill algoritmus esetén, az eltérés nagysága, negatív érték estén kikapcsol |
| o, output | Kimeneti, eredmény videó generálása |
| t, test | Részeredmények kinyerése, exportálása |

2. táblázat Paraméterek

Más és más beállításokat érdemes használni egy autó vagy egy labda, ember követésénél.

A már látott pattogó labda esetében, például

>py -m thesis -in dataset/bball03.mp4 -fst -size 30 -bgs mog -tracker simple   
-area 1000 -1 -o

Autópálya megfigyelés esetén:

>py -m thesis -in dataset/video01.mp4 -size 50 -tracker poly -pp -yolo 128 -area 450 -1

## Előfeldolgozás

A *size* paraméternek megfelelően, területi interpolációval átméretezi a képet.

A hivatalos dokumentáció szerint a legmegfelelőbb ez az interpolációs módszer, abban az esetben, ha csökkentjük a kép méretét, és a legtöbb esetben a gyorsabb feldolgozás érdekében vagy a felesleges részletek elvétése miatt csökkenteni érdemes a méretet, melyben az új értékek függenek az eredeti képben megfelelő ablak alatti értékek súlyozott átlagától.

Az előfeldolgozás során elvégzik a szükséges kalibrációkat, inicializálásokat, például az alapértelmezett *kernel* kiszámítását, háttér kivonásos algoritmusok beállítását, 2 dimenziós üres mátrixok létrehozását.

A kalibrációs modulban számlálódnak az alap adatok, például az FPS szám, ezek alapján a szükséges ciklikus függvényhívások, például a hőtérkép meghatározása, amelyek csak bizonyos időközönként számolódnak ki.

Ha a *preprocess* be van kapcsolva, akkor a LAB színteret alkalmazva, más hasonló tudományos alkalmazások is ezt használják szín alapú szegmentációra [9], homályosítással, megtörténik az előfeldolgozás, azonban érdemes megvizsgálni az alkalmazott háttérkivonási, objektum detektálási módszereket, hiszen lehetséges, hogy egyes algoritmusok maguknak kezelik ezt a folyamatot.

**def** startProcess(capture):  
 **while True**:  
 ret, frame = capture.read()  
 **if** frame **is None**:  
 **return  
  
 if** len(config.SIZE) == 1:  
 frame = imutils.resize(frame, int(frame.shape[1] \* config.SIZE[0] / 100))  
 **if** len(config.SIZE) == 2:  
 frame = cv2.resize(frame, (config.SIZE[0], config.SIZE[1]), interpolation=cv2.INTER\_AREA)  
  
 calibration.init(frame)  
  
 contoursFrame = numpy.copy(frame)  
 tests = [numpy.copy(frame)]  
  
 calibration.fps()**if** config.PP:  
 preprocessed = preprocess.denoise(frame)  
 **else**:  
 preprocessed = frame  
  
 **if** config.INTERACTIVE:  
 cv2.imshow(**'preprocessed'**, preprocessed)

15. ábra Feldolgozás indítása az inicializálással és előfeldolgozással a csővezetékben

A feldolgozás indításakor átméretezzük a megadott paraméterek szerint, mindegy egyes képkockát, és szükség esetén meghívjuk az egyéb előfeldolgozási függvényeket.

## Szegmentálás

A *bgs*, vagy *yolo* paraméter alapján megtörténik a háttérkivonás, objektum szegmentálás. Ha létezik *yolo* paraméter a *bgs*-t figyelmen kívül hagyja.

YOLO alkalmazás esetén figyelni kell a betanított adatbázison alapuló eredmények mennyire relevánsak a ténylegesen alkalmazott felhasználáskor. A felmerülő objektumok lehetséges osztályozó listáját és módosítani lehet a minimum konfidencia értéket.

BGSLibrary használata esetén, általában az implementált algoritmusok maguk végzik az előfeldolgozási lépéseket, ellenben az OpenCV által megvalósított algoritmusok pedig nem, továbbá beállítható a tanulási ráta, a minimum határértékek.

A floodfill algoritmus használata opcionális, nagyban függ a választott háttérkivonási algoritmus eredményességétől. Kikapcsolásához negatív érték megadása szükséges a *fd* paraméter segítségével*.*

A morfológia, a zajok eltávolítására, és a markáns objektumok megtartatását célozza meg, ezért a Open műveletet úgy végezzük el, hogy az erózióhoz használt kernel mértéke eggyel kisebb legyen, mint a dilatációhoz használt.

Ezek után a kontúr keresés következik, olyan kontúrokat veszünk figyelembe, amik területe, a megadott intervallumba esik, és a hierarchia szerint szülő nélküliek, tehát a belsőbb találatokat nem vesszük figyelembe, ilyenkor az esetek nagy részében elhagyható lenne a morfológia, hiszen nem szükséges szabályos maszkok kialakítása, ha ignoráljuk a kisebb hibás kontúrokat.

**for** i **in** range(len(contours)):**if** hierarchy[i][3] != -1:  
 **continue** cnt = contours[i]  
 **if** config.YOLO > 0:  
 x, y, w, h = cnt  
 **else**:  
 x, y, w, h = cv2.boundingRect(cnt)  
 **if** x > 0 **and** y > 0 **and** w != 0 **and** h != 0 **and** config.CONTOUR\_MIN\_AREA < w \* h < config.CONTOUR\_MAX\_AREA:**if** config.SHADOW\_CROP:  
 shadow.detectShadowPart(frame[y:y + h, x:x + w])  
 cx, cy, cw, ch = shadow.crop(x, y, w, h)  
valid\_contours.append((x, y, w, h))  
 masker.addContour(cx, cy, cw, ch)

16. ábra Kontúr feltételek vizsgálata, és gyűjtése

Végig iterálunk az kontúrokon, és megvizsgáljuk, hogy megfelelnek-e a méretbeli és hierarchiás feltételeknek, majd szükség esetén meghívjuk, az árnyék kereső és a 4.6 fejezetben látható metódusokat.

Fontos megjegyezni, hogy a program alapvetően approximált befoglaló téglalapokkal dolgozik az egyszerűség kedvéért, és mert a legtöbb általános tárgy megfelel ennek a követelménynek.

## Kontúrok és árnyékok

A kontúrok által határolt képrészekből próbáljuk kiszűrni az árnyékok arányát, ehhez viszont biztos objektumot tartalmazó kép kell, a teljesen fekete színű detektált kontúrok hibás eredményt adhatnak, így ezeket elvetjük. Az említett módszerek szerint tároljuk egy listában az arányokat, majd ezek középértékét használjuk, a további vágásokra.

Ezt a *shadow* paraméterrel lehet szabályozni, a *shadowDir*-rel pedig a vágások irányát.

**def** shadowRate(img, contour, threshold=0.75, edge=1.2):  
 **global** directions  
 found = **False** x, y, w, h = contour  
  
 *# top* **if** 0 **in** config.SHADOW\_DIRECTIONS:  
 **if** y > (img.shape[0] \* (1 - threshold)):  
 c = (y \* (1 - (edge - 1))) / (img.shape[0])  
 **if** 0 < c < 1:  
 directions[0].append(c)  
 found = **True** *# right* **if** 1 **in** config.SHADOW\_DIRECTIONS:  
 **if** (x + w) <= (img.shape[1] \* threshold):  
 c = ((x + w) \* edge) / (img.shape[1])  
 **if** 0 < c < 1:  
 directions[1].append(c)  
 found = **True**

17. ábra Kódrészlet árnyék arányok megállapítására

**def** crop(x, y, w, h):  
 **global** directions  
  
 **for** direction **in** range(len(directions)):  
 **if** len(directions[direction]) != 0:  
 alpha = numpy.mean(directions[direction])**if** direction == 0:  
 y = y + math.ceil(h \* alpha)  
 h = h - math.ceil(h \* alpha)  
 **if** direction == 1:  
 w = math.ceil(w \* alpha)

18. ábra Vágások elvégzése, az előzetesen gyűjtött arányokkal

Ha nem található semmilyen adat, vagy ki van kapcsolva a vágási funkció, akkor az inputtal azonos adatokkal tér vissza a metódus.

Az összenőt, egyben detektált több objektum szétdarabolásához, távolságtérképet használok, aminek segítségével megkapom az objektumok központját, ezt használom a daraboláshoz.

Ezeket gyűjtve hőtérképet kapok, ami egy-egy terület fontosságát, frekvenciát jelenti meg.

**def** calcCurrentContours():  
 **global** \_\_mask, pixel\_min\_occurrence, pixel\_max\_oldness, current\_mask  
 **if** pixel\_max\_oldness == 0:  
 pixel\_max\_oldness = calibration.frame\_counter \* 1.2  
  
 x = \_\_mask[:, :, 0]  
 avg = numpy.average(x[x > 0])  
 **if** numpy.isnan(avg):  
 **return** pixel\_min\_occurrence = int(avg \* 0.8)  
  
 res = numpy.where((\_\_mask[:, :, 0] >= pixel\_min\_occurrence) &  
 (\_\_mask[:, :, 1] >= calibration.frame\_counter - pixel\_max\_oldness), 255, 0)  
 \_\_mask[:, :, 0] = 0  
 current\_mask = res.astype(numpy.uint8)

19. ábra Hőtérkép számítása a gyűjtött adatokból

## Osztályozás

A megkapott objektumokat, osztályozni kell, hogy bekerültek-e már a rendszerbe, vagy új belépő objektum. Ez egy többlépcsős fokozat.

Ehhez elsőnek megvizsgálom, hogy tartalmazza-e az adott objektum koordinátákat, valamely előző nem lezárt objektum határvonalai, hiszen stabil, magas FPS-nél, nem valószínű, hogy egy tárgy elmozdulása nagyobb, mint a tárgy nagysága. Ha több ilyen van, ezeket sorbarendezem a szerint, hogy mikor volt róluk utoljára információ frissítés, majd veszem a legkorábbit.

**class** Object:  
 **def** \_\_init\_\_(self):  
 self.id = uuid.uuid4()  
 self.path = []  
 self.entry = Point()  
 self.exit = Point()  
 self.closed = **False** self.direction = **None** self.lostIndex = 0  
 self.\_contour = **None** self.histogram = []  
 **if** tracking.builtin:  
 self.tracker = config.TRACKER.value()  
 self.tracked = **False** self.contourPosition = []  
 cr = (list(numpy.random.choice(range(256), size=3)))  
 color = [int(cr[0]), int(cr[1]), int(cr[2])]  
 self.corrected = **False** self.color = color  
 self.poly = []

20. ábra Egy objektumot reprezentáló osztály

Ha nem volt ilyen, vizsgálom a hozzá maximum 4 legközelebbiket beleértve a hőtérkép által generált széleket.

Ezt a négyet, azért választottam mert lehetséges, hogy megfigyelés közben bekerül valami hasonló, de hibás, nem releváns, nem sokáig aktív tárgy az objektum mellé, ilyenkor nem lesznek hasonlóak, de mégis a legközelebb lesz hozzá, így hibás lehet a csak legközelebbit venni, ezt a legközelebbi objektumok darabszámát, módosítani lehet a gyűjtő beállítások, *CLOSEST\_N* attribútumával.

newObject = createNewObject(cx, cy, x, y, w, h, contour)  
**if** len(list(filter(**lambda** o: o.closed **is False**, objects))) == 0:  
 **return** [**True**, newObject, **'noone opened'**]  
insides = []  
**for** o **in** objects:  
 **if not** o.closed **and** pointInContour(o.contourPosition, [cx, cy]):  
 insides.append(o)  
**if** len(insides):  
 insides.sort(key=**lambda** x: x.lostIndex, reverse=**False**)  
 sims = similarity(insides[0],newObject)  
 **if** numpy.max(sims) > 0.7 :  
 **return** [**False**, insides[0], **'inside another'**]  
#center point  
cp = Point(cx, cy)  
closests = list(filter(**lambda** o: o.closed **is False**, objects))  
closests.sort(key=**lambda** o: o.path[-1].distance(cp), reverse=**False**)  
closests = closests[0:config.CLOSEST\_N]  
closests = sorted(closests, key=**lambda** o: numpy.sum(similarity(o, newObject)), reverse=**True**)  
  
**for** closest **in** closests:  
 sims = similarity(closest, newObject)  
 **if** config.LINEAR\_MOVEMENT:  
 **if** closest.path[-2].y > closest.path[-1].y:  
 **if not** (closest.path[-1].y > (cy \* 0.85)):  
 result = [**True**, newObject, **'backwards'**]  
 **continue  
  
 if** closest.path[-2].y < closest.path[-1].y:  
 **if not** (closest.path[-1].y < (cy \* 1.15)):  
 result = [**True**, newObject, **'backwards'**]  
 **continue**

**if** (sims[0] > 0.65 **or** sims[1] > 0.65) **or** sims[2] >= 0.5:  
 dists = [99999]  
 **if** len(closest.path) > 2:  
 dists = []  
 **for** i **in** range(1,len(closest.path)-1):  
 dists.append(closest.path[i].distance(closest.path[i+1]))  
 **if** closest.path[-1].distance(cp) < numpy.median(dists) \* config.MAX\_LOST \* calibration.fps\_rate:  
 **return** [**False**, closest, '**similar'**]  
  
**if** len(result) != 0:  
 **return** result  
  
**return** [**True**, newObject, **'nothing'**]

21. ábra Legközelebbi, nem lezárt objektumok vizsgálata

Ezeket a közeli objektumokat az említett hasonlósági módszerek szerint rendezem, és a legjobbat találom meg, ha teljesíti a feltételeket, ellenkező esetben, új objektumként illesztem be a rendszerbe, azzal a megjegyzéssel, hogy kulcspont párok kereséséhez és illesztéséhez ORB (Oriented FAST and Rotated BRIEF) [10], mivel elkészítés időpontjában még, szabadalmi problémák miatt, nem állt rendelkezésre a SIFT [11] algoritmus.

Siker esetén az aktuális új képen lévő adatok is bekerülnek az objektumhoz, például koordináta, kimetszett kép, ilyenkor ebben a körben már nem fut semmilyen követési algoritmus ezen a tárgyon, hiszen már bekerültek az új adatok.

## Követési módszerek

Ha a fent említettek szerint sikerül párosítani egy objektumot, akkor a tétlenség, inaktivitás jelzője nullázódik. Ellenben pedig minden eltelt képkockával nő és a megadott paraméterek szerinti idő, jelentse itt az eltelt képkockák számát, eltelte után lezár egy objektum, ilyenkor nem lehet már hozzáadni sem új adatot, sem a párosításnál nem játszik szerepet.

A poly követési módszernél, ha a polinom által határozott pont kívül esik a hasznos területen akkor is lezár az objektum.

Hogy elkerüljük a rövid életű, fals objektumok tömeges megjelenítését, az objektum úthosszának elkell érni a *MIN\_PATH\_LEN* számot.

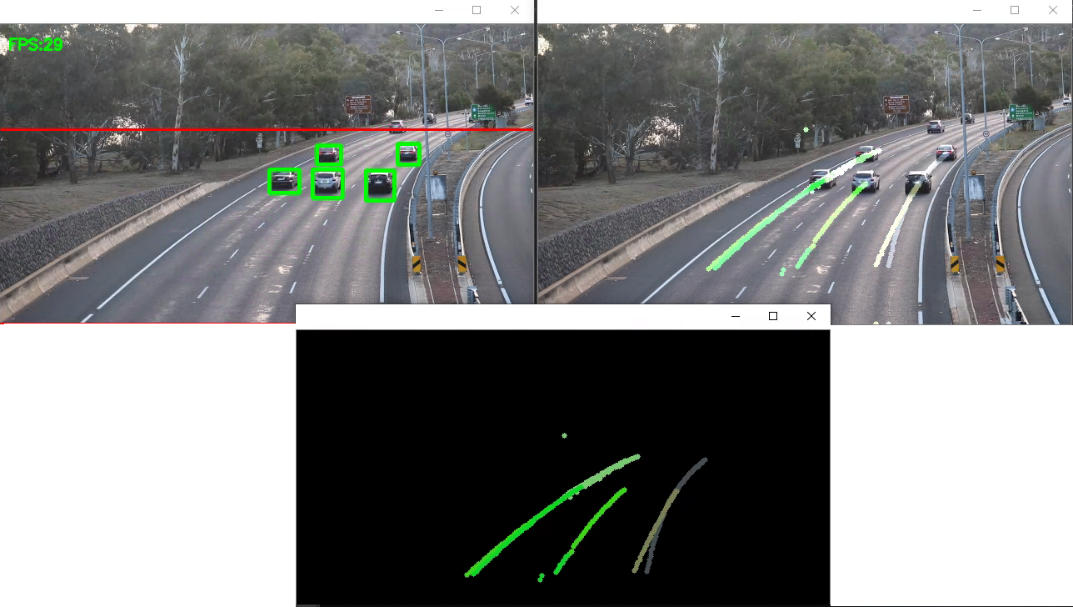
**for** object **in** objects:  
 **if** object.closed:  
 **continue  
  
 if** builtin:  
 (success, box) = object.tracker.update(frame)  
 **if** success:  
 x, y, w, h = [int(v) **for** v **in** box]  
 cx = int(x + w / 2)  
 cy = int(y + h / 2)  
  
 cv2.rectangle(dotsframe, (x, y), (x + w, y + h), (0, 255, 0), 1)  
 **if** object.corrected:  
 object.corrected = **False  
 else**:  
 object.addToPath(Point(cx, cy), x, y, w, h)  
 object.contour = frame[y:y + h, x:x + w]  
  
 **if** y < actionScene.top **or** y + h > actionScene.bottom:  
 object.close()  
 **else**:  
 object.lostIndex += 1  
 **elif** config.TRACKER == TRACKERS.SIMPLE:  
 object.lostIndex += 1  
 **elif** config.TRACKER == TRACKERS.POLY:  
 **if** object.corrected:  
 object.corrected = **False  
 else**:  
 object.lostIndex = object.lostIndex + 1  
 success, [cx, cy] = (object.getNextCoordinates())  
 y = object.contourPosition[1]  
 h = object.contourPosition[3]  
  
 **if** success:  
 object.addToPath(Point(cx, cy))  
 **if** y < actionScene.top **or** y + h > actionScene.bottom **or** cy < actionScene.top **or** cy > actionScene.bottom:  
 object.close()

22. ábra Minden képkockán lefutó karbantartó, követő metódus

A különböző módszerekkel kiszámoljuk az eredményeket. Minden frame-re lefuttatjuk ezt, hiszen az OpenCV által szolgáltatott algoritmusoknak szükséges minden frame bevitele, máskülönben, olyan mintha lassú, alacsony FPS-el dolgoznánk. Továbbá szükséges az inaktivitás, és *lostIndex* karbantartása majd ezek megjelenítése és kirajzolása feltételek szerint.

## Összegzés

A kirajzolt útvonalakat pontszerűen jeleníti meg a program, azzal a színnel, amit az objektum inicializáláskor random módon választ. Ezt egy külön képként jeleníti meg, illetve külön kerül megjelenítésre a szegmentált objektumok határoló téglalapja és a hasznos, hőtérkép által határolt terület széle. Emellett egy harmadik képben összevonja ezeket a könnyebb megjelenítés és értelmezés érdekében.



23. ábra Teljes eredmény kép autók követésénél

Ezeket számadatként exportálni is tudja a program a megfelelő beállításokkal és paraméterekkel.

A fejlesztés során generált tesztekhez és demókhoz használt kódok megtalálhatóak, vagy egyszerű szerkezetben az implementációban, vagy pedig külön fájlként, így ezek futtatása megismételhető.

# Eredmények

Az dolgozat alapvetően az online azonnali feldolgozásra épít, tiszta, egyértelmű objektumokat képes követni, összehasonlítani, és előrejelzést készíteni a múltbéli mozgásuk alapján. Mindezt egy általános, autonóm, nem specializált rendszerként próbáltam implementálni, ennek a céljára igyekeztem részletes paraméterezést biztosítani a programban, illetve a rendszer felépítését átláthatóbbá, és könnyen kezelhetővé tenni.

A tesztelés során igyekeztem különböző tematikájú és helyzetű videókat használni, az ebben a fejezetben bemutatott képek a megfelelően konfigurált program futtatási eredményei.

A program képes mozgás alapján felismerni és követni objektumokat rögzített kamera szögből.

|  |  |
| --- | --- |
| A képen út, állás, sárga, utca látható  Automatikusan generált leírás | A képen virág látható  Automatikusan generált leírás |

24. ábra Követés futtatása simple (bal) és poly (jobb) módszerrel

|  |  |  |
| --- | --- | --- |
| A képen tábla, alapozás, férfi látható  Automatikusan generált leírás | A képen alapozás, szőnyeg látható  Automatikusan generált leírás |  |
| A képen tábla, alapozás, férfi, utca látható  Automatikusan generált leírás | A képen férfi, utca, alapozás, tábla látható  Automatikusan generált leírás |  |

25. ábra Egyedülálló egyszerű objektum követése a programmal

A képen medve, kültéri, állat, emlősök látható

Automatikusan generált leírás

26. ábra Program futtatása egy kedvező, fix kamerás felvétel, viszonylag állandó háttérrel

Az algoritmus képes több, különálló objektumot is követni felvételeken megfelelő beállításokkal, de nehézségekbe ütközhet.



27. ábra Parkban mozgó emberek követése a programmal

Az algoritmus hiányossága, hogy nem veszi figyelembe, ha egy tárgy visszatér a megfigyelt régióba, mert ha egyszer kívül esik ezen akkor lezárásra kerül, ellenben, ha csak nem kerül detektálásra, tehát csak az inaktivitási jelző nő, akkor visszatérhet.

Továbbá a fizikai modellek sem valósulnak meg, egyedül a lineáris mozgás, de annak is az ellenőrzése, illetve a múltbéli helyzet alapján való függvény illesztés, de ez egy véletlenszerűen, vagy közel véletlenszerűen mozgó tárgynál nem jó leíró eszköz. Ennek megoldására múltbéli mozgástól, és alkalmazástól függő predikciókat lehet használni, lásd Kalman Filter és Extended Kalman Filter (EKF).

A képen fű, kültéri, férfi, papírsárkány látható

Automatikusan generált leírás

28. ábra Járókelők követése a programmal egy teszt felvételen

A járókelők útja direkt keresztezi egymást ezen a teszt felvételen, egyes objektumok túl nagy hasonlóságot mutatnak, így az algoritmus összekeveri ezeket, de az útvonalak még mindig kivehetőek. Ez így más szcenárió, mint a már látott jármű követés.

|  |  |
| --- | --- |
| A képen jelenet, út, fű, kültéri látható  Automatikusan generált leírás | A képen jelenet, út, fű, kültéri látható  Automatikusan generált leírás |

29. ábra Mozgás alapú detekcióval végzett követés (balra) és YOLO (320) által végzett detekció (jobb)

Előfordulhat olyan eset is, legtöbbször az egyszerű elmozdulásoknál, illetve egyszerű objektumoknál amikor a háttérkivonásos algoritmus hatékonyabb, mint egy előre betanított neurális hálózat. Érdemes megemlíteni, hogy a háló nem specifikusan előre betanított adatokon alapul tanított.

30. ábra Az osztályozás eredményei eloszlása egy felvételen

Az osztályozás eredménye legtöbbször a belső helyzeten alapul, ami megfelelően kamerával és képkocka sebességgel elegendő lehet, de ezek hiányában téves eredményt adhat.

Hasonlóság ellenőrzésnél nagyban lehetne támaszkodni, kulcspontok gyűjtésére, de ezzel az a probléma, hogy általában ezek túl kis felbontású képekké torzulnak a perspektivikus nézet miatt, így nem lehet rajtuk kiemelni kulcspontokat.

31. ábra Detektált objektum méretek megoszlása

A szemléltetés érdekében a teszt videón árnyék vágást végeztem, a minimum méret pedig 450 pixel volt, maximumot nem állítottam be, a háttérkivonás MOG algoritmussal történt. Az adatok közé azért kerülhet 450-nél kisebb terület mert ez a követelmény ellenőrzés még az árnyék csökkentés elött van. A kép teljes mérete (360, 640) pixel, több objektummal, autópályás felvételen. Ezeken a kis méretű képeken nehéz szignifikáns, leíró tulajdonságot találni, tehát a perspektivikusan elhelyezett kamera miatt, a kisebb és hasonló autók összetéveszthető adatokat produkálnak.

Az objektumokat befoglaló téglalapok helyett alkalmazás szerinti fejlesztés lehet, a pont halmazból álló kijelölések használata, illetve az árnyékok dinamikus vágása, mivel a jelenlegi módszer a konstans fényforrásoknál működik jól, viszont a változó irányú és nagyságúaknál nem.

## Teljesítmény

A legtöbb algoritmus, és a neurális hálózatok, kezdve a tanuláson át a felismerésig jobban teljesítenek a megfelelő GPU-n. A használt keretrendszert a 2000 évek elején kezdte el fejleszteni az Intel, ennek megfelelően a projektet mai napig támogatja, és általában Intel termékekre optimalizált, de a keretrendszer GPU modulja CUDA-ra épül, így más hardwarek is támogatják.

A képen képernyőkép látható

Automatikusan generált leírás

32. ábra Tesla C2050 versus Core i5-760 2.8Ghz, SSE, TBB [12]

Ennek megfelelően a csúcsteljesítmény eléréséhez érdemes megfelelően felszerelt videókártyát választani, akár a neurális hálózaton alapuló felismerésről, akár más képfeldolgozási algoritmusról van szó, ennek hiányában a CPU számítási teljesítményére vagyunk kénytelenek hagyatkozni.

## Más módszerek

Más módszer és megközelítések lehetne például a Kálmán-szűrű használata beépítve a módszerek közé, ami, valószínűségi modell alapján próbálja jósolja meg az objektumok helyzetét a valóságból vett modellek alapján, ennek egy implementációja a SORT [13].

A YOLO detektáló rendszeren túl, lehet helyettesíteni mással is a keresést, esteleg eleve megadott koordinátákkal dolgozni.

|  |  |  |  |
| --- | --- | --- | --- |
| **Metódus** | **Input** | **mAP (%)** | **FPS** |
| Faster R-CNN | - | 72.70 | 11.23 |
| YOLO | 448 x 448 | 62.52 | 42.34 |
| YOLOv2 | 416 x 416 | 73.82 | 64.65 |
| YOLOv2 | 544 x 544 | 75.96 | 39.14 |
| SSD300 | 300 x 300 | 74.18 | 58.78 |
| SSD512 | 512 x 512 | 76.83 | 27.75 |
| R-SSD512 | 512 x 512 | 77.73 | 24.19 |

3. táblázat Különböző detektálási algoritmusok eredményei az University at Albany versenyéről [14]

Az eredmények közül [15] kimagasló teljesítménnyel rendelkezik egy módszer, melynek alapja a képi információk nélküli követés, és ennek tovább fejlesztett változata, ugyanazon szerző által [16].

# Irodalomjegyzék

|  |  |
| --- | --- |
| [1] | S. Sural, G. Qian és S. Pramanik, „Segmentation and histogram generation using the HSV color space for image retrieval,” in *Proceedings. International Conference on Image Processing*, 2002. |
| [2] | „CIELAB”. |
| [3] | A. Sobral, „lrslibrary”. |
| [4] | A. Sobral, „List of available algorithms”. |
| [5] | P. K. a. R. Bowden, An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection In: Video-Based Surveillance Systems, New York, 2002. |
| [6] | J. Redmon és A. Farhadi, „YOLOv3: An Incremental Improvement,” *CoRR,* %1. kötetabs/1804.02767, 2018. |
| [7] | M. T. Pike, „Computer Vision and Deep Learning in Autonomous Drones”. |
| [8] | OpenCV, „Tracking API”. |
| [9] | MatLAB, „Color-Based Segmentation Using the L\*a\*b\* Color Space”. |
| [10] | V. R. ,. K. K. ,. G. B. Ethan Rublee, „ORB: an efficient alternative to SIFT or SURF”. |
| [11] | D. G. Lowe, „Distinctive Image Features from Scale-Invariant Keypoints,” 2003. |
| [12] | Opencv, „CUDA performance,” 2020. |
| [13] | Z. G. L. O. F. R. B. U. Alex Bewley, „Simple Online and Realtime Tracking,” in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016. |
| [14] | F. &. L. C. &. Y. F. Zhang, „Vehicle Detection in Urban Traffic Surveillance Images Based on Convolutional Neural Networks with Feature Concatenation,” 2019. |
| [15] | S. U. o. N. Y. (. Research IT Group of the University at Albany, „UA-DETRAC Benchmark”. |
| [16] | V. E. a. T. S. E. Bochinski, „High-Speed tracking-by-detection without using image information,” in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2017. |

# Nyilatkozat

Alulírott Sári Gábor programtervező informatikus BSc. szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Képfeldolgozás és Számítógépes Grafika Tanszékén készítettem, programtervező informatikus BSc. diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Dátum

Aláírás

# Köszönetnyilvánítás

2020