Gépi látás

GKLB_INTM038

Mozgás detektálása videófelvételen

Varga Zoltán

GIRJ8O

Tatabánya, 2019/20 2. félév

Tartalomjegyzék

1 Elméleti háttér

A mozgás – azaz az optikai áramlás - érzékelésének lényege, hogy adott képpontokat figyelve képkockáról képkockára megállapítható, hogy változik-e azok pozíciója a képen, vagyis azok X,Y koordinátái. Ezzel az időbeli elmozdulás figyelhető meg. Két féle módon tehető meg: megfeleltetés vagy variációs alapú módszerekkel.

Az előbbi eljárások, azaz a **megfeleltetéses módszer**ek esetében kitűntetett képpontok elmozdulását vizsgálják, vagyis jellemzők alapján jól meghatározható pontokét. Az ilyen eljárások esetében valószínűsíthető egy megelőző, objektum felismerő algoritmus futtatása, amely meghatározza azon képpontok halmazát, amelyeknek majd az időbeni elmozdulására vagyunk kíváncsiak. Mivel ebben az esetben nincs minden egyes képpont vizsgálat alá vetve, kevesebb erőforrást igényel mivel kevesebb számítást kell elvégezni. Ez a módszer nagy előnnyel alkalmazható abban az esetben is, amikor nem statikus háttér előtt mozog a vizsgált objektum, így a képkerethez képest elmozduló háttér nem okoz gondot az algoritmus számára. A említett mobilalkalmazásokhoz ez megfelelő módszer lehet.[1]

A **variációs alapú módszer**ek esetén, a képkockák minden egyes pontja vizsgálat alá kerül. Ennek megfelelően ezen eljárás során nagyobb mennyiségű számítás kerül elvégzésre, ami lassabb működést és nagyobb erőforrás igényt jelent. Mivel ebben az esetben minden képpont vizsgálat alá kerül képkockáról képkockára, ha nem statikus a háttér vagy túl sok mozgó objektum van a megfigyelt képen (például szél fújta fák) akkor a mozgás detektálás túl nagy eredmény halmazt eredményezhet amely az értelmezhetőséget nehezíti. Ez az eljárás megfelelő lehet biztonsági kamerák képének megfigyeléséhez vagy az iparban a gyártó környezetbe telepített kamerákkal való termék vagy egyéb megfigyelni kívánt eszköz követésére.[1]

2 Mozgás detektálás a gyakorlatban

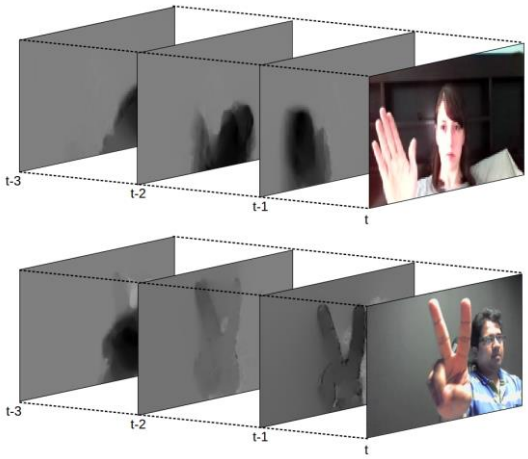
Én elsősorban egy olyan kódot szerettem volna készíteni, ami egy stabilan álló kamera képén detektálja a mozgást, vagyis valamely tárgy vagy személy elmozdulását. Ez az előzőekben említett variációs alapú módszernek felel meg, mivel a kamera képén a környezet nem vagy csak kis mértékben változik. Ebben az esetben nem szükséges egy adott tárgy előzetes felmérése és a kód egyszerűbbé válik, valamint kevesebb erőforrást használ fel.

Különböző forrásokat böngészve több féle megoldással találkoztam. Az eljárások lépései és azok sorrendje azonban túlnyomó részt nagyon hasonló vagy megegyező volt, amely szinte minden algoritmus gerincét képezte:

1. Kamerakép vagy video felbontása képkockákra
2. Két egymást követő képkocka különbségképzése
3. A kapott kép szürkeárnyalatossá konvertálása
4. A szürkeárnyalatos kép kis mértékű elmosása
5. Majd egy küszöbérték szerinti bináris átalakítás
6. Az élek folytonosabbá tétele
7. Kontúrok meghatározása a kisimított élekből
8. Kontúrok megjelölése
9. Eredménykép megjelenítése

2.1 Kamerakép vagy video képkockákra történő felbontása

Mozgás álló képen, rögzített kamerával nem detektálható hacsak egy relatív hosszabb záridővel készült képen nem látszik egy mozgó tárgy vagy egyéb más elmosódva. Mivel a kamerakép vagy videófelvétel nem csak egy időpillanatot ad vissza, azon sokkal könnyebben meghatározható, ha valami megváltoztatja a kamerához viszonyított helyzetét a térben, az idő függvényében. Ehhez azonban arra van szűkség, hogy a mozgókép felbontásra kerüljön annak statikus képkockáira. Mivel a képek időben eltérő pillanatban készülnek, a mozgó objektum azokon különböző pozícióban kell, hogy megjelenjen.



1. ábra Mozgókép t, t+1, t+2...képkockákra való felbontása [4]

Éppen ez az a tulajdonság amelyből a mozgás detektálható. Azonban érdemes megjegyezni, hogy ha a mozgás nagyon lassú, vagy a felvétel, időegység alatt nagyon nagy számú készülő képpel rendelkezik (magas fps szám), nem minden egymást követő képen lehet majd különbséget és ezáltal mozgást azonosítani. Ekkor jó megoldás, ha nem minden egymást követő kép kerül összehasonlításra, hanem az adott képkocka után meghatározott számú képkockával későbbi kép.

2.2 Két képkocka összehasonlítása

A mozgás érzékeléséhez két egymást (nem feltétlenül közvetlenül) követő képkocka összehasonlítására van szűkség. Ha a kamera képen vagy videón egy objektum elmozdul, akkor annak képpontjai két egymást követő képkockán máshol jelennek meg. Ez a pixel eltolódás az amely alapot nyújt a mozgás helyének és mértékének meghatározásában. A különböző programnyelvek és keretrendszerek valamint függvény könyvtárak segítségével a programozók könnyen megkerestethetik az eltérést két képkocka között. Az általam választott Python nyelvhez túlnyomó részt a nyílt forráskódú openCV függvény könyvtárat használják széles körben, amellyel valós idejű gépi látás funkciók valósíthatók meg egyszerűen.[2] Én differencia keresésre a cv2.absdiff() függvényt használtam, amely kiszámolja az abszolút különbséget két tömb vagy egy tömb és egy skalár mennyiség közt. Ebben az esetben az eredmény egy képkockaként értelmezhető tömb, amely azokat a képpontokat tartalmazza, amelyek nem egyezik meg a függvénynek paraméterként átadott két egymást követő képkocka között.

2.3 Grayscale konvertálás

Az eredményül kapott RGB különbség kép pontjai még mindhárom szín információit tartalmazza. A különbség keresés során erre már nincs szűkség, valamint a további képtranszformációkat is nagy mértékben meg nehezítené. Ezért szűkséges a differenciát tartalmazó kép (vagy különbség halmaz) szürkeárnyalatossá konvertálása, amellyel már elérhető, hogy minden egyes pixelhez a három szín információ helyett már csak egy szürkeárnyalatos intenzitás információ tartozzon.

2. ábra Szürkeárnyalatos konverzió [1]

Az openCV függvénykönyvtárban erre a cv2.cvtColor() és a cv2.COLOR_BGR2GRAY eljárások alkalmazhatóak. Az előbbi input paraméterként a különbség képet és a cv2.COLOR_BGR2GRAY flag-et, amely meghatározza, hogy az adott képet mely színtérből melybe kell konvertálni.

2.4 Kép elmosása

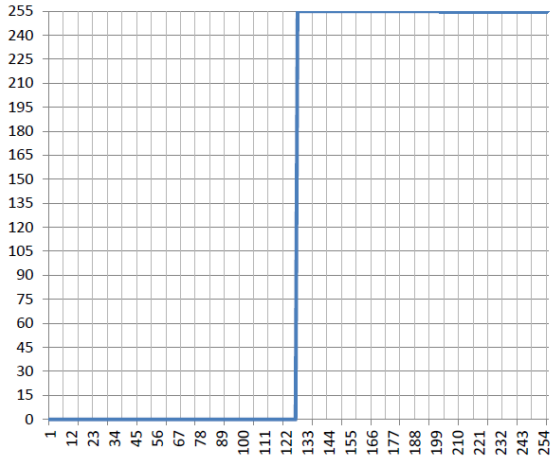
Általában egy fénykép annál jobb, minél élesebb. Persze ez alól kivételt képeznek a művész fotók, melyek témája indokolhatja a kép szándékos elmosását. Egy másik ilyen kategória a portré képek, amelyeknél az alany körüli teret szokás kissé homályosabbá tenni. Az ilyen fotókon az vehető észre, hogy azokon a területeken, ahol a háttérben valamilyen nem egyenletes kontúrral rendelkező objektum található, ott az elmosás hatására az egyenetlen körvonalak folytonosabbnak hatnak. Ez a gépi látás területén az élkeresési eljárások során általánosan használt módszer a kontúrok simábbá tételére. A legtöbb esetben a Gauss elmosást alkalmazzák, amely alapvetően egy zajszűrésre használható konvolúciós eljárás, ahol a kernel úgy van kialakítva, hogy a módosítandó képponthoz közelebbi képpontok nagyobb súllyal bírnak.[1]

3. ábra Gauss elmosás [5]

Én a cv2.GaussianBlur() függvényt használtam, ami paraméterként a szürkeárnyalatos képet kapja meg valamint azt, hogy a használt kernel mekkora méretű legyen illetve az X és Y irányú eltolás mértékét. A kernel mérete általában 3x3 vagy 5x5-ös mérettel ad vissza optimális eredményt, azonban ez a felvétel vagy kamerakép több tulajdonságaitól is változhat (pl. mozgó objektum alakja).

2.5 Bináris kép létrehozása

A szürkeárnyalatos kép már jó közelítéssel megadja a mozgás kontúrját, azonban ez még nem elegendő a mozgást körül határoló kontúrok meghatározásához. Ehhez sokkal alkalmasabb lenne egy két színből álló úgy nevezett bináris kép, ahol minden képpont intenzitás értéke 0 vagy 255. Ez egy nagy kontraszttal rendelkező kép, amelyen már jól meghatározható két képpont közti különbség. Ez feltétlenül szűkséges a minél pontosabb éldetektáláshoz.

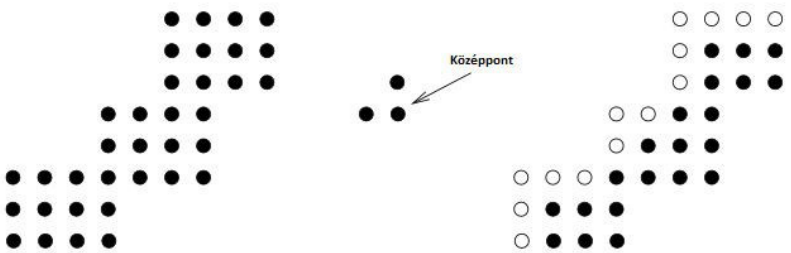


4. ábra Bináris konverzió diagrammja. A küszöbérték itt ~128. [1]

Az openCV segítségével ez a kép transzformáció egy függvény használatával elvégezhető. Ez a függvény a cv2.threshold() amely négy paramétert vár: a módosítani kívánt képet, a küszöbértéket, azt az intenzitás értéket amelyet majd minden pixel megkap a transzformáció során amelyek a lefutás előtt küszöbérték felett voltak valamint azt, hogy a művelet milyen módon hajtódjon végre. Ebben az esetben a bináris kép lenne a legoptimálisabb kimenet, így a negyedik paraméter ez esetben cv2.THRESH_BYNARY. Így azon képpontok intenzitása, amelyek a küszöbérték felett vannak, megkapják a paraméterként adott értéket (ezt én fehérnek vettem azaz 255-nek), a többi pedig 0-s (azaz fekete) értéket kap.

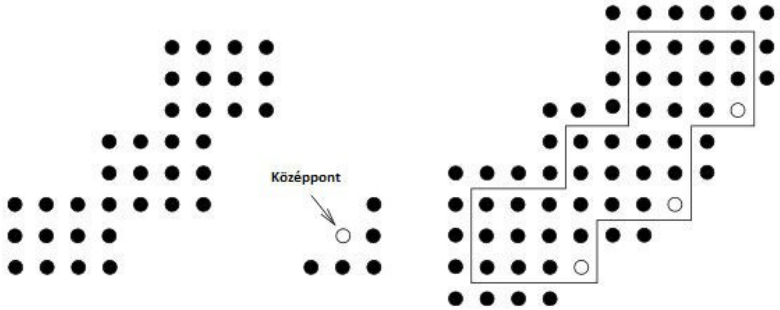
2.6 Élfolytonosság

A két képkocka közti különbség megállapítása során szinte biztos, hogy az objektum alakja, mozgási iránya (pl. forgó mozgás) vagy helyenként a háttérrel való színbeli összemosódása miatt a különbség képpontok nem fognak folytonos vonalba esni vagy éppen több külön álló objektum körvonalai összemosódnak. Ez miatt nehezebb meghatározni a mozgó tárgy körvonalait. Ennek kiküszöbölésére alkalmazhatók bináris képeken az úgynevezett morfológiai eljárások. Ennek két változata az erózió és a dilatáció. Sok esetben a két módszert együttesen használják megfelelő sorrendben végrehajtva.
Az előbbi, vagyis az erózió önmagában abban az esetben használható, ha a képen több objektum átfedésben vagy érintkezésben van egymással. Ekkor a kontúrvonalaik nem különülnek el egymástól. Az erózió során egy strukturáló elemet a kép pontjain végig csúsztatva csak azok a képpontok maradnak eredeti intenzitásúak, amelyeket a strukturáló elem lefed. Ezt a műveletet általában többször kell végre hajtani egymás után, hogy az eltérő objektumok élei elváljanak egymástól. A kisebb zajok és egyedülálló pontok (binárisok) eltűntethetőek ilyen módon.[1]



5. ábra eróziós eljárás [1]

A dilatációs eljárás hasonló elven működik. A művelet során a strukturáló elem szintén végig léptetésre kerül a kép összes pixelén. Azonban ebben az esetben ha a strukturáló elem egy pontja egybe esik az adott képponttal, akkor azon képpont körül a többi pont intenzitása is módosításra kerül a strukturáló elem pontjainak megfelelően. Ezzel a megoldással a nyitott, nem összefüggő élek folytonosabbá tehetőek. A legjobb eredmény eléréséhez mindkét módszer alkalmazására van szűkség. Először a az eróziós eljárást a zajok csökkentésére, majd a dilatációs folyamatot a kontúrok folyamatosabbá tételére. Mindkét eljárást érdemes többször egymás után végrehajtani.[1]

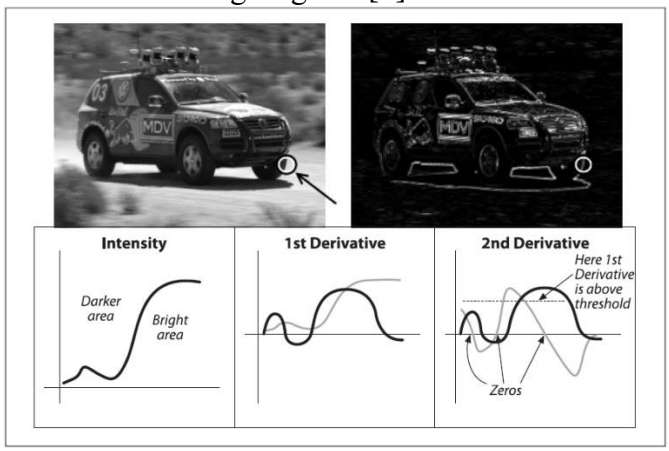


6. ábra Dilatációs eljárás [1]

Én csak a dilatációs funkciót alkalmaztam. Az openCV könyvtárából ezt az eljárást a cv2.dilate() függvény meghívásával implementáltam. A függvény három paramétert vár: a módosítani kívánt képet, a kernel típusát valamint a végrehajtások számát. Én az előzőekben binárissá konvertált képet, az alapértelmezett kernelt, ami egy 3x3-as mátrix és kétszeri végrehajtást adtam meg.

2.7 Kontúrok meghatározása

Ahhoz, hogy a mozgó objektumot a kameraképen vagy videófelvételen jól körül határolhatóan meg lehessen jelölni, annak egyértelmű élekkel kell rendelkeznie. Az éldetektálás azon az elven alapul, hogy ahol a szomszédos képpontok között hirtelen nagy az intenzitás különbség, ott valószínűsíthetően él található. Ezt a leg könnyebben egy bináris képen lehet meghatározni. A kép egy kétdimenziós függvényként értelmezhető. Élek keresésekor a kép deriváltja egyértelműen megadja az élek helyét, mivel azon a helyen ahol a szomszédos pixelek intenzitás különbsége nagy, ott a derivált értéke is magas lesz. A jóval nagyobb számítási erőforrást igénybe vevő pontos derivált kiszámítása helyett a kép deriváltja jó közelítéssel meghatározható konvolúciós szűrők segítségével.[1]



7. ábra Kontúrok meghatározása [5]

Én a cv2.findContours() függvényt használtam amelynek három paramétert adtam át: a dilatációs eljárással feljavított képet, a kontúrok visszakeresési módját valamint a kontúr közelítési módszert. Én itt a CV_RETR_TREE opciót választottam, ami esetén beolvassa az összes kontúrt és rekonstruálja a beágyazott kontúrok teljes hierarchiáját, valamint a CV_CHAI_APPROX_SIMPLE közelítési módszert amely csak a függőleges, vízszintes és átlós kontúrokat tömöríti oly módon, hogy csak azok végpontjait rögzíti.[3]

2.8 A kontúrok megjelölése

Mivel a fő cél a mozgás megjelenítése olyan formában ami a felhasználó számára értelmezhető, valamilyen módon jelölni kell a mozgást amely így egy a kameraképhez vagy videófelvételhez hozzáadott plusz információként jelenik meg. Az előzőekben végrehajtott képi transzformációk mind azt a célt szolgálták, hogy minél pontosabban meghatározható legyen a mozgókép azon területe ahol a mozgás történik. A legutóbbi konverziós eljárással létrejött egy bináris kép amely pontosan a mozgás helyét tartalmazza. Azonban ez a bináris kép kevésbé értelmezhető a felhasználó számára. Így szűkség van a képen lévő ezen információ megjelölésére. A leg célravezetőbb mód, ha az eredeti képkockán körbe rajzolásra kerül a két képkocka közti különbségből adódó kontúrvonal. Ez Python nyelv alatt a cv2.drawContours() függvénnyel egyszerűen kivitelezhető, ami 4 paramétert vár: elsőként a képet amelyre a jelölést szeretnénk (fontos, hogy a két összehasonlított képkocka közül az időben korábbira kerüljön, hogy a későbbi kép eredeti állapotában maradjon, hogy az összehasonlíthatóvá váljon az időben azt követővel), a körbe rajzolandó vagy kitöltendő kontúrvonalakat tartalmazó bináris képet, a kirajzolni kívánt kontúrok meghatározását (ha negatív érték, akkor az összes kontúrt figyelembe veszi), a kirajzolás vonalának színét RGB értékekkel valamint annak vastagságát (amennyiben ez negatív érték, a kontúr nem körbe rajzolódik, hanem kitöltésre kerül a megadott színnel).[3]

2.9 Az eredmény megjelenítése

Ha már meghatározásra került a két képkocka közti különbség, azaz a mozgás pontos helye valamint különböző képi transzformációk után az megjelölésre került az eredeti képkockán, akkor már annak megjelenítése van hátra. Mivel a cv2.drawContours() függvénnyel már módosítva lett az eredeti képkocka elég azt megjeleníteni. Erre alkalmas a cv2.imshow() függvény, amelynek paraméterként elég egy a megjelenítendő ablak elnevezését megadni, valamint azt a képet (ezesetben kamerakép vagy videófelvétel képkockája) amelyet meg kívánunk jeleníteni. Ezzel megjelölésre került az adott képkockán a mozgás helye.[3]

2.10 Videókép tovább léptetése

Az aktuális képkocka megjelenítése után fontos, hogy az azt követő másik – amellyel az összehasonlítást végeztük – áthelyezésre kerüljön a megjelenített képkockát tároló változóba, majd a helyére (az azt eddig tároló változóba) a mozgókép következő képkockája kerüljön beolvasásra. Így a fent leírt ciklus előröl indulhat.

3 Összegzés

Úgy gondolom, hogy az általam létrehozott kód egy alap eljárás amely számos ponton fejleszthető. Azonban szinte minden ma használatos mozgás detektálási algoritmus alapja az az elv amely mentén haladva építettem fel én is a programot. Azonban az általam leírt kódot sajnos soha sem sikerült futtatnom úgy, hogy értékelhető eredményt kapjak, így a leírtakat csupán elméleti síkon tudtam véghez vinni.

Először egy Raspberry PI B+-on próbáltam Linux környezetben, magát a kód írást is ott végeztem. A többszöri hibára futás után már csak egy hibaüzenetet kaptam újból és újból amely az openCV függvénykönyvtár használatával volt kapcsolatban. A Python és az openCV többszöri telepítése után sem sikerült a hibaüzenetet megszűntetnem. Hosszasabb kutatómunka árán arra a felismerésre jutottam, hogy többek visszajelzése alapján, Linux alatt a cv2.imshow() függvény nem tud probléma mentesen futni. Ezen információ ismeretében megpróbáltam a kódot egy másik környezetben futtatni, egy MS Windows7 operációs rendszerrel rendelkező notebook-on VisualStudio2015 segítségével. Ekkor ugyan már a kód futtatható volt és nem kaptam semmilyen figyelmeztetést vagy hibaüzenetet, azonban látható eredményt sem adott. A VisualStudio, a Python és az openCV újra telepítése sem hozott eredményt. Majd segítséggel módosítottam a kódon, azonban úgy sem sikerült más eredményt elérnem.

Úgy vélem, hogy a kód felépítése és a gondolatmenet alapvetően jó, azonban a hibamentes futtatás, szemantikai vagy pedig fejlesztő környezethez köthető hiba miatt egyenlőre nem lehetséges.

Célom, hogy a programot működőképessé tegyem amennyiben logikai, teoretikai hibát tartalmaz. Jelenleg több megoldáson gondolkodom amely működhet: más verziójú Python és openCV telepítése, szemantikai hibák keresése vagy pedig a kód átültetése C++ nyelvre. Ha sikerrel járok és működőképessé válik a program célom annak továbbfejlesztése.

4 Felhasznált irodalom

[1] Gépi látás – Hollósi János, Széchenyi István Egyetem
[2] OpenCV team - https://opencv.org/about/
[3] OpenCV contents - https://docs.opencv.org/2.4/index.html
[4] Motion Fused Frames: Data Level Fusion Strategy for Hand Gesture Recognition - Okan Köpüklü, Neslihan Köse, Gerhard Rigoll
[5] Learning OpenCV – Gary Bradsky and Adrian Kaehler