Biztonsági kamera alkalmazás
Dokumentáció

Gépi látás (TKNB_INTM038)

Készítette: Fieszl Bence ISOJQW

Tartalomjegyzék

1 Bevezetés

Az általam választott beadandó feladat egy videófelvételen vagy kamerán egyidejűleg egy mozgást nyomon követő biztonsági kamerarendszer megalkotása. A rendszer egy megadott kamerából érkező, vagy egy megadott fájlból betöltött videófelvételen keres meg egy mozgást és követi nyomon azt.

A feladat egy fix kamerára van tervezve, ami mozgás esetén a mozgást végző objektumot egy zöld kerettel veszi körbe. Az alkalmazás Python programnyelven készült, OpenCV,Numpy,Pafy(youtube-dl) csomagokkal kiegészítve. Ezen kívül még használ pár, Pythonon belüli programkönyvtárat is(sys,time,argprase,os).

2 Megoldás elméleti háttere

A feladat megoldásához először is szükségünk van a mozgást végző objektumok kontúrjainak megtalálására. Ehhez viszont szükséges a mozgóképet olyan állapotra hozni, ami alapján már vizsgálható lesz. Ezt a program az egymás utáni képkockák elemzésével végzi.

Először is vesz 2 egymás utáni képkockát, amit szükséges átalakítani szürkeárnyalatosra a későbbi műveletek elvégzése érdekében. A szürkeárnyalat konverzió egy sima súlyozott átlagolás alapján végzi, aminél az eredeti kép összes képpontjának mindhárom csatornájának az átlagát veszi, és ez adja ki az új képen a fekete képpont intenzitását. Matematikailag ez a következőképpen néz ki:

$$J(x,y) = 0{,}3 \times I_R(x,y) + 0{,}59 \times I_g(x,y) + 0{,}11 \times I_b(x,y)$$

OpenCV-ben a következő képlet alapján történik a számolás:

$$J(x,y) = 0{,}299 \times I_R(x,y) + 0{,}587 \times I_g(x,y) + 0{,}114 \times I_b(x,y)$$

A szürkeárnyalati konverzió után következnie kell egy különbségképzésnek. Ehhez először is a 2 képkockának minden pixelét ki kell vonni a másikból.

$$J(x,y) = I_1(x,y) - I_2(x,y)$$

Mivel a képkockák így kieshetnek a]0;255[tartományból (negatív irányba), ezért egy abszolútérték vétel szükséges, hogy minden értéket visszajuttassunk a pozitív tartományba.

$$J(x,y) = |I(x,y)|$$

Célszerű alkalmazni egy Gauss-szűrőt, hogy csökkentsük a zajt a képen. A Gauss-szűrő működését tekintve egy konvolúciós művelet, ami az adott képkocka intenzitását a körülötte lévő képkockák intenzitásának súlyozásával számolja ki. A konvolúcióhoz szükség van egy kernelre, aminek az elemeit a következő képlet segítségével tudjuk kiszámolni

$$K(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Ezek után pedig konvolúcióval ki lehet számolni az adott képpontok milyen intenzitásúak lesznek

$$J(x,y) = K(u,v) * I(x,y)$$

A zaj csökkentése után pedig egy küszöbölést végzünk, hogy csak az éles mozgást végző kontúrok maradjanak meg a képen. A küszöbölés egy olyan művelet, hogyha a beállított küszöbértéken belül van az értékünk akkor azt felerősíti, ha kívül akkor nulla lesz.

$$J(x,y) = \begin{cases} 0, & I(x,y) < 20 \\ 255, & k\ddot{u}l\ddot{o}nben \end{cases}$$

A küszöbölés után a program végez pár morfológiai transzformációt, hogy a kontúrdetektáló algoritmusnak egyszerűbb legyen megtalálni a kontúrokat. Ezek közül is egy morfológiai nyitást, ami a zajok eltávolítására jó. A nyitás egy erózió és azután egy dilatáció.

$$(X \ominus B) \oplus B$$

Az erózió egy olyan művelet, ami akkor ad teljes intenzitást a képpontnak, ha annak a képpontnak a vizsgált képpontjai egybeesnek a strukturáló elem(B) **minden** hasznos pontjával.

$$X \ominus B = \{x : B_x \subseteq X\}$$

A dilatáció pedig egy olyan művelet, amikor a vizsgált képpont akkor kap teljes intenzitást, ha a kép adott helyén vett képpontjának vizsgált képpontjai egybeesik a strukturáló elem **legalább egy** hasznos pontjával.

$$X \oplus B = \{x: B_x \cap X \neq \emptyset\}$$

A nyitás után pedig még egy dilatáció szükséges, hogy a hibás kontúrokat ki lehessen javítani.

Ezek után a kontúrdetektálás következik, ami megkeresi a megfelelő kontúrokat, tehát az elmozdulásokat, amiket eltárol. Majd a kontúrok közül kiválasztja a legnagyobb területűt

$$\mathrm{c} = \max\left(contours, key = ContourArea\right)$$

vagy azt amelyik a legnagyobbak közül az előző észlelt mozgáshoz közel található. A távolságot két dologgal lehet megállapítani.

Először vizsgáljuk a kontúr közepét. A kontúr közepét úgy tudjuk kiszámolni, hogy vesszük a kontúr momentumait, és ezekből a következő összefüggés szerint megkaphatjuk a kontúrt középpontját (Cx,Cy).

$$\begin{bmatrix} Cx \\ Cy \end{bmatrix} = \begin{bmatrix} \frac{M_{10}}{M_{00}} \\ \frac{M_{01}}{M_{00}} \end{bmatrix}$$

Ezek után még megállapítjuk a kontúrt körbevevő négyzet jobb felső sarokpontjának a koordinátáját. Ha valamelyik pont megadott határértéken belül van

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

akkor a két elmozdulás egy mozgásnak tekinthető.

3 Megvalósítás

3.1 A megvalósítás menete

A program megvalósításának első lépése a futási környezet kiválasztása. A program nem GUI felülettel kommunikál a felhasználóval, hanem parancssorral, csak a kép megjelenítése történik grafikus felületen. A programmal a felhasználó argumentumokon keresztül tud kommunikálni, így elkerülhető az, hogy a programban kelljen „nyúlkálni", emellett gyorsabb és felhasználóbarátabb is.

Ehhez azonban szükséges deklarálni a parancssori argumentumok feldolgozásához szükséges programrészt. Az argumentumokat a Python beépített, argprase és sys programkönyvtáraival lehet kinyerni, és a program számára használható formára hozni. Először az argpraser argumentumait szükséges megadni, amiket használhat a felhasználó. Ezek tetszés szerint variálhatóak, hozzáadható, elvehető belőle bármelyik, jelenleg 6+1 argumentum van definiálva, **-c**, **-v**, **-s**, **-y**, **-a**, **-d** és -**h**. Ezek sorrendben a következőt tudják: kamera használata (itt meg kell adni az eszközkódot),videófile használata, online videostream használata, youtube link használata, minimálisan detektálandó terület, maximális távolság az egy mozgást végző objektumok között és a segítség menü.

Ezek után következik a programban az argumentumok vizsgálata, Először megnézi, hogy lett-e megadva kamera, ha igen akkor azt fogja használni input deviceként, ha nem akkor megy tovább a videófilera ha az meg van adva és létezik akkor azt használja, ha nem létezik akkor hibával kilép a programból. Ha ez sincs megadva, akkor megy tovább a videostreamre, ha az sincs megadva akkor megy tovább a youtube linkre, amit ellenőriz is, ha hibás a link akkor kilép a programkból, ha nem akkor a pafy nevű programkönyvtár segítségével letölti a youtube videót, és ezt a videót küldi tovább input adatként. Ha egyik sincs megadva, akkor megnyitja a súgó menüt és kilép a programból.

Ezek után következik a megadott forrás beolvasása **VideoCapture** fügvénnyel. A függvény egy **cap** nevű változóban tárolja az objektumot. Szükséges még deklarálni egy **x**,**y**,**cX**,**cY** változót amik majd a későbbiekben fognak szerepet vállalni a programban, alapértékük 0 lesz.

A változók deklarálása után szükséges az első képkocka beolvasása. Ezt a képkockát egy **frame** nevű változóban tároljuk, amivel később tudunk dolgozni. A megfelelő képkockasebességhez még szükséges egy **fps** érték is, amit a **cap** objektum **get** függvényével tudunk lekérni. a **get** fv. **CAP_PROP_FPS** argumentumával a megnyitott videóforrás képkockasebességét tudjuk lekérni. Történik itt még egy milliszekundum pontosságú időbeolvasás is, amit a **ms** változóban tárolódik. Ennek majd később lesz jelentősége a képkockák sebességének szempontjából.

Ezek után következik egy **while** ciklus, amit addig csinál, ameddig meg van nyitva a videóforrás (tehát ameddig van érvényes képkocka).

A cikluson belül először is eltárolja az előző képkockát a **frame2** változóba, és beolvas egy újabb képkockát a **frame** változóba. Azután következik egy elágazás, amiben megnézi, hogy sikerült-e képkockát beolvasnia. Ha nem sikerült akkor továbbmegy a program vége felé. Azonban, ha sikerült akkor belelép a felismerő programba.

Első lépésben mind a kettő BGR csatornás képkockát (**frame** és **frame2**) átalakítja a program szürkeárnyalatos képre (**cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)**), emellett pedig végez egy típuskonverziót is minden képpontra a numpy tömbben, hogy a kivonás esetén tudja kezelni a negatív értékeket is a program (**.astype(int)**).

A szürkeárnyalati konverzió után következik az egymás utáni 2 képkocka kivonása egymásból, amint a **f_sub** tömbben tárol a program. A kivonás után pedig, hogy minden érték a megfelelő tartományban legyen szükséges egy abszolútérték vétel az **np_abs()** fügvénnyel, amit az **f_abs** változóban tárol el.

A kivonást követi egy Gauss-szűrő, amit a **cv2.GaussianBlur()** függvénnyel lehet meghívni. Ez a szűrő egy 9x9-es kernelt használ arra, hogy a képkockákat a mellettük lévő képkockák alapján módosítsa konvolúciós lépésekkel. Így egy elmosottabb, de zajtalanabb képet kapunk. Ezt követi egy küszöbölés, ami ha 20-as intenzitás alatt van a kép akkor legyen 0 az intenzitása (**f_gauss.astype(int) > 20**, ez egy logikai értéket ad vissza, 0 vagy 1), azonban ha 20 felett, akkor legyen 255(az előző lépést meg kell szorozni 255-el).

A küszöbölés után következnek a morfológiai transzformációk. Először egy nyitás szükséges, hogy az esetleges maradék zajokat eltüntessük a képről. Ezt egy 3x3-as kernellel végezzük, ahol minden érték 1-es. Ezt kétszer végzi el a program (**cv2.morphologyEx(f_mov.astype(np.uint8), cv2.MORPH_OPEN, kernel, iterations=2)**). Ezután következik egy dilatáció, hogy a kontúrokat a felsimerő algoritmus jobban meg tudja találni (**cv2.dilate**), ez háromszor fut le.

Ezen műveletek után következik a kontúrkereső algoritmus (**cv2.findContours(f_dil, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)**). Az algoritmus megkeresi az összes kontúrt, ami van az adott dilatált képen, és ezeket eltárolja egy **contours** változóban.

Ezek után a program megnézi, hogy van-e eltárolva kontúr, ha van akkor elemzi őket, ha nincs akkor továbbmegy a megjelenítésre. Ha található kontúr, akkor ezeket előszöris egy **cns** tömbbe eltárolja kontúrméret (**cv2.contourArea**) szerint csökkenő sorrendben. Majd egy **c** változóba eltárolja a legnagyobb kontúr adatait, hogyha a következő lépések nem adnak eredményt akkor a legnagyobb kontúr lesz a mozgást végző kontúr.

A legnagyobb kontúr eltárolása után végigmegy a csökkenő listán, és nézi a kontúrokat. Előszöris a kontúrokat körbevevő „téglalapnak” a szélét határozza meg a **cv2.boundingRect** fv. segítségével, ami visszaadja a téglalap jobb felső koordinátákat és a téglalap méreteit. Ezek után az adott kontúr közepét is megállapítja (**cv2.moments** és a momentsekből az osztásokkal a kontúr közepe). Majd ezek után az előzőleg mozgást végző kontúr ezen adataival összeveti a jelenleg vizsgált kontúr adatait, és ha a beállított tűrésen belül vannak ezek az értékek, akkor a két mozgás „összetartozik”, egy mozgásnak vehető. Ez egyfajta optical flow vizsgálatnak is felfogható, bár attól teljesen különböző módszert használ, inkább csak hasonlóság van a működés szempontjából (a hasznos pontok az előző mozgás közepe és „téglalap” széle és ezt „tolja” el).

Ezek után megvizsgálja, hogy az adott kontúr, amit kiválasztottunk (legnagyobb, vagy a „mozgást végző” kontúr) megfelel-e a minimális kontúr méretnek (amit állíthatunk parancssorral, de az alapérték 500), ha igen akkor eltárolja a momentseket és a téglalap adatokat és a frame2 képre rajzol egy zöld téglalapot, ahol mozgás volt (**cv2.rectangle**).

Ezek után átméretezi a program a **frame2**-t hogy megfelelő nagyságú legyen a kép, ne lógjon ki a képernyőből. Majd ezek után az, hogy a program indulási idejét vizsgáljuk, mivel a tesztek alapján a program gyors működéséből kiindulva felgyorsítja a felvételeket, ezért szinkronizálni kell a képkockákat a videó frameratejével. Ezt egy egyszerű módon oldja meg a program, a kezdeti frameratéből kiszámolja, hogy egy képkockának hány ms időablakja van, és addig nem engedi megjeleníteni a következőt, ameddig az előző képkocka időablakja le nem járt, így már megfelelős sebességgel fogja visszajátszani a program a videófelvételeket.

3.2 Rövid programvázlat

Lényeges Változók:

- args: argumentumok
- inp: input mód
- cap: Capture objektum
- x,y: négyzet sarkai
- cX,cY: kontúr közepe
- frame, frame2: képkockák
- f_gray: szürkeárnyalatos kép
- f_sub: kivont képek
- f_abs: Értelmezési tartomány korrekció után
- f_gauss: Gauss-szűrt kép
- f_mov: Küszöbölt kép
- f_open és kernel: Morfológiai nyitás után a kép és hozzá a kernel
- f_dil: Dilatáció után a kép
- contours: kontúrok
- c: kiválaszottt kontúr
- x,y,w,h: négyzet sarka és méretei
- frame2rs: átméretezett kimeneti kép

A program futása:

- Argumentumvizsgálat
- Forrásbetöltés
- Képkockabeolvasás
- szürkeárnyalati konverzió
- kivonás és abszolútérték
- Gauss-szűrő
- Küszöbölés
- Morfológiai nyitás
- Dilatáció
- Kontúrkeresés
- Kontúrrendezés
- Legnagyobb kontúr megtalálása
- A legutóbbi mozgáshoz képest van-e határétéken belül mozgást végző kontúr
- Kontúrterület vizsgálata
- Rajzolás a képre
- Képkocka szinkronizálás
- Képkocka kirajzolása
- Vissza a képkockabeolvasáshoz

4 Tesztelés

A program tesztelve lett különböző esetekkel és videófelvételekkel. A program feldolgozási sebessége megfelelőnek mondható, elég gyorsan végzi el a műveleteket, nem késik a képkockákkal. A videókat is problémamentesen olvassa be, azonban a youtube livestream felvételekkel akadnak problémái, valószínűleg bufferelési gondokból adódóan 3-5 sec elteltével a youtube livestream felvételek befagyasztják a programot és vagy force quittel lehet kilépni, vagy megvárni ameddig 3-5 perc múlva a cv2 nincs beolvasandó képkocka hibaüzenettel leállítja a programot. De a sima youtube videók tökéletesen működnek a programmal.

A program tesztelése alap argumentumokkal történt, csak az input argumentum lett megadva

A youtube videók, amivel a program le lett tesztelve (ezek linkek szintén megtalálhatóak a test mappa yttest fileban):

https://www.youtube.com/watch?v=MNn9qKG2UFI Elfogadható mértékben ismeri fel és követi a mozgásokat, főleg torlódáskor és fókuszáláskor/rázkódáskor veszíti el az objektumokat.

https://www.youtube.com/watch?v=2dysaG-q6Lc A mozgást szinte tökéletesen felismeri, a lassú sebességnél kicsit későn észlelt, a takarás miatt nem teljesen tudta felismerni a teljes objektumot, mint mozgó objektum, de az eredmények elfogadhatóak.

https://www.youtube.com/watch?v=xeuWNm72YRg A mozgást követi, néha ugrál, mivel elveszíti az objektumot, illetve az objektum felületén is eléggé ugrál, az úton közlekedő járműveket a pózna kitakarása után nem követte le.

https://www.youtube.com/watch?v=mRe-514tGMg Live felvétel, kifagy pár másodperc után

https://www.youtube.com/watch?v=jjlBnrzSGjc A kamera mozgása miatt rendszeresen elveszíti a mozgást végző objektumot, emellett még a távolodás miatt nem is képes teljes mértékben lekövetni őket

https://www.youtube.com/watch?v=lZsDve8_DkM A sétáló férfit viszonylag jól követi, bár ugrál rajta a tracker, de a mozgását képes lekövetni

https://www.youtube.com/watch?v=CkVJyAKwByw A harmadik videón a fiút kicsit későn észlelte, az ugrálóváras videónál eléggé össze volt zavarodva, a hóesés és ebből kiindulva valószínűleg az eső be tudja zavarni, emellett a lassú mozgásokat nem olyan jól érzékeli, mint a gyorsakat.

A stream, amivel a program le lett tesztelve (a videó megtalálható róla a test mappában)

stream.mp4 Jól követi a mozgást, a kezem, folyamatos a stream local hálózaton

A kamera eszköz, amivel a program le lett tesztelve (A videók megtalálhatóak a test mappában)

cam.mp4 Jól követi a mozgást szintén, a tollat nem mindig követi csak a kezemet, itt kicsi csalás van, mivel a videojel streamben érkezik csak a droidcam applikációval emulálom kameraként, és így a számítógépen kameraként van jelen

A file, amivel a program le lett tesztelve (A videó megtalálhatóak a test mappában)

VID.mov Elfogadhatóan követi a mozgást, nem mindig az egész kezem, van, amikor szétesik a tracker de összességében elfogadható ahogy követi.

5 Felhasználói leírás

5.1 A program működéséhez szükséges programok és külső programkönyvtárak

- Python 3.8
    - NumPy 1.18.2
    - OpenCV 4.2.0.34
    - youtube-dl 2020.3.24
    - pafy 0.5.5

5.2 A program használata

- A program Githubról való letöltés után a main.py fileból indítható. Az indításhoz csak a main.py file szükséges
- Az indítást parancssorból kell végezni, **python main.py [argumentumok]** formában
- A programhoz használatos argumentumok
    - -c [SZAM] vagy --cam [SZAM] :Kamera bevitel választása, a számhoz a kamera eszközkódját szükséges megadni(alapértelmezésként 0)
    - -v [UTVONAL] vagy --video [UTVONAL] : Bemeneti videó abszolút vagy relatív útvonala
    - -s [URL] vagy --stream [URL] : Az online videóstream(RTSP, RTP stb…) linkje
    - -y [URL] vagy –youtube [URL] : Youtube videó linkje
    - -a [TERULET] vagy --min-area [TERULET] A minimális érzékelendő kontúr terület, alapból 500px
    - -d [px] vagy --distance [px] Az egy mozgást végző kontúrok közötti maximális távolság, alapból 10px
- Az argumentumok közül a területet és a távolságot nem kötelező megadni, viszont a bemeneti források közül egy forrást meg kell adni, különben a program nem indul el. Több forrás megadása esetén az előzőleg felsorolt lista szerinti sorrendben a legelsővel fog lefutni a program.
- A program folyamatosan fut, kilépni a q gombbal lehet belőle, az ablak bezárása esetén újranyílik

6 Felhasznált irodalom

[1] Órai diák és programkódok

[2] OpenCV Dokumentáció: https://docs.opencv.org/4.2.0/

[3] OpenCV Python Tutorials https://opencv-python-tutroals.readthedocs.io/

[4] Intel: Motion detection using OpenCV https://software.intel.com/en-us/node/754940

[5] Python dokumentáció:argrpase https://docs.python.org/3/library/argparse.html

[6] Basic motion detection and tracking with Python and OpenCV https://www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/