Gépi látás (TKNB_INTM038)

Biztonsági kamera

Kamera videófelvételen egyidejűleg egy mozgás felismerése és nyomon követése.

Zettis Dániel János

I2Y53C

Tartalom

Bevezetés

A modern világunkban kamerák találhatóak a számos ponton elszórva, gyakran folyamatosan bekapcsolva. Számos alkalmazási módjuk van, kezdve a forgalom figyeléstől és védett területen való behatolás megelőzésétől az önvezető autókig és videótelefonálásig. Az adatok feldolgozása pedig a rengeteg kamera képéből egyre nagyobb üzlet, a mozgás felismerése pedig kritikus szempont. A választott témámban erre, és a mozgás követésére keresem a megoldást.

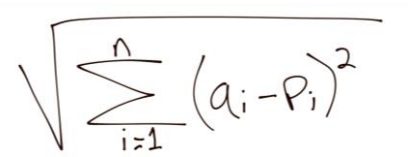
Az elkészült program valós időben képes megmutatni több, egyidőben zajló mozgást és nyomonkövetni azokat egy mozdulatlan kamera képén.

Olyan megoldást kerestem, amihez a lehető legkevesebb külső könyvtárat kell alkalmazni és törekedtem, hogy a kód egyszerű ás átlátható maradjon.

Elméleti háttér

A feladat megoldásához az **Euklideszi normált** választottam.

Az euklideszi távolság két képpont közötti egyenes távolság, és az euklideszi norma alapján kerül kiértékelésre. Pontosabban a két pont közötti különbség négyzetes összegének négyzetgyökeként számítjuk. Mindig nagyobb vagy egyenlő nullával.



1. ábra: kettes vagy euklideszi norma képlete

A képelemzésben a távolságtranszformáció az egyes tárgypontok távolságát méri a legközelebbi határtól, és fontos eszköze a számítógépes látásnak, a képfeldolgozásnak és a mintafelismerésnek.

Felhasználói dokumentáció

Indításhoz szükséges könyvtárak:

- NumPy
- OpenCV
- math

A program indításához szükséges fájlok:

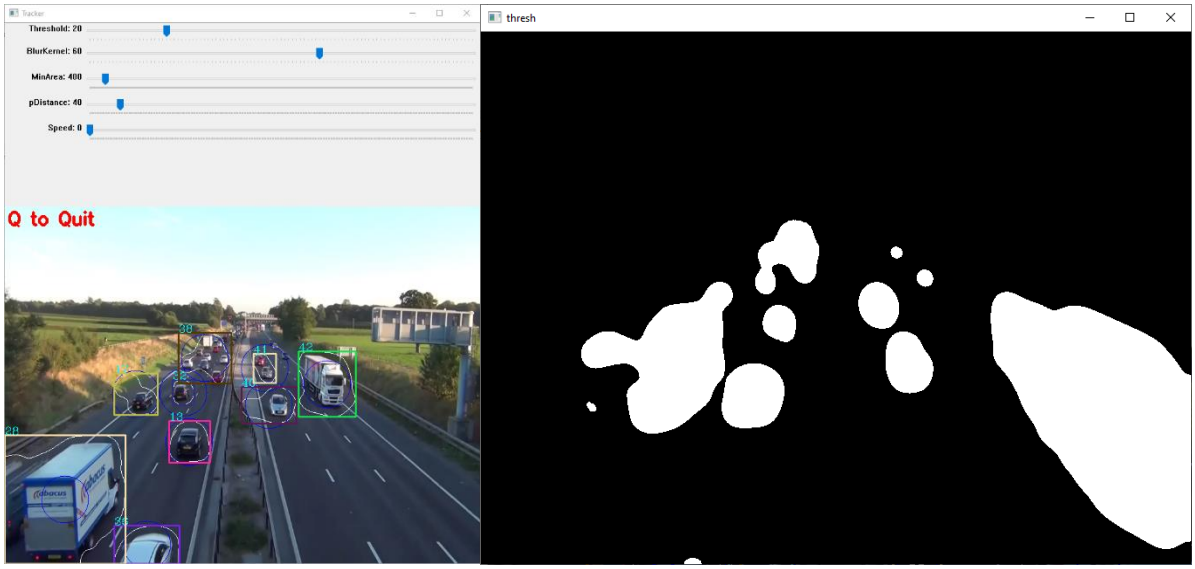
- motionTracker.py
- euclidianTracker.py
- tetszőleges videófájl

A githubról való letöltés és kicsomagolás után, a motionTracker.py fájlt egy python nyelv feldolgozására alkalmas programban be kell olvasni és futtatni. Az alapértelmezett „ball.mp4” helyére tetszőlegesen megadható más videó neve is, a videoPath változóban.

```
videoPath = 'videok/traffic1.mp4'
…
video = cv2.VideoCapture(videoPath)
```

A kiválasztott videófájlon a program automatikusan megkeresi az egyes képkockák közötti különbséget, amit felcímkéz és megmutatja színkódolt négyzetek segítségével.

A program folyamatosan ismétli a felvételt, bezárni a lejátszást a ’q’ billentyű megnyomásával lehet.



2. ábra: GUI

A program két ablakot jelenít meg, a „Tracker” és „thresh” neveken.

A Tracker a valós videóképet mutatja a mozgó objektumokkal, ahol lehetőségünk van finom hangolni a keresést, a thresh ablak pedig megmutatja, hogy a változtatások milyen eredményt adnak a „szűrökön”.

elnevezés	**alapérték**	**maximum**	**funkció**
Threshold	20	100	Minden pixelre küszöbértéket állít, így befolyásolja a „jó” pixelek mennyiségét.
BlurKernel	60	100	A homályosítás mértékét állítja.
MinArea	400	10000	A minimális négyzet területét szabályozza. Növelésével elkerülhető a zajok címkézése.
pDistance (distance of points)	40	500	Mekkora távolságban vegyen egynek két pontot. Növelésével nagyobb mozgást végző pontok is követhetőek.
Speed	450	499	A képkockák közötti ezredmásodperc 500-ból kivont értéke. Csökkentésével lassul, növelésével gyorsul a felvétel.

3. ábra: Csúszkák

Megvalósítás

A program elkészítéséhez az OpenCV 4.5.5, NumPy 1.21.5 és Python 3.10.2 verzióját, hozzátartozó math könyvtárat használtam, az indításához is szükséges Thonny fejlesztőkörnyezetben.

A program a könyvtárak importálásával kezdődik.

```
import cv2
import numpy as np
```

A program indulást követően elkezdi egyesével beolvasni a videó képkockáit, azaz lejátszani a videót és az két függvény segítségével megkeresi rajta a kontúrokat.

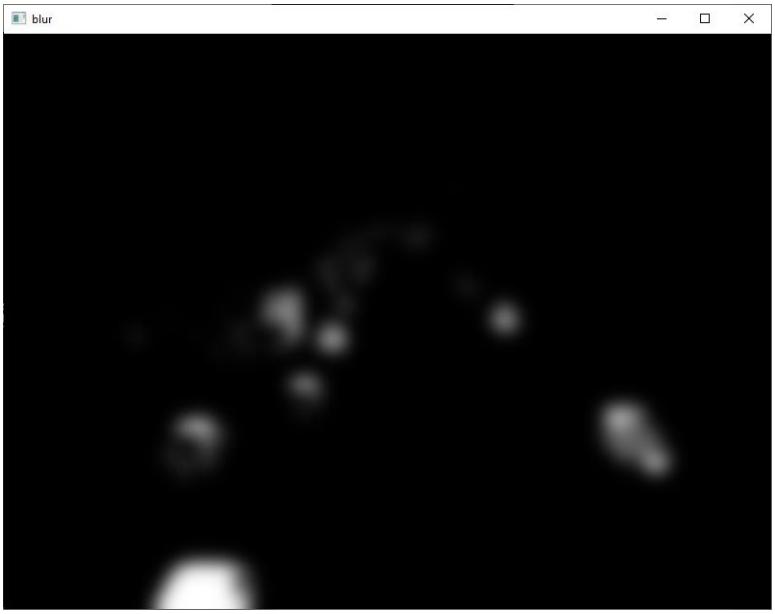
```
def callBack(x):
…

def findContours(frame):
…
```

A `callBack(x)` szükséges, hogy csúszkákon lévő érték frissüljön, a `findContours(frame)` pedig a kapott képen `(frame)` keresi kontúrokat.

```
def findContours(frame):
    diff = MOG.apply(frame)
    blur = cv2.GaussianBlur(diff, (blurKernel,kernel), 0)
    _, thresh = cv2.threshold(blur, threshold, 255,
    cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
    cv2.imshow("thresh", thresh)
    return contours
```

4. ábra: diff kép

A `diff = MOG.apply(frame)` a `MOG = cv2.createBackgroundSubtractorMOG2(history=100, varThreshold=70)` alkalmazása a képen, képes az algoritmus elkülöníteni a mozgó és álló, mozdulatlan részeket egy képen, így láthatóvá téve az aktivitást. A két paramétere, a `history=100` és a `varThreshold=70` pedig megadja, hogy hány képkockát vizsgáljon és mekkora eltérést enged az egyes pixelek mozgásában.



5. ábra: blur kép

```
blur = cv2.GaussianBlur(diff, (blurKernel,blurKernel), 0)
    _, thresh = cv2.threshold(blur, threshold, 255,
    cv2.THRESH_BINARY)
```

A következő sor a zajt hivatott eltüntetni a képről Gauss-szűrő segítségével. A paraméterei a következőek: a `diff` az előzőleg már átalakított kép. `(blurKernel,blurKernel)` egy `blurKernel` * `blurKernel` méretű mátrixot jelent, ami Gauss-eloszlást követ, vagyis a szomszédok nagyobb súllyal szerepelnek. A végén a `0` pedig a szegély nélkülözése.

```
_, thresh = cv2.threshold(blur, threshold, 255, cv2.THRESH_BINARY)
```

A bemenete szintén az előzőleg módosított kép, a `threshold` a *Threshold* nevezetű csúszka értéke. A `255` a maximálisan beállítható érték lenne, míg a `cv2.THRESH_BINARY` a küszöbölés milyensége, jelen esetünkben kétállású. A `threshold` alatt 0, felette 255 értéket ad vissza.

6. ábra: thresh kép

```
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
```

Kontúrokat a `cv2.findContours(…)` függvény segítségével keresünk a képen. A `thresh` bemeneten kívül megadjuk neki, hogy csak a külső kontúrokat adja vissza a `cv2.RETR_EXTERNAL` és a csúcsok tárolását a `cv2.CHAIN_APPROX_SIMPLE` paraméterrel, így gyorsabb megvalósítást elérve. A **`return`** `contours` pedig visszatér a kontúrok tömbjével.

A következő lépésben ezeken a talált elemeken megyünk végig.

```
for contour in contours:
        area = cv2.contourArea(contour)
        if area > minArea:
            cv2.drawContours(frame,[contour],-1,(255,255,255),1)
            x,y,w,h = cv2.boundingRect(contour)
            cv2.circle(frame, ((x + x + w) // 2, (y + y + h) //
            2), int(distanceOfPoints), (255, 0, 0), 1)
            centers.append([x,y,w,h])
```

Az `area` meghatározza a kontúr területét, amit a `minArea` segítségével megszűrünk. Lényege, hogy a túl kicsi, zajból adódó kontúrok ne kapjanak körvonalat. A `cv2.boundingRect()` jelölőnégyzetet rak a megfelelt elemekre. A `cv2.circle()` segítségével pedig a *pDistance*, azaz két pont távolságvizsgálatának mértékét láthatjuk. Majd a `centers.append([x,y,w,h])` segítségével eltárolom azon kontúrok négyzetének az adatait, amik megfeleltek.

A Tracker osztály végzi a pontok középpontjából számolt távolság alapján az osztályozást.

```
    def Euclidean(self, objects, distanceOfPoints):
        objectsIDs = []
        for (x, y, w, h) in objects:
            cx = (x + x + w) // 2
            cy = (y + y + h) // 2
            sameObject = False
            for id, pt in self.centerPoints.items():
                dist = math.hypot(cx - pt[0], cy - pt[1])
                sameObject = False
```

Az `Euclidean(…)` bemenetként a négyzetek koordinátáit és egy skaláris számot vár.

Első lépésben meghatároz két középpontot a `cx` és `cy` változókban. A `math.hypot()` az előbbi kettő és a futás során már eltárolt középpontok segítségével határozza meg az euklideszi távolságukat.

```
                if sameObject is False:
                    self.centerPoints[self.idCount] = (cx, cy)
                    objectsIDs.append([x,y,w,h, self.idCount])
                    self.idCount += 1
```

Ellenőrzi, és amennyiben még nincs detektált elem, úgy új középpontokat vesz fel a `objectsIDs.append([…])`.

```
            if dist < distanceOfPoints:
                self.centerPoints[id] = (cx, cy)
                objectsIDs.append([x,y,w,h, id])
                sameObject = True
                break
```

Amennyiben új elemet talál, ahhoz az elemhez azonosítót rendel.

```
        newCenterPoints = {}
        for objectsID in objectsIDs:
            _, _, _, _, newID = objectsID
            center = self.centerPoints[newID]
            newCenterPoints[newID] = center
        self.centerPoints = newCenterPoints.copy()
```

Eltávolítja a már nem használt azonosítókat és frissíti az elemek listáját.

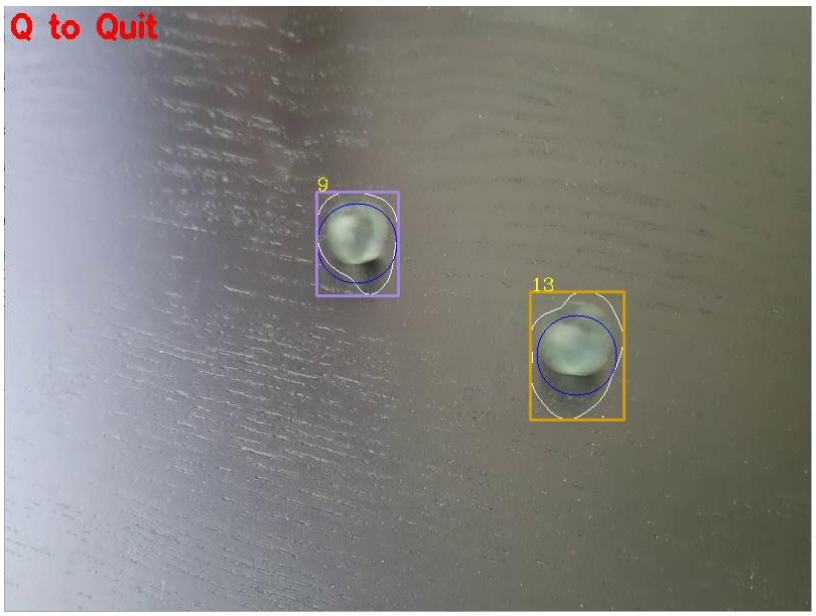
```
for i in centerSort:
        x,y,w,h,id = i
        cv2.rectangle(frame,(x,y),(x+w,y+h),color[id].tolist(),2)
        cv2.putText(frame,str(id),(x,y - 1),
        cv2.FONT_HERSHEY_COMPLEX, 0.6, (0,255,255),1)
```

Végül a talált pontok középpontjához egy véletlen színű négyzetet rajzol, és kiírja az azonosító számát.

Tesztesetek

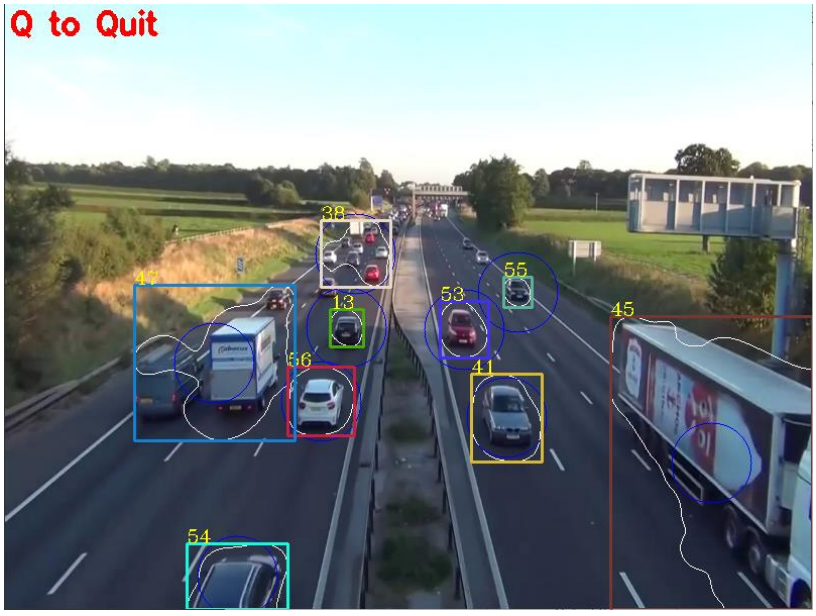
A programot gyorsaságra megfelelő, képes valósidőben visszaadni a történéseket.

1. Az első teszteset egy általam rögzített üveggolyók felvétele stabilnak mondható kamerán. A golyók elkülönülve, szépen látszanak, nagy mértékű hiba nem tapasztalható. Alapbeállítások.



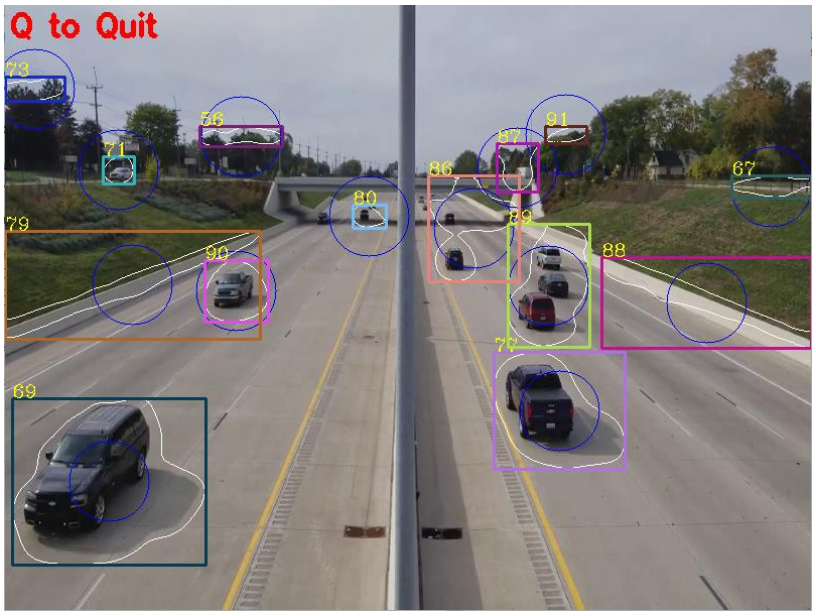
7. ábra: balls.mp4

2. A második teszteset egy rögzített autópálya felvétele. Az autókat többnyire jól felismeri, bár a távolban lévők, illetve az egymáshoz közel haladó járművek problémája jól látszik. Az alapbeállításokat használtam.



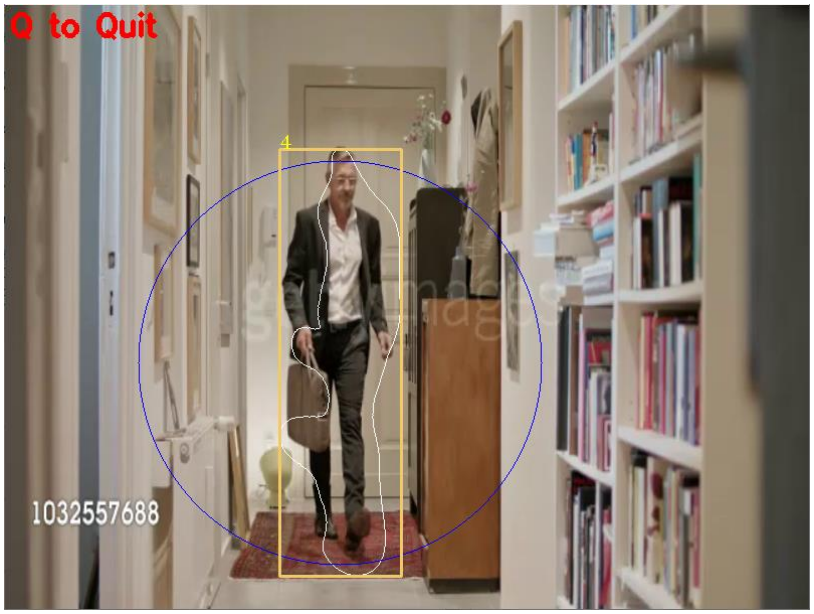
8. ábra: traffic1.mp4

3. A következő egy hasonló, bár kis mértékű mozgást végző felvétel. Látható, hogy nem létező, pontokat is jelöl, pusztán a kamera elmozdulásából adódóan. Szintén alapbeállításokkal.



9. ábra: traffic.mp4

4. Egy egyetlen embert követő mozgáshoz elfogadható követéséhez, szükség volt a *BlurKernel* növelésére 100-ra hogy ne vegyen észre kisebb, például ajtónyitásnyi mozgást, a *MinArea* 4000-re állítására a közeli kamerakép miatt és *pDistance* 200-as értékekre.



10. ábra: oneman.mp4

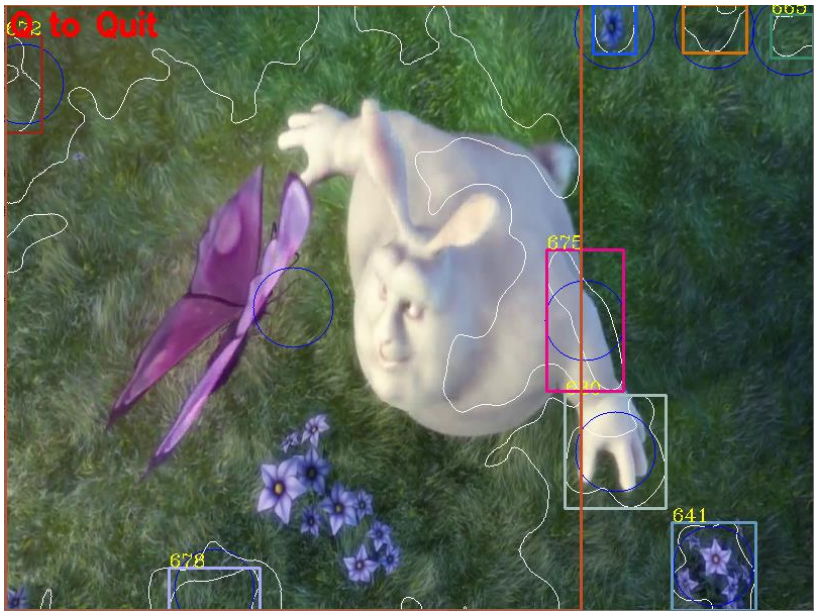
5. Az ötödik egy több embert egyszerre és külön-külön is megörökítő felvétel. Sajnos ideális beállításoknál is nehezen tesz különbséget két egymás mellett elhaladó ember között, vagy túl alacsony *BlurKernel* értéknél több pontot is rendel egy emberhez. A *MinArea* ajánlott hogy legyen legalább 7000, a közeli kamerakép miatt.

11. ábra: fewman.mp4

6. Az utolsó felvételen egy embertömeg látható. Sajnos az előzőhöz hasonlóan, nem sikerült megfelelő beállításokat találni, gyakorta a tömegtől jól elkülönülő egyének vagy a világosság elött elhaladó mozgás érzékelése. A *Threshold* 10, *BlurKernel* 65, a *MinArea* 90, a *pDistance* 20.

12. ábra: manyman.mp4

7. A teljesen használhatatlan eset. A videón a kamera, a környezet és követendő elemek is folyamatosan változásban vannak.



13. ábra: bunny.mp4

A tesztelést követően a következőket állapítottam meg a program működésével kapcsolatban.

- A kameramozgást nem kezeli.
- Jól elkülöníthető elemek követésében a leghatékonyabb.
- A beállítások finomhangolásával érdemes foglalkozni, de nem minden esetben hoz kellő eredményt.

Felhasznált források

[1] https://docs.python.org/3/library/math.html

[2] https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

[2] https://docs.opencv.org/4.x/d6/d17/group__cudabgsegm.html

[4] https://pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/

[3] https://www.includehelp.com/python/math-hypot-method-with-example.aspxmog

[4] https://data-flair.training/blogs/abandoned-object-detection/

[5] https://gobertpartners.com/how-does-euclidean-distance-work/

[6] https://www.analyticsvidhya.com/blog/2022/04/building-vehicle-counter-system-using-opencv/