

# Gépi látás- GKLB\_INTM038

## Contents

Bevezetés, megoldandó feladat kifejtése (5%).....	1
Elméleti háttér (30%) .....	2
Step#1 – Pozitív minta .....	2
Step#2 – Negatív minta.....	2
Step#3 – Pixelenkénti HOG jellemzők.....	2
Step#4 – Cellánkénti HOG jellemzők, hisztogram.....	2
Step#5 – Blokk normálás.....	3
Step#6 – HOG jellemzők összegyűjtése és SVM modell tanítása .....	4
Megvalósítási terv és kivitelezés (40%).....	4
Github repository .....	4
Adatok, minták.....	4
Alkalmazások.....	4
Build instrukciók.....	4
Alkalmazások leírása .....	4
Teszt (20%).....	6
Tesztelési terv .....	6
Teszteredmények.....	7
Képek.....	7
Vizualizált eredmények .....	7
Teszteredmények kiértékelése .....	10
Mutatók csoport szintű bontásban.....	10
Felhasználói leírás (5%).....	10
Követelmények .....	10
Installáció/Build .....	11
Programfutás .....	11
References .....	11

## Bevezetés, megoldandó feladat kifejtése (5%)

A választott feladat típusát tekintve objektumdetektálásként sorolható be. A fő feladat olyan alkalmazás készítése, mely drónfelvétel alapján képes a felvételen előforduló embereket detektálni, illetve az előfordulásokat összeszámolni. Elsődleges use case egy olyan program készítése, mely képes egy utcai demonstráció képei alapján összeszámolni a résztvevők számát.

Az objektumdetektáláshoz a HOG (histogram of oriented gradients) + SVM (support vector machine) módszer került alkalmazásra, melynek fő előnye a viszonylagos egyszerűsége, módszertani eleganciája. A modell tanításához saját képi adatbázis készült, illetve ez kiegészült az internetről ingyenesen elérhető képekkel, melynek célja az alkalmazás robusztusságának javítása.

A feladat OpenCV könyvtárt használja és C++-ban van implementálva, Linux operációs rendszer alatt. Fontos, hogy feladatban a valós embereket LEGO™ figurák helyettesítik, a tanuló algoritmus is ezekkel lett tanítva. A programnak képesnek kell lennie egyaránt videóstream és képek alapján is dolgozni, ezeket inputként kezelni. A feladat tekinthető egyfajta járókelő detektálási problémának is (pedestrian/human detection), melynek kiterjett irodalma található az interneten, illetve az OpenCV könyvtár is nyújt beépített detektorokat, melyek a `getDefaultPeopleDetector()` és `getDaimlerPeopleDetector()` függvényekkel hívhatók. Ezek szándékosan nem kerültek alkalmazásra, mivel egyrészt ezek valós emberek detektálására lettek készítve, másrészt cél volt a kész modulok számának alacsonyan tartása.

## Elméleti háttér (30%)

A HOG (histogram of oriented gradients) módszertan kiegészítve SVM modellel, nálal Navneet és Bill Triggs 2005-ös munkája után lett népszerű és vált széles körben alkalmazottá. A módszer lényege a következő pontokban foglalható össze:

### Step#1 – Pozitív minta

Pozitív minta alatt értjük azokat a képeket, melyeket a tanuló algoritmus használ és melyeken szerepel az objektum, melynek detektálására a feladat irányul. Ezekből a képekből nyerhető ki a HOG jellemzője a detektálandó objektumnak, melyeket az SVM modell használ a detektáláshoz.

### Step#2 – Negatív minta

Azon képek halmaza, melyek nem tartalmazzák a detektálandó objektumot, így az SVM modell ezek alapján tanulja meg, hogy melyek azok a HOG jellemzők, melyek nem a detektálandó objektumhoz tartoznak.

### Step#3 – Pixelenkénti HOG jellemzők

A mintákban található képekből a HOG jellemzők kinyerése. A mintát alkotó képek 64x128-as méretben vannak elmentve. Az eredeti tanulmányban is ez a méret van, mivel ebben kényelmesen elfér egy ember és a valódi embereket szimuláló LEGO™ figurák esetében is megfelelő.

A HOG jellemző tulajdonképpen a pixelenkénti gradienst jelenti. Ennek egy x és y (vízszintes és függőleges) irányú komponense ( $G_x$  és  $G_y$  gradiens) az adott pixel szomszédjainak intenzitásából számítható. A  $G_x$  és  $G_y$  meghatározzák a gradiens nagyságát (Magnitude) és szögét ( $\theta$ ):

$$G_x(r, c) = I(r, c + 1) - I(r, c - 1) \text{ és } G_y(r, c) = I(r - 1, c) - I(r + 1, c)$$

, ahol  $r, c$  a sorokra és oszlopokra vonatkoznak

$$\text{Magnitude}(\mu) = \sqrt{G_x^2 + G_y^2} \text{ és } \theta = \arctan \frac{G_y}{G_x}$$

### Step#4 – Cellánkénti HOG jellemzők, hisztogram

A pixelenkénti HOG jellemzőket egy 8x8 (vagy 4x4) cellákba rendezve előállíthatók a cellára jellemző gradiens komponensenkénti (magnitude – nagyság és direction – szög) mátrixok. A mátrixokban a szögekre vonatkoztatott értékkészlet 0-360 fok, hossz/nagyság értékkészlet 0-255. Szögek esetében

jellemző ugyanakkor, hogy csak a 0-180 fokos értéktartomány szerepel (unsigned gradients), mivel a 0-360 fokos tartomány nem javítja érdemben a modelleket.

A cellára vonatkozó 2 mátrix értékeiből előállítható a cellára számított hisztogram, mely a 0-180 fokos (szög)értékeket tartalmazza, jellemzően 20 fokos beosztásban, összesen 9 kategóriát meghatározva a hisztogramon:

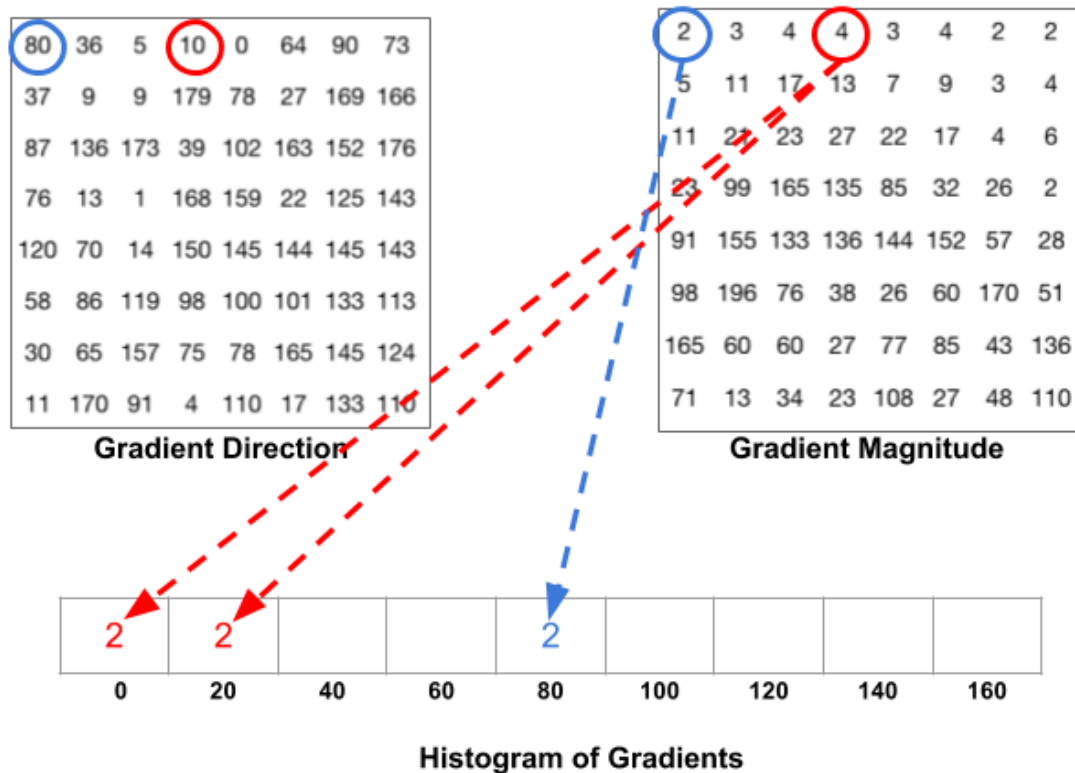


Figure 1 – Gradiens mátrixok és hisztogram 8x8-as cellára számítva, Forrás: [Histogram of Oriented Gradients explained using OpenCV \(learnopencv.com\)](https://www.learnopencv.com/histogram-of-oriented-gradients-explained-using-opencv/)

A késsel jelölt értékek következő módon kerülnek besorolásra a hisztogramban: a szög értéke alapján kiválasztjuk a megfelelő hisztogram kategóriát. A szögérték 80, ami pontosan az 5. kategóriát (bucket-et) jelenti. A kategóriának megfelelő helyre beírható a gradiens mértékét mutató érték (Magnitude), ami 2.

A pirossal jelölt elemek esetében nem egyértelmű a besorolás, mivel a szögérték (10) pontosan 2 kategória (0 és 20) közé esik a hisztogramon. Ezért ennek az elemnek a nagysága a hisztogram osztályokba esés mértékének megfelelően két részre bontható. Mivel a 10-es szögérték pontosan a két hisztogram osztály határára esik, így fele-fele arányban tartozhat a 0 és 20 hisztogram osztályba. Tehát a 4-es méretből 2 kerül a 0-ás és 2 kerül a 20-as osztályba.

#### Step#5 – Blokk normálás

A kialakított hisztogramok normálása annak érdekében, hogy az eredmény a fényviszonyokra kevésbé legyen érzékeny. A normáláshoz egy nagyobb blokk választható a stabilabb eredményért. Ezt az OpenCV algoritmus automatikusan elvégzi, feltéve, hogy a paraméterezésnél figyelembe van véve a 16x16-os alapérték, és az előző lépésben használt cellák osztói ennek a 16x16-os alapértéknek (tehát 8x8 vagy 4x4 stb. megfelelő választás az előző lépésben).

## Step#6 – HOG jellemzők összegyűjtése és SVM modell tanítása

Az összes lépés elvégzése után lehetséges a HOG jellemzők összegyűjtése és az SVM modell illesztése.

## Megvalósítási terv és kivitelezés (40%)

### Github repository

A GitHub repository az alábbi linken érhető el: <https://github.com/lefodor/peoplecounter>

A dokumentum írása pillanatában privát, de a 2021/22 tavaszi félév végén és az értékelés befejeztével publikusan is elérhető lesz.

### Adatok, minták

A mintákhoz saját, generált adatok kerültek alkalmazásra, nem történt külső forrásból származó adatok használata. A minta megoszlását az alábbi táblázat mutatja:

Pozitív minta	Negatív minta	Teljes minta
#339 (45%)	#415 (55%)	#754

A pozitív minta szétbontható abból a szempontból is, hogy milyen perspektívából láthatók a detektálandó figurák:

Perspektíva	Leírás	Darabszám #
Front + Fourtyfive	Teljesen szemből, vagy szemből, ~45 fokos magasságból	282 (83.18%)
Above	Felülről készült kép	21 (6.2%)
Side	Oldalnézetből készült kép	36 (10.62%)

A képek 64x128-as felbontással készültek (width x height).

### Alkalmazások

#### Build instrukciók

Az applikációk build-je CMAKE-kel történik. A Github repository-ban minden applikációhoz megtalálható a CMakeLists.txt file, mely tartalmazza a build-hez szükséges beállításokat. Fontos, hogy a CMake.txt file alábbi sorában az installált OpenCV könyvtár elérési útvonala szerepeljen.

```
set(OpenCV_DIR "~/opencv/build")
```

A könyvtárban, melyben a letöltött forrásfájlok vannak, létre kell hozni egy build alkönyvtárat, és innen végrehajtani a

```
cmake --build .
```

parancsot és a – például a hogpedestrians esetében – ./hogpedestrians parancssorba beírásával indítható a program.

### Alkalmazások leírása

#### Hogpedestrians

Megadott input pozitív és negatív minták alapján kiszámítja a HOG jellemzőket, SVM modellt illeszt és output file-ba menti a készített HOG struktúra elemeit.

#### Argumentumok:

- pos\_dir: pozitív minta abszolút v. relatív elérési útvonala
- neg\_dir: negatív minta abszolút v. relatív elérési útvonala

- `obj_det_filename`: output file (.yml) elérése, ahova elmenti
- `detector_width`, `detector_height`: mintákhoz használt képméret (64x128), jelenleg minden képnak azonosan 64x128 pixel méretűnek kell lennie.

### Függvények

**`void ResizeBox(cv::Rect& box)`**: Detektálások köré rajzolt téglalapot átméretezi, nincs használatban.

**`std::vector< float > get_svm_detector(`**

**`const cv::Ptr< cv::ml::SVM >& svm )`**: Egy SVM modell segítségével inicializálja egy HOGDescriptor objektum detektorját.

**`void load_images(`**

**`const cv::String & dirname,`**

**`std::vector< cv::Mat > & img_lst,`**

**`bool showImages = false )`**: Dirname könyvtárban megadott image file-okat betölti egy image vektorba.

**`void sample_neg(`**

**`const std::vector< cv::Mat > & full_neg_lst,`**

**`std::vector< cv::Mat > & neg_lst,`**

**`const cv::Size & size )`**: negatív minta esetén megengedett, hogy a mintakép eltérjen a 64x128 pixeles képmérettől. Ebben az esetben ez a függvény véletlenszerűen kiválaszt egy ekkora méretű részletet az eredeti képből és azt használja a mintában.

**`void computeHOGs(`**

**`const cv::Size wsize,`**

**`const std::vector< cv::Mat > & img_lst,`**

**`std::vector< cv::Mat > & gradient_lst, bool use_flip )`**: előállítja a HOG jellemzőket a pozitív és negatív minták alapján

**`void convert_to_ml(`**

**`const std::vector< cv::Mat > & train_samples,`**

**`cv::Mat& trainData )`**: előkészíti az adatokat, az SVM modellhez

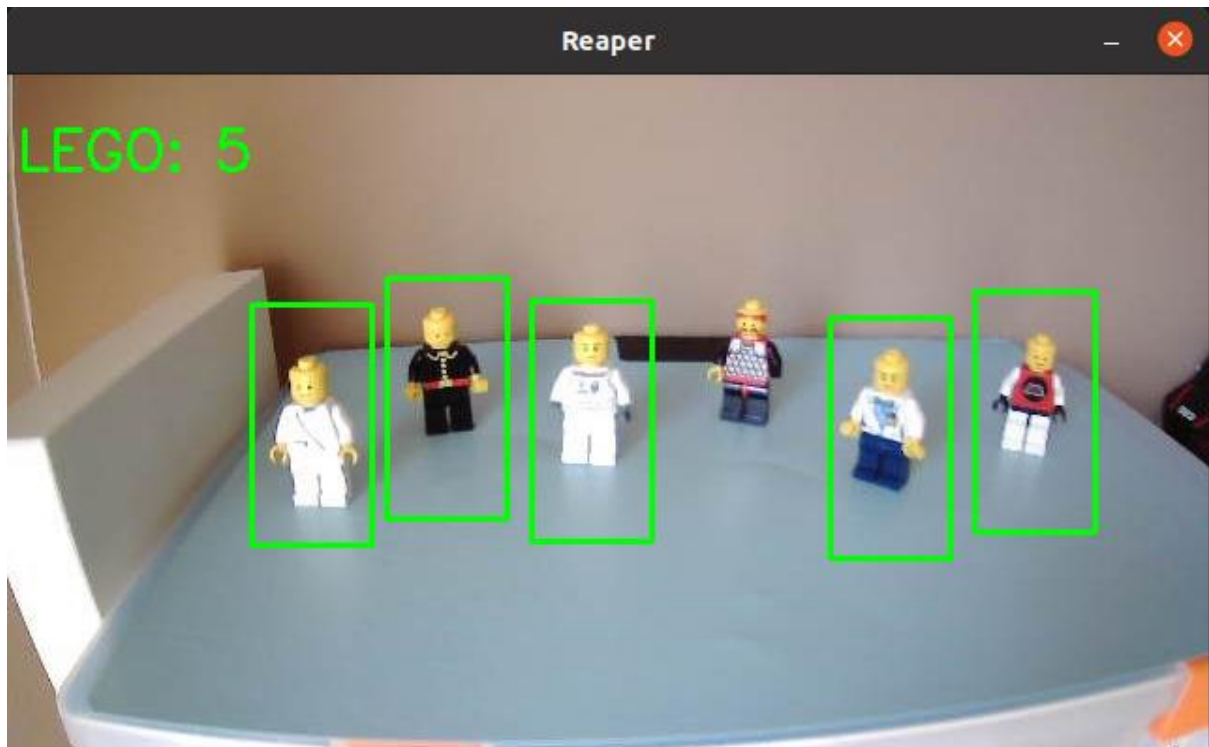
### Hogtesting

Parancssorban megadott képfájltra lefuttatja a megadott objektumfelismerő modellt (detectionoutput.yml).

### Argumentumok

- `obj_det_filename`: .yml file elérhetősége, melyben a hogpedestrians program által elmentett output szerepel
- parancssorban megadott képfájl elérési útvonala

Példa a kimenetre:



#### Hogteststream

Webkamerás stream-re lefuttatja a megadott objektumfelismerő modellt (detectionoutput.yml).

#### Argumentumok

- `obj_det_filename`: .yml file elérhetősége, melyben a hogpedestrians program által elmentett output szerepel

Példa a kimenetre:



hogteststream\_demo  
811.avi

## Teszt (20%)

### Tesztelési terv

Az alkalmazás tesztelése képekre lett végezve a Github repo-ban található hogtesting és hogeststream programokkal. Alapvetően az alkalmazás használata az alábbi scénáriókban kerültek tesztelésre:

- Front: a detektálandó objektum a kamerával szemben helyezkedik el – 4 tesztadat
- Fourtyfive: a detektálandó objektum a kamera alatt ~45 fokos szögben helyezkedik el - 7 tesztadat
- Above: a detektálandó objektum a kamera alatt helyezkedik el (nem pontosan 90 fokos szög, de megközelíti a derékszögű beesést) – 4 tesztadat
- Structure: a detektálandó objektumok valamilyen struktúrán vannak elhelyezve (épület szimulálása) – 3 tesztadat

A front és fourtyfive csoportban levő teszteseteknél szerepelnek átfedések esetek is, ezek kiértékelő táblázatban külön (o) szimbolummal vannak jelezve.

## Teszteredmények

Az eredmények a következő részekben található táblázatokban kerülnek bemutatásra. A táblázat oszlopai:

- Actual: ténylegesen mennyi LEGO figura helyezkedik el a képen, ezeket kell detektálni
- Detected#: korrekt detektálások száma
- Detected, False+: hibás pozitív.
- Result: százalékos érték, mutatja hány százalékos detektálási arány az adott tesztadat esetében. Számítás:  $\text{Detected\#} / \text{Actual}$ .

A False+ oszlopban a (\*) jelölés mutatja, ha a hibás pozitív észlelés elkerülhető lenne a non-maxima suppression módszer alkalmazásával, ami az átfedő detektálások esetében összevonja ezeket. Ez nem került implementálásra az alkalmazásban. Ebben az esetben 1 korrekt detektálás átfed 1 fals pozitív észleléssel, amit a módszer kiszűrne. A csillagok száma azt jelzi, hány fals pozitív lenne elkerülhető, ezek a fals pozitív detektálások tulajdonképpen 1 korrekt észlelés duplikációinak tekinthetők.



## Képek

Group	Name	Actual #	Detected (#, False+)		Result (%)
Front	front_12_37_17	3	2	0	66.67%
Front	front_12_37_42	3	2	1(*)	66.67%
Front	front_12_38_06 (o)	3	2	1(*)	66.67%
Front	front_12_38_30 (o)	2	0	0	0%
Fourtyfive	fourtyfive_12_39_00	3	3	1(*)	100%
Fourtyfive	fourtyfive_12_39_14	2	2	1(*)	100%
Fourtyfive	fourtyfive_12_40_05	5	4	1	80%
Fourtyfive	fourtyfive_12_40_37 (o)	3	3	0	100%
Fourtyfive	fourtyfive_12_40_42 (o)	2	1	0	50%
Fourtyfive	fourtyfive_12_41_04 (o)	3	1	0	50%
Fourtyfive	fourtyfive_12_41_22 (o)	2	2	0	100%
Above	above_12_45_37	5	0	0	0%
Above	above_12_45_40	3	1	0	33.33%
Above	above_12_45_45	4	0	0	0%
Above	above_12_45_49	4	0	0	0%
Structure	structure_12_57_40	5	3	3(*)	60%
Structure	structure_12_58_18	5	4	2	80%
Structure	structure_12_58_41	5	3	4(**)	60%


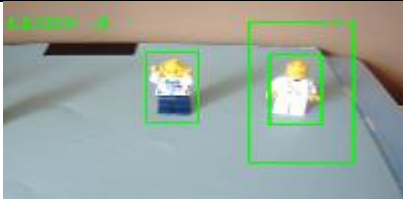




## Vizualizált eredmények

Group: Front


Name	Image output
front_12_37_17	
front_12_37_42	

front_12_38_06 (o)		
front_12_38_30 (o)		

*Group: Fortyfive*

Name	Image output	
fourtyfive_12_39_00		
fourtyfive_12_39_14		
fourtyfive_12_40_05		
fourtyfive_12_40_37 (o)		
fourtyfive_12_40_42 (o)		
fourtyfive_12_41_04 (o)		



fourtyfive_12_41_22 (o)	
-------------------------	--

*Group: Above*

Name	Image output
above_12_45_37	
above_12_45_40	
above_12_45_45	
above_12_45_49	

*Group: Structure*

Name	Image output
structure_12_57_40	
structure_12_58_18	
structure_12_58_41	

## Teszteredmények kiértékelése

### Mutatók csoport szintű bontásban

Group	Mean Result %	Mean False+ (%)
Front	50%	25%
Fourtyfive	83%	15.79%
Above	8.33%	0%
Structure	66.67%	54.54%

Mean False+ (%) =  $\sum \text{Detected, False+} / \sum (\text{Detected\#} + \text{Detected, False+})$ , vagyis csoporton belül összes detektálás hány százaléka fals pozitív.

A csoportszintű aggregált mutatókól látható, hogy a LEGO felismerő alkalmazás a Front és Fourtyfive csoportokban teljesít jól (a front csoportban a front\_12\_38\_30 jelzésű kép ad 0%-t, ami elrontja az egyébként 66.67%-os eredményt). A kapott számok visszatükrözik a tanulási minta jellemzőit, abban az értelemben, hogy a mintában szereplő legtöbb elem a tesztelésnél használt Front és Fourtyfive (a teljesen szemből és 45 fokos magasságból) csoportokból kerültek ki.

A Structure csoport is jól teljesít, ezt tulajdonképpen a Front és Fourtyfive csoportokba sorolt képek keveréke, megtoldva egyéb elemekkel, amelyek a magasabb, 54.54%-os fals pozitív találatokat okozzák. A részeredmény alapján elmondható, hogy szükséges lehet a negatív minta bővítése olyan adatokkal, melyekben a teszteseteken látható struktúrák szerepelnek, illetve értelemszerűen olyan környezeti elemekkel, melyek alkalmazás során szintén fals pozitív eredményeket hozhat.

A minta arányain is szükséges lehet módosítani, mivel a minta jelenlegi megoszlása pozitív/negatív részre 45/55%-os, tehát hozzávetőlegesen 1:1 az arány. A szakirodalomban általában 1:10, de minimum 1:4 szerepel a negatív mintaelemek javára.

További javítási lehetőségek:

- Átfedő detektálások összevonására non-maxima suppression módszer alkalmazása, ezzel csökkenthető a fals pozitív detektálások száma.
- Detektor kalibrációja, a detektor paraméterei jelenleg állandóak, de lehetséges ezek futásidőben történő dinamikus változtatása, a detektor körülményekhez / üzemmódhoz való kézi kalibrálása. (+) lehetséges mindez automatikus módon is.
- Külső adatok használata az adatbázis (tanulási és tesztelési, validációs) bővítésére, valamint pozitív / negatív mintaarány javítása (eltolása negatív irányba).

## Felhasználói leírás (5%)

### Követelmények

1. OpenCV könyvtár 4.x verzió. Letöltés, dokumentáció:  
<https://opencv.org/>  
<https://docs.opencv.org/4.x/>
2. Github repository, mindhárom könyvtár szükséges:  
<https://github.com/lefodor/peoplecounter>
  - hogpedestrians: modellillesztő program
  - hogtestimg: illesztett modell alkalmazása képen
  - hogteststream: illesztett modell alkalmazása videó inputra (kamerastream)
3. Cmake a program build-hez.

## Installáció/Build

Github repository 3 könyvtárának letöltése, majd az inputnak megfelelő alkalmazás build-je CMAKE-kel:

1. CmakeLists.txt fájlban az OpenCV könyvtár megfelelő elérési útvonalát meg kell adni.
2. build könyvtár létrehozása, pl.: ../hogtestimg/build és ugrás ebbe a könyvtárba
3. cmake konfiguráció futtatása (példa: `$ cmake ..` )
4. cmake build futtatása (példa: `$ cmake --build .` )
5. alkalmazás meghívása adott fájlra (példa: `$ ./hogtestimg "c:/sample.jpg"` )

## Programfutás

A program nyit egy új ablakot, melyben a megadott inputkép másolata látható, világoszöld téglalappal bekeretezve a megtalált célobjektumokat, a bal felső sarokba kiírva a találatok számát. A megjelenített kép az ESC billentyű lenyomásával eltűnik és a program kilép.

## References

Dalal, N., & Triggs, B. (2005). *Histograms of Oriented Gradients for Human Detection*. Forrás: INRIA: <http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf>

Mallick, S. (2016. December). *LearnOpenCV*. Forrás: LearnOpenCV: <https://learnopencv.com/histogram-of-oriented-gradients/>

Mallick, S. (dátum nélk.). *LearnOpenCV*. Forrás: Read, Write and Display a video using OpenCV CPP and Python: <https://learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/>

OpenCV. (dátum nélk.). *OpenCV 4.x documentation*. Forrás: [docs.opencv.org/4.x/examples.html](https://docs.opencv.org/4.x/examples.html): <https://docs.opencv.org/4.x/examples.html>

Rosebrock, A. (2014. November 10). *pyimagesearch*. Forrás: [pyimagesearch.com](https://pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/): <https://pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/>

tochiVision. (dátum nélk.). *youtube.com*. Forrás: <https://www.youtube.com/watch?v=cvGEWBOOVho>