



M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Irányítástechnika és Informatika Tanszék

M8 – Sávkövetés

MÉRÉSI ÚTMUTATÓ

IRÁNYÍTÁSTECHNIKA ÉS KÉPFELDOLGOZÁS
LABORATÓRIUM 1.

Szemenyei Márton, Reizinger Patrik

Tartalomjegyzék

1. Autonóm járművek	2
1.1. Sávdetektálás	2
2. A mérés környezete	5
2.1. A Python nyelv	5
2.2. OpenCV	9
2.2.1. Adattípusok	9
2.2.2. Numpy tömbök manipulációja	9
2.2.3. Képek olvasása és megjelenítése	10
3. Mérési feladatok	12
4. Hasznos kódrészletek	13
5. Ellenőrző kérdések	16

1 Autonóm járművek

Napjainkban az autonóm járművek egyre hangsúlyosabb szerepet töltenek be a mindennapi életben. Kontrollált ipari környezetekben már jó néhány éve alkalmaznak ilyen robotokat, azonban a számítógépes látás és a mesterséges intelligencia rohamos fejlődésének köszönhetően már az utcákon is megjelentek önvezető autók formájában. A legmagasabb autonómiai szinttel rendelkező járművek fejlesztése azonban a jelen labor tárgyán messze túlmutat, így a mérés során a látás alapú autonóm viselkedések legegyszerűbb formájával, a sávdetektálással és -követéssel fogunk megismerkedni.

1.1. Sávdetektálás

A sávdetektálás elvégzésére alapvetően két fontos módszer létezik. Az egyik a sávok intenzitás alapú detektálása, míg a másik a sávok élkeresés alapján történő megtalálása. Mindkét módszer megbízhatósága önmagában kérdéses, így gyakorta szokás a kettő módszer párhuzamosan, egymás korrigálására felhasználni. Az intenzitás alapú detektálás legegyszerűbb módja a küszöbözés, amikor egy előre meghatározott küszöbérték egyik oldalán lévő pixeleket 0-ba, míg a másik oldalon lévőket 1-be állítjuk, így egy bináris képet kapunk. Fontos megjegyezni, hogy egy előre meghatározott küszöbérték használata esetén a fényviszonyok megváltozása a módszer eredményét is nagymértékben befolyásolhatja, így a gyakorlatban gyakran használunk adaptív – az adott képen lévő intenzitások eloszlásából meghatározott – küszöbértéket.

A Sávdetektálás elvégezhető képi élek segítségével is, melyek definíció szerint a képen található szomszédos pixelek között végbemenő nagy mértékű, egy irányú intenzitás változások. Lényeges tulajdonságuk, hogy az intenzitás csak az egyik irányban változik, míg a másikon konstans, valamint, hogy a változás éles, ugrásszerű. Az éldetektálás során rendkívül gyakran előfordul, hogy különféle egyszerű alakzatok (téglalap, kör) határvonalait keressük, amely esetben célszerű lehet a megtalált élpontokra egy egyenes modellt illeszteni, így a képen megtalált egyenes szerű elrendezésben található pontokat egy paraméteres modellel leírhatjuk, amely az alakzatok detektálását rendkívül megkönnyíti.

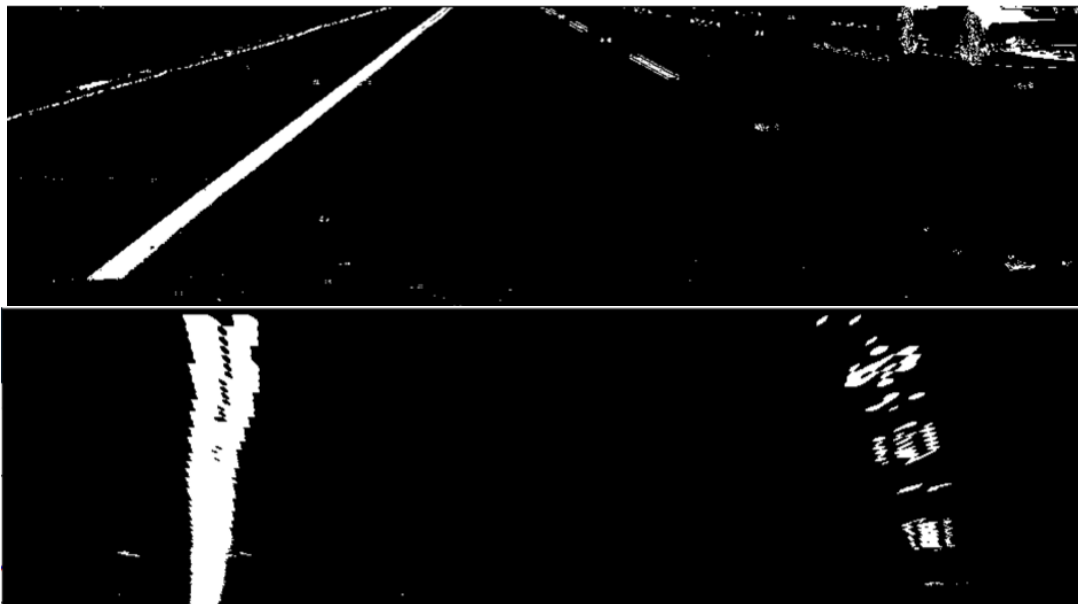
A probléma nehézsége, hogy természetesen a képen talált élpontok csak egy része fog egyenesekre illeszkedni, és azok közül is számos pont külön-külön egyenesre illeszkedik, így egyszerre kell azt meghatároznunk, hogy mely pontok illeszkednek egy egyenesre, és hogy milyen paraméterek írják le ezt az egyenest. Ha két kérdés közül bármelyikre tudnánk a választ, a probléma megoldása triviális volna. Erre a problémára az egyik legnépszerűbb algoritmus a Hough transzformációra épülő alakzat detektálás. A küszöbözés, éldetektálás, valamint a Hough-transzformáció részletei megtalálhatók a Számítógépes Látórendszerek c. tárgy jegyzetében [1].

Az egyenesek meghatározása a sávdetektálás szempontjából rendkívül hasznos lépés lehetne, azonban a valóságban a megtalált sávok számos esetben nem egyenes alakúak, hanem az út görbületének megfelelő módon változnak. Éppen ezért ilyen esetekben a Hough transzformáció nem fog pontos eredményt adni, ráadásul az egyenes görbületét sem képes meghatározni. E megfontolásokból érdemes egy másik, alternatív megoldást alkalmazni a sávok élképből történő meghatározására.

Ez a módszer első lépésként egy perspektív transzformációt alkalmaz a képen, aminek segítségével a képet egy számunkra kedvezőbb formára alakítjuk. A perspektív transzformáció egy olyan általános projektív transzformáció, amely egy tetszőleges kétdimenziós síkot egy másik kétdimenziós síkra vetít le. Ha belegondolunk, a képalkotás során pontosan ez történik: az autópálya síkja a kamera képsíkjára vetül. Ennek a transzformációnak az egyik alapvető hatása, hogy számos geometriai jellemzőt (méret, szögek, párhuzamosság) torzít, melynek következtében a képen látható sávokat meghatározó egyenesek nem párhuzamosak. Ha azonban ezt a transzformációt meg tudjuk határozni, és valamilyen módon visszacsinálni, akkor egy sokkal könnyebben feldolgozható képet kapunk.

A perspektív transzformáció meghatározása rendkívül könnyen elvégezhető: mivel a transzformáció mátrixa 3×3 -as, és ez egyik elemét szabadon megválaszthatjuk, ezért elég egyszerűen négy pontpárt kiválasztanunk, melyek értékeiből a transzformáció meghatározható. Ez a gyakorlatban ezt jelenti, hogy kiválasztunk az eredeti képen négy pontot, majd megadjuk, hogy hova szeretnénk, hogy ezek a pontok kerüljenek a transzformáció után. Mivel a jelen esetben célunk, hogy a sávot meghatározó két egyenes az új képen függőleges legyen, így ez a négy pont célszerűen a sáv széleinek 2-2 végpontja.

Miután ez a transzformált kép előállt, előállítunk egy speciális hisztogramot: a kép minden oszlopához összeszámoljuk, hogy hány darab egyes pixel található az adott oszlopban. Mivel a sávok görbülhetnek, ezért célszerű csak a kép alsó, a járműhöz közeli felén elvégezni ezt a számlálást. Ha az így kapott hisztogram két maximumát megkeressük, akkor ebből megkaphatjuk a sávok pozícióit. A maximumok értékéből



1.1. ábra. Az él kép perspektív transzformáció előtt és után.

következtethetünk arra is, hogy az adott sáv folytonos vagy szaggatott felfestésű. Az eljárás utolsó lépéseként külön-külön összegyűjtjük a két megtalált vonalhoz tartozó pontokat az egész képről, majd a legkisebb négyzetek módszerének segítségével másodfokú polinomot illesztünk a pontokra, melyből a sáv görbülete meghatározható.



1.2. ábra. A perspektív transzformáció eredménye kanyarodó út esetén.

2 A mérés környezete

A mérés során a *PyCharm* elnevezésű IDE áll rendelkezésre, amely rendkívül sokoldalú szolgáltatásokkal könnyíti meg a szoftverfejlesztést, például konfigurálható automatikus formázási lehetőségek állnak rendelkezésünkre. További részletekért érdemes lehet a JetBrains ide vonatkozó weboldalát [2] felkeresni. Függvények, objektumok esetében a **Ctrl+P** billentyűkombináció pop-up segítségként szolgálva mutatja nekünk a paramétereket. A mérés során használt programnyelv a Python 3-as verziója lesz.

2.1. A Python nyelv

A Python programnyelv egy *interpretált szkriptnyelv* (a saját bytekódját hajtja végre), mely alapvetően az *objektumorientált* paradigma által vezérelve került kialakításra. Jelenleg kétféle, kismértékben eltérő főverziója érhető el, ezek közül a mérésen a 3-as verziót használjuk majd.

Az OOP szemlélet olyannyira központi szerepet tölt be Pythonban, hogy minden változó objektumnak tekinthető – ami azt jelenti, hogy az integer és float típusok is objektumok. A nyelv számára jelentős hátrány adatintenzív számítási feladatok elvégzésekor ennek ténye, ugyanis arról beszélünk, hogy a Python **nem rendelkezik natív számtípussal**. Nem véletlen, hogy a hatékony modulok alacsonyabb szintű nyelveken nyugszanak, így a következő alfejezetben ismertetett PyTorch is.

További nem szokványos jellemzője a nyelvnek, hogy **nem erősen típusos**, a változónevekhez futási időben rendelődik hozzá a referált objektum (azaz egy *változónév* igazából egy referens, vagyis adott *példányra mutató referenciát* tartalmaz). Habár Pythonban a referenciák teljesen úgy viselkednek, mint a változók (vagyis semmilyen szintaktikai kiegészítésre nincs szükség, még argumentumok átadása esetében sem, mint ahogy azt C++-ban láttuk). (A félreértések elkerülése végett fontos szem előtt tartani, hogy amennyiben a segédletben a továbbiakban változó szerepel, akkor is igazából referencia van a háttérben, ezért hangsúlyozott általában, hogy nem a változó, hanem a változónév tartalmazza az objektumra mutató referenciát.)

Vagyis lehetőség van arra, hogy egy referenshez (változónévhez) a programban különböző típusú objektumokat rendeljünk hozzá – ez teljesen logikusnak tűnik, ha

belegondolunk abba, hogy a „Pythonban minden objektum” korábbi kijelentés lényegében azt sejteti, hogy minden az objektum őosztály leszármazottja. A referenciák kezelése referenciaszámláló segítségével történik – a koncepció analóg a például C++-ban megtalálható *shared_ptr* típus esetén használttal.

Pythonban nincsen továbbá pointer sem, az argumentumok átadása **referencia szerint** történik. Itt azonban meg kell különböztetnünk az objektumokat az alapján, hogy módosíthatóak-e. A következőket érdemes alaposan átgondolni, különben hibás működésű programot kaphatunk.

Kétféle objektumtípus létezik, **módosítható** (mutable), ill. **nem módosítható** (immutable). Nem módosíthatóak többek között az egyszerű adattípusok (POD, plain old data), mint az egész vagy lebegőpontos számok, valamint a sztringek. Habár a gondolatmenet e megfontolás mögött elsőre furcsának tűnik, a következőképpen magyarázható: minden adott szám vagy szöveg egyedi, vagyis ha pl. két változóhoz hozzárendeljük az 5 értéket, akkor a kettő tartalma egymással teljesen azonos, ha az egyiket megváltoztatjuk, akkor az már egy másik objektum lesz: nem egy tagváltozót módosítunk, hanem magát az objektumot teljes mértékben és kizárólagosan azonosító elemet. Talán érdemes a fordított programnyelvekből egy szemléletes példát átgondolni: a fordító ugyanolyan értékeket helyettesít be a gépi kódba (érdeklődők számára: a Pythonnak esetében is elérhető egyfajta disassembly a `dis` modul segítségével).

Módosítható lényegében minden egyéb típus, így a Pythonba beépített konténer jellegű típusok, mint `list`, `dict` (ezek különlegessége, hogy nem csak egy típust képesek egyidejűleg magukba foglalni, hanem bármilyen objektumot), de a saját osztályok példányai is ide tartoznak. Ebben az esetben a módosítás nem eredményezi új objektum létrehozását. A `tuple` egy különleges eset, ugyanis ez a konténer típus immutable, azonban, ha mutable objektumokat tartalmaz, akkor azok módosulhatnak.

Ezek a különbségek objektumok másolása esetében is jelentkezik. Fontos különbség, hogy Pythonban alapvetően a hozzárendelés (assignment) igazából C++-szemszögből inkább a `copy constructor` hívásának feleltethető meg. Immutable esetben az eredeti objektumhoz tartozó referenciaszámláló kerül megnövelésre, módosítás esetén pedig új objektum jön létre, értelemszerűen ugyanazt az értéket tartalmazó változók egyikének megváltoztatása nem hat ki a többire. Mutable esetben azonban nem ilyen egyszerű a helyzet: mivel alapvetően referenciákat tartalmaznak a változónevek, amelyek módosítható objektumokat referálnak, így ugyanarra a példányra mutató referenciák bármelyikének módosítása változást eredményez a referált egy darab objektum esetében, vagyis bármelyik változóval hivatkozunk rá, a változás mindegyik esetben látható lesz számunkra. Az ilyen jellegű másolást *shallow copy*-nak szo-

kás nevezni, melynek párja a deepcopy (elérhető a copy modulban), ami Pythonban mutable esetben is új példányt hoz létre, így az új objektum független lesz a többitől.

Paraméterek átadása hozzárendeléssel történik, ez azonban nagy objektumok esetében sem okoz komolyabb problémát, ugyanis a referenciák kerülnek csak másolásra. Ez a láthatóságra a következőképpen van hatással: ha a paraméter immutable és a függvénytörzsben módosításra kerül, akkor lényegében a függvény scope-jában egy ideiglenes objektum kerül létrehozásra, a függvényből visszatérve a változó megtartja eredeti értékét. Mutable esetben, mivel a referencia kerül másolásra, az objektumok másolásánál láttuk, hogy az általuk referált objektumok módosítása érvényes, bármelyik referenciájával hivatkozunk is rá, így a függvény visszatérését követően az objektum már módosult értékével használható. Rendkívül hasznos tulajdonság, hogy a Python gyakorlatilag bármennyi visszatérési értéket támogat. A nyelvi koncepciók ismertetése után a következőkben a szintaktikai részletek kerülnek összefoglalásra.

Pythonban a programkód tagolása indentálással történik, vagyis a kódblokkokat egy tabulátorral beljebb kell kezdeni (ha valamilyen okból üres függvényt, ciklust, stb, kívánunk írni, akkor is kell egy indentált blokk, ezt egy sorban, a pass utasítással valósíthatjuk meg, ami nem hajt végre semmilyen műveletet).

Modulok betöltésére az import utasítással van lehetőségünk, mégpedig kétféle módon: importálhatjuk a teljes modult, ekkor a modul minden osztálya/függvénye a modul neve után írt „.” (pont) operátorral érhető el, ha from-ot használunk, lehetőségünk van csak egyes elemeket betölteni, ekkor a modul nevét nem kell az importált elem neve elé kiírni.

```
import module
m = module.MyClass()

import module as md
m = md.MyClass()

from module import MyClass, my_func
m = myClass()
```

Függvények a következő módon hozhatók létre:

```
def func(x):
    x += 1
    print("x = ", x)
```

A def kulcsszó után a függvény neve, majd a paraméterlista kerül megadásra, azt követően pedig az indentált függvénytörzs következik.

Osztályok esetében sem bonyolult a konstrukció:


```
class my_class(object):  
    def __init__(self):  
        self.x = 5
```

Pythonban a konstruktort az `__init__` (2-2 aláhúzással) rutin testesíti meg, mint látható, a tagfüggvények is majdnem teljesen megegyeznek az általános függvényekkel, azzal a különbséggel, hogy az első argumentum mindenképpen az adott példányra vonatkozik (mint ahogy a `this` C++-ban) – ezt konvenció szerint `self`-nek szoktuk nevezni. Öröklés esetén nincs más teendők, mint az `object` osztály helyett megadni az általunk választott őssztályt, majd a konstruktorban meghívni az őssztály konstruktorát a `super`, általánosan az őssztályra használható objektum segítségével.

```
class base_class(object):  
    def __init__(self):  
        print("I am Groot")
```

```
class inherited_class(base_class):  
    def __init__(self):  
        super().__init__()  
        print("I am inherited")
```

Aki mélyebben érdeklődik a Python nyelv iránt, annak érdemes felkeresnie további példaprogramokat és kódrészleteket [3], valamint a TMIT SmartLab blogjának bejegyzéseit [4], [5] (angolul).

A Python programnyelvhez számos hasznos függvénykönyvtár tartozik, melyek a mérési feladatok megvalósítását nagymértékben megkönnyítik. A Python nyelv egyik rendkívül kényelmes funkciója a beépített package manager, amelynek segítségével az egyes könyvtárak automatikusan telepíthetők, telepítsük után pedig minden további beállítás nélkül használhatók. A Pythonhoz két ilyen package manager is tartozik, az egyik a Pip, amely a legtöbb telepíthető Python verzió mellé automatikusan települ, a másik pedig az Anaconda [6], ami a könyvtárkezelési funkciókon túl virtuális környezeteket is képes kezelni.

A Python egyik legfontosabb függvénykönyvtára a Numpy, amely tömbök kezelésére, illetve számtalan numerikus algoritmus használatára ad lehetőséget. A Numpy funkcionalitását kiegészíti a Matplotlib, melynek segítségével különböző ábrákat készíthetünk a tömbjeinkről. Egy harmadik rendkívül hasznos könyvtár család a scikit, ami számos tudományos számítható szükséges alkönyvtárt foglal össze. A scikit-image képek kezelésére, a scikit-learn gépi tanulás algoritmusok használatára, míg a scikit-fuzzy fuzzy logika használatára ad lehetőséget. Ezek a könyvtárak tulajdonképpen együttesen kiadják a Matlab funkcionalitásának jelentős részét.

2.2. OpenCV

Az OpenCV [7] egy nyílt forráskódú számítógépes látás algoritmusokat tartalmazó függvénykönyvtár. Az OpenCV elsődleges nyelve a C++, azonban elérhetőek hozzá hivatalos wrapperek többek között Java és Python nyelven. Az OpenCV rengeteg hivatalosan támogatott algoritmust tartalmaz, melyen felül a külön letölthető Contrib modulban harmadik felek által kifejlesztett további funkciók is elérhetők.

2.2.1. Adattípusok

Az OpenCV könyvtár a Python verzióban a Numpy könyvtár által definiált n -dimenziós számtömböket (ndarray) használja a képek tárolására. Ezek a tömbök különböző adattípusokat tartalmazhatnak, méretük minden kép esetén $c \times H \times W$, ahol c a csatornák, H a képsorok, W pedig az oszlopok száma. Az egyszerűbb képi adattípusok egy azonosítóval is definiálhatóak az alábbi formában:

$$\text{CV_}<\text{bitmélység}>\text{U|S|FC}<\text{csatornák száma}>$$

Bár tömbök segítségével közvetlenül is tárolhatnánk többdimenziós adatokat, a csatornák számának közvetlen megadása kényelmesebbé és szemléletesebbé teszi az adattárolást és programozást. Például színes képek esetén három csatornára van szükségünk, esetleg négyre, ha az időt is tárolni akarjuk. Példák:

- CV_32F: 32 bites lebegőpontos szám
- CV_8UC3: 8 bites 3 csatornás szám/kép

2.2.2. Numpy tömbök manipulációja

Üres 3×4 tömb létrehozása:

```
arr = np.ndarray((3,4))
```

Nullákkal/egyesekkel feltöltött 3×4 tömb létrehozása:

```
arr = np.zeros((3,4))  
arr = np.ones((3,4))
```

Nullákkal/egyesekkel feltöltött, egy másik tömbbel megegyező méretű és típusú tömb létrehozása:

```
arr1 = np.zeros_like(arr)
arr1 = np.ones_like(arr)
```

Lista-tömb konverzió:

```
arr = np.array ([[1,2,3],[4,5,6]])
```

Típuskonverzió:

```
arr1 = arr.astype('float32')
arr1 = arr.astype('float64')
arr1 = arr.astype('uint8')
arr1 = arr.astype('int16')
```

Tömb mérete és adattípusa:

```
arr.shape
arr.dtype
```

Tömb eleméhez történő hozzáférés:

```
arr [2,2] # egy elem
arr [-1,-1] # utolsó sor utolsó eleme
arr [1:4,2:5] # részmátrix kiemelése (FONTOS: A felső limit nem inkluzív, vagyis az [1-4) sor
            és a [2-5) oszlop vannak benne)
arr [1:3,:] # [1-3) sorok összes eleme

rowInd = [1,3,5,11]
arr[rowInd] # Az 1,3,5 és 11 sorok összes eleme

colBin = [True,True,False,True,False,False]
arr[:, colBin] # az összes sor 0,1 és 3 oszlopai
```

2.2.3. Képek olvasása és megjelenítése

Az első programunk egy keretprogram, melyet a mérés során folyamatosan módosítunk, bővítünk. A program rendkívül egyszerű, az OpenCV imread(), namedWindow() és imshow() függvényei megjelenítik egy ablakban a képet, majd az „ESC” gomb hatására program visszatér az operációs rendszerhez.

```
import cv2
import numpy as np

img = cv2.imread("image.jpg",cv2.IMREAD_COLOR)
cv2.imshow("Image",img)
cv2.waitKey(0) # Waits for keyboard press - necessary after imshow
```

A `cv2.IMREAD_COLOR` egy logikai változó (ún. flag), mely megmondja az `imread` függvénynek, hogy a képet színesként, három csatornán töltsse be. Alternatívaként, a `cv2.IMREAD_GRAYSCALE` flag használatával lehetőség van a képet szürkeárnyaltosként, egyetlen csatornán betölteni. Fontos opció továbbá a `cv2.IMREAD_UNCHANGED` melyet gyakran használunk RGB-D kamerák esetén a mélység kép betöltésére, mivel ezek általában egy csatornás, 16 bites képek, így a többi flag esetén elvégzett 8 bites konverzió adatvesztéshez vezetne.

A mérés során az eredmények mentéséhez és a mérési jegyzőkönyvhöz szükségünk lesz a képek kiírására, amihez az alábbi utasítás használható:

```
cv2.imwrite("img_out.jpg",img)
```

3 Mérési feladatok

A mérés folyamán az alábbi feladatokat kell elvégezni:

1. Készítsen eljárást a sávokat jelentő élek detektálására! Az éldetektálás során kapott gradienseket szűrje nagyság és irány alapján, valamint végezzen szín alapú szűrést is!
2. Torzítsa a képet perspektív transzformáció segítségével úgy, hogy a sávok párhuzamosak legyenek, majd határozza meg a sávok helyzetét!
3. Készítsen robusztus becslőt a sávok görbületének, az autó helyzetének és ez utóbbi változásának meghatározására!
4. Valósítson meg egy sávtartó algoritmust egyszerű fuzzy irányítás segítségével!
5. Ellenőrizze az algoritmus helyes működését előre felvett videofelvételek segítségével!

4 Hasznos kódrészletek

Videó betöltése

```
clip = cv2.VideoCapture("original.mp4")
```

Képkocka olvasása videóból

```
success, img = clip.read()
if not success:
break
```

Szürkeárnyaltos konverzió

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Sobel operátor futtatása x és y irányban

```
sobelx = cv2.Sobel(gray, cv2.CV_32F, 1, 0, ksize=sobel_kernel)
sobely = cv2.Sobel(gray, cv2.CV_32F, 0, 1, ksize=sobel_kernel)
```

Abszolút érték, maximum számolása

```
abs = np.absolute(array)
max = np.max(abs_sobelx)
```

Típuskonverzió

```
converted = np.uint8(array)
```

Tartományon belüli küszöbözés

```
binary = cv2.inRange(rarray, low_thresh, high_thresh)
```

Atan2 tömbön

```
dir = np.arctan2(y, x)
```

Elemenkénti logikai függvények

```
result = np.zeros_like(a)
combined[((a == 255) & (b == 255)) | (c == 255)] = 1
```

Képrészlet kivágása

```
roi = image[y:Y, x:X]
```

Perspektív transzformációhoz szükséges pontok

```
src = np.float32 ([[380, 0],[875, 235],[60, 235],[470, 0]])
dst = np.float32 ([[150, 0],[800, 260],[150, 260],[800, 0]])
```

Perspektív transzformáció és az inverzének számítása

```
M = cv2.getPerspectiveTransform(src, dst)
Minv = cv2.getPerspectiveTransform(dst, src)
```

Kép transzformálása

```
warped = cv2.warpPers
```

Hisztogram számítása

```
histogram = np.sum(warped[warped.shape[0]//2:,:], axis=0)
```

Hisztogram megjelenítése

```
plt.figure()
plt.plot(histogram)
plt.show()
```

Tömb shiftelése

```
a = np.roll(a,1)
```

Átlag számítás

```
avg = np.mean(a)
```

LS becslés

```
lineEst = np.linalg.lstsq(X,Y, rcond=None)[0]
```

Fuzzy változó

```
universe = np.linspace(-2, 2, 5)
var = ctrl.Antecedent(universe, 'name')
names = ['nb', 'ns', 'ze', 'ps', 'pb']
var.automf(names=names)
```

Fuzzy szabály készítése

```
rule0 = ctrl.Rule(antecedent=((error['nb'] & delta['nb']) |
(error['ns'] & delta['nb']) |
(error['nb'] & delta['ns'])),
consequent=output['pb'], label='rule pb')
```

Fuzzy szabályzó készítése

```
system = ctrl.ControlSystem(rules=[rule0, rule1, rule2, rule3, rule4])
controller = ctrl.ControlSystemSimulation(system)
```

Szabályzó bemenetének megadása

```
controller .input['error'] = devProc*4
controller .input['delta'] = slope*40
```

Szabályzó kimenetének számolása

```
controller .compute()
control = controller.output['output']
```


5 Ellenőrző kérdések

1. Mit jelent a küszöbözés? Hogyan lehet változó fényviszonyok esetében robusztusan elvégezni?
2. Ismertesse az éldetektálás első és második deriváltakon alapuló elvégzésének lehetőségeit! Írjon fel éldetektáló konvolúciós szűrőket! Mi a közös tulajdonságuk?
3. Ismertesse a Canny algoritmus működését!
4. Mire jó a Hough transzformáció? Hogyan működik?
5. Milyen módszerrel lehet nem egyenes sávok pozícióját és görbületét könnyedén meghatározni?
6. Milyen programozási nyelvet használunk a mérés során? Miért előnyös ez a nyelv multi-platform fejlesztés esetén?
7. Hogyan lehet osztályt és függvényt definiálni Python nyelven? (pszeudokód elég)

Bibliography

- [1] M. Szemenyei, *Számítógépes Látórendszerek*, M. Szemenyei, Ed. BME, 2019. [Online]. Available: <http://deeplearning.iit.bme.hu/jegyzetFull.pdf>.
- [2] *PyCharm Quick Start Guide*. [Online]. Available: <https://www.jetbrains.com/help/pycharm/quick-start-guide.html>.
- [3] P. Reizinger, *Python Példaprogramok*. [Online]. Available: <https://gist.github.com/search?utf8=%5C%E2%5C%9C%5C%93%5C&q=user%5C%3Apatrik96%5C&ref=searchresults>.
- [4] —, *Python under the hood — tips and tricks from a C++-programmers' perspective (Part I)*. [Online]. Available: <https://medium.com/@SmartLabAI/python-under-the-hood-tips-and-tricks-from-a-c-programmers-perspective-01-b5f96895663>.
- [5] —, *Python under the hood — tips and tricks from a C++-programmers' perspective (Part II)*. [Online]. Available: <https://medium.com/@SmartLabAI/b52675c7c0af>.
- [6] *Anaconda*. [Online]. Available: <https://www.anaconda.com/>.
- [7] *OpenCV Documentation*. [Online]. Available: <https://docs.opencv.org/4.1.1/>.