
# M09 – Távérzékelés és Teleoperáció Virtuális Cellában

## Mérési útmutató

## *Irányítástechnika és képfeldolgozás laboratórium 1.*

Szemenyei Márton

Irányítástechnika és Informatika Tanszék

2018

# Tartalomjegyzék

# 1. Autonóm járművek

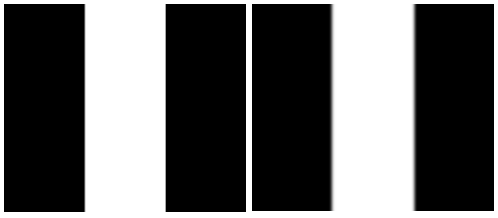
Napjainkban az autonóm járművek egyre hangsúlyosabb szerepet töltenek be a mindennapi életben. Kontrollált ipari környezetekben már jó néhány éve alkalmaznak ilyen robotokat, azonban a számítógépes látás és a mesterséges intelligencia rohamos fejlődésének köszönhetően már az utcákon is megjelentek önvezető 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.

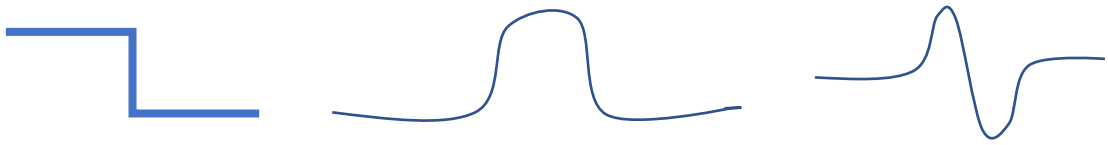
*ábra 1: Eredeti kép és a küszöbözött változata (Eredeti kép forrása: MIT)*

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 másikban konstans, valamint, hogy a változás éles, ugrásszerű. A valóságban természetesen a különböző képi hibák, zajok és a véges felbontás miatt a fent leírt ideális élekhez képest a valóságban az átmenet fokozatos, elmosott lesz, valamint lokálisan más irányú változás is elképzelhető.



*ábra 2: Ideális élek (jobbra) és elmosódott valóságos élek (balra)*

A képi élek keresésének legegyszerűbb módja a pixelek egyes irányok szerinti deriváltjának számolása, amelyet a konvolúciós szűréshez hasonló elven tehetünk meg numerikusan. A képen végighaladva minden pozícióban kiszámítjuk az adott pixel és a jobb, illetve alsó szomszédja különbségét, megkapva a kép x, illetve y irányú deriváltjait. A kettő négyzetes összegéből megkapható a teljes derivált nagysága, amely az adott pont élszerűségének mértékeként értelmezhető.



*ábra 3: Deriváló szűrő (jobb), Gauss szűrő (közép) és a Gauss deriváltja (bal) egy dimenzióban*

Habár a fent leírt módszer rendkívül gyors és egyszerű, alapvető problémája, hogy a véletlen képi zajok hamis deriváltakat eredményeznek a képen, így a kapott él kép rendkívül zajos lesz. Ez elkerülhető, ha a képet egy Gauss szűrő segítségével szűrjük, így mérsékelve a zaj hatását. A gyakorlatban azonban az algoritmus gyorsításának érdekében kihasználjuk, hogy mind a deriváló, mint a Gauss szűrők lineáris műveletek, így összevonhatók egyetlen műveletté. Így a két szűrő egymás utáni alkalmazása helyett csak egyetlen szűrést végzünk a Gauss szűrő deriváltja által meghatározott konvolúciós szűrővel, amely az előző módszerrel megegyező eredményt ad. [2]

A Gauss szűrő deriváltja mellett elterjedtek továbbá további konvolúciós éldetektáló szűrők, amelyek hasonló elven működnek. Ezek közül a legismertebbek a Prewitt [4] és a Sobel [5] operátorok, melyek irányfüggő éldetektorok. Alkalmazásuk esetén amennyiben bármilyen irányultságú éleket szeretnénk detektálni akkor az adott operátor mindkét változatát futtatni kell a képen.

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

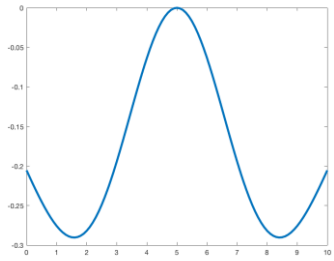
| | | |
|---|---|---|
| 1 | 0 | -1 |
| 2 | 0 | -2 |
| 1 | 0 | -1 |

*ábra 4: Vízszintes élek deriválására használható Prewitt (jobb) és Sobel (bal) operátorok. A függőleges élekhez használt szűrők ezek transzponáltjai.*

Az első deriválton alapuló szűrők egyik legnagyobb hátránya, hogy a simítás miatt az él homályosan, elkenődve fog látszani, így annak pontos lokalizációja nehézkes. Ez a probléma kiküszöbölhető, ha az élképet még egyszer lederiváljuk, és a deriváltak nullátmenetének pontját keressük meg. Ezt a műveletet természetesen egyetlen lépésben végezzük el, egy második derivált konvolúciós szűrő segítségével, melyet Laplace szűrőnek nevezünk. A Laplace szűrőt gyakorta szokták alakja miatt Sombrero-kalap szűrőnek is nevezni.

A gyakorlatban a Laplace szűrő közelíthető két eltérő szórású Gauss szűrő különbségével helyettesíteni. Ezt a megoldást DoG [2] (az angol Difference of Gaussians kifejezés rövidítése) szűrőnek nevezzük, és számos alkalmazásban használatos. Gyakorta előfordul, hogy képeket egyszerre több, különböző méretű DoG szűrő segítségével szeretnénk megszűrni. Ekkor bevett

szokás a folyamatot úgy gyorsítani, hogy először külön előállítjuk a különböző Gauss szűrők által simított képeket, majd magukat a szűrt képeket vonjuk ki egymásból. A kapott eredmény az elvégzett műveletek linearitása miatt megegyezik.

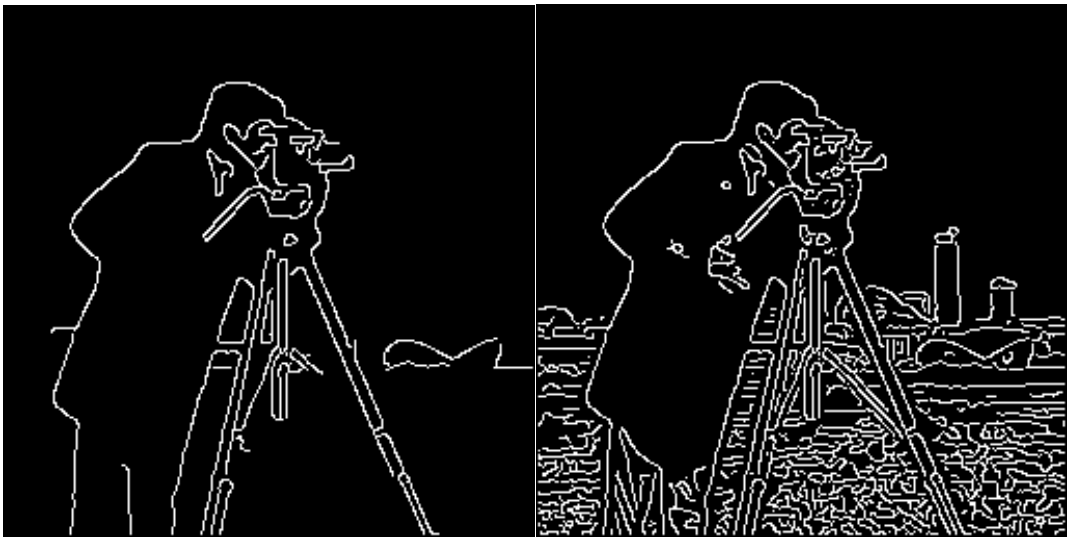


*ábra 5: A DoG szűrő egy dimenzióban*

Megfigyelhető, hogy szemben a simító szűrőkkel, ahol minden szűrő elemeinek összege 1 volt, itt minden élkereső szűrő elemeinek összege 0. Léteznek olyan szűrők is, amelyek, habár értékeik elrendezésében inkább az élkereső szűrőkre hasonlítanak (negatív és pozitív értékek más-más oldalon), mégis az értékeik összege 1. Ezeket élesítő szűrőknek [2] nevezzük, és – mint azt nevük is sugallja – képesek képeket a finom részleteket, változásokat kiemelni, ezzel a képet élesebb érzetűvé tenni. Ezt a szűrőfajtát gyakorta alkalmazzák fotós alkalmazásokban.

Az eddigi éldetektáló algoritmusok különböző deriváltszámítási módszerekre alapultak, amelyek során megkaptuk minden pixel élszerűségének mértékét. Innentől kezdve azonban a mi feladatunk tervezőként eldönteni, hogy pontosan mekkora határérték felett tekintünk egy képpontot élnek. Ha ezt a hatértéket túlságosan magasra választjuk, akkor előfordulhat, hogy valós élek kevésbé kontrasztos részeit nem fogjuk tudni detektálni, ha viszont túl alacsonyra választjuk, akkor rengeteg hamis élt fogunk kapni a zajnak és egyéb hatásoknak köszönhetően. A cél persze a kettő közötti kompromisszum megtalálása, azonban ez sem lesz tökéletes.

Ezt a dilemmát igyekszik kezelni az éldetektáló algoritmusok state-of-the-art módszere, a Canny algoritmus [7]. A Canny eljárás egy többlépcsős algoritmus, melynek első lépése, hogy egyszerű deriváló szűrők segítségével kiszámoljuk a képen a vízszintes és a függőleges irányultságú deriváltakat. Ezt követően a két irányú deriváltakból minden képpontban meghatározzuk a képi gradiens (a legnagyobb intenzitásváltozás iránya) nagyságát és irányát.



*ábra 6: A Canny eljárás eredménye különböző küszöbértékek esetén*

Ezt követően minden egyes élszerű pontból kiindulva a gradiens irányát követve meghatározzuk a legnagyobb derivált értékkel rendelkező pontot. Az ilyen pontokat meghagyjuk, míg a náluk kisebb derivált értékkel rendelkező szomszédjaikat nullába állítjuk. Ezzel a módszerrel elérjük, hogy a deriváló szűrő használata után kapott homályos élkép pontosan olyan éles legyen, mintha második deriváltat használtunk volna.

Ezt követően a gradiens nagyságokat tartalmazó képet küszöbözzük, azonban egy helyett két különböző küszöbértéket használunk (és így két különböző bináris kép keletkezik). A kisebb küszöbértékkel készített bináris képen nagy valószínűséggel megmarad az összes valódi élpont, azonban számos nem valódi élhez tartozó zaj is keletkezni fog. A nagyobb küszöbértékkel készített bináris képen a valódi élekből biztosan látszani fog valahány képpont, azonban hiányosak lesznek. Cserébe természetesen csak minimális számú zaj fog keletkezni.

A Canny algoritmus fő erénye, hogy utolsó lépésben ennek a két bináris képnek a felhasználásával képes egy mindkettőnél lényegesen jobb minőségű élképet létrehozni. Ezt úgy teszi, hogy a nagyobb küszöbértékkel rendelkező, hiányos élképhez hozzá veszi a másik élképről azokat az élpontokat, amelyek szomszédosak a hiányos képen is megtalálható élpontokkal. Ezt iteratívan addig végzi, amíg már nem tud több élpontot hozzáadni a hiányos élképhez. Ennek eredményeképp a Canny algoritmus fel tudja használni a megengedőbb küszöbértékkel készült élképet arra, hogy a szigorúbb képen keletkező hiányokat betöltse, az valódi élekhez nem kapcsolódó zajokat azonban nem veszi át.

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.

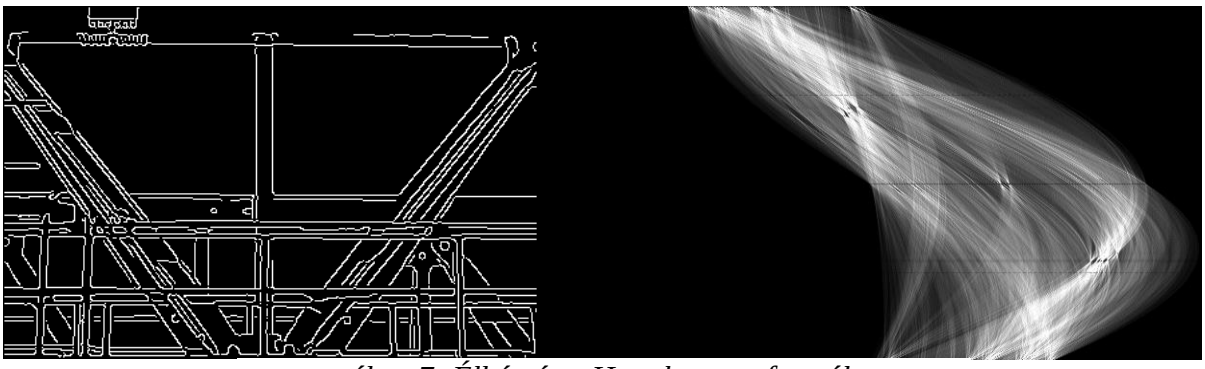
Erre a problémára az egyik legnépszerűbb algoritmus a Hough transzformációra épülő alakzat detektálás. A Hough transzformáció [8] egy koordináta transzformáció, amely a kép pontjait a megszokott képkoordinátarendszerből (x,y) az egyenesek paraméterei (r,θ) által kifeszített térbe viszi át. Ebben a térben egy egyenest egyetlen pont (számpár) ír le, amelynek egyik eleme az origóból az egyenesre állított merőleges szakasz hossza (r), a másik eleme pedig ennek a szakasznak az x tengellyel bezárt szöge (θ).

Érdekesebb azonban, hogy a kép egyes pontjai hogyan képződnek le a Hough térbe. Ezt a leképzést a Hough transzformáció során úgy végezzük el, hogy az (r,θ) által kifeszített Hough térben egy képpont képe az összes olyan egyenes lesz, amelyre az adott pont illeszkedik. Habár

egy adott pont végtelen számú egyenesre illeszkedik, mégsem illik rá az összes létező egyenesre, így egy konkrét (x,y) pont képét a Hough térben az alábbi görbe adja meg:

$$x \cos\theta + y \sin\theta = r$$

Vagyis egy képpont képe a Hough térben egy szinusz görbe lesz. Ha a Hough térben két pont görbéje metszi egymást, akkor az azt jelenti, hogy mind a két pont ráillik arra az egyenesre, amelyiket a két görbe metszéspontja ír le (emlékezzünk: a Hough térben egy pont egy egyenest ír le).



*ábra 7: Élkép és a Hough transzformált*

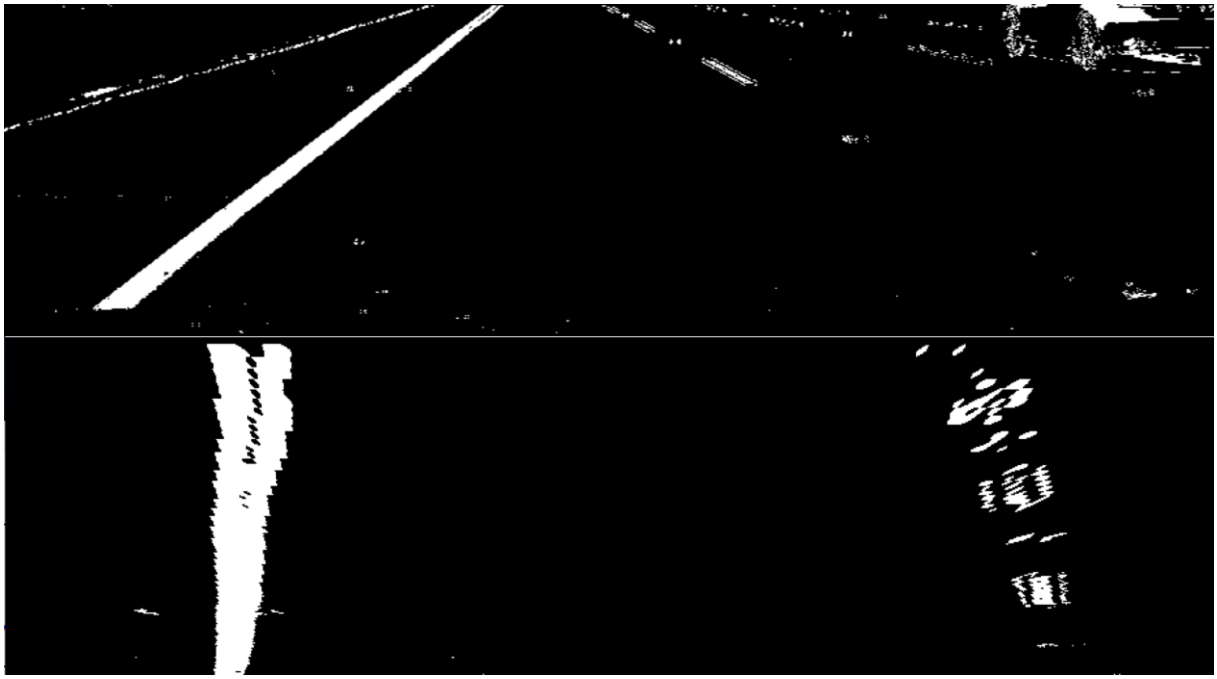
Ebből az összefüggésből egy egyszerű következtetés vonható le: Ha a Hough térben sok olyan görbénk van, amik megközelítőleg egy pontban metszik egymást, akkor ez azt jelenti, hogy a képtérben sok olyan pont van, amelyek ráillenek ugyanarra az egyenesre. Ebből kiindulva a Hough transzformációs egyenes detektálás algoritmusa könnyedén adja magát: Először transzformáljuk az összes élpontot a Hough térbe, majd megkeressük azt az egyenest amelyikre a legtöbb ráillő élpontunk található. Ezt követően ezeket az élpontokat töröljük a Hough térből, majd újra megkeressük a legtöbb pontra illeszkedő egyenest. Ezt addig ismételjük, amíg találunk olyan egyenest, amelyre legalább egy küszöbértéknyi pont illeszkedik.

Fontos megjegyezni, hogy a Hough transzformációt nem csak egyenesek, hanem bármilyen egyszerű, paraméterek által leírható formára el lehet végezni. Különböző típusú Hough transzformációkat alkalmazva tehát különböző egyszerű alakzatokat lehetünk képesek detektálni. Egyenesek mellett gyakorta alkalmazzák a Hough transzformációt körök, illetve ellipszisek detektálására is. Érdemes megemlíteni az általánosított Hough transzformációt, melynek segítségével általános formák is észlelhetők.

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 éképből 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 autóút 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.



*ábra 8: Az él kép perspektív transzformáció előtt és után.*

A perspektív transzformáció meghatározása rendkívül könnyen elvégezhető: mivel a transzformáció mátrixa 3x3-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 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.

*ábra 9: 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 felkeresni (https://www.jetbrains.com/help/pycharm/quick-start-guide.html). 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.

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, ami a könyvtárkezelési funkciókon túl virtuális környezeteket is képes kezelni. További információk az Anacondáról elérhetők itt: https://www.anaconda.com/

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 könyvtárcsalád a scikit, ami számos tudományos számításhoz szükséges alkönyvtárt foglal össze. A sckit-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.

A mérés során használt autó feldolgozó egysége egy Raspberry Pi Model 3B beágyazott számítógép, melyre USB-n keresztül egyetlen webkamera kapcsolódik. A Raspberry Pi operációs rendszere Linux alapú, így természetesen a Python interpreter alapértelmezett módon installálva van rajta. A korábban említett Pip használata miatt Python nyelven rendkívül egyszerű olyan programok készítése, melyeknek több platformon is futni kell. Érdemes megjegyezni, hogy a legtöbb olyan Python könyvtár, ami jelentős számítási igényű algoritmusokat tartalmaz a konkrét implementációkat egy előre lefordított bináris állományban tartalmazza. Ebben az esetben a Pip és az Anaconda képesek az aktuális platformnak megfelelő állományt kiválasztani és felinstallálni.


*ábra 10: A Raspberry Pi modul*

# 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 videófelvételek, valamint tényleges robotautó segítségével!

# 4. Hasznos kódrészletek

Videó betöltése

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

Képkocka olvasása videóból

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

Szürkeárnyalatos 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(rray, 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) & (d == 255)) | (e ==
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.warpPerspective(img, M, img_size, flags=cv2.INTER_LINEAR)
```

Hisztogram számítása

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

Tömbök összefűzése

```
np.array(a1,a2)
```

Számsorozat készítése

```
np.linspace(start, stop, num=elementNum)
np.ones(elementNum)
```

Transzponálás

```
X = np.transpose(X)
```

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=(var['nb'] | (var['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']
```

Kocsi mozgatása

```
car.start()
car.steer(x)
car.stop()
```

# 5. Ellenőrző kérdések

1. Mit jelent a küszöbözés? Hogyan lehet változó fényviszonyok esetében robusztusan elvégezni?
2. Ismertesse az éldetektálás első deriváltakon alapuló elvégzésének lehetőségeit!
3. Ismertesse az éldetektálás második deriváltakon alapuló elvégzésének lehetőségeit!

4. Hogyan lehet az éldetektálást konvolúciós szűrések segítségével elvégezni? Ismertessen és hasonlítson össze ilyen szűrőablakokat! Mi a közös tulajdonságuk?
5. Ismertesse a Canny algoritmus működését!
6. Mire jó a Hough transzformáció? Hogyan működik?
7. Milyen módszerrel lehet nem egyenes sávok pozícióját és görbületét könnyedén meghatározni?
8. Milyen programozási nyelvet használunk a mérés során? Miért előnyös ez a nyelv multi-platform fejlesztés esetén?