

Structure from Motion from Two Views

# 1. fejezet

## Algoritmus

A Structure from motion (SfM) folyamat segítségével 3D rekonstrukciót hajthatunk végre egy képpár segítségével.

1. Két kép közötti ritka ponthalmazok megfeleltetése (pontmegfeleltetés): az első kép sarkainak azonosítása a *detectMinEigenFeatures* függvénnyel, majd azok követése a második képre a *vision.PointTracker* segítségével.
2. Az esszenciális mátrix becslése *estimateEssentialMatrix* használatával.
3. Kamera elmozdulásának kiszámítása *estrelpose* függvénnyel.
4. Két kép közötti sűrű ponthalmazok megfeleltetése (pontmegfeleltetés): több pont kinyeréséhez újra kell detektálni a pontokat a *detectMinEigenFeatures* függvény segítségével a '*MinQuality*' opciót használva. Ezt követi a sűrű ponthalmaz követése a második képre a *vision.PointTracker* használatával.
5. Az illeszkedő pontok 3D helyzeteinek meghatározása a *triangulate* segítségével (háromszögelés).

## 2. fejezet

# Kód magyarázata

### 2.1. Képpár betöltése

1. *fullfile(string1, string2, ...)* = az argumentumként kapott stringekből összeállít egy elérési útvonalat, pl.:

```
path = fullfile('myfolder', 'mysubfolder')  
path = 'myfolder\mysubfolder\'
```

*toolboxdir(toolbox)* = visszaadja az argumentumként kapott toolbox abszolút elérési útvonalát.

2. *imageDatastore(path)* = létrehoz egy ImageDatastore objektumot a kapott elérési útvonallal meghatározott képekből. Az ImageDatastore objektum segítségével egy mappában található összes képet össze lehet gyűjteni egy változóba (de alapból nem lesz az összes kép egyszerre betöltve).
3. *readimage(datastore, n)* = betölti az n. képet a megadott datastore-ból.
4. *figure* = létrehoz egy új, üres ábra ablakot.
5. *imshowpair(image1, image2, 'montage')* = a meghatározott két képet egymás mellé helyezi a legutolsó ábrán.
6. *title('string')* = hozzáad egy címet a legutolsó ábrához.

## 2.2. A Camera Calibrator alkalmazás segítségével előre kiszámolt kamera paraméterek betöltése.

1. `load(file_name.mat)` = betölti egy korábban elmentett workspace adatait a jelenlegi workspace-be. A workspace egy ideiglenes tároló amely a MATLAB elindítása óta létrehozott változókat tárolja. Alapértelmezetten a MATLAB ablak jobb oldalán látható. A workspace-t el lehet menteni, így a benne tárolt változókat később vissza lehet tölteni a MATLAB-ba.

## 2.3. Lencse által okozott torzítás eltávolítása.

1. `undistortImage(image, intrinsics)` = a második argumentumként megadott kamera paramétereket felhasználva eltünteti a kamera lencséje által okozott torzítást a megadott képről.

A kamera kalibrációja során kapott kamera paramétereket és a torzítási együtthatókat felhasználva kiszámítjuk a bemeneti kép minden pixelének eredeti pozícióját. Az egyes pixelek pozícióját az alábbi torzítások módosítják:

- **Radiális torzítás** = kiváltó oka, hogy a lencse szélén áthaladó fény jobban törik, mint a lencse közepén környezetében áthaladó fény. Ez kiszámolható:

$$x_d = x_u(1 + k_1r^2 + k_2r^4)$$

$$y_d = y_u(1 + k_1r^2 + k_2r^4)$$

(Ahol  $x_u, y_u$  = torzulásmentes koordináták;  $x_d, y_d$  = torzított koordináták;  $k_1, k_2$  = radiális torzítási együtthatók;  $r^2 = x_u^2 + y_u^2$ )

- **Tangenciális fordítás** = előfordul, ha a kameraszensor és a lencse nem állnak tökéletesen párhuzamosan. Ez kiszámolható:

$$x_d = 2p_1x_uy_u + p_2(r^2 + 2x_u^2)$$

$$y_d = 2p_2x_uy_u + p_1(r^2 + 2y_u^2)$$

(Ahol  $x_u, y_u$  = torzulásmentes koordináták;  $x_d, y_d$  = torzított koordináták;  $p_1, p_2$  = tangenciális torzítási együtthatók;  $r^2 = x_u^2 + y_u^2$ )

Az egyes pixelek korrigált helyének kiszámítása nem egész számú értékeket is előállít. Mivel a nem egész szám nem lehet pixel koordináta, ezért bilineáris interpolációt is végre kell hajtani. A bilineáris interpoláció során, a legközelebbi négy szomszédot felhasználva először lineáris interpolációt hajtunk végre az egyik irányba (pl. az x tengely mentén), majd pedig a másik irányba (az y tengely mentén):

$$out_P = I_1(1-\Delta X)(1-\Delta Y) + I_2(\Delta X)(1-\Delta Y) + I_3(1-\Delta X)(\Delta Y) + I_4(\Delta X)(\Delta Y)$$

(Ahol  $I_1, I_2, I_3, I_4$  = a szomszédos négy koordináta intenzitása az eredeti, torzított képen;  $\Delta X, \Delta Y$  = a nem egész értékű koordinátákkal rendelkező vizsgált pixel és a vizsgált pixelhez legközelebb eső, egész értékű koordinátákkal rendelkező szomszédai közötti távolság;  $out_P$  = végeredményként kapott pixel intenzitás)

A szomszédos pixelek efféle súlyozott átlagolásával, az interpoláció eredményeképp egy pixel intenzitás értéket kapunk, amely a legközelebbi egész érték koordinátával rendelkező pixel intenzitása lesz.

Az előállított, torzítatlan képen néhány pixel (leginkább a kép szélein) nem rendelkezik megfelelő pixel párral az eredeti, torzított képről (ezek azok a területek, ahol az eredeti képből nincs információ). Ezek a pixelek alapértelmezetten 0 értéket kapnak (feketék lesznek).

## 2.4. Pontmegfeleltetés a képek között.

1. *detectMinEigenFeatures(grayImage, MinQuality=0.1)* = a Shi és Tomasi féle minimum sajátérték algoritmust (Shi & Tomasi, Minimum Eigenvalue Algorithm) használva keresi meg a kép sarokpontjait (a sarokpont jelen esetben olyan pixeleket jelentenek, amelyek éles változást mutatnak a környező pixelekhez képest). Szürkeárnyaltos képet vár

argumentumként, ezért a képet még előtte az *im2gray* függvénnyel szürkeárnyalatosra változtatjuk. A *MinQuality* argumentum a detektált sarokpontok minőségét határozza meg. Az értékének  $[0, 1]$  tartományból választhatunk. Magasabb érték, jobb minőségű, viszont kevesebb sarokpontot is eredményez. Pontmegfeleltetés esetén gyakran a sarokpontok detektálása a preferált módszer, ugyanis a sarokpontok általában könnyen azonosíthatók különböző nézőpontból és stabilak (azaz kisebb elmozdulás, zaj vagy torzítás hatására is megismerhetők).

A Shi és Tomasi féle sajátérték algoritmus a Harris sarokpont detektáló algoritmuson alapszik. A Harris sarokpont detektáló algoritmus esetén első lépésben meghatározunk egy csúszóablakot a vizsgált képen. Ha azt tapasztaljuk, hogy ezt az ablakot bármelyik irányba is mozgassuk el, nagy lesz a különbség az ablak eredeti pozíciója alatti terület és az elmozdított ablak új pozíciója alatti terület között, akkor sikeresen detektáltunk egy sarokpontot. Ezt a változást az alábbi képlet szerint mérjük:

$$E(u, v) = \sum_{x,y} w(x, y) [I(x + u, y + v) - I(x, y)]^2$$

Ahol:

- $E$  = a négyzetkülönbség a csúszóablak eredeti és elmozdított pozíciója között.
- $u$  = a csúszóablak elmozdításának értéke az x tengely mentén.
- $v$  = a csúszóablak elmozdításának értéke az y tengely mentén.
- $w(x, y)$  = a csúszóablak az (x,y) pozíción
- $I$  = a kép intenzitása a zárójelekben meghatározott pozíción, pl:
  - $I(x, y)$  = az ablak eredeti pozíciója alatti terület intenzitása.
  - $I(x+u, y+v)$  = az elmozdított ablak alatti terület intenzitása.

A cél olyan csúszóablakokat találni, ahol ez az  $E$  érték nagy, bármelyik irányba is toljuk el az ablakot. Azaz olyan pozíció kell, ahol (a képletben) a szögletes zárójelben található kifejezés értéke nagy. Tehát a

$$\sum_{x,y} [I(x + u, y + v) - I(x, y)]^2$$

részt kell maximalizálni. Ehhez először Taylor-sort alkalmazunk, amely után az alábbi egyenletet kapjuk (ezek után már csak közelítő eredményt kapunk):

$$E(u, v) \approx \sum_{x,y} [I(x, y) + ul_x + vl_y - I(x, y)]^2$$

A  $I(x, y) - I(x, y)$  rész kiüti egymást, majd a elvégezzük a négyzetre emelést  $((a + b)^2 = a^2 + 2ab + b^2)$ :

$$E(u, v) \approx \sum_{x,y} u^2 l_x^2 + 2uv l_x l_y + v^2 l_y^2$$

Ezt mátrixá alakítjuk:

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \left( \sum \begin{bmatrix} l_x^2 & l_x l_y \\ l_x l_y & l_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

A mátrixot (talán második momentum mátrixnak hívják)  $M$  betűvel fogjuk jelezni és a  $w(x, y)$  is csodával határos módon visszakerült:

$$M = \sum w(x, y) \begin{bmatrix} l_x^2 & l_x l_y \\ l_x l_y & l_y^2 \end{bmatrix}$$

Az  $M$ -et behelyettesítve:

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

To be continued...

[https://docs.opencv.org/3.4/dc/d0d/tutorial\\_py\\_features\\_harris.html](https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html)

<https://aishack.in/tutorials/harris-corner-detector/>

## 3. fejezet

### Forrás

- <https://www.mathworks.com/help/vision/ug/structure-from-motion-from-two-views.html>
- [https://www.mathworks.com/help/vision/ref/undistortimage.html?s\\_tid=doc\\_ta](https://www.mathworks.com/help/vision/ref/undistortimage.html?s_tid=doc_ta)
- <https://www.mathworks.com/help/visionhdl/ug/image-undistort.html>
- [https://e-learning.ujs.sk/pluginfile.php/23441/mod\\_resource/content/1/01-ProjektivKamera.pdf](https://e-learning.ujs.sk/pluginfile.php/23441/mod_resource/content/1/01-ProjektivKamera.pdf)
- <https://www.mathworks.com/help/vision/ref/detectmineigenfeatures.html>
- <https://aishack.in/tutorials/features/>
- <https://aishack.in/tutorials/harris-corner-detector/>
- <https://aishack.in/tutorials/shitomasi-corner-detector/>
- [https://docs.opencv.org/3.4/dc/d0d/tutorial\\_py\\_features\\_harris.html](https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html)