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 datastoreból.

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 eltűnteti 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:

     $$x_d = x_u(1 + k_1 r^2 + k_2 r^4)$$
     $$y_d = y_u(1 + k_1 r^2 + k_2 r^4)$$

     (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 kameraszenzor és a lencse nem állnak tökéletesen párhuzamosan. Ez kiszámolható:

     $$x_d = 2p_1 x_u y_u + p_2(r^2 + 2x_u^2)$$
     $$y_d = 2p_2 x_u y_u + p_1(r^2 + 2y_u^2)$$

(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 súlyzott 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). Szürkeárnyalatos 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 erdeti 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 elmozgatott 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 ($(a+b)^2 = a^2 + 2ab + b^2$):

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

Ezt mátrixá alakítjuk:

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

A mátrixot (talán második momentum mátrixnak hívják és a kép intenzitásának változását méri az x és y irányokban) 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:

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

Az így kapott E értékből meg tudjuk mondani, hogy jelentős változás van-e a csúszóablak eredeti, valamint elmozdított pozíciója alatti terület között. Ha az intenzitásváltozás jelentős, akkor az alábbi képlettel döntjük el, hogy az adott ablak tartalmaz-e sarokpontot:

$$R = det(M) - k(trace(M))^2$$

Ahol:

- *det(M)* = amely a második momentum mátrix determinánsát határozza meg. Megadja, hogy mekkora az intenzitásváltozás mértéke mindkét irányban. Értéke: $det(M) = \lambda_1 \lambda_2$, ahol $\lambda_{1,2}$ az M mátrix sajátértékei, és megmondják, hogy a képgradiens milyen irányba, milyen mértékben változik.

- k = egy állandó, amely a sarokpont detektálás szigorúságát határozza meg. Általában [0.04, 0.05] tartományba eső érték.
- *trace(M)* = a mátrix nyomát adja meg, amely a főátlók, azaz a két irány intenzitásváltozásának összege. Értéke: $trace(M) = \lambda_1 + \lambda_2$

Tehát az M mátrix λ_1, λ_2 sajátértékeinek értéke határozza meg, hogy a vizsgált régió sarokpont, él vagy felület-e:

- Ha R $>$ 0, akkor az adott területen sarokpont van.
- Ha R $<$ 0, akkor az adott területen él van.
- Ha R $\approx$ 0, akkor az adott terület homogén.

A Shi és Tomasi féle minimum sajátérték algoritmus egyetlen apró változtatást hajt végre a Harris féle sarokpont detektáló algoritmuson: A sarokpontok minőségét nem a sajátértékek kombinációjával, hanem a két sajátérték közül a kisebbik sajátérték alapján határozza meg. Az R értékét ez az algoritmus az alábbi képlet szerint számolja ki:

$$R = min(\lambda_1, \lambda_2)$$

Ha R nagyobb, mint egy előre meghatározott érték, akkor a vizsgált terület sarokpontot tartalmaz.

2. *imshow(image, InitialMagnification = value)* = a meghatározott kép megjelenítése *value*%-os megnagyításban.

3. *hold on* = a következőkben végrehajtott grafikus (pl.: grafikonok kirajzolása, stb.) parancsokat a legutolsó, aktuális ábrára fogja rárajzolni.

4. *plot* = létrehoz kétdimenziós grafikont.

5. *selectStrongest(featurePoints, N)* = visszaadja az N legerősebb (talán ez az intenzitások különbségének nagyságát jelenti) jellemzőpontot (nálunk sarokpontok lesznek) a megadott *featurePoints* változóból.

6. *tracker = vision.PointTracker(MaxBidirectionalError=value1, NumPyramidLevels=value2)* = létrehoz egy Kanade-Lucas-Tomasi (KLT) algoritmus szerint működő pontkövető objektumot.
   A KLT algoritmus kifejezetten jól működik olyan objektumok követésére, amely nem változtat alakot, valamint egyedi és részletes textúrával

rendelkezik. A KLT algoritmus a Lucas-Kanade (LK) optikai áramlás becslés algoritmuson alapul. Az optikai áramlás az objetumok egy látszólagos/vizuális elmozdulása (nem feltétlenül egyezik meg a valós elmozdulással). Általános feltételezése, hogy egy objektum pixeleinek intenzitása elmozdulástól függetlenül állandó:

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

Ahol:

- *I(x,y,t)* = *(x,y)* pozíción lévő pixel intenzitása t időben
- u = elmozdulás az x tengelyen
- v = elmozdulás az y tengelyen

Ebből kifejezhető az optikai áramlás egyenlete Taylor sort alkalmazva:

$$I(x + u, y + v, t + 1) \approx I(x, y, t) + I_x u + I_y v + I_t$$

$$I(x + u, y + v, t + 1) - I(x, y, t) = I_x u + I_y v + I_t$$

$$0 \approx I_x u + I_y v + I_t$$

Ahol:

- $I_x u$ = az intenzitás változása az x tengelyen.
- $I_y v$ = az intenzitás változása az y tengelyen.
- I_t = az intentizás idő szerinti változása.

Tehát, ha az u és v elmozdulások helyesen vannak meghatározva, akkor az intenzitások különbsége megközelítőleg 0. Azonban u és v ismeretlenek és meghatározásukat nehezíti a **nyílás/rekesz/apertúra probléma (aperture problem)**, azaz amikor a tényleges mozgást egy kicsi rekeszen keresztül figyelve próbáljuk meghatározni. Egy objektum tényleges elmozdulásának kiszámítása nehéz, ha csak egy kis területet látunk belőle a résen keresztül.
A Lucas-Kanade algoritmus feltételezi, hogy az optikai áramlás (u és v) konstans és a képen textúrázott tárgyak láthatók. A megfigyelt pixel kezdeti intenzitása a, valamint a rés mögött látható tárgy elmozdítása után észlelt pixel intenzitása b. Ezek különbsége *(b - a)*, azaz az időbeli

intenzitáskülönbség $I_t(x,y)$. A *(x,y)* pozíción megfigyelt pixel intenzitásváltozás $I_x(x,y)$ az x tengelyen, valamint $I_y(x,y)$ az y tengelyen. Az x tengelyen történő u és y tengelyen történő v elmozdulás esetén a pixel intenzitása:

$$I_x(x,y)u + I_y(x,y)v = -I_t(x,y)$$

Nem tudom miért negatív az I_t :(

Az algoritmus helyes működéséhez nem elegendő egyetlen pixelt nézni, így ezt ki kell terjeszteni egy pixelszomszédságra. Egy 3x3 szomszédság esetén az egyenlet:

$$I_x(x+\Delta x, y+\Delta y)u + I_y(x+\Delta x, y+\Delta y)v = -I_t(x+\Delta x, y+\Delta y)$$

ahol Δx = -1,0,1 (a három szomszédos pixel az x tengelyen), Δy = -1,0,1 (a három szomszédos pixel az y tengelyen) Tömör formában a képlet:

$$S \begin{bmatrix} u \\ v \end{bmatrix} = \vec{t}$$

Ahol:

- S = 9x2 mátrix amely tartalmazza: $[I_x(x+\Delta x, y+\Delta y), I_y(x+\Delta x, y+\Delta y]$.
- $\vec{t}$ = vektor amely tartalmazza: $-I_t(x+\Delta x, y+\Delta y)$.

Ezen két ismeretlenes egyenlet megoldása a legkisebb négyzetek módszerével történik, amihez megszorozzuk az egyenletet S^T (én sem értem miért):

$$S^T S \begin{bmatrix} u \\ v \end{bmatrix} = S^T \vec{t}$$

Ebből $\begin{bmatrix} u \\ v \end{bmatrix}$ kifejezve:

$$\begin{bmatrix} u \\ v \end{bmatrix} = (S^T S)^{-1} S^T \vec{t}$$

Röviden (ha jól fogtam fel) a Lucas-Kanade optikai áramlás becslés algoritmusa a képkockák közötti mozgást próbálja nyomon követni. Ehhez két képkockán kiválaszt egy pixelszomszédságot és azok térbeli elmozdulását elemzi. Ezután kiszámolja a pixelértékek intenzitásváltozását, amelyből próbálja kiszámolni az objektum tényleges elmozdulását.

Gyorsan mozgó vagy nem ritkásan textúrázott objektumok esetén nem megbízható.
A tényleges pontmegfeleltetést a Kanade-Lucas-Tomasi (KLT) algoritmus végzi, amelyből a MATLAB vision.PointTracker függvény a pyramid KLT továbbfejlesztett verziót használja. Először minden kiválasztott jellemzőponthoz egy kis ablakot/apertúrát illeszt. A következő lépésben létrehot egy kép piramist: a kiinduló képből több szintet hoz létre, ahol az alsó szinten található a teljes (eredeti) felbontású kép, a piramis teteje felé haladva a vizsgált kép egyre kisebb felbontású verzi-ója fog szerepelni. Ezután kiszámolja a kiválasztott pixelek elmozdulás vektorait a piramis legfelső szintjétől kezdve, az előbb tárgyalt Lucas-Kanade optikai áramlás becslés algoritmusát felhasználva. A becsült mozgásokat használva frissíti a jellemzőpontok pozicióit a következő, magasabb felbontású piramisszintent. Ezt addig folytatja, amíg el nem éri a legalsó szintet. Az alacsonyabb felbontású képeken végzett kezdeti nyomkövetés csökkenti a számítási igényt és javítja a nyomkövetési pontosságot.

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/visionhdl/ug/image-undistort.html
- 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://www.mathworks.com/help/vision/ref/vision.pointtracker-system-object.html
- https://lorenzopeppoloni.com/lkttracker/

- https://www.baeldung.com/cs/optical-flow-lucas-kanade-method
- https://www.inf.u-szeged.hu/~kato/teaching/IpariKepfeldolgozas/08-Motion.pdf
- http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/tutorials/Lucas-Kanade2.pdf
- https://link.springer.com/chapter/10.1007/978-3-319-29451-3_29