Project 4:

Visual motion based Human-Computer Interface

Számítógépes Látás kurzus 2007/08.

3. ellenőrzési pont

(2007-12-11)

Tartalomjegyzék

Csapattagok

Jaksa Zsombor	Jaksa.Zsombor@stud.u-szeged.hu
Németh József	Nemeth.Jozsef@stud.u-szeged.hu
Ungi Tamás	Ungi.Tamas@stud.u-szeged.hu
Utasi Tamás	Utasi.Tamas@stud.u-szeged.hu

Feladat

Human-Computer Interface (HCI) fejlesztése olcsó webkamera segítségével. A kamera mozgását a kamera által továbbított képek alapján kell megbecsülni. Az interface vezérlése a kézben tartott kamera mozgatásával történik, így 3D mutatóként használható. Egy egyszerű képnézegető program vezérlését is meg kell valósítani az alábbi módon:

- vízszintes mozgás az képet bal-jobb irányban mozgatja
- függőleges mozgás a képet fel-le mozgatja
- előre-hátra mozgás a képet nagyítja és kicsinyíti

Háttér

1963-ban a Stanford Kutatóintézeben Douglas Engelbart felvetette az egér gondolatát. Mára egy mindenki által használt eszköz lett, de azóta nemigazán történt előrelépés a hétköznapi használatra szánt számítógépek perifériáinak piacán. A technika fejlődésével megjelent az igény a számítógépek könnyű használhatóságára. Szükségessé vált, hogy az ember ne csak billentyűzeten tudjon kommunikálni a géppel, telefonnal. Újfajta ember-számítógép interfészek jelentek meg. Modern autókban a fedélzeti számítógépek megértik a szóbeli utasításokat, intelligens kivetítőkre lehet ujjal rajzolni. Az ilyen interfészek megkönnyíthetik a fogyatékkal élők életét is, mint például jelbeszédfelismerő rendszerek, vagy a vakokat segítő hangalapú rendszerek.

A mobil számítógépek (telefonok) esetében is komoly probléma a kezelhetőség kis méretük miatt. Ezért mára már kihagyhatatlan részének számító kamera is egy fontos beviteli perifériaként használható. Képnézegetők, játékok, webböngészők irányítását lehet megkönnyíteni a kamera képéből kinyerhető mozgásinformáció felhasználásával. Ez a technika egészen új lehetőségeket is teremtett, melynek jelentőségét növeli, hogy a mobil számítástechnikai eszközök növekvő teret foglalnak az informatikai üzletágban [1].

Meglévő hasonló rendszerek

Egy megvalósítás leírása elérhető [2], melyet megvalósítva azt összehasonlíthatjuk az általunk fejlesztett módszerrel.

A feladattal rokon alkalmazás, amikor a kamera rögzített és a felhasználó a kezét vagy az arcát mozgatja, ezzel irányítva egy programot. Ilyen alkalmazás is létezik, mely képes az arc egyes részeit külön is értelmezni [3].

OpenCV

Az OpenCV egy C++ nyelven írt open-source függvénygyűjtemény, melynek elsődleges célja a valós idejű számítógépes látás segítése. Fejlesztését az Intel kezdeményezte, de mára nagyon sokan járultak hozzá. Platformfüggetlen forráskódja letölthető a következő címről:

http://sourceforge.net/projects/opencvlibrary/

Optical flow

Mivel a mozgásérzékelő programunkat általános felhasználásra tervezzük, nem használhatunk ki semmilyen előzetes információt a kamera által érzékelt objektumokról. Ezért azokat a módszereket, amelyek objektum felismerésen alapszanak, nem használhatjuk. Általános esetben a képi elmozdulást az optical flow-val jellemezzük.

Az optical flow egy olyan módszer, amely vizuálisan reprezentált tárgyak elmozdulását határozza meg. Jellemzően vektorokkal írjuk le, melyek képpontokból erednek vagy képpontokba mutatnak egy képsorozatban [4].

Az optical flow számítására több módszer is létezik, röviden felsorolunk néhányat a legfontosabb jellemzőikkel együtt:

Phase correlation

Ez a módszer azt használja ki hogy az eredeti képen az eltolás a frekvenciatérben fáziseltolódásként jelenik meg [5]. Gyors módszer, real-time megvalósításra alkalmas, de mivel csak az eltolást és a forgatást detektálja, számunkra nem teljesen megfelelő, mivel nekünk a zoom funkcióhoz a nagyítási faktorra is szükségünk van.

Block matching

Ez a módszer egy egyszerű template matching algoritmus szabályosan kiválasztott képrészletekre alkalmazva [6]. A leírások alapján nem elég pontos módszer.

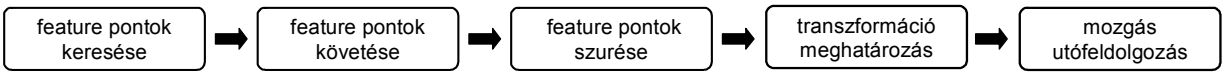
Lucas Kanade módszer

Ez a legnépszerűbb optical flow meghatározó módszer, mely viszonylag régóta ismert [7]. Érdemes csak néhány előre meghatározott feature pontokra alkalmazni [8], így real-time megvalósítása gyengébb hardware-en is lehetséges.

Módszerek

1. módszer

Először egy már publikált módszert implementáltunk[2], hogy lehetőségünk legyen a saját megvalósítási ötleteinket összehasonlítani. Ennek a módszernek lényege az alábbi ábrán látható:



1. ábra: Az első módszer folyamatábrája.

Feature pont kiválasztás

A videó képet 4×4 régióra bontjuk, és mindegyikben meghározunk egy feature pontot (FP). Az lesz a régióhoz tartozó FP, amelyiknek a környezetében maximális a horizontális és vertikális deriváltak összege. Egy (x,y) pontban jelölje G(x,y) a két parciális derivált összegét:

$$G(x,y) = (I(x,y) - I(x-1,y))^2 + (I(x,y) - I(x,y-1))^2$$

Ezzel a jelöléssel a FP kiválasztás az alábbi kritérium alapján történik:

$$p_i = \max_{(x,y)\in R_i} \left(\sum_{(x,y)\in B_i} G(x,y) \right)$$

ahol p_i az i-edik FP, R az aktuális kép régió, B_i az i-edik FP-hez tartozó kép blokk, amely egy FP középpontú kis méretű négyzet alakú része a képnek (a mi esetünkben pl. 7×7 méretű).

FP elmozdulásának meghatározása

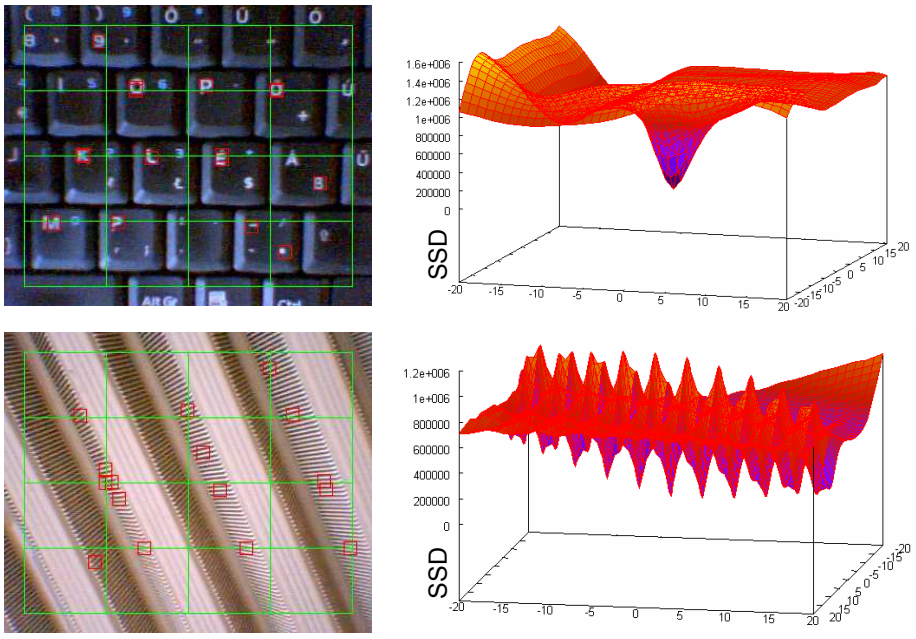
Az FP-k kiválasztása után a következő képkockán meg kell határoznunk azok elmozdulását. Ehhez bevezetjük az i-edik FP-hez és v elmozduláshoz tartozó SSD mértéket, melyet az FP körüli B_i blokkban számítunk az alábbi képlet alapján:

$$D(v,B_i) = \sum_{(x,y)\in B_i} (I_k(x,y) - I_{k+1}(x+v_x, y+v_y))^2$$

Azt az elmozdulást választjuk minden ponthoz, amely egy adott keresési ablakon (W) belül a minimális SSD értéket adja:

$$v_i = \min_{v\in W_i} D(v,B_i)$$

Ezt a módszert nevezik block-matching-nek. Működését két teszt képpel illusztráljuk, melyeken piros keretekkel vannak jelölve az FP középpontú blokkok.

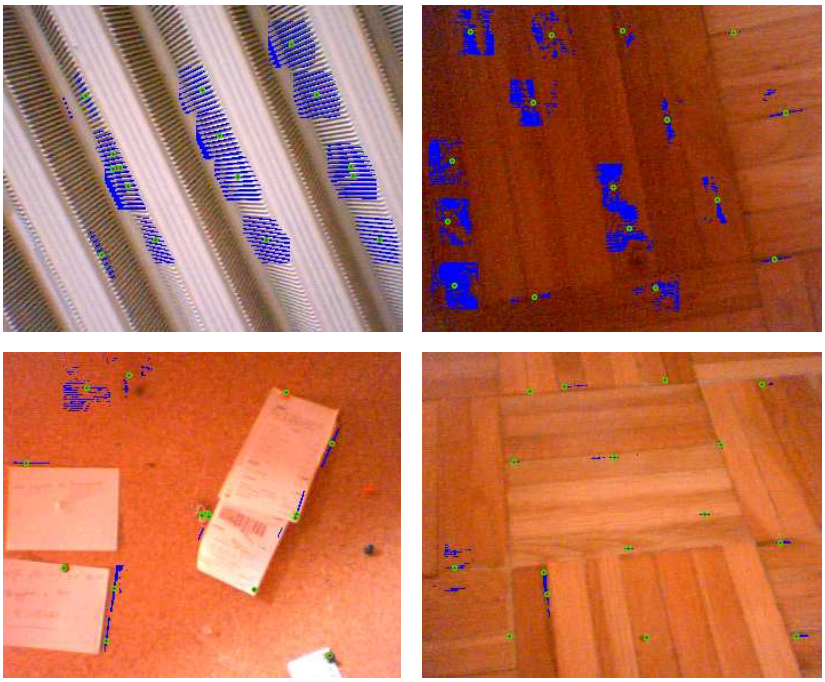


2. ábra: Átlagos block-matching SSD értékek a 16 darab FP-tól számított -20...+20 távolságban (vízszintes tengelyek) két különböző input képen. A felső képen a FP-ok egyedi környezettel bírnak, míg az alsó képen periodikus mintázatban helyezkednek el.

Jól látható módon ennek a módszernek az a hátránya, hogy a valós elmozduláshoz képest az SSD minimuma ettől eltérő helyre kerül, melynek oka egyrészt a kép diszkretizációja, másrészt a képen ismétlődő kisebb minták jelenléte. Ezért a legjobb blokk illeszkedésen kívül figyelembe vesszük azokat a pontokat is, amelyek még bizonyos valószínűséggel takarhatják a valós FP elmozdulást. p_i-hez tartozó lehetséges elmozdulások halmaza legyen V_i:

$$V_i = \{v | D(v, B_i) < k_1 G(B_i) + k_2\}$$

ahol $G(B_i)$ a B_i blokkon belüli (x,y) pontok G(x,y) értékeinek összege. k1 és k2 konstansokat kísérleti úton határozzuk meg, a mi esetünkben a k_1=0,6 és k_2=4 értékek bizonyultak használhatónak. A szóbajövő elmozdulások illusztrálására bemutatjuk az alábbi négy képet, melyeken kékre színeztük azokat a pontokat, melyek lehetséges elmozdulásai a zöld-piros FP-oknak. Érdemes megfigyelni hogy a sarokpontok elmozdulását nagy biztonsággal meg tudjuk határozni, az éleken elhelyezkedő pontok elmozdulása az éllel párhuzamosan bizonytalan, míg a viszonylag homogén területen az elmozdulás meghatározása minden irányban bizonytalan, amit a nagy kiterjedésű kék területek jeleznek:



3. ábra: A zöld-piros FP-k lehetséges elmozdulásait kék pontok jelölik.

V_i halmaz eltárolása helyett, csak azok kovarianciamátrixát (kis módosítással) tároljuk:

$$C_i = \frac{1}{N}\sum_{v \in V_i}(v-\bar{v})(v-\bar{v})^T + \frac{1}{12}I$$

Ezt a formulát részletesen indokolják a szerzők egy korábbi munkájukban[9].

Globális transzformáció meghatározás

Az elmozdulás leírására nem a hagyományos transzformációs mátrixokat használjuk (affin transzformáció esetén 2x2 méretű mátrix), hanem egy θ-val jelölt 4x1 méretű mátrixot:

$$p = \begin{bmatrix} x \\ y \end{bmatrix} \quad v = v(\theta, p) = \begin{bmatrix} 1 & 0 & x & y \\ 0 & 1 & y & -x \end{bmatrix} \cdot \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} = H(p) \cdot \theta$$

Ebben a módszerben tehát elmozdulást (θ_1 és θ_2), valamint forgatást és skálázást (θ_3 és θ_4) határozunk meg, melyet globális transzformációnak nevezünk az egyes képkockák között.

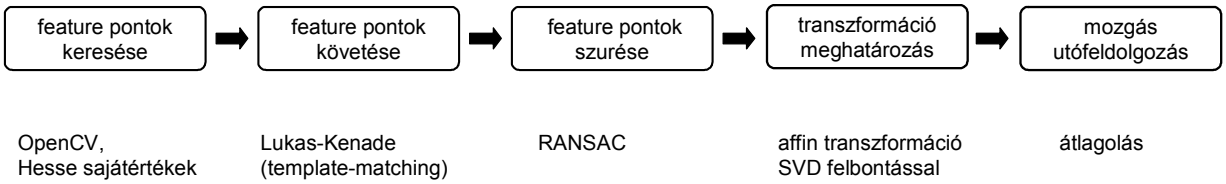
A 16 FP elmozdulás közül elvetjük azokat amelyek nincsenek összhangban a többség által meghatározott transzformációval, ezt nevezzük *outlier analysis*-nek. Ehhez egy RANSAC-hoz hasonló módszert használunk, azzal a különbséggel hogy a 16 FP által alkotott minden lehetséges pontpárhoz meghatározzuk a velük összhangban elmozduló pontok számát, valamint figyelembe vesszük hogy nem egy diszkrét elmozdulás érték tartozik egy ponthoz, hanem még egy kovariancia mátrix is, amely tartalmazza az elmozdulás bizonytalanságát két különböző irányban.

A legtöbb "szavazatot" kapó pontpár transzformációs csoportjában lévő elmozdulásokból BLUE (Best Linear Unbiased Estimator) módszerrel határozzuk meg a globális transzformációt, a szerzők által leírt módon.

A mozgás utólagos símítását Kálmán filter helyett jelenleg három képkocka egyszerű átlagolásával oldottuk meg, amely a gyakorlatban kielégítőnek bizonyul. Ez az egyetlen algoritmikus eltérés amelyet a szerzők leírásához képest a saját implementációnkban végeztünk.

2. módszer (saját ötletek alapján)

A meglévő megoldások és az elérhető könyvtárak alapján a következő saját megvalósítást tervezzük a feladat megoldására:



4. ábra: Az általunk fejlesztett HCI algoritmus vázlata.

A feature pontok meghatározására jó és hatékony implementációt találtunk OpenCV-ben. A függvény neve GoodFeaturesToTrack. Ez a pontok közül azokat választja ki, melyeknek lokálisan a legnagyobbak a Hesse mátrix sajátértékei.

A pontok Lukas-Kenade szerinti követése szintén meg van valósítva OpenCV függvényként (cvCalcOpticalFlowPyrLK). Ez hatékony, mivel a pontkövetést két-szintű képpiramison végzi, template matching segítségével.

Mivel a pont megfeleltetések túldefiniálják a transzformációt, ezért először az outlier-eket ki kell szűrni, amire a RANSAC algoritmust használjuk. Véletlenszerűen kiválasztott pontok által meghatározott transzformációra megnézzük hogy a többi pontmegfeleltetést mennyire elégítik ki valamilyen küszöb alatti hibával.

$$T_{p_1,p_2,p_3} = \begin{bmatrix} p_1' & p_2' & p_3' \end{bmatrix} \begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix}^{-1}$$

ahol p-k a véletlenszerűen kiválasztott pontok, T pedig az affin transzformáció.

Az i-edik pont szavazatszáma a j_1, j_2, j_3 pontokra:

$$V_i(p_{j_1}, p_{j_2}, p_{j_3}) = \begin{cases} K - E(p_i) & E(p_i) = d(p_i', T_{p_{j_1},p_{j_2},p_{j_3}} p_i) < K \\ 0 & \textit{különben} \end{cases}$$

ahol K tapasztalati úton választott küszöbérték (a mi esetünkben 15), *d* az euklideszi távolság, *E* pedig a hiba mértéke.

A legjobb pontokat és a rájuk szavazó pontokat tartjuk meg. A megmaradó pontokból még mindig túldefiniált egyenletrendszert kapunk:

$$T = \begin{bmatrix} p_1' & p_2' & \dots & p_n' \end{bmatrix} \begin{bmatrix} p_1 & p_2 & \dots & p_n \end{bmatrix}^{-1}$$

A jobboldalon található inverz számítást SVD felbontással közelítjük.

A mozgás utófeldolgozását jelenleg a 3 utolsó elmozdulás súlyozott átlagolásával oldjuk meg.

A fejlesztés technikai részletei

Egyik célunk, hogy a fejlesztett forráskódban csak ingyenes, open-source eszközöket használjunk, valamint a kód változtatás nélkül fordítható legyen az különböző platformokon (Windows és Linux).

A fordítás előkészítését CMake-kel végezzük, amely egy konfigurációs text file-ból különböző platformokra előkészíti a fordítást, pl. GCC-hez makefile-t csinál, Visual Studio-hoz projekt file-t.

Fordítási környezetün részleteit a következő címen publikussá tettük:

http://opencvlibrary.sourceforge.net/Getting_started

A kamera képeinek beolvasása, képfeldolgozó műveletek és numerikus műveletek elvégzéséhez az OpenCV könyvtár függvényeit használjuk.

A program használata

A programot parancsori terminálból neye.exe futtatásával indítjuk, mely egy kötelező paramétert vár, a böngészni kívánt képfájl nevét. Második opcinális paraméterként teszt input videófájlt lehet megadni, ezesetben ennek a képeit használja webkamera helyett.

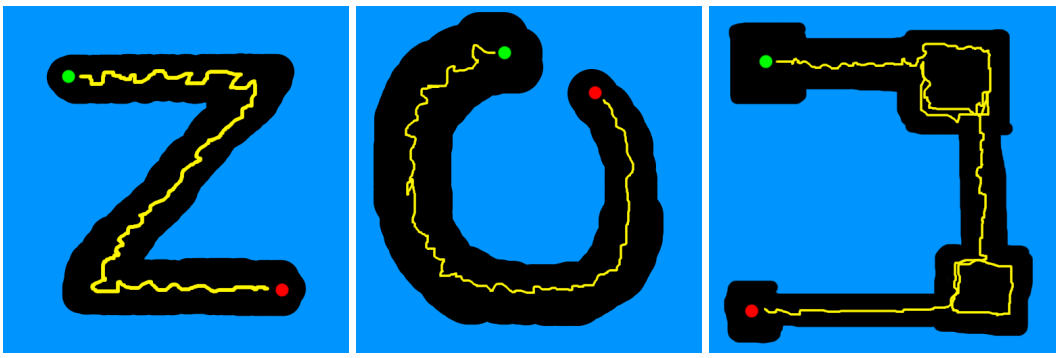
Futtatás közben az alább funkcióbillentyűket használhatjuk:

's' : pozíció alaphelyzetbe állítás; 'r' : forgatás engedélyezés/tiltás; 'x' : kilépés; +/- : zoomolás.

A képben való navigáció a feladatkiírásnak megfelelően a webkamera mozgatásával történik.

Tesztelő környzet

Az általunk fejlesztett HCI minőségét az alábbi teszt segítségével mérjük. Az 5. ábrán látható képeket nyitjuk meg navigáció céljából. A HCI segítségével úgy kell navigálni a képeken hogy a zöld pontból eljussunk a piros pontig, miközben a látómező csak a fekete és sárga mezőket érintheti (a kéket nem). A kék mezők érintése hibának számít. A fekete utakon úgy kell haladni hogy mindig legyen sárga mező a látómezőben. Ha a sárga mezőn hurok van, akkor azon a helyen az egész hurkot látómezőbe kell hozni. A három képen végighaladva számoljuk a hibák számát.



5. ábra: Tesztképek navigációhoz.

Eredmények

1. ellenőrzési pont

A fejlesztői környezetet, az OpenCV és VTK könyvtárak összelinkelését megoldottuk. Készítettünk két egyszerű demóprogramot. Az egyik a feladatot oldja meg, a másik egy 3D-s objektumot forgat a fej jobbra-balra mozgatásával egyszerre a monitorra rögzített kamera segítségével.

A demó programokról készült videóink a következő linkeken elérhetők:

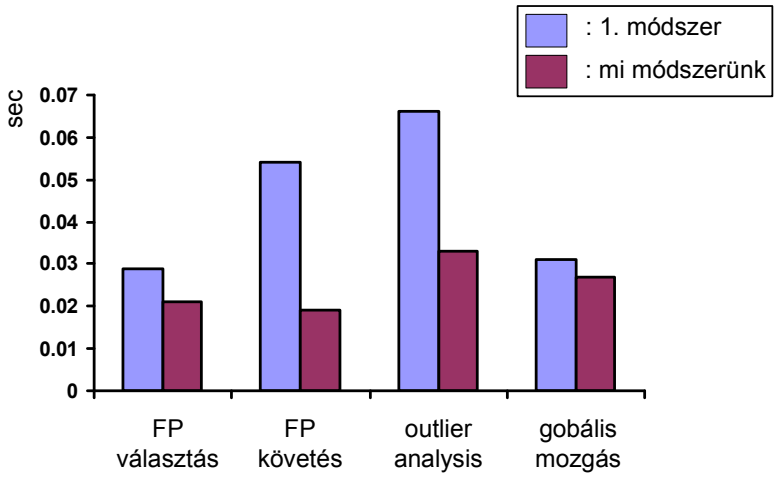
http://www.youtube.com/watch?v=KNiY-u6H7WM

http://www.youtube.com/watch?v=hm-FWnr5UZU

2. ellenőrzési pont

Elkészítettük a [2] referenciában leírt módszer implementációját, azzal az egyetlen módosítással, hogy a mozgás utófeldolgozása nem Kálmán filterrel, hanem egyszerű átlagolással történik. Mivel véleményünk szerint e módszer bonyolultsága nem áll arányban a hatékonyságával, úgy döntöttünk hogy saját ötleteken alapuló módszert is implementálunk.

A két módszer futásidejének elemzését az alábbi ábrán mutatjuk be:



5. ábra: A két implementáció futásidejének elemzése.

Az FP kiválasztás és követés az 1. módszerben műveletigényesebb mint az OpenCV úgyanezt a feladatot ellátó függvényei. Az outlier analízisnél a bonyolultabb mátrixműveletek lelassítják a RANSAC működését az 1. módszer esetén. Ennek oka az elmozdulás meghatározás hibájának továbbvitele kovarianciamátrixok formájában.

3. ellenőrzési pont

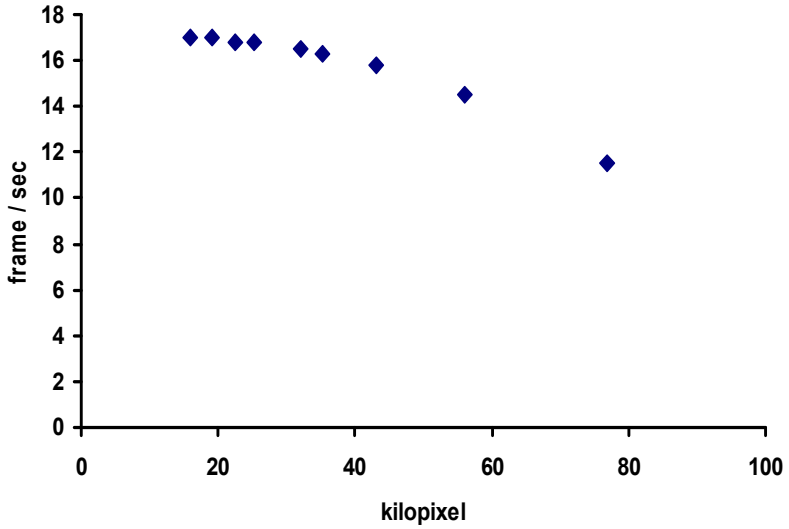
Megpróbáltuk a frame-ek közötti globális fényerő változást frame-enkénti hisztogram kiegyenlítéssel kompenzálni, de mivel ez lokálisan nagy eltéréseket okozott az átlag pixelértékekben, az eredményeinket rontotta, ezért ezt az ötletet elvetettük.

A navigációs tesztképekkel bizonyítottuk hogy a mi módszerünkkel is el lehet érni a kívánt pontosságot, hiszen a feladatok hiba nélkül teljesíthetők.

Célunk volt a számítások gyorsítása. A mátrixműveletek futási idejét sikerült csökkentenünk az implementáció hatékonyabbá tételével, kevesebb szükségtelen memóriafoglalás használatával.

A használhatóságot javítottuk az utófeldolgozás közben végzett átlagolás súlyozásával.

Tesztelés közben megfigyeltük hogy a kamerától kapott kép mérete jelentősen befolyásolja a futás sebességét, ezért méréseket végeztünk az összefüggés felderítése céljából (6. ábra). Ezek alapján sikerült megtalálni a leghasználhatóbb felbontást a mi tesztelő hardverünkre (160x120).



6. ábra: Input képméret és futási sebesség összefüggése.

További lehetőségek

Mozgás predikció

Feature pont szegény input kép esetén a korábbi információk alapján valamekkora időtartományban jósolható lehet a mozgás. Ezzel ellensúlyozni lehet homogén területek fölötti kamera áthúzást.

Térképnézegető

A feladatban meghatározott képnézegető program speciális esete amikor térképet nézünk mobiltelefonon vagy PDA-n. Ez az felhasználás különösen hasznos.

Térképszoftveren lokalizáció

A GPS helymeghatározó rendszerek egyik korlátja hogy csak néhány méteres pontossággal képesek az elmozdulást érzékelni. Ezért pl. amikor elindulunk egy irányba akkor akár több tíz métert is megtehetünk mire a GPS jel alapján a térképszoftver el tudja dönteni hogy milyen irányba indultunk és

ahhoz képest merre kanyarodjunk. Ha a navigáló készüléket kiegészítjük egy földre irányuló kamerából származó mozgásinformációval (pl. autó aljára szerelve), akkor ez nagyon jól kiegészítené a GPS pontatlanságát kis távolságokon.

Készülékek általános vezérlése

A mobil készülékek mozgásinformációit nem csak másféle mozgássá transzformálhatjuk (pl. kép mozgatása) hanem egyéb parancsokat is rendelhetünk a mozgásokhoz. Például a jobbra-balra gyors mozgatás megnyithatja a telefonkönyvet, a körbe mozgatás megnyithat egy programot stb. Egy fotó sorozat megnézése közben kameránkkal apró mozdulatot tehetünk jobbra vagy balra ezáltal lapozva oda-vissza a képek között.

3D tervezőprogramok segítése

Úgy kell tennünk, mintha a valóságban akarnánk megnézni bármilyen 3D-s objektumot. A kamera előtt ülve csak balra-jobbra, előre -hátra dőlünk, az alkalmazásunk pedig a kamera képe alapján elforgatja, átméretezi a monitoron látható objektumot. Ez nagyban segítheti a 3D-s szerkesztőprogramok kényelmes használatát, mivel általában nagyon sok interakció kell a pontos szerkesztéshez.

Hivatkozások

1. Judy York and Parag C. Pendharkar. Human-computer interaction issues for mobile computing in a variable work context. International Journal of Human-Computer Studies. *HCI Issues in Mobile Computing, 2004 (60); 771-797.*
2. Hannuksela J, Sangi P. Heikkilä J. Vision-based motion estimation for interaction with mobile devices. *Computer Vision and Image Understanding: Special Issue on Vision for Human-Computer Interaction, 108(1-2):188-195.*
3. Jilin Tu, Hai Tao and Thomas Huang. Face as mouse through visual face tracking. *Computer Vision and Image Understanding. 2007 (108); 35-40.*
4. http://en.wikipedia.org/wiki/Optical_flow
5. E. De Castro and C. Morandi. Registration of Translated and Rotated Images Using Finite Fourier Transforms. *IEEE Transactions on pattern analysis and machine intelligence, Sept. 1987.*
6. http://www.fxguide.com/article333.html
7. Lucas BD and Kanade T. An iterative image registration technique with an application to stereo vision. Proceedings of Imaging understanding workshop 1981. pp 121–130.
8. Tomasi C, Kanade T. Detection and Tracking of Point Features. Tech Report 1991. http://www.ces.clemson.edu/~stb/klt/tomasi-kanade-techreport-1991.pdf
9. Sangi P, Heikkila J, Silven O. Motion analysis using frame differences with spatial gradient measures. Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on, Volume 4, Issue , 23-26 Aug. 2004: 733 - 736.