Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aleš Hribar

Pretvorba vsesmerne fotografije v panoramsko sliko

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Borut Batagelj

Ljubljana 2012

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00341/2012
Datum: 04.09.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ALEŠ HRIBAR**

Naslov: **PRETVORBA VSESMERNE FOTOGRAFIJE V PANORAMSKO SLIKO**
OMNIDIRECTIONAL TO PANORAMIC IMAGES TRANSFORMATION

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Za video nadzor ter na drugih področjih kjer hočemo s kamero zajeti čim večji prostor se uporabljajo vsesmerne kamere. Takšne kamere lahko zajamejo sliko prostora iz vseh 360 stopinj. Takšna slika pa za nadaljnjo uporabo ni uporabna, ker so predmeti na nji popačeni.

Kandidat naj preuči različne metode s pomočjo katerih lahko takšno sliko iz vsesmernih kamer pretvorimo v panoramsko sliko, ki bo primerna za nadaljnjo obdelavo z metodami računalniškega vida.

Mentor:

viš. pred. dr. Borut Batagelj

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Aleš Hribar, z vpisno številko **63010041**, sem avtor diplomskega dela z naslovom:

Pretvorba vsesmerne fotografije v panoramsko sliko.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr Boruta Batagelja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 24. septembra 2012 Podpis avtorja:

Zahvala za to diplomsko delo gre moji družini ter mentorju, viš. pred. dr. Borutu Batagelju. Brez njih mi ne bi uspelo.

Kazalo

Povzetek

Vsesmerna fotografija ali videoposnetek za praktično rabo ni preveč uporaben, ker je popačen. Z uporabo ustrezne programske opreme ga je možno predelati v panoramo. Panorama je za človeka lažje gledljiva, računalnik pa na njej lažje prepozna objekte s pomočjo računalniškega vida. V tej diplomski nalogi so opisani tipi vsesmernih kamer, njihova uporaba ter postopki transformacije slike. Dva postopka sta v sklopu diplomskega dela tudi realizirana in sta podrobneje opisana. Eden je spisan v programskem okolju Matlab, drugi pa s pomočjo knjižnice CUDA. Algoritmično sta si podobna in oba uporabljata inverzno transformacijo slik. Naštete so tudi možne uporabe vsesmernih kamer.

Abstract

Omnidirectional photos or videos are not very useful in practice. However, they can be transformed into a panoramic picture or video using relevant software. Panoramic pictures are easily viewable by human and also computer software can much easily recognize objects using computer vision on panoramic pictures. This thesis describes different types of omnidirectional cameras, their use and transformations to panoramic image. Furthermore, two different programs are implemented and described in thesis, one being implemented in Matlab platform, the other using CUDA library. They are algorithmically similar and they both use reverse image transformation. Additionally, different applications of omnidirectional cameras are stated in thesis.

Poglavje 1

Uvod

Vsesmerna kamera je sestavljena iz objektiva običajne kamere in nanj pritrjenega zrcala, ki zbira žarke z vseh strani. Informacija je shranjena v kolobarju med dvema črnima krogoma. En krog je črn od znotraj, drugi od zunaj, vmes je kolobar s popačeno sliko. Ne oziraje se na črnino je treba za lažjo obdelavo slike in prepoznavo objektov na njih sliko pretvoriti v panoramsko sliko. Panoramska slika naj bo taka, da čim bolje ohranja kote v naravi, oziroma kote, pod katerim žarki pridejo na zrcalo (da so na različnih koncih enako obravnavani). Pri tem moramo upoštevati obliko zrcala. Od zrcal smo si ogledali hiperbolično in parabolično ter tudi krogelno.

Možnosti uporabe so: avtonomna navigacija, teleprisotnost (prisotnost na daljavo), navidezna resničnost in oddaljeni nadzor [2]. Z dvema zamaknjenima posnetkoma ali z dvema kamerama lahko posnamemo 3-dimenzionalno 360° panoramo [3]. Za oddaljeni nadzor je uporaba teh kamer zanimiva, ker jih, da pokrijemo ves prostor, potrebujemo manj kot običajnih kamer. Pri telekonferenci se z uporabo zaznavanja obrazov lahko loči posamezne osebe, ujete v objektiv. Možne pa so seveda tudi zaznavanja drugih objektov s pomočjo računalniškega vida. Računalniški vid se uporablja skupaj s vsesmerno kamero tudi pri avtonomni navigaciji robotov. Pri teleprisotnosti sistem lahko sledi glede na azimut osebi v okolju kot je muzej ali razstava [2].

Nekateri primeri detekcij so: izdelava zemljevida, navigacija glede na ze-

mljevid ali zaznane točke, izogibanje trkom, nadzor nad plinovodi in vodovodi s pomočjo robota...

Pri tem diplomskem delu je bil izdelan program za pretvorbo vsesmerne fotografije v panoramo in program za avtomatično določanje centra kamere, če je zrcalo zamaknjeno. Pri ugotavljanju centra je bila uporabljena Houghova transformacija za kroge, kjer se izbere največji krog na sliki, če je le dovolj velik. Najprej so bile uprabljene standardne Matlabove funkcije za inverzne transformacije slik (preslikuje se od ponora do izvora), nato smo implementirali algoritem še s CUDA grafično knjižnico. Rezultata sta identična, če ne uporabimo filtriranja. Opisane pa so tudi druge metode pretvorb vsesmernih slik v panoramo.

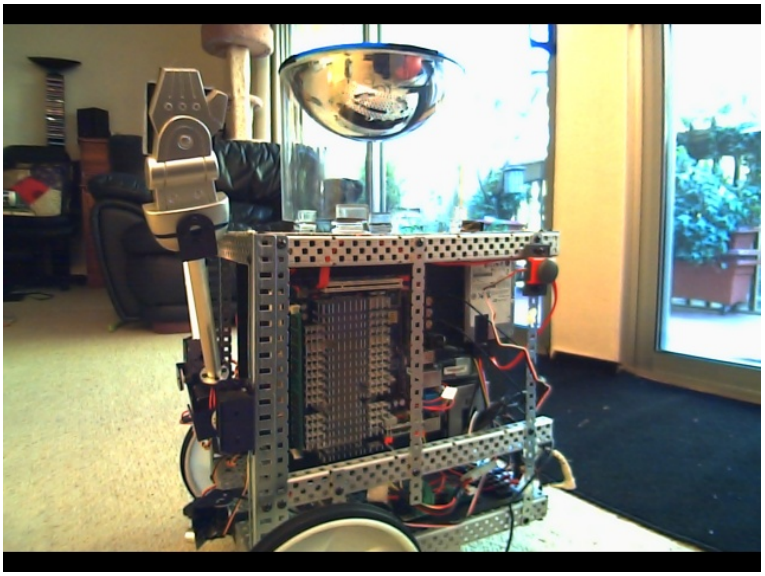
1.0.1 Avtonomna navigacija [2]

Prednosti vsesmerne kamere niso samo velik zoren kot, ampak tudi posebne lastnosti kot sta nespremenljivost azimutov in krožna kontinuiteta.

Ker se navpične linije v prostoru preslikajo v linije, ki gredo skozi središče slike, so lahko zanesljivo zaznane. Poleg tega kot v sliki sovpada z azimutom, kar tudi pomaga pri zanesljivi oceni lokacije linij v prostoru.

Poglavje 2

Opis kamere



Slika 2.1: Fotografija robota Π s krogelnim zrcalom [4].

Kamere, ki zavzamejo velik zorni kot, delimo na katadioptrične in dioptrične. Katadioptrične kamere so sestavljene iz konkavnega zrcala in običajne kamere. Dioptrične kamere imajo lečo pred običajno kamero, ki zbere žarke z želenih smeri. Med katadioptrične kamere sodijo vsesmerne kamere (slika 2.2) s hiperboličnim, paraboličnim ali krogelnim zrcalom (slika 4.5) [5].

Slika 2.2: Kamera 0-360° [6].

Zrcala so lahko različnih dimenzij in parametrov, važna pa je tudi oddaljenost od objektiva [6]. Med dioptrične kamere spada ribje oko [6] (slika 2.3).

Slika 2.3: "Fisheye" kamera [7].

S kamero ribje oko, ki ima kot pogleda 180°, lahko s transformacijami in več slikami v različne smeri dobimo z lepljenjem 360° panoramo [8]. Vsesmerne slike morajo biti pred obdelavo z računalniškim vidom obdelane, ker so dosedanja orodja tipično prirejena perspektivnim slikam. Pogost cilj je izdelava perspektivnih izrezkov, ki omogočajo uporabo teh orodij brez modifikacij. S tem izgubimo prednost velikega zornega kota. Zato je valjčna panorama pogosto najboljša možnost. Pri tej preslikamo vsesmerno sliko na notranjost valja. Taka slika je podobna perspektivni in omili krožna popačenja katadioptričnih in dioptričnih projekcij.

V nadaljevanju bomo podrobneje opisali hiperbolična, parabolična in krogelna zrcala ter njune pretvorbe v valjčne slike.

2.1 Hiperbolično zrcalo

Povzeto po [7]. Naj bo C = (0, 0,-2e) središče kamere na luknjico. Če je žariščna točka F hiperboloida C^2 izvor prostora koordinatnega sistema, je hiperboloid izražen z:

$$(x, y, z, 1) \begin{pmatrix} \frac{1}{a^2} & 0 & 0 & 0 \\ 0 & \frac{1}{a^2} & 0 & 0 \\ 0 & 0 & -\frac{1}{b^2} & -\frac{e}{b^2} \\ 0 & 0 & -\frac{e}{b^2} & -\frac{e^2}{b^2}+1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = 0 \qquad (2.1)$$

V enačbi (2.1) je

$$e = a^2 + b^2 \tag{2.2}$$

Projekcija točke $\mathbf{X} = (X, Y, Z)^\top$ v 3D prostoru v točko $\mathbf{x} = (x, y, z)^\top$ na hiperboloidu C^2 je izražen z $\mathbf{x} = \chi\mathbf{X}$, kjer je (2.3).

$$\chi = \frac{a^2}{b|\mathbf{X}| - eZ} \tag{2.3}$$

Projekcija točke $\mathbf{x}$ v ustrezno točko $\mathbf{m} = (u, v)^\top$ v ravnini slike π je izražena kot:

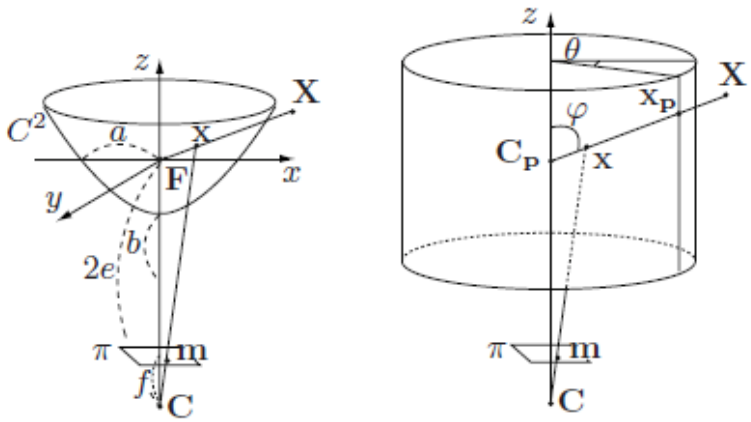
$$\begin{pmatrix} \mathbf{m} \\ 1 \end{pmatrix} = \frac{1}{z + 2e} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} \tag{2.4}$$

Preslikava iz $\mathbf{X}$ v $\mathbf{m}$ je:

$$u = \frac{fa^2X}{(a^2 - 2e^2)Z + 2be|\mathbf{X}|} \tag{2.5}$$

$$v = \frac{fa^2Y}{(a^2 - 2e^2)Z + 2be|\mathbf{X}|} \tag{2.6}$$

Transformacija hiperbolične v valjčno sliko je prikazana desno na sliki 2.4.



Slika 2.4: Pretvorba hiperbolične v valjčno sliko. Levo: sistem kamere s hiperboličnim zrcalom. Desno: hiperbolični in valjčni kombiniran sistem [7].

.

Naj bo $\mathbf{C}_p$ središče valjčne projekcije Naj bo $\mathbf{C_p} = \mathbf{F}$. Potem točki $\mathbf{x_p}$ na valjčni sliki in točka $\mathbf{x}$ na hiperboloidu ležita na črti, ki povezuje točko $\mathbf{X}$ v 3D prostoru s točko žarišča bfF na hiperboloidu. V valjčnem koordinatnem sistemu je točka $\mathbf{x_p} = (\mathbf{x_p}, \mathbf{y_p}, \mathbf{z_p}$ izražena z $x_p = r\cos\theta$, $y_p = r\sin\theta$ in $z_p = r\tan\varphi$. Tu lahko brez izgube splošnosti nastavimo $r = 1$. Enačbi za pretvorbo točke med valjčno sliko $I(u, v)$ in točko na valju (slika 2.4) $I(\theta, \varphi)$ sta potem

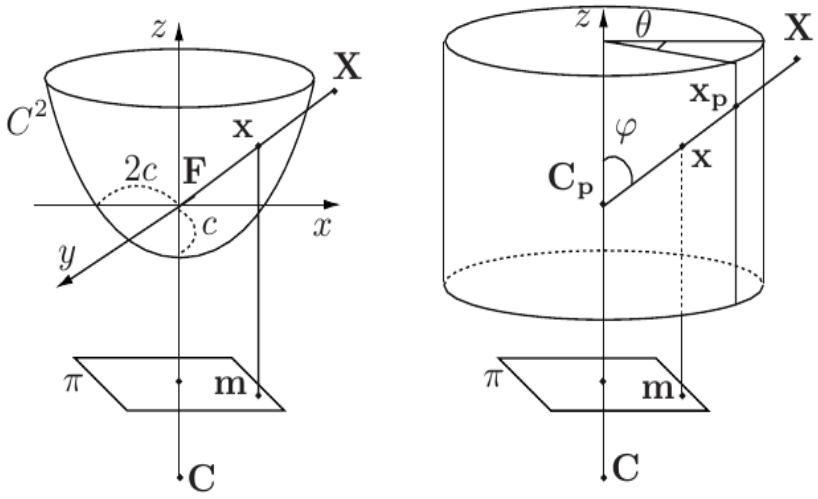
$$u = \frac{fa^2\cos\theta}{(a^2 \mp 2e^2)\tan\varphi \pm 2be\sqrt{1+\tan^2\varphi}} \tag{2.7}$$

$$v = \frac{fa^2\sin\theta}{(a^2 \mp 2e^2)\tan\varphi \pm 2be\sqrt{1+\tan^2\varphi}} \tag{2.8}$$

kjer je $e = a^2 + b^2$, $0 \leq \theta < 2\pi$ in $-\pi/2 \leq \varphi < \pi/2$. Tako dobljeno točko potem interpoliramo z bi-linearnim, bi-kubičnim ali kakšnim drugim filtrom.

Opisane enačbe izpeljujejo preslikavo s središčno valjčno projekcijo. Valjnče slike so lahko osnovane tudi na drugačnih projekcijah (recimo, če nastavimo $z_p = r\varphi$, dobimo ekvi-pravokotno projekcijo).

2.2 Parabolično zrcalo



Slika 2.5: Pretvorba parabolične v valjčno sliko. Levo: sistem kamere s paraboličnim zrcalom. Desno: parabolični in valjčni kombiniran sistem [7].

Najprej si poglejmo skico, prikazano levo na sliki 2.5. Naj bo $\mathbf{C} = (0, 0, -\infty)$ središče ortografske kamere. Če je žarišče $\mathbf{F}$ paraboloida C^2 izvor prostora koordinatnega sistema, je paraboloid C^2 izražen z (2.9).

$$(x, y, z, 1) \begin{pmatrix} \frac{1}{4c} & 0 & 0 & 0 \\ 0 & \frac{1}{4c} & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = 0 \tag{2.9}$$

Projekcija točke $\mathbf{X} = (X, Y, Z)^\top$ v 3D prostoru v točko $\mathbf{x} = (x, y, z)^\top$ na paraboloidu je izražen z $\mathbf{x} = \chi\mathbf{X}$, kjer je (2.10).

$$\chi = \frac{2c}{|\mathbf{X}| - Z} \tag{2.10}$$

Projekcija točke $\mathbf{x}$ v ustrezno točko $\mathbf{m} = (u, v)^\top$ v ravnini slike π je izražena kot (2.11):

$$\begin{pmatrix} \mathbf{m} \\ 1 \end{pmatrix} = \frac{1}{z + 2e} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} \tag{2.11}$$

Preslikava iz $\mathbf{X}$ v $\mathbf{m}$ je (2.12) in (2.13).

$$u = \frac{f2cX}{|\mathbf{X}| - \mathbf{Z}} \tag{2.12}$$

$$v = \frac{f2cY}{|\mathbf{X}| - \mathbf{Z}} \tag{2.13}$$

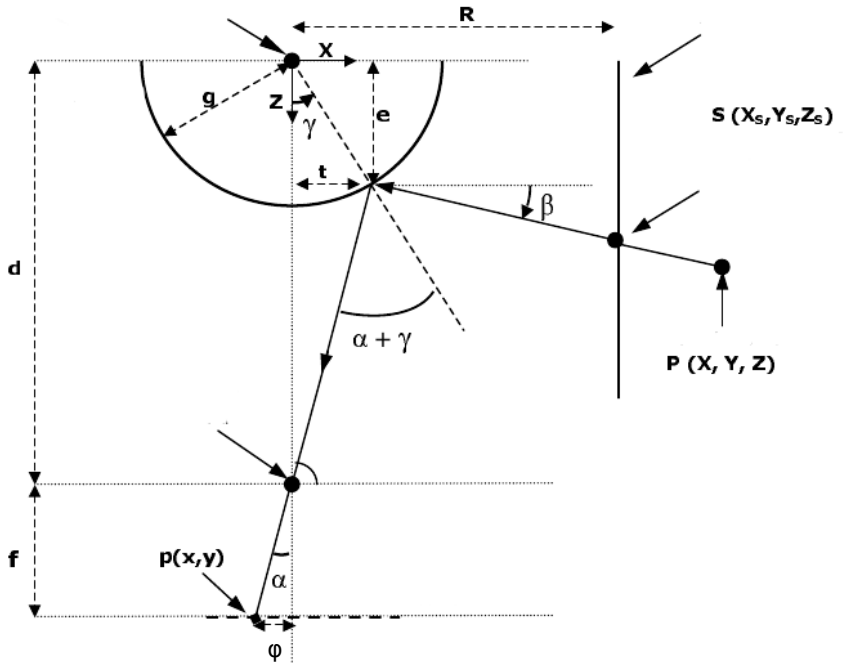
Če nastavimo $\mathbf{C_p} = \mathbf{F}$, točka x_p na panoramski sliki in točka $\mathbf{x}$ na paraboloidu ležita na črti, ki povezuje točko $\mathbf{X}$ v 3d prostoru z žariščno točko $\mathbf{F}$ na paraboloidu. Naj bo $\mathbf{x}_p = (x_p, y_p, z_p)$ točka na površini valja. Enačbi za pretvorbo točke med valjčno sliko $I(u, v)$ in točko na valju (slika 2.5) $I(\theta, \varphi)$ sta (2.14) in (2.15).

$$u = \frac{2c\cos\theta}{\sqrt{1 + \tan^2\varphi} - \tan\varphi} \tag{2.14}$$

$$v = \frac{2c\sin\theta}{\sqrt{1 + \tan^2\varphi} - \tan\varphi} \tag{2.15}$$

Tako dobljeno točko potem interpoliramo z bi-linearnim, bi-kubičnim ali kakšnim drugim filtrom.

2.3 Krogelno zrcalo



Slika 2.6: Shema kamere s krogelnim zrcalom [9].

Premica, ki določa kot α, je (2.16) [1]. Enačba kroga v ravnini je (2.17) [1].

$$y = kx = \cot(\alpha)x = \frac{f}{\varphi}x \tag{2.16}$$

$$R^2 = x^2 + (y - d)^2 \tag{2.17}$$

Rešimo sistem enačb za α in posebej za $x(\varphi)$. Dobimo (2.18).

$$\alpha = \cot^{-1}\left(\frac{d + \sqrt{g^2 - x^2}}{x}\right) \tag{2.18}$$

$$x = \frac{d\frac{f}{\varphi}}{1 + \frac{f^2}{\varphi^2}} - \frac{\sqrt{d^2\frac{f^2}{\varphi^2} - (d^2 - g^2)(1 + \frac{f^2}{\varphi^2})}}{1 + \frac{f^2}{\varphi^2}} \tag{2.19}$$

Izpeljemo enačbe za u (2.20) in v (2.21), kjer je x funkcija (2.19).

$$u = \frac{f}{\left(\frac{d+\sqrt{g^2-x^2}}{x}\right)} \cos\theta \tag{2.20}$$

$$v = \frac{f}{\left(\frac{d+\sqrt{g^2-x^2}}{x}\right)} \sin\theta \tag{2.21}$$

Poglavje 3

Pretvorba v panoramo z inverzno transformacijo

V tem diplomskem delu bomo opisali najprej povratno (inverzno) transformacijo slike. Pri teh transformacijah za vsako slikovno piko na ciljni sliki poiščemo preslikavo v izvorni sliki.

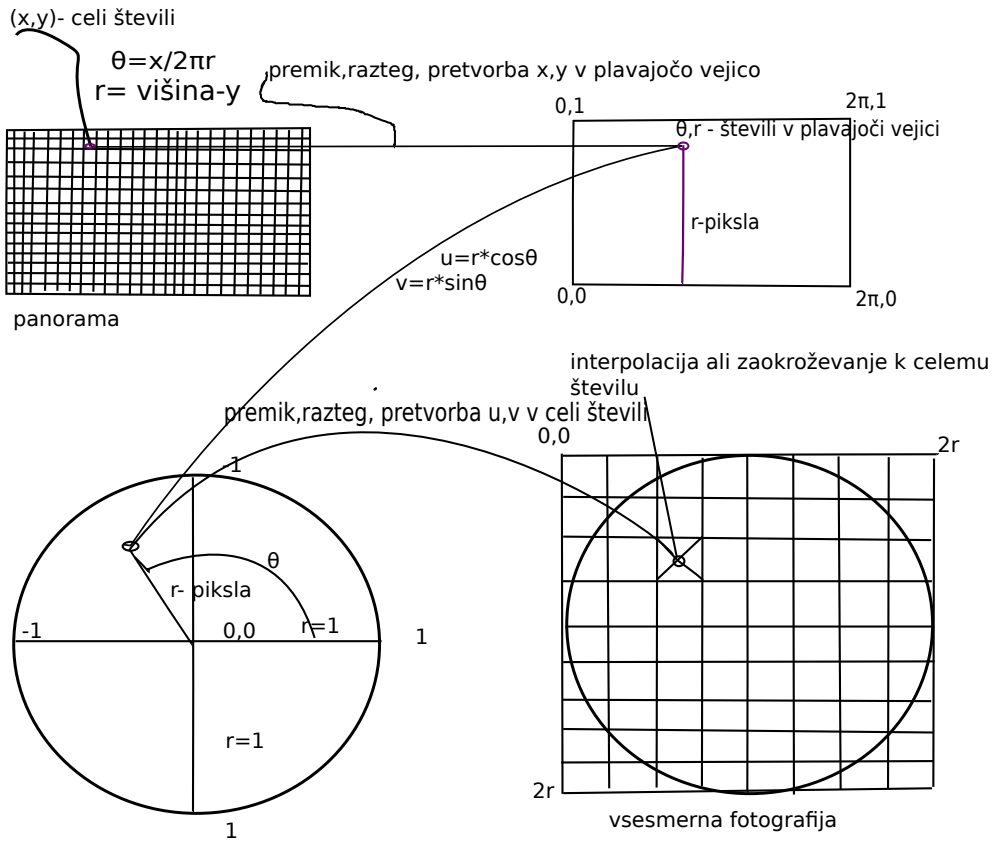
Končna slika zavzame kot 360° valja, torej $2\pi r$. Velikost izvorne slike je višina krat širina, dimenziji sta enaki in sta enakovredni dvakratnemu polmeru kroga, torej $2r$. Obseg kroga 360° izvorne slike je $2\pi r$. Optimalna višina je r. Tako na vrhu (obodu) končne slike dobimo sliko točka za točko, nižje pa se ena izvorna točka preslika v več končnih točk, če namesto interpolacije uporabimo zaokroževanje k najbližjemu celemu številu. Za preslikavo uporabimo enačbo preslikave kartezičnega v polarni koordinatni sistem, ki je:

$$u = r * cos\theta \tag{3.1}$$

$$v = r * sin\theta \tag{3.2}$$

Najprej pretvorimo točko s končne slike (x, y) v prostor(x', y'), $x \in (0, ...2\pi r)$, $y \in (r, ...0)$. Slikovna pika (x, y) postane točka (x', y'). Pri tem sta x' in y' elementa v plavajoči vejici. Z enačbama (3.1) in (3.2) pa pretvorimo vsak tak element v prostor (u, v), u$\in$ [1,1] ter v $\in$[-1,1]. Nato element (u,v) raztegnemo na izvorno sliko (x_0, y_0), x_i $\in$0,...,2r, y_i $\in$0,...,2r. Elementu

(x_i, y_i) lahko porežemo decimalke ali ga zaokrožimo na najbližje celo število in dobimo koordinate izvore slikovne točke (x, y), ki ga nato preslikamo na končno sliko, kot je opisano. Lahko pa nad (x_i, y_i) uporabimo filtriranje. Shema je predstavljena na sliki 3.1.



Slika 3.1: Shema za pretvorbo vsesmerne fotografije v panoramo

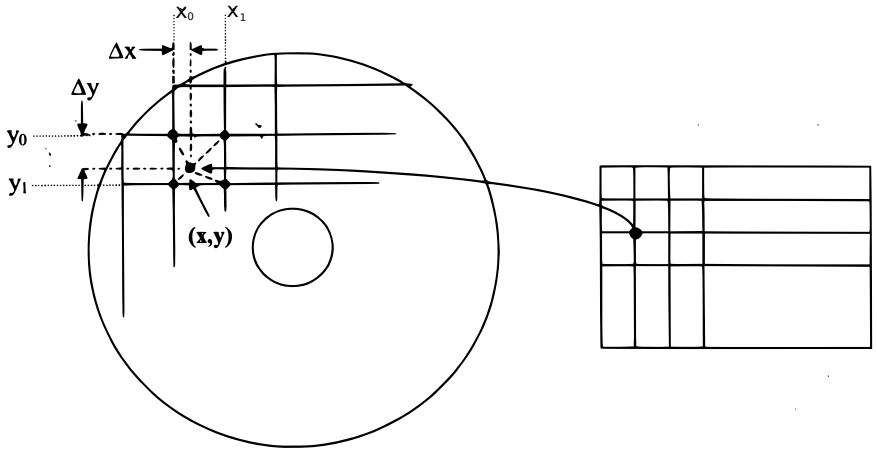
V tem delu smo opisali bi-linearno filtriranje. V jeziku Matlab je filtriranje opcijsko, izberemo ga z ukazom pri transformaciji.

3.1 Bi-linearna interpolacija [1]

Pri bi-linearni interplolacijski metodi je intenziteta slikovne pike v izhodni sliki odvisna od štirih slikovnih pik izvorne slike v okolici elementa (x_0, y_0) po enačbi (3.3) in sliki 3.2.

$$\begin{aligned} I(x,y) \quad &= I(x_0,y_0) + [I(x_1,y_0) - I(x_0,y_0)] * \Delta x + [I(x_0,y_1) - I(x_0,y_0) * \Delta y + \\ &+ I(x_1,y_1) + I(x_0,y_0) - I(x_0,y_1) - I(x_1,y_0)] * \Delta x * \Delta y \end{aligned} \tag{3.3}$$

[1]



Slika 3.2: Interpolacija z bilinearnim filtrom [1].

Obstajajo še druge interpolacije, kot je bi-kubična, vendar jih v tem delu ne bomo obravnavali.

3.2 Iskanje centra vsesmerne fotografije s Houghovo transformacijo

Zrcalo za vsesmerno sliko je težko centrirati na objektiv. Zato potrebujemo postopek, ki ugotavlja center slike. Najlažje za programerja je ročni vnos, kjer uporabnik sam določi center razvijanja. Boljši je avtomatični način, kjer s Houghovo transformacijo poiščemo kroge (slika 3.3). Center največjega kroga je zelo verjetno center pripenjanja zrcala.

Slika 3.3: Detekcija krogov s Houghovo transformacijo.

Ni pa nujno. Krog mora biti v celoti na sliki, če ne, ne moremo pretvoriti slike v panoramo (nastopajo slikovne točke, ki niso na izvorni sliki). Tudi krog, s premerom, manjšim od 80 % višine slike verjetno ni pravi, ker običajno fotograf poskusi kar se da optimalno namestiti zrcalo. Poleg tega obstaja še ena črna notranjost kroga, kjer kamera posname objektiv in je precej manjši. Z upoštevanjem teh postopkov lahko precej natančno določimo center razvijanja slike.

Več o detekciji krogov s Houghovo transformacijo si lahko preberete v [10].

3.3 Implementacija s programsko knjižnico CUDA

Programska knjižnica CUDA je namenjena poganjanju GPGPU (General-purpose computing on graphics processing units) kode na Nvidia grafičnih karticah. Koda se izvaja paralelno na več procesnih enotah. Programska koda se piše v C/C++ podobni sintaksi. Ločimo kodo, ki se izvaja na centralni procesni enoti (CPU) in kode (imenovano kernels), ki se izvajajo na GPU-jih. Konkurenčna knjižnica programski knjižnici CUDA (Compute Unified Device Architecture) je programska knjižnica OpenCL, ki poleg na Nvidijinih deluje tudi na GPU-jih ostalih proizvajalcev (AMD, Intel). Transformacije v programu na GPU-jih so predstavljene v kodi 3.1.

Koda 3.1 (Transformacije s pomočjo knjižnice CUDA)

```
...
//poiscemo x in y izhodne slike s pomocjo indexa i, ki
//tece cez vse elemente
int x = remainder((double)i,(double)newwidth);
int y = floorf((double)(i/newwidth));

//postavimo x1 med 0 in 2*pi*r, r = visina
double x1 = -((double)(x)*2*CUDART_PI_F/(double)newwidth);

//postavimo y med 0 in ena po obratni vrednosti r
double y1 = ((double)(y)/(double)newheight)-1;

// enacbi transformacije
double oldx1 = y1*cosf(x1);
double oldy1 = y1*sinf(x1);

// razteg na koncno sliko izvornega x (oldx) in y (oldy).
double oldx = ((oldx1 + 1)*(double)width/2);
double oldy = ((oldy1 + 1)*(double)height/2);
```

```
//dolocimo nov index slike i,  ki tece od
zacetka od leve proti desni
int oldi = floorf(oldy)*width+floorf(oldx);

//filtriranje
filter(imagem, outimage, width, height, oldx, oldy, i);
...
```

Poglavje 4

Pretvorba v panoramo z Matlabom

Drugi program za pretvorbo vsesmerne slike v panoramo smo spisali s pomočjo Matlaba. Slike lahko naložimo kot matriko v Matlab in ji določimo spremenljivko. Program vsebuje pet funkcij: unroll.m, unroll_hyperbolic.m, unroll_parabolic.m, unroll_basic.m in unroll_krogla.m. Poleg teh funkcija unroll.m kliče zunanji C++ program, unroll_center (unroll_center.exe v okolju Win32), ki določi center preslikave in polmer kroga za preslikavo. Polmer se v nadaljnjih izračunih izpusti. Tu je samo zaradi kompatibilnosti s CUDA programom, v katerega je ta program tudi vključen. Za določanje kroga preslikave in njegovega centra ter polmer smo uporabili knjižnico OpenCV, namenjeno uporabi pri računalniškem vidu.

4.1 Program za izbiro vrste slike

Vrsta slike se poda funkciji unroll.m v Matlabovi lupini v obliki:

```
unroll('Slika.jpg;', 'tip', parametri);
```

`Slika.jpg` kaže na pot do fotografije, ki jo želimo pretvoriti, `tip` je ena od črk `'b', 'h', 'p' ali 'k'` in določa tip transformacije, oziroma vrsto uporabljenega objektiva v izvorni fotografiji.

Za osnovno pretvorbo moramo vpisati v Matlabovo lupino

```
unroll('vsesmerna_slika','b);.
```

To nam naredi osnovno pretvorbo. Funkcija zahteva uporabnikovo interakcijo za obrezovanje slike. Določi se samo spodnja in zgornja meja.

Funkcija unroll vrača dva parametra: sliko in čas računanja. Pokličemo jo kot:

```
[img,time]=unroll(...);.
```

4.2 Osnovna pretvorba brez upoštevanja zrcala

Osnovna pretvorba je bila matematično opisana v prejšnjem poglavju. Matlab ima močno orodje za transformacijo slik in nam ni treba skrbeti za malenkosti, kot so razteg slike. Treba je samo napisati enačbo inverzne transformacije in območje [x, y] končne slike 4.2 in [u, v] začetne 4.1 slike.

Slika 4.1: Fotografija, posneta s 0-360° zrcalom [7].

Slika 4.2: Razširjena fotografija 4.1 s programom unroll [7].

4.3 Objektiv s hiperboličnim zrcalom

Slika 4.3: Fotografija, posneta s hiperboličnim zrcalom [11].

Slika 4.4: Razširjena fotografija 4.3 s programom unroll [11].

Pri pretvorbi vsesmernih slik bi želeli, da se enaki koti, ki prihajajo do zrcala, preslikajo po navpični osi v enako število slikovnih točk.

Hiperbolična kamera je sestavljena iz hiperboličnega ogledala in običajnega digitalnega fotoaparata [8,10]. Vsi svetlobni žarki, ki pridejo na kamero, sekajo navidezno skozi točko F hiperboloida, kot je prikazano na sliki 2.4.

Iz enačb za u in v se vidi, da je razlika med njima samo v funkcijah cos in sin, ostali členi pa so si identični, in tvorijo razteg po y-osi v končni sliki. Funkcija nas zanima samo med njeno ničlo in $-\pi/2$, ker drugače dobimo dvojno sliko. Ničlo funkcije je v Matlabu enostavno izračunati s funkcijo `fsolve`, kot na primeru programa 4.1.

Koda 4.1 (Transformacije s pomočjo Matlabovih funkcij)

```
function [ imgWrite ] = unroll_hyperbolic(imeSlike, ...
x, y, a, b, focus)
%UNROLL_HYPERBOLIC Vrne razvito panoramsko sliko
% vsesmerne fotografije.
sourcev = imread(imeSlike);

%... celoten program je v prilogi

e=sqrt((a.^2)+(b.^2));

% Funkcije preslikave:
%Izpostavimo qt v x in y
qt=@(x) focus.*(a.^2)./(((a.^2)-2.*(e.^2)).*tan(x) + ...
    2.*b.*e.*sqrt(1+tan(x).^2));

iks=@(x) qt(x(:,2)).*cos(x(:,1));
ypsilon=@(y) qt(y(:,2)).*sin(y(:,1));
ipanorama=@(x) [iks(x), ypsilon(x)];
inverse=@(x, t) ipanorama(x);

% Iskanje ničle funkcije qt1 z fsolve in izračun slike;

qt1=@(x) qt(x)-1;

% Začnemo v -pi/2;
```

```
x0=-pi/2;
tform2 = maketform('custom', 2, 2, [], inverse, []);
imgWrite = imtransform(source, tform2, 'bilinear', ...
                       'UData', [-1 1], ...
                       'VData', [-1 1], 'XData', [2*pi 0], ...
                       'YData', [fsolve(qt1, x0)  -pi/2], ...
                       'Size', [newheight newwidth]);
end
```

Za pretvorbo slike s hiperboličnim zrcalom poženemo

`unroll('hiperbolicna_slika.jpg', 'h', 39.292, 19.646, 2.3);`

Parametri: a = 39.292, b = 19.646, žariščna razdalja = 2.3. Lahko pa so tudi drugačni, glede na lastnosti zrcala.

4.4 Objektiv s paraboličnim zrcalom

Za pretvorbo parabolične slike vpišemo v lupino Matlaba:

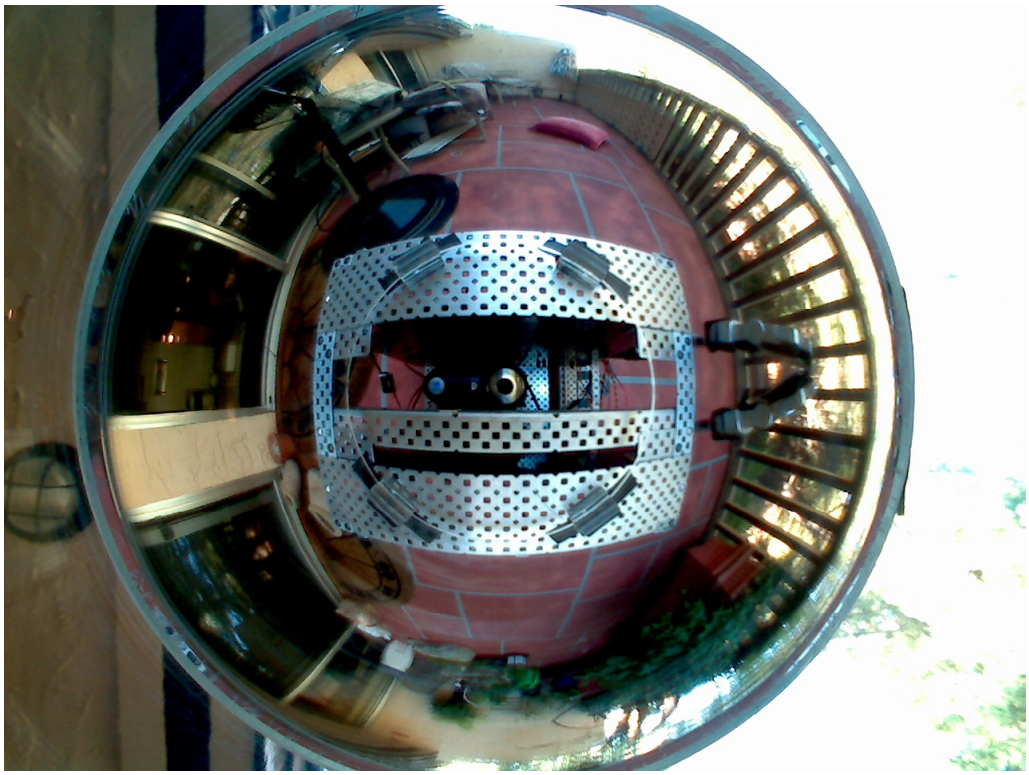
`unroll('parabolicna_slika.jpg','p', 2.343);`,

kjer zadnji parameter 2.343 določa parameter c parabole.

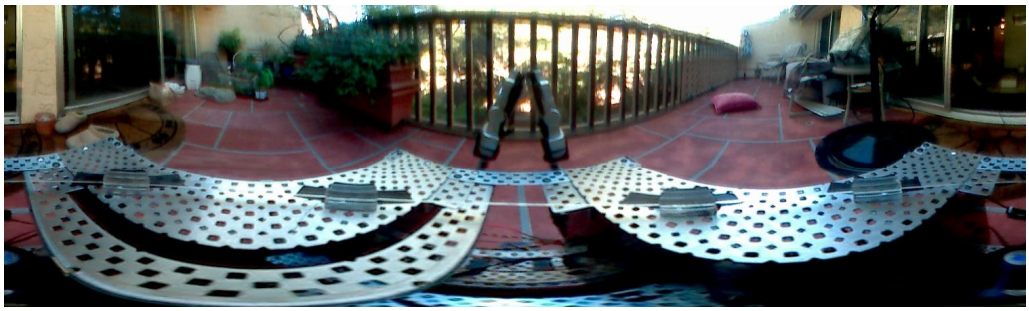
4.5 Objektiv s krogelnim zrcalom

Za pretvorbo parabolične slike vpišemo v lupino Matlaba:

`unroll('krogelna_slika.jpg','k',g,d,focus);`, kjer je `g` polmer zrcala, `d` razdalja med središčem krogelnega zrcala in kamero ter `focus` žariščna razdalja kamere.



Slika 4.5: Fotografija, posneta s krogelnim zrcalom [4].



Slika 4.6: Razširjena fotografija 4.5 s programom unroll [4].

Poglavje 5

Sklepne ugotovitve

Za inverzno transformacijo so paralelni algoritmi smiselni. Čas izvajanja na CUDA pri 96 procesnih enotah (Nvidia Optimus 540M) je za 10-krat hitrejši, kot pa algoritem, ki deluje v Matlabu na istem računalniku, Core i7. Konkretno, za transformacijo brez upoštevanja zrcala je CUDA program na sliki Marsket0.jpg velikosti 2448x2448 (obrezana slika na podlagi Hughove transformacije), potreboval 0,19 sekund, proogram s programsko zanko 0,62 sekunde, zgoraj opisan Matlabov program pa 2,27 sekunde 5.1. Matlabovim programom se prišteje še 10 % k času izvajanja, ko se upošteva oblika zrcala.

Z uporabo grafičnih procesnih enot za preslikavo lahko preslikavo v valjčno panoramo ob ustrezni strojni podpori izvajamo tudi v realnem času, kar programsko pri visoki ločljivosti še ni mogoče.

Realizacija	čas (s)
Nvidia Optimus 540M	0,19
Programska zanka	0,62
Matlab	2,27

Slika 5.1: Realizacije in časi izvajanja.

Slike

Literatura

[1] Y. Baştanlar, "Parameter extraction and image enhancement for catadioptric omnidirectional cameras," Dostopno na: http://www.ii.metu.edu.tr/~yalinb/publications/Bastanlar-Thesis.pdf, 2012.

[2] Y. Yagi, "Omnidirectional sensing and its applications." v *IEICE Transactions on Information and Systems*, jul 1995, stran 568 – 579.

[3] A. Simon, R. Smith, in R. Pawlicki, "Omnistereo for panoramic virtual environment display systems," v *Virtual Reality, 2004. Proceedings. IEEE*, marec 2004, stran 67 – 279.

[4] "Pi robot," Dostopno na: http://www.pirobot.org/blog/0001/, 2012.

[5] T. Svoboda, "Central panoramic cameras: Geometry and design," 1997.

[6] "Some omnidirectional shots durin flying," Dostopno na: http://wwwlasmea.univ-bpclermont.fr/Personnel/Maxime.Lhuillier/OmniPara.html, 2012.

[7] A. Torii in R. Klette, "Panoramic and 3d computer vision," v *ArtsIT, LNICST*, 2012, stran 9 – 16.

[8] H. Bakstein in T. Pajdla, "Panoramic mosaicing with a 180 deg; field of view lens," v *Omnidirectional Vision, 2002. Proceedings. Third Workshop on*, 2002, stran 60 – 67.

[9] A. Ohte, O. Tsuzuki, in K. Mori, "A practical spherical mirror omnidirectional camera," v *Robotic Sensors: Robotic and Sensor Environments, 2005. International Workshop on*, 30 2005-oct. 1 2005, stran 8 – 13.

[10] D. Kerbyson in T. Atherton, "Circle detection using hough transform filters," v *Image Processing and its Applications, 1995., Fifth International Conference on*, jul 1995, stran 370 –374.

[11] "Catadioptric cameras for 360 degree imaging," Dostopno na: http://www1.cs.columbia.edu/CAVE/projects/cat_cam_360/, 2012.

Poglavje 6

Priloga

Priloženi so programi, ki smo jih razvili za potrebe te diplomske naloge. Prvi je program, napisan za knjižnico CUDA, opisan v poglavju 3.

Priloga k 3.1:

Program unroll_cuda

Program poženemo kot: `unroll vhodna_datoteka.jpg izhodna_datoteka.jpg`. *Pri tem je* `vhodna_datoteka.jpg` *vhodna datoteka tipa* `jpg` *in* `izhodna_datoteka.jpg` *izhodna datoteka tipa* `jpg`.

Program prebere vhodno vsesmerno sliko iz datoteke, s Hughovo transformacijo poišče center in izpiše panoramsko sliko vhodne slike v izhodno datoteko.

Razdeljen je v tri datoteke:

unroll.cpp

Glavni program s funkcijo `main()`*:*

```
1  #include "unroll_kernels.h"
2  #include <opencv/cv.h>
3  #include <opencv/highgui.h>
4  #include <iomanip>
5  #include <iostream>
6  #include <math.h>
7  #include <sys/timeb.h>
8  #include <time.h>
9
10
11 #define DEBUG_TIME
12
13 int notEntirely(int x, int y, int r, int width, int height) {
14    if ( (x-r<0) || (x+r>width) || (y-r<0) || (y+r>height)) return 1;
15    else return 0;
16 }
17
18 int main(int argc, char** argv)
19 {
20    if (argc < 3)
21      {
22        std::cout << "Uporabi: " << argv[0] << " <input>" << " <output>" <<
23                  << std::endl;
24        exit(1);
25      }
26
```

```
27    IplImage* input_image = NULL;
28    input_image = cvLoadImage(argv[1], CV_LOAD_IMAGE_UNCHANGED);
29    if(!input_image)
30      {
31        std::cout << "ERROR: Failed to load input image" << std::endl;
32        return -1;
33      }
34
35    IplImage* gray = cvCreateImage(cvGetSize(input_image), 8, 1);
36    CvMemStorage* storage = cvCreateMemStorage(0);
37    cvCvtColor(input_image, gray, CV_BGR2GRAY);
38    cvSmooth(gray, gray, CV_GAUSSIAN, 9, 9);
39    CvSeq* circles = cvHoughCircles(gray, storage,
40                                    CV_HOUGH_GRADIENT, 2, gray->height/8,
41                                    200, 100, gray->height/8, (gray->width>
42                                                               gray->height ?
43                                                               gray->height :
44                                                               gray->width));
45
46    int width = input_image->width;
47    int height = input_image->height;
48
49    //ugotovi najvecji krog
50    int maxx=0;    int maxy=0;      int maxr=0;      int i;
51
52    for (i = 0; i < circles->total; i++)
53      {
54        float* p = (float*)cvGetSeqElem( circles, i );
55
56        int x=cvRound(p[0]); int y=cvRound(p[1]); int r=cvRound(p[2]);
57
58        if (r>maxr) {
59          maxx=x; maxy=y; maxr=r;
60        }
61
62        /*          cvCircle( simg, cvPoint(cvRound(p[0]),cvRound(p[1])),
63                    3, CV_RGB(0,255,0), -1, 8, 0 );
64                    cvCircle( simg, cvPoint(cvRound(p[0]),cvRound(p[1])),
65                    cvRound(p[2]), CV_RGB(255,0,0), 3, 8, 0 );
66        */
67      }
68
69    // Ce krog ni v celoti na sliki, vzemi kar geometrijsko sredisce slike ...
70    if (notEntirely(maxx,maxy,maxr, gray->width,gray->height)) {
71      maxx=cvRound(gray->width/2);
72      maxy=cvRound(gray->height/2);
73      maxr=( (gray->width > gray->height) ?
74             cvRound(gray->height/2) : cvRound(gray->width/2));
75    }
76
77    // Ce je krog manjsi od 80% najmanjse zunanje dimenzije slike,
78    //vzemi geometrisko sredisce slike
79
80    if ( maxr<0.8*((gray->width > gray->height) ?
81                   cvRound(gray->height/2) : cvRound(gray->width/2))){
82      maxx=cvRound(gray->width/2);
83      maxy=cvRound(gray->height/2);
84      maxr=( (gray->width > gray->height) ?
85             cvRound(gray->height/2) : cvRound(gray->width/2));
86    }
87
88    int r=0;
89    int newheight=r;
90
```

```
91    if (height > 2 * maxy ) newheight = maxy;
92    else newheight = height-maxy;
93
94    r=maxr-(maxr%8);
95    newheight=r+(16-r%16);
96    int newwidth= CUDART_PI_F*r+(16-(int)(CUDART_PI_F*r)%16);
97
98    printf("%d_%d_%d\n",newheight,newwidth,r);
99
100   cvSetImageROI( input_image,  cvRect( maxx-r, maxy-r, 2*r, 2*r ) );
101   IplImage* img = cvCreateImage(cvSize(2*r, 2*r), input_image->depth,
102                                 input_image->nChannels );
103   cvCopy( input_image, img,NULL );
104   cvResetImageROI( input_image );
105   /*
106     cvNamedWindow( "circles", 1 );
107     cvShowImage( "circles", img );
108     cvSaveImage("circle_output.jpg", img);
109     cvWaitKey(0);
110   */
111
112   int bpp = input_image->nChannels;
113
114 #ifdef DEBUG
115   std::cout << ">>_Width:" << width << std::endl <<
116     ">>_Height:" << height << std::endl <<
117     ">>_Bpp:" << bpp << std::endl;
118
119   std::cout << std::endl << ">>>_Debugging_Original_data:" << std::endl;
120   for (int i=0; i < width*height*bpp; i+=bpp)
121     {
122       if (!(i % (width*bpp)))
123         std::cout << std::endl;
124
125       std::cout << std::dec << "R:"<< (int) input_image->imageData[i] <<
126         "_G:" << (int) input_image->imageData[i+1] <<
127         "_B:" << (int) input_image->imageData[i+2] << "_____";
128     }
129   std::cout << std::endl << std::endl;
130 #endif
131
132   float* cpu_image = new float[width * height * 4];
133   if (!cpu_image)
134     {
135       std::cout << "ERROR:_Failed_to_allocate_memory" << std::endl;
136       return -1;
137     }
138
139   for (int i = 0; i < 2*r * 2*r; i++)
140     {
141       cpu_image[i * 4 + 0] = (unsigned char)img->imageData[i * bpp + 0] / 255.f;
142       cpu_image[i * 4 + 1] = (unsigned char)img->imageData[i * bpp + 1] / 255.f;
143       cpu_image[i * 4 + 2] = (unsigned char)img->imageData[i * bpp + 2] / 255.f;
144     }
145
146   float* cpu_outimage = new float[newwidth * newheight * 4];
147   if (!cpu_outimage)
148     {
149       std::cout << "ERROR:_Failed_to_allocate_memory" << std::endl;
150       return -1;
151     }
152
153 #ifdef DEBUG_TIME
154   //Start clock
```

```
155     struct timeb start_time_st;
156     ftime(&start_time_st);
157   #endif
158
159     float* gpu_image = NULL;
160     cudaError_t cuda_err = cudaMalloc((void **)(&gpu_image),
161                                       (2*r * 2*r * 4) * sizeof(float));
162     if (cuda_err != cudaSuccess)
163       {
164         std::cout << "ERROR: Failed cudaMalloc" << std::endl;
165         return -1;
166       }
167
168     cuda_err = cudaMemcpy(gpu_image, cpu_image,(2*r * 2*r * 4) * sizeof(float),
169                           cudaMemcpyHostToDevice);
170     if (cuda_err != cudaSuccess)
171       {
172         std::cout << "ERROR: Failed cudaMemcpy" << std::endl;
173         return -1;
174       }
175
176     float* gpu_outimage = NULL;
177     cudaError_t cuda_err1 = cudaMalloc((void **)
178                                        (&gpu_outimage),
179                                        (newwidth * newheight * 4)
180                                        * sizeof(float));
181     if (cuda_err1 != cudaSuccess)
182       {
183         std::cout << "ERROR: Failed cudaMalloc" << std::endl;
184         return -1;
185       }
186
187     dim3 block(16, 16);
188     dim3 grid((int)ceil(double((newwidth * newheight) / 256.0)));
189
190     cuda_transform(gpu_image, gpu_outimage, 2*r, 2*r,
191                    newwidth, newheight, grid, block);
192
193     cudaMemcpy(cpu_outimage, gpu_outimage, (newwidth * newheight * 4)
194                * sizeof(float), cudaMemcpyDeviceToHost);
195     if (cuda_err != cudaSuccess)
196       {
197         std::cout << "ERROR: Failed cudaMemcpy" << std::endl;
198         return -1;
199       }
200
201   #ifdef DEBUG_TIME
202     // Stop clock
203     struct timeb stop_time_st;
204     ftime(&stop_time_st);
205     double elapsed = ((double) stop_time_st.time +
206                       ((double) stop_time_st.millitm * 0.001)) -
207       ((double) start_time_st.time + ((double) start_time_st.millitm * 0.001));
208
209     std::cout << "* Time elapsed: " << std::setprecision(5) <<
210               << elapsed << " sec" << std::endl;
211   #endif
212
213     char* buff = new char[newwidth * newheight * bpp];
214     if (!buff)
215       {
216         std::cout << "ERROR: Failed to allocate memory" << std::endl;
217         return -1;
218       }
```

```
219
220
221     for (int i = 0; i < (newwidth * newheight); i++)
222       {
223         buff[i * bpp + 0] = (char)floor(cpu_outimage[i * 4 + 0] * 255.f);
224         buff[i * bpp + 1] = (char)floor(cpu_outimage[i * 4 + 1] * 255.f);
225         buff[i * bpp + 2] = (char)floor(cpu_outimage[i * 4 + 2] * 255.f);
226       }
227
228   #ifdef DEBUG
229     std::cout << std::endl << ">>> Debugging Output data:" << std::endl;
230     for (int i=0; i < width*height*bpp; i+=bpp)
231       {
232         if (!(i % (width*bpp)))
233           std::cout << std::endl;
234
235         std::cout << std::dec << "R:"<< (int) buff[i] <<
236           " G:" << (int) buff[i+1] <<
237           " B:" << (int) buff[i+2] << "     ";
238       }
239     std::cout << std::endl << std::endl;
240   #endif
241
242     IplImage* out_image = cvCreateImage(cvSize(newwidth, newheight),
243                                         input_image->depth, bpp);
244     if (!out_image)
245       {
246         std::cout << "ERROR: Failed cvCreateImage" << std::endl;
247         return -1;
248       }
249
250     out_image->imageData = buff;
251
252     if (!cvSaveImage(argv[2], out_image))
253       {
254         std::cout << "ERROR: Failed cvSaveImage" << std::endl;
255       }
256
257     cuda_err = cudaFree(gpu_image);
258     if (cuda_err != cudaSuccess)
259       {
260         std::cout << "ERROR: Failed cudaFree" << std::endl;
261         return -1;
262       }
263
264     cuda_err1 = cudaFree(gpu_outimage);
265     if (cuda_err1 != cudaSuccess)
266       {
267         std::cout << "ERROR: Failed cudaFree" << std::endl;
268         return -1;
269       }
270
271     cvReleaseImage(&input_image);
272     cvReleaseImage(&out_image);
273
274
275     delete[] cpu_image;
276     delete[] buff;
277
278     return 0;
279   }
```

unroll_kernels.h

Zaglavna datoteka:

```
1
2  #include <cuda_runtime_api.h>
3  #include <cuda.h>
4  #include <math_constants.h>
5  #include <cutil_math.h>
6
7  extern "C" void cuda_transform(float* imagem, float* outimage, int width,
8                                 int height, int newwidth, int newheight,
9                                 dim3 blocks, dim3 block_size);
```

unroll_kernels.cu

Funkcije, ki se izvajajo na grafični kartici:

```
1  #include "internal/cutil_math_bugfixes.h"
2  #define WEIGHTS bspline_weights
3  #include "internal/bspline_kernel.cu"
4  #include <math_constants.h>
5
6  __device__ void filter (float4* img, float4* outimage, int width, int height,
7              double x, double y,int i)
8  {
9          float dx=remainder(x,1);
10                         float dy=remainder(y,1);
11                         int oldi = floor(y-0.5)*(width)+floor(x-0.5);
12
13                         int bpp = 4;
14                         int px = (int)x; // floor of x
15                         int py = (int)y-0.5; // floor of y
16                         const int stride = width;
17
18   // Calculate the weights for each pixel
19                         double fx = dx;
20                         double fy = dy;
21                         double fx1 = 1.0f - fx;
22                         double fy1 = 1.0f - fy;
23
24                         double w1 = 0.01+ fx1 * fy1 * 0.9f;
25                         double w2 = 0.01+ fx  * fy1 * 0.9f;
26                         double w3 = 0.01+ fx1 * fy * 0.9f;
27                         double w4 = 0.01+ fx  * fy * 0.9f;
28
29                         float r = (img[oldi].x * w1 +
30                                    img[oldi+1].x * w2 +
31                                    img[oldi+stride].x * w3 +
32                                    img[oldi+stride+1].x * w4);
33                         float g = (img[oldi].y * w1 +
34                                    img[oldi+1].y * w2 +
35                                    img[oldi+stride].y * w3 +
36                                    img[oldi+stride+1].y * w4);
37                         float b = (img[oldi].z * w1 +
38                                    img[oldi+1].z * w2 +
39                                    img[oldi+stride].z * w3 +
40                                    img[oldi+stride+1].z * w4);
41         outimage[i] = make_float4(r, g, b, 0);
42 }
```

```
43
44  __global__ void transform(float4* imagem, float4* outimage, int width,
45              int height, int newwidth, int newheight)
46  {
47          const int i = blockIdx.x * (blockDim.x * blockDim.y) +
48           blockDim.x * threadIdx.y + threadIdx.x;
49
50          if(i < newwidth * newheight)
51          {
52
53                  //int x = blockIdx.x*blockDim.x + threadIdx.x;
54                  //int y = blockIdx.y*blockDim.y + threadIdx.y;
55                  int x = remainder((double)i,(double)newwidth);
56                  int y = floorf((double)(i/newwidth));
57                  double x1 = -((double)(x)*2*CUDART_PI_F/(double)newwidth);
58                  double y1 = ((double)(y)/(double)newheight)-1;
59                  double oldx1 = y1*cosf(x1);
60                  double oldy1 = y1*sinf(x1);
61                  double oldx = ((oldx1 + 1)*(double)width/2);
62                  double oldy = ((oldy1 + 1)*(double)height/2);
63                  int oldi = floorf(oldy)*width+floorf(oldx);
64                  //float r = imagem[oldi].x;
65                  //float g = imagem[oldi].y;
66                  //float b = imagem[oldi].z;
67                  //outimage[i] = make_float4(r, g, b, 0);
68                  filter(imagem,outimage,width,height,oldx,oldy,i);
69          }
70  }
71
72  extern "C" void cuda_transform(float* imagem, float* outimage, int width,
73          int height,int newwidth, int newheight, dim3 blocks, dim3 block_size)
74  {
75           transform <<< blocks, block_size >>> ((float4*)imagem,
76            (float4*)outimage, width, height, newwidth, newheight);
77  }
```

Program unroll_center

Program poženemo kot: `unroll_center vhodna_datoteka.jpg izhodna_datoteka.jpg`. Pri tem je `vhodna_datoteka.jpg` vhodna datoteka tipa `jpg` in `izhodna_datoteka.jpg` izhodna datoteka tipa `jpg`.

Program določi center in premer kroga za program v Matlabu. Matlabov program, opisan v nadaljevanju ga kliče. Program mora biti v istem imeniku, kot Matlabovi programi.

unroll_center.cpp

```
1  #include <cv.h>
2  #include <highgui.h>
3  #include <math.h>
4
5  int notEntirely(int x, int y, int r, int width, int height) {
6          if ( (x-r<0) || (x+r>width) || (y-r<0) || (y+r>height)) return 1;
7          else return 0;
8  }
9
10 int main(int argc, char** argv)
11 {
12     IplImage* img = cvLoadImage(argv[1], 1);;
13     IplImage* gray = cvCreateImage(cvGetSize(img), 8, 1);
14     CvMemStorage* storage = cvCreateMemStorage(0);
15     cvCvtColor(img, gray, CV_BGR2GRAY);
16     cvSmooth(gray, gray, CV_GAUSSIAN, 9, 9);
17     CvSeq* circles = cvHoughCircles(gray, storage,
18         CV_HOUGH_GRADIENT, 2, gray->height/4, 200, 100);
19     int i;
20
21         //ugotovi najve?ji krog
22     int maxx=0;
23         int maxy=0;
24         int maxr=0;
25         for (i = 0; i < circles->total; i++)
26     {
27         float* p = (float*)cvGetSeqElem( circles, i );
28
29         int x=cvRound(p[0]);
30         int y=cvRound(p[1]);
```

```
31          int r=cvRound(p[2]);
32
33          if (r>maxr) {
34            maxx=x;
35            maxy=y;
36            maxr=r;
37          }
38
39          /*cvCircle( img, cvPoint(cvRound(p[0]),cvRound(p[1])),
40                3, CV_RGB(0,255,0), -1, 8, 0 );
41           cvCircle( img, cvPoint(cvRound(p[0]),cvRound(p[1])),
42                cvRound(p[2]), CV_RGB(255,0,0), 3, 8, 0 );*/
43
44          }
45          // Ce krog ni v celoti na sliki, vzemi kar geometrijsko
46          // sredisce slike ...
47          if (notEntirely(maxx,maxy,maxr, gray->width,gray->height)) {
48            maxx=cvRound(gray->width/2);
49            maxy=cvRound(gray->height/2);
50            maxr=( (gray->width > gray->height) ?
51              cvRound(gray->height/2) : cvRound(gray->width/2));
52          }
53
54          // Ce je krog manjsi od 80% najmanse zunanje dimenzije
55          //slike, vzemimo geometrisko sredisce slike
56
57          if ( maxr<0.8*((gray->width > gray->height) ?
58              cvRound(gray->height/2) : cvRound(gray->width/2))){
59            maxx=cvRound(gray->width/2);
60            maxy=cvRound(gray->height/2);
61            maxr=( (gray->width > gray->height) ?
62              cvRound(gray->height/2) : cvRound(gray->width/2));
63          }
64
65      //cvNamedWindow( "circles", 1 );
66      //cvShowImage( "circles", img );
67          //cvSaveImage("circle_output.jpg", img);
68      //cvWaitKey(0);
69      printf("%d %d %d\n",maxx,maxy,maxr);
70      return 0;
71  }
```

Program unroll: programska zanka

Program poženemo kot: `unroll vhodna_datoteka.jpg izhodna_datoteka.jpg`. Pri tem je `vhodna_datoteka.jpg` vhodna datoteka tipa `jpg` in `izhodna_datoteka.jpg` izhodna datoteka tipa `jpg`.

Program prebere vhodno vsesmerno sliko v vhodni datoteki, s Hughovo transformacijo poišče center in izpiše panoramsko sliko vhodne slike v izhodno datoteko.

unroll.cc

```
1  #include <opencv/cv.h>
2  #include <opencv/highgui.h>
3  #include <iomanip>
4  #include <iostream>
5  #include <math.h>
6  #include <sys/timeb.h>
7  #include <time.h>
8
9
10 #define DEBUG_TIME
11
12 void filter (float* img, float* outimage, int width, int height, double x,
13              double y,int i)
14 {
15   float dx=remainder(x,1);
16   float dy=remainder(y,1);
17   int oldi = floor(y-0.5)*(width)+floor(x-0.5);
18
19   int bpp = 4;
20   int px = (int)x; // floor of x
21   int py = (int)y-0.5; // floor of y
22   const int stride = width;
23
24   // Calculate the weights for each pixel
25   double fx = dx;
26   double fy = dy;
27   double fx1 = 1.0f - fx;
28   double fy1 = 1.0f - fy;
29
30   double w1 = 0.01+ fx1 * fy1 * 0.9f;
```

```
31    double w2 = 0.01+ fx   * fy1 * 0.9f;
32    double w3 = 0.01+ fx1 * fy * 0.9f;
33    double w4 = 0.01+ fx   * fy * 0.9f;
34
35    outimage[i*4] = (img[oldi*4+0] * w1 +
36                     img[(oldi+1)*4+0] * w2 +
37                     img[(oldi+stride)*4+0] * w3 +
38                     img[(oldi+stride+1)*4+0] * w4);
39    outimage[i*4+1] = (img[oldi*4+1] * w1 +
40                       img[(oldi+1)*4+1] * w2 +
41                       img[(oldi+stride)*4+1] * w3 +
42                       img[(oldi+stride+1)*4+1]* w4);
43    outimage[i*4+2] = (img[oldi*4+2] * w1 +
44                       img[(oldi+1)*4+2] * w2 +
45                       img[(oldi+stride)*4+2] * w3 +
46                       img[(oldi+stride+1)*4+2]* w4);
47  }
48
49  void transform(float* imagem, float* outimage, int width, int height,
50                 int newwidth, int newheight)
51  {
52    for (int y=0; y<newheight;y++){
53      for (int x=0; x<newwidth;x++){
54        //int x = remainder((double)i,(double)newwidth);
55        //int y = floorf((double)(i/newwidth));
56        int i = floorf(y)*newwidth+floorf(x);
57        double x1 = -((double)(x)*2*M_PI/(double)newwidth);
58        double y1 = ((double)(y)/(double)newheight)-1;
59        double oldx1 = y1*cosf(x1);
60        double oldy1 = y1*sinf(x1);
61        double oldx = ((oldx1 + 1)*(double)width/2);
62        double oldy = ((oldy1 + 1)*(double)height/2);
63        int oldi = fabs(floorf(oldy)*width+floorf(oldx));
64        //float r = imagem[oldi].x;
65        //float g = imagem[oldi].y;
66        //float b = imagem[oldi].z;
67        //outimage[i] = make_float4(r, g, b, 0);
68        filter(imagem,outimage,width,height,oldx,oldy,i);
69      }
70    }
71  }
72
73  int notEntirely(int x, int y, int r, int width, int height) {
74    if ( (x-r<0) || (x+r>width) || (y-r<0) || (y+r>height)) return 1;
75    else return 0;
76  }
77
78  int main(int argc, char** argv)
79  {
80    if (argc < 3)
81      {
82        std::cout << "Uporabi: " << argv[0] << " <input>" << " <output>" << std::endl;
83        exit(1);
84      }
85
86    IplImage* input_image = NULL;
87    input_image = cvLoadImage(argv[1], CV_LOAD_IMAGE_UNCHANGED);
88    if(!input_image)
89      {
90        std::cout << "ERROR: Failed to load input image" << std::endl;
91        return -1;
92      }
93
94    IplImage* gray = cvCreateImage(cvGetSize(input_image), 8, 1);
```

```
95    CvMemStorage* storage = cvCreateMemStorage(0);
96    cvCvtColor(input_image, gray, CV_BGR2GRAY);
97    cvSmooth(gray, gray, CV_GAUSSIAN, 9, 9);
98    CvSeq* circles = cvHoughCircles(gray, storage,
99                                    CV_HOUGH_GRADIENT, 2, gray->height/8, 200,
100                                   100, gray->height/8,
101                                   (gray->width> gray->height ?
102                                    gray->height : gray->width));
103
104   int width = input_image->width;
105   int height = input_image->height;
106
107   //ugotovi najve?ji krog
108   int maxx=0;    int maxy=0;      int maxr=0;      int i;
109
110   for (i = 0; i < circles->total; i++)
111     {
112       float* p = (float*)cvGetSeqElem( circles, i );
113
114       int x=cvRound(p[0]); int y=cvRound(p[1]); int r=cvRound(p[2]);
115
116       if (r>maxr) {
117         maxx=x; maxy=y; maxr=r;
118       }
119
120       cvCircle( input_image, cvPoint(cvRound(p[0]),cvRound(p[1])),
121                 3, CV_RGB(0,255,0), -1, 8, 0 );
122       cvCircle( input_image, cvPoint(cvRound(p[0]),cvRound(p[1])),
123                 cvRound(p[2]), CV_RGB(255,0,0), 3, 8, 0 );
124
125     }
126
127   // Ce krog ni v celoti na sliki, vzemi kar geometrijsko sredisce slike ...
128   if (notEntirely(maxx,maxy,maxr, gray->width,gray->height)) {
129     maxx=cvRound(gray->width/2);
130     maxy=cvRound(gray->height/2);
131     maxr=( (gray->width > gray->height) ? cvRound(gray->height/2) :
132            cvRound(gray->width/2));
133   }
134
135   // Ce je krog manjsi od 80% najmanjse zunanje dimenzije slike,
136   //vzemi geometrisko sredisce slike
137
138   if ( maxr<0.8*((gray->width > gray->height) ? cvRound(gray->height/2) :
139                  cvRound(gray->width/2))){
140     maxx=cvRound(gray->width/2);
141     maxy=cvRound(gray->height/2);
142     maxr=( (gray->width > gray->height) ? cvRound(gray->height/2) :
143            cvRound(gray->width/2));
144   }
145
146   int r=0;
147   int newheight=r;
148
149   if (height > 2 * maxy ) newheight = maxy;
150   else newheight = height-maxy;
151
152   r=maxr-(maxr%8);
153   newheight=r+(16-r%16);
154   int newwidth= M_PI*r+(16-(int)(M_PI*r)%16);;
155
156   printf("%d %d %d\n",newheight,newwidth,r);
157
158   cvSetImageROI( input_image,  cvRect( maxx-r, maxy-r, 2*r, 2*r ) );
```

```
159     IplImage* img = cvCreateImage(cvSize(2*r, 2*r), input_image->depth,
160                                   input_image->nChannels );
161     cvCopy( input_image, img,NULL );
162     cvResetImageROI( input_image );
163
164     cvNamedWindow( "circles", 1 );
165     cvShowImage( "circles", img );
166     cvSaveImage("circle_output.jpg", img);
167     cvWaitKey(0);
168
169
170     int bpp = input_image->nChannels;
171
172   #ifdef DEBUG
173     std::cout << ">> Width:" << width << std::endl <<
174       ">> Height:" << height << std::endl <<
175       ">> Bpp:" << bpp << std::endl;
176
177     std::cout << std::endl << ">>> Debugging Original data:" << std::endl;
178     for (int i=0; i < width*height*bpp; i+=bpp)
179       {
180         if (!(i % (width*bpp)))
181           std::cout << std::endl;
182
183         std::cout << std::dec << "R:"<< (int) input_image->imageData[i] <<
184           " G:" << (int) input_image->imageData[i+1] <<
185           " B:" << (int) input_image->imageData[i+2] << "     ";
186       }
187     std::cout << std::endl << std::endl;
188   #endif
189
190     float* cpu_image = new float[width * height * 4];
191     if (!cpu_image)
192       {
193         std::cout << "ERROR: Failed to allocate memory" << std::endl;
194         return -1;
195       }
196
197     for (int i = 0; i < 2*r * 2*r; i++)
198       {
199         cpu_image[i * 4 + 0] = (unsigned char)img->imageData[i * bpp + 0] / 255.f;
200         cpu_image[i * 4 + 1] = (unsigned char)img->imageData[i * bpp + 1] / 255.f;
201         cpu_image[i * 4 + 2] = (unsigned char)img->imageData[i * bpp + 2] / 255.f;
202         cpu_image[i * 4 + 3] = (unsigned char)img->imageData[i * bpp + 2] / 255.f;
203       }
204
205     float* cpu_outimage = new float[newwidth * newheight * 4];
206     if (!cpu_outimage)
207       {
208         std::cout << "ERROR: Failed to allocate memory" << std::endl;
209         return -1;
210       }
211
212   #ifdef DEBUG_TIME
213     //Start clock
214     struct timeb start_time_st;
215     ftime(&start_time_st);
216   #endif
217
218
219     transform(cpu_image, cpu_outimage, 2*r, 2*r, newwidth, newheight);
220
221
222   #ifdef DEBUG_TIME
```

```
223     // Stop clock
224     struct timeb stop_time_st;
225     ftime(&stop_time_st);
226     double elapsed = ((double) stop_time_st.time +
227                       ((double) stop_time_st.millitm * 0.001)) -
228       ((double) start_time_st.time + ((double) start_time_st.millitm * 0.001));
229
230     std::cout << "* Time elapsed: " << std::setprecision(5) << elapsed << " sec" << std::endl;
231   #endif
232
233     char* buff = new char[newwidth * newheight * bpp];
234     if (!buff)
235       {
236         std::cout << "ERROR: Failed to allocate memory" << std::endl;
237         return -1;
238       }
239
240
241     for (int i = 0; i < (newwidth * newheight); i++)
242       {
243         buff[i * bpp + 0] = (char)floor(cpu_outimage[i * 4 + 0] * 255.f);
244         buff[i * bpp + 1] = (char)floor(cpu_outimage[i * 4 + 1] * 255.f);
245         buff[i * bpp + 2] = (char)floor(cpu_outimage[i * 4 + 2] * 255.f);
246       }
247
248   #ifdef DEBUG
249     std::cout << std::endl << ">>> Debugging Output data:" << std::endl;
250     for (int i=0; i < width*height*bpp; i+=bpp)
251       {
252         if (!(i % (width*bpp)))
253           std::cout << std::endl;
254
255         std::cout << std::dec << "R:"<< (int) buff[i] <<
256           " G:" << (int) buff[i+1] <<
257           " B:" << (int) buff[i+2] << " ";
258       }
259     std::cout << std::endl << std::endl;
260   #endif
261
262     IplImage* out_image = cvCreateImage(cvSize(newwidth, newheight),
263                                         input_image->depth, bpp);
264     if (!out_image)
265       {
266         std::cout << "ERROR: Failed cvCreateImage" << std::endl;
267         return -1;
268       }
269
270     out_image->imageData = buff;
271
272     if (!cvSaveImage(argv[2], out_image))
273       {
274         std::cout << "ERROR: Failed cvSaveImage" << std::endl;
275       }
276
277     cvReleaseImage(&input_image);
278     cvReleaseImage(&out_image);
279
280
281     delete[] cpu_image;
282     delete[] buff;
283
284     return 0;
285   }
```

Drugi program je spisan v programskem okolju Matlab, opisan v poglavju 4:
Priloga k 4.1:

Program unroll

Program kličemo iz Matlabove lupine kot `unroll('slika.jpg', parametri);`. *Parametri za posamezne transformacije so opisani v poglavju 4.*

unroll.m

```
function [ imgWrite, TOC ] = unroll( imeSlike, tip, a, b, c )

%% Strojno dolocanje centra
sourcev = imread(imeSlike);
command='./unroll_center ';
command=horzcat(command,imeSlike);
[status, center]=system(command);
[x, center] = strtok(center);
x=str2double(x);
[y, center] = strtok(center);
y=str2double(y);
[r, center] = strtok(center);
r=str2double(r);

% nova optimalna velikost slike, maksimalna resolucija
newwidth=double(int64(pi*r));
tic;
if (tip=='b')
    panorama=unroll_basic(imeSlike, x, y);
else
    if (tip=='h')
        if (not(exist('a','var')))
            a=39.292;
        end
        if (not(exist('b','var')))
            b=19.646;
        end
        if (not(exist('c','var')))
            c=2.3;
        end
        panorama=unroll_hyperbolic(imeSlike, x, y, a,b,c);
    else
        if (tip=='k')
            panorama=unroll_krogla(imeSlike, x, y, a,b,c);
        else
            if (tip=='p')
                panorama=unroll_parabolic(imeSlike, x, y, a);
            else
                 panorama=unroll_basic(imeSlike, x, y);
```

```
40              end
41          end
42      end
43  end
44  TOC=toc;
45  if ((tip~='f'))
46          imshow(panorama);
47          [~,y]=ginput(2);
48
49          if ( double(int64(y(1))) > double(int64(y(2))) )
50              panorama=imcrop(panorama,[0 double(int64(y(2))) newwidth double(int64(y(1)-y(2)))]);
51          else
52              panorama=imcrop(panorama,[0 double(int64(y(1))) newwidth double(int64(y(2)-y(1)))]);
53          end
54  end
55  %imshow(panorama);
56  imgWrite=panorama;
57
58  end
```

unroll_basic.m

Funkcije za osnovno transformacijo brez upoštevanja zrcala:

```
1   function [ imgWrite ] = unroll_basic( imeSlike, x, y )
2   %UNROLL Vrne razvito panoramsko sliko omnidirekcijske fotografije.
3   %   imeSlike - ime slike
4   %   x - center offset za x
5   %   y - center offset za y
6   sourcev = imread(imeSlike);
7   % Velikost slike
8   [height,width]=size(sourcev);
9   % Odkomentiraj za barvne slike
10  width = width/3;
11  if (height > double(int64(2.*y)) )
12      height = double(int64(2.*y));
13  else
14      height = double(int64(2*(height-y)));
15  end
16  %Premer
17  if (width<height)
18      r=double(int64(width/2));
19  else
20      r=double(int64(height/2));
21  end
22  newwidth=double(int64(pi*r));
23  newheight=double(int64(r));
24  source=sourcev(y-r+1:y+r-1, x-r+1:x+r-1, :);
25  % Funkcije preslikave
26  u=@(x) x(:,2).*cos(x(:,1));
27  v=@(x) x(:,2).*sin(x(:,1));
28  ipanorama=@(x) [u(x), v(x)];
29  inverse=@(x, t) ipanorama(x);
30  tform2 = maketform('custom', 2, 2, [], inverse, []);
31  imgWrite = imtransform(source, tform2, 'bilinear', 'UData', [-1 1], ...
32                         'VData', [-1 1], 'XData', [2*pi 0], 'YData', [1 0], ...
33                         'Size', [newheight newwidth]);
34  end
```

unroll_hiperbolic.m

Funkcije za hiperbolično zrcalo:

```
function [ imgWrite ] = unroll_hyperbolic(imeSlike, x, y, a, b, focus)
%UNROLL_HYPERBOLIC Vrne razvito panoramsko sliko omnidirekcijske fotografije.
%   Detailed explanation goes here
sourcev = imread(imeSlike);

% Velikost slike
[height,width]=size(sourcev);
% Odkomentiraj za barvne slike
width = width/3;
if (height > double(int64(2.*y)) )
    height = double(int64(2.*y));
else
    height = double(int64(2*(height-y)));
end

%Premer
if (width<height)
    r=double(int64(width/2));
else
    r=double(int64(height/2));
end
% nova optimalna velikost slike, maksimalna resolucija
newwidth=double(int64(pi*r));
newheight=double(int64(r));
source=sourcev(y-r+1:y+r-1, x-r+1:x+r-1, :);
e=sqrt((a.^2)+(b.^2));
% Funkcije preslikave
qt=@(x) focus.*(a.^2)./(((a.^2)-2.*(e.^2)).*tan(x) + ...
    2.*b.*e.*sqrt(1+tan(x).^2));

u=@(x) qt(x(:,2)).*cos(x(:,1));
v=@(x) qt(x(:,2)).*sin(x(:,1));
ipanorama=@(x) [u(x), v(x)];
inverse=@(x, t) ipanorama(x);

% Iskanje nicle funkcije qt1 z fsolve in izracun slike;
qt1=@(x) qt(x)-1;
x0=-pi/2;
tform2 = maketform('custom', 2, 2, [], inverse, []);
imgWrite = imtransform(source, tform2, 'bilinear', 'UData', [-1 1], ...
                       'VData', [-1 1], 'XData', [2*pi 0], ...
                       'YData', [fsolve(qt1, x0) -pi/2], ...
                       'Size', [newheight newwidth]);
end
```

unroll_krogla.m

Funkcije za krogelno zrcalo:

```
function [ imgWrite ] = unroll_krogla (imeSlike, x, y, a,b,c)
sourcev = imread(imeSlike);
% Velikost slike
[height,width]=size(sourcev);
% Odkomentiraj za barvne slike
width = width/3;
if (height > double(int64(2.*y)) )
```

```
8       height = double(int64(2.*y));
9   else
10      height = double(int64(2*(height-y)));
11  end
12  %Premer
13  if (width<height)
14      r=double(int64(width./2));
15  else
16      r=double(int64(height./2));
17  end
18  % nova optimalna velikost slike, maksimalna resolucija
19  newwidth=double(int64(pi*r));
20  newheight=double(int64(r));
21  source=sourcev(y-r+1:y+r-1, x-r+1:x+r-1, :);
22  % Konstante krogle.
23  g=a;
24  d=b;
25  focus=c;
26  f=focus;
27  % Funkcije preslikave
28  iks=@(x) ((d.*f./x)./(1+(f./x).^2))-(sqrt((d^2.*(f./x).^2)-(d.^2-g.^2).* ...
29              (1+(f./x).^2))./(1+(f./x).^2));
30  qt=@(x) focus./((d+sqrt(g.^2-iks(x).^2))./iks(x));
31  u=@(x) qt(x(:,2)).*cos(x(:,1));
32  v=@(x) qt(x(:,2)).*sin(x(:,1));
33  ipanorama=@(x) [u(x), v(x)];
34  inverse=@(x, t) ipanorama(x);
35  % Iskanje nicle funkcije qt1 z fsolve in izracun slike;
36  qt1=@(x) (qt(x))-1;
37  x0=1;
38  tform2 = maketform('custom', 2, 2, [], inverse, []);
39  imgWrite = imtransform(source, tform2, 'bicubic', 'UData', [-1 1], ...
40                         'VData', [-1 1], 'XData', [pi -pi], ...
41                         'YData', [fsolve(qt1, x0) 0], ...
42                         'Size', [newheight newwidth]);
```

unroll_parabolic.m

Funkcije za paraboličnо zrcalo:

```
1   function [ imgWrite ] = unroll_parabolic(imeSlike, x, y, c)
2   %UNROLL_PARABOLIC Vrne razvito panoramsko sliko omnidirekcijske fotografije.
3   %   Detailed explanation goes here
4   sourcev = imread(imeSlike);
5   % Velikost slike
6   [height,width]=size(sourcev);
7   % Odkomentiraj za barvne slike
8   width = width/3;
9   if (height > double(int64(2.*y)) )
10      height = double(int64(2.*y));
11  else
12      height = double(int64(2*(height-y)));
13  end
14  %Premer
15  if (width<height)
16      r=double(int64(width/2));
17  else
18      r=double(int64(height/2));
19  end
20  % nova optimalna velikost slike, maksimalna resolucija
21  newwidth=double(int64(pi*r));
```

```
22  newheight=double(int64(r));
23  source=sourcev(y-r+1:y+r-1, x-r+1:x+r-1, :);
24  qt=@(x) (sqrt(1+tan(x).^2)-tan(x));
25  % Funkcije preslikave
26  u=@(x) 2*c*cos(x(:,1))./qt(x(:,2));
27  v=@(x) 2*c*sin(x(:,1))./qt(x(:,2));
28  ipanorama=@(x) [u(x), v(x)];
29  inverse=@(x, t) ipanorama(x);
30  % Iskanje nicle funkcije qt1 z fsolve in izracun slike;
31  qt1=@(x) (2*c/qt(x))-1;
32  x0=-pi/2;
33  tform2 = maketform('custom', 2, 2, [], inverse, []);
34  imgWrite = imtransform(source, tform2, 'bilinear', 'UData', [-1 1], ...
35                         'VData', [-1 1], 'XData', [2*pi 0], ...
36                         'YData', [fsolve(qt1, x0+0.05) -pi/2], ...
37                         'Size', [newheight newwidth]);
38  end
```