УНИВЕРЗИТЕТ
У НОВОМ САДУ

ФАКУЛТЕТ
ТЕХНИЧКИХ НАУКА

Трг Доситеја Обрадовића 6, 21000 Нови Сад, Република Србија
Деканат: 021 6350-413; 021 450-810; Централа: 021 485 2000
Рачуноводство: 021 458-220; Студентска служба: 021 6350-763
Телефакс: 021 458-133; e-mail: ftndean@uns.ac.rs

ИНТЕГРИСАНИ
СИСТЕМ
МЕНАЏМЕНТА
СЕРТИФИКОВАН ОД:

Напредна роботика

Стереовизија и рачунање дубине

Мастер студије
Летњи семестар 2018.

доц. др Милутин Николић
milutinn@uns.ac.rs

Sadržaj

1. Увод

Након разматрања о моделима камере, сада смо у ситуацији да се позабавимо стереовизијом. Сви смо свесни способности наших очију да нам дају тродимензионалну слику. Поставља се питање до ког степена можемо да емулирамо ову способност на рачунару. Рачунар испуњава овај задатак проналазећи парове тачака на левој и десној слици који долазе са исте тачке у простору. Помоћу тога и познатог положаја камера у стању смо да израчунамо 3Д локацију тачака. Иако је упаривање тачака са слика веома рачунски захтевно, можемо искористити познавање геометрије система како бисмо сузили простор у коме се траже парови што је више могуће. У пракси, стереовизија (у случају када се користе две камере) има четири корака:

(1) Уклањање изобличења које уводи сочиво, о чему је било речи. Излаз из овог корака су две слике без изобличења.
(2) Подешавање углова и растојања између камера ради лакшег упаривања тачака. Овај процес се назива ректификација и излаз су две слике које су поравнате по редовима. То значи да су равни слика компланарне и да су редови тачно поравнати.
(3) Упаривање тачака са леве слике са тачкама на десној слици. Овај процес се назива кореспонденција. Излаз из овог корака је мапа *диспаритета*, где диспаритет представља разлике између x-координата исте тачке виђене са леве и десне камере: $x^l - x^r$. Обратите пажњу на чињеницу да разлика по y-координати не постоји пошто су слике поравнате по редовима.
(4) Ако знамо геометријски распоред камера, можемо из мапе диспаритета одредити растојања коришћењем триангулације. Овај корак се назива репројекција и излаз је дубинска мапа.

Почећемо од последњег корака, како би се видео процес одређивања позиције тачке у простору на основу њене слике посматране са две различите локације, што ће нас мотивисати да се позабавимо осталим корацима.

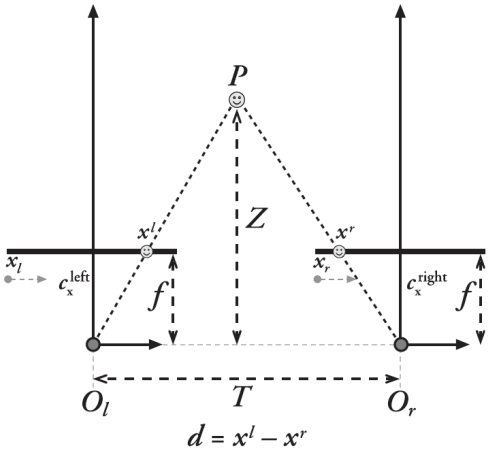
2. Триангулација

Претпоставимо да имамо две слике без изобличења, поравнате и постављене на познатом растојању као што је приказано на слици 1: две камере чије су равни слике потпуно компланарне, са потпуно паралелним оптичким осама које су на познатом растојању и једнаких жижних даљина $f_l = f_r$. Такође, претпоставимо да су основне тачке c_x^{left} и c_x^{right} једнаке. Обратите пажњу на разлику између основне тачке и центра слике. Основна тачка је место где оптичка оса сече раван слике. Као што је речено раније, раван слике готово никад није тачно поравната са сочивом тако да се центар слике готово никад не поклапа са основном тачком. Такође, претпоставимо да су слике поравнате по редовима тако да је сваки ред пиксела једне камере поравнат са редом пиксела на другој камери. Називаћемо распоред камера са свим овим особинама *"фронтално паралелна конфигурација"*. Такође ћемо претпоставити да можемо наћи тачку P коју виде обе камере и чија је позиција на левој и десној слици p_l и p_r и чије су хоризонталне координате x^l и x^r.

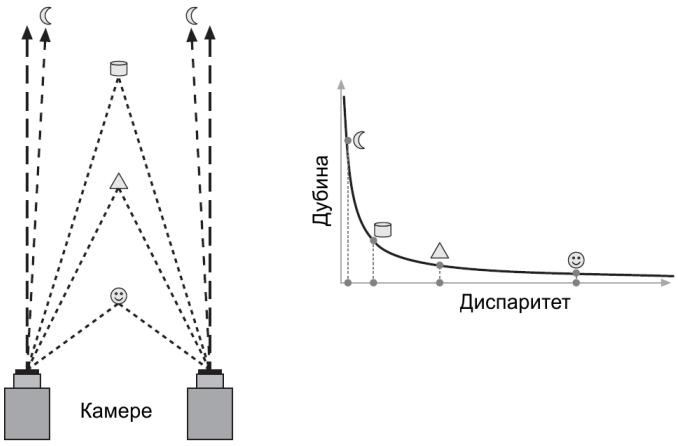
У овом упрошћеном случају, уколико узмемо да су x^l и x^r хоризонталне позиције

на левој и десној слици можемо показати да је дубина инверзно пропорционална диспаритету између ова два погледа, где је диспаритет дат као $d = x^l - x^r$. Овај пример је дат на слици 1, са које коришћењем сличности троуглова можемо одредити дубину Z:

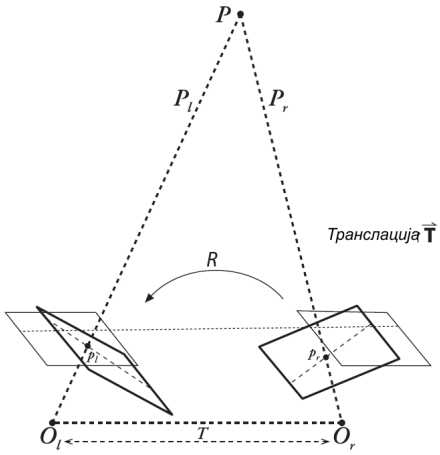
$$\frac{T - \left(x^l - x^r\right)}{Z - f} = \frac{T}{Z} \Rightarrow Z = \frac{fT}{x^l - x^r} \tag{1}$$



Slika 1. Идеалан распоред камера за стеровизију



Slika 2. Зависност дубине од диспаритета



Slika 3. Реалан распоред камера

Пошто је дубина инверзно пропорционална диспаритету, очигледно постоји нелинеарна релација. Када је диспаритет приближно нула, мале разлике у диспаритету праве велику разлику у дубини. Када је диспаритет велики, мале промене у диспаритету не мењају дубину значајно. Као последица тога, стереовизија има високу дубинску резолуцију само за објекте који су релативно близу камерама, као што је приказано на слици 2.

Са оваквим распоредом камера, веома је лако одредити дистанцу. Сада ћемо уложити одређен труд да бисмо разумели како можемо да мапирамо стварни распоред камера на геометрију која представља идеалан распоред. У стварности, камере готово никад нису у фронталној паралелној конфигурацији као што је призаказано на слици 3. Због тога ћемо математички наћи пројекције које ће ректификовати леву и десну слику у фронталну паралелну кофигурацију. Приликом пројектовања стерео система, најбоље би било да су камере приближно фронтално паралелене и што више хоризонтално поравнате. Овакав распоред ће олакшати математичко поравнавање. Уколико се камере не поравнају ни приближно онда ће резултујуће математичко поравнање произвести значајна изобличења и тиме умањити преклапање слика које је потребно да би се вршило упаривање тачака и одређивање дубине. Уз то, за добијање квалитетних резултата камере морају бити синхронизоване. Уколико камере не узимају слику у истом тренутку, постојаће проблем ако се нешто помера у сцени. Ако нису синхронизоване, ограничени смо на коришћење стационарних камера који снимају статичне сцене.

Слика 3 приказује стварну ситуацију и стварну позицију камера као и математичко поравнање које желимо да добијемо. Да бисмо то могли да урадимо, морамо научити додатне ствари о геометрији камера које посматрају сцену. Када се то научи и дефинише одређена терминологија, вратићемо се на проблем поравнавања слика.

Вежба 1 Триангулација

Одредити где се у простору налази тачка која се пресликава на пиксел $x^l = 25$, $y^l = 14$ на левој слици и пиксел $x^r = 15$, $y^r = 14$ на десној слици. Претпоставити да су камере постављене у фронталној паралелној конфигурацији и да су параметри обе камере исти и да су на међусобном растојању од 1.25. Димензије слике су 32 пута 24 пиксела и $f_x = 8$, $f_y = 6$, $c_x = 17$ и $c_y = 11$.

Решење вежбе 1

Унутрашња матрица камере за дати случај је:

$$\mathbf{M} = \begin{bmatrix} 8 & 0 & 17 \\ 0 & 6 & 11 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

На основу једначине 1 можемо одредити растојање предмета од камере:

$$Z = \frac{f_x T}{x^l - x^r} = \frac{8 \times 1.25}{25 - 15} = \frac{10}{10} = 1. \tag{3}$$

На тај начин смо одредили колико је предмет удаљен од камере. Преостаје само да одредимо још и X и Y координате, које ће бити рачунате у односу на леву камеру. На основу модела камере важи:

$$w \begin{bmatrix} x^l \\ y^l \\ 1 \end{bmatrix} = \mathbf{MQ} \tag{4}$$

Одакле следи:

$$\mathbf{Q} = w\mathbf{M}^{-1} \begin{bmatrix} x^l \\ y^l \\ 1 \end{bmatrix} = w \begin{bmatrix} 8 & 0 & 17 \\ 0 & 6 & 11 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 25 \\ 14 \\ 1 \end{bmatrix} = \frac{w}{48} \begin{bmatrix} 6 & 0 & -102 \\ 0 & 8 & -88 \\ 0 & 0 & 48 \end{bmatrix} \begin{bmatrix} 25 \\ 14 \\ 1 \end{bmatrix} = w \begin{bmatrix} 1 \\ 0.5 \\ 1 \end{bmatrix} \tag{5}$$

Пошто знамо да је $Z = 1 = w * 1$ добија се да је $w = 1$, одакле следи:

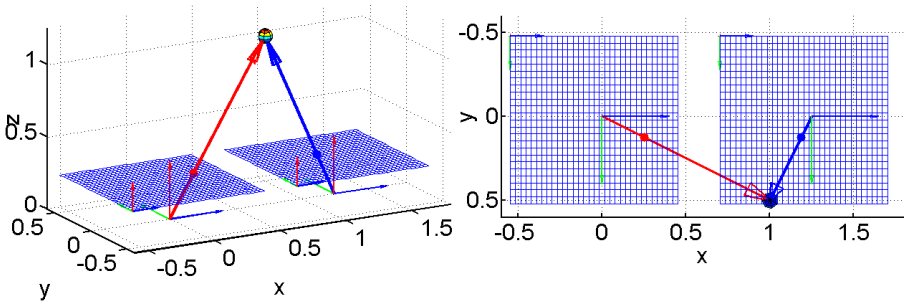
$$\mathbf{Q} = \begin{bmatrix} 1 \\ 0.5 \\ 1 \end{bmatrix} \tag{6}$$

Графички приказ решења је дат на слици 4.

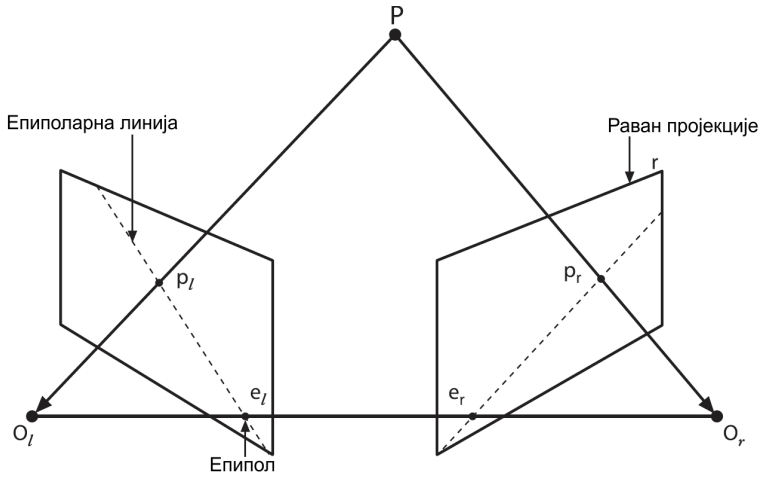
3. Епиполарна геометрија

Геометрију система са две камере називамо епиполарна геометрија. У суштини, комбинују се две камере моделоване као камера са отвором и неке тачке од интереса које називамо епиполови (слика 5). Пре него што дамо објашњење шта су епиполови и чему служе, дефинисаћемо их и увести нову терминологију. Након тога, имаћемо потпуно разумевање геометријског распореда и такође ћемо видети да можемо значајно сузити претрагу могућих локација парова тачака на две камере. Ова особина је врло битна приликом имплементације стереовизије.

За сваку од камера постоји засебан центар пројекције O_l и O_r и одговарајућа раван пројекције Π_l и Π_r. Тачка P у реалном свету се пројектује на равни пројекције у тачке p_l и p_r. Епипол e_l (или e_r) на равни Π_l (или Π_r) је дефинисан као слика центра



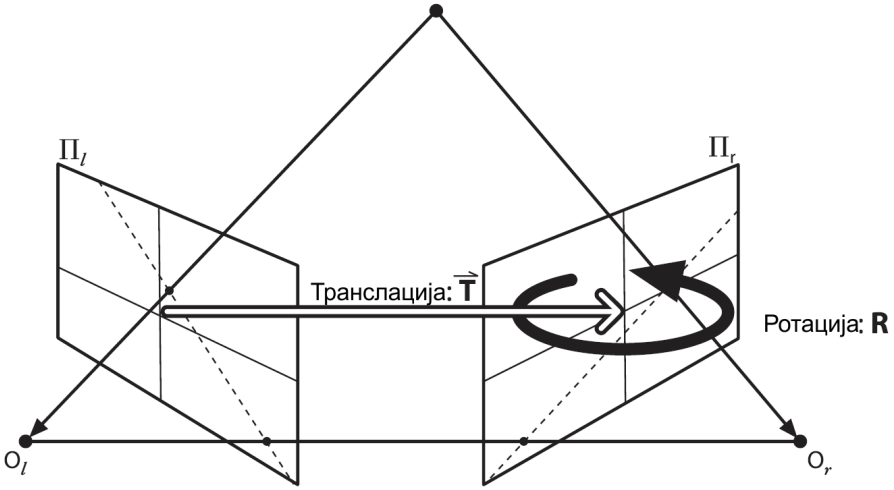
Slika 4. Резултат вежбе 1



Slika 5. Основни делови епиполарне геометрије

пројекције друге камере O_r (или O_l). Раван у простору коју формирају тачка P и два епипола e_l и e_r назива се епиполарна раван и праве $p_l e_l$ и $p_r e_r$ се називају епиполарне линије. Обратите пажњу на чињеницу да уколико су равни пројекција паралелне са правом $O_l O_r$ епиполови су у бесконачности.

Да бисмо разумели корист епиполова, прво се сетимо да када посматрамо тачку у простору пројектовану на леву или десну слику, та тачка се може налазити било где дуж целе полуправе која креће из O_r и пролази кроз p_r (или креће из O_l пролази кроз p_l), јер посматрајући само са једном камером не можемо одредити удаљеност једне тачке коју посматрамо. На пример, посматрајмо тачку P и како је види десна камера. Пошто камера види само њену пројекцију p_r, стварна тачка P може бити било где на полуправој дефинисаној са O_r и p_r. Ова права очигледно садржи тачку P али



Slika 6. Приказ релативне трансформације између две камере

садржи и бесконачно тачака поред ње. Пројекција те праве на раван леве слике Π_l је у ствари епиполарна линија дефинисана тачкама p_l и e_l. Другачије речено, слика свих могућих позиција посматране тачке са једне слике је линија која пролази кроз одговарајућу тачку и епипол на другој слици.

Сада ће се сумирати неке чињенице о епиполарној геометрији стерео камера:

- Свака тачка у 3Д простору коју виде обе камере је на епиполарној равни која сече равни слике у епиполарним линијама
- За задату тачку на једној слици, њена слика на другој слици се мора налазити дуж одговарајуће епиполарне линије. Ово је познато као епиполарно ограничење.
- Епиполарно ограничење нам говори да упаривање пиксела са леве и десне слике, уместо могуће дводимензионалне претраге по целој слици, постаје једнодимензионална претрага дуж одговарајуће епиполарне линије. Осим што значајно смањује потребно рачунање, такође смањује могућност погрешног упаривања тачака.

4. Есенцијална и фундаментална матрица

Есенцијалну матрицу ћемо означавати са **E** док ће фундаментална матрица бити означавана са **F**. Матрица **E** садржи информације о транслацији и ротацији између две камере у простору (слика 6), док **F** садржи исте информцације као и **E** уз додате информације о унутрашњим параметрима камере. Пошто је у **F** уграђена информација о унутрашњим параметрима камере, она повезује позиције тачака у пикселима.

Прво, рашчистимо разлику између **E** и **F**. Есенцијална матрице је везана чисто за геометрију и у њу не улазе никакве информације о слици. Она даје релацију

између локације у спољашњим координатама тачке P задате у односу на леву камеру и задате у односу на десну камеру (даје релацију $\mathbf{p}_l$ у односу на $\mathbf{p}_r$). Фундаментална матрица задаје релацију између позиције тачака на слици једне камере у координатама слике (пикселима) и позиције тачака на слици друге камере у пикселима ($\mathbf{q}_l$ и $\mathbf{q}_r$).

4.1. *Рачунање есенцијалне матрице*

За задату тачку P, желели бисмо да изведемо релацију која повезује локације које видимо из левог и десног центра пројекције $\mathbf{p}_l$ и $\mathbf{p}_r$ тачке P. Из ове везе ћемо добити дефиницију есенцијалне матрице. Ово се може добити коришћењем епиполарне геометрије.

Изаберимо да је координатни почетак целог система O_l, односно координатни систем везан у центру леве камере. Према тој конвенцији, локација посматране тачке је $\mathbf{P}_l$ и координатни почетак друге камере је лоциран у $\mathbf{T}$. Координата тачке P у односу на десну камеру је $\mathbf{P}_r$ и може се израчунати као $\mathbf{P}_r = \mathbf{R}\left(\mathbf{P}_l - \mathbf{T}\right)$[a] Први корак је увођење епиполарне равни. Као што је рађено из линеарне алгебре, један начин за задавање једначине равни је задавања вектора нормале на раван $\mathbf{n}$ и координате једне тачке $\mathbf{a}$ коју садржи та раван. Свака тачка $\mathbf{x}$ на равни испуњава следећи услов:

$$(\mathbf{x} - \mathbf{a}) \cdot \mathbf{n} = 0, \tag{7}$$

што речима значида је свака дуж у равни нормална на задати вектор нормале. Приметимо да вектори $\mathbf{P}_l$ и $\mathbf{T}$ припадају епиполарној равни чија нормала је $\mathbf{T} \times \mathbf{P_l}$ и она садржи тачку $\mathbf{T}$. То значи да за сваку тачку $\mathbf{P}_l$ важи:

$$\left(\mathbf{P}_l - \mathbf{T}\right)^T \left(\mathbf{T} \times \mathbf{P}_l\right) = 0 \tag{8}$$

Подсетимо се да нам је циљ да нађемо везу између $\mathbf{p}_l$ и $\mathbf{p}_r$ тако што ћемо прво наћи везу између $\mathbf{P}_l$ и $\mathbf{P}_r$. Ако везу $\mathbf{P}_r = \mathbf{R}\left(\mathbf{P}_l - \mathbf{T}\right)$ напишемо у облику $\mathbf{R}^T\mathbf{P}_r = \left(\mathbf{P}_l - \mathbf{T}\right)$[b] добијамо:

$$\left(\mathbf{R}^T\mathbf{P}_r\right)^T \left(\mathbf{T} \times \mathbf{P}_l\right) = 0 \tag{9}$$

Векторски производ два вектора је могуће представити као производ кососиметричне матрице и једног од вектора. Тако добијамо:

$$\mathbf{T} \times \mathbf{P}_l = \mathbf{S}_T\mathbf{P}_l;\ \mathbf{S}_T = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix} \tag{10}$$

Одакле добијамо коначну зависност из које следи дефиниција епиполарне матрице:

$$\mathbf{P}_r^T\mathbf{R}\mathbf{S}_T\mathbf{P}_l = 0 \tag{11}$$

где се есенцијална матрица дефинише као $\mathbf{E} = \mathbf{R}\mathbf{S}_T$. Одакле се добија:

$$\mathbf{P}_r^T\mathbf{E}\mathbf{P}_l = 0 \tag{12}$$

[a]Обратите пажњу да је нотација мало другачија од оног што је рађено из индустријске роботике. Према ономе што је тамо рађено било би $\mathbf{P}_r = \mathbf{R}_r^l\mathbf{P}_l + \mathbf{d}_r^l$ и $\mathbf{P}_l = \mathbf{R}_l^r\mathbf{P}_r + \mathbf{d}_l^r$. Тако да треба посматрати горње $\mathbf{R}$ као $\mathbf{R}_r^l$ и $\mathbf{T}$ као $\mathbf{d}_l^r$

[b]Подсетимо се да је матрица ротације ортогонална па је $\mathbf{R}^{-1} = \mathbf{R}^T$

Наравно, оно што смо заиста желели је веза између тачака које видимо на пројективној равни. Да бисмо то добили једноставно можемо заменити $\mathbf{p}_l = f_l\mathbf{P}_l/Z_l$ и $\mathbf{p}_r = f_r\mathbf{P}_r/Z_r$ и онда поделимо целу једначину са Z_lZ_r/f_lf_r да бисмо добили коначан резултат:

$$\mathbf{p}_r^T\mathbf{E}\mathbf{p}_l = 0 \tag{13}$$

Делује да задавањем једне тачке можемо у потпуности да добијемо координате друге тачке. Међутим ово је скаларна једначина тако да се може добити само права на којој лежи пројекција тачке на другој равни пројекције. Такође, ранг матрице $\mathbf{E}$ је 2, тако да она нема инверзну и за њу нам су нам потребна 3 параметра за ротацију и два за правац транслације (интензитет није битан) уз још два ограничења. Прво ограничење је да је детерминанта 0, а друго је да су две сингуларне вредности матрице једнаке, јер је $\mathbf{S}_T$ кососиметрична и $\mathbf{R}$ ортогонална. Подсетимо се да $\mathbf{E}$ не даје везу између координата у пикселима, већ везу између тачака на равни пројекције. Треба поново нагласити да је претходна једначина скаларна, што значи да се из ње може одредити само један параметар а не и цела матрица $\mathbf{E}$.

Вежба 2 Есенцијална матрица

Дата је тачка на десној пројективној равни са координатама $\mathbf{p}_r = [0.2\ 0.3\ 0.25]^T$. Одредити на којој правој на левој пројективој равни се мора налазити тачка чија се пројекција на десну пројективну раван поклапа са $\mathbf{p}_r$. Растојање између камера је $d = 1$. Камере одступају од паралелне фронталне конфигурације, јер су обе заротиране ка унутра за угао од 30°. Претпоставити да је $f_l = 0.25$.

Решење вежбе 2

Прво ћемо да одредимо матрицу ротације $\mathbf{R}$ и вектор транслације $\mathbf{T}$. Са индексима $_{l,r}$ и $_0$ ћемо означавати да је координата задата у координатном систему леве камере, десне камере и координатном систему који је би се односио на паралелну фронталну конфигурацију пре него што је било ротација камере. Онда важи:

$$\mathbf{T}_0 = \begin{bmatrix}1\\0\\0\end{bmatrix};\ \mathbf{R}_0^l = Rot_{y,30^\circ} = \begin{bmatrix}\frac{\sqrt{3}}{2} & 0 & \frac{1}{2}\\ 0 & 1 & 0\\ -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2}\end{bmatrix};\ \mathbf{R}_0^r = Rot_{y,-30^\circ} = \begin{bmatrix}\frac{\sqrt{3}}{2} & 0 & -\frac{1}{2}\\ 0 & 1 & 0\\ \frac{1}{2} & 0 & \frac{\sqrt{3}}{2}\end{bmatrix} \tag{14}$$

Онда је вектор транслације:

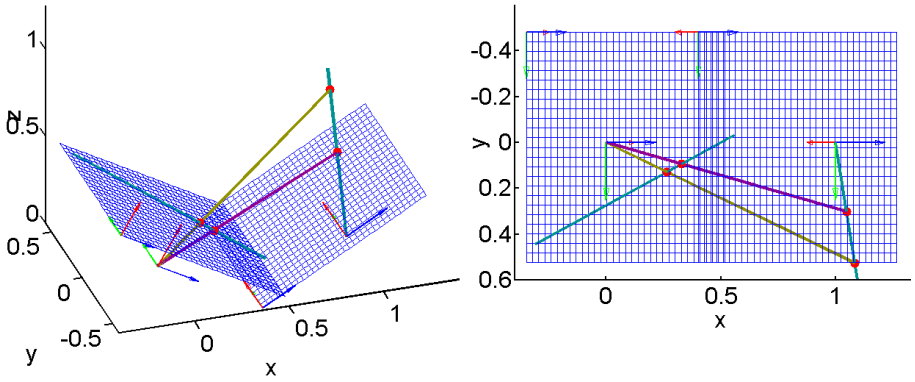
$$\mathbf{T} = \mathbf{R}_l^0\mathbf{T}_0 = \left(\mathbf{R}_0^l\right)^T\mathbf{T}_0 = \begin{bmatrix}\frac{\sqrt{3}}{2} & 0 & -\frac{1}{2}\\ 0 & 1 & 0\\ \frac{1}{2} & 0 & \frac{\sqrt{3}}{2}\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix} = \begin{bmatrix}\frac{\sqrt{3}}{2}\\0\\\frac{\sqrt{1}}{2}\end{bmatrix} \tag{15}$$

и матрица ротације је:

$$\mathbf{R} = \mathbf{R}_r^l = \mathbf{R}_r^0\mathbf{R}_0^l = \left(\mathbf{R}_0^r\right)^T\mathbf{R}_0^l = \begin{bmatrix}\frac{\sqrt{3}}{2} & 0 & \frac{1}{2}\\ 0 & 1 & 0\\ -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2}\end{bmatrix}\begin{bmatrix}\frac{\sqrt{3}}{2} & 0 & \frac{1}{2}\\ 0 & 1 & 0\\ -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2}\end{bmatrix} = \begin{bmatrix}\frac{1}{2} & 0 & \frac{\sqrt{3}}{2}\\ 0 & 1 & 0\\ -\frac{\sqrt{3}}{2} & 0 & \frac{1}{2}\end{bmatrix}. \tag{16}$$

Одатле добијамо есенцијалну матрицу:

$$\mathbf{E} = \mathbf{R}\mathbf{S}_T = \begin{bmatrix}\frac{1}{2} & 0 & \frac{\sqrt{3}}{2}\\ 0 & 1 & 0\\ -\frac{\sqrt{3}}{2} & 0 & \frac{1}{2}\end{bmatrix}\begin{bmatrix}0 & -\frac{1}{2} & 0\\ \frac{1}{2} & 0 & -\frac{\sqrt{3}}{2}\\ 0 & \frac{\sqrt{3}}{2} & 0\end{bmatrix} = \begin{bmatrix}0 & \frac{1}{2} & 0\\ \frac{1}{2} & 0 & -\frac{\sqrt{3}}{2}\\ 0 & \frac{\sqrt{3}}{2} & 0\end{bmatrix} \tag{17}$$



Slika 7. Приказ решења вежбе 2. На десној камери је приказан зрак који пролази кроз тачку $\mathbf{p}_r$ заједно са још једном тачком на њему. Приказани су зраци од те две тачке ка левој камери заједо са местима где они продиру раван леве слике. Спајањем те две тачке је добијена одговарајућа епиполарна права.

Када сад познате ствари убацимо у 13, добијамо:

$$0 = \mathbf{p}_r^T \mathbf{E} \mathbf{p}_l = \begin{bmatrix} 0.2 & 0.3 & 0.25 \end{bmatrix} \begin{bmatrix} 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & -\frac{\sqrt{3}}{2} \\ 0 & \frac{\sqrt{3}}{2} & 0 \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ f_l \end{bmatrix} = \begin{bmatrix} \frac{3}{20} & \frac{1}{10} + \frac{\sqrt{3}}{8} & -\frac{3\sqrt{3}}{20} \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ \frac{1}{4} \end{bmatrix} \tag{18}$$

Одакле добијамо једначину праве:

$$\frac{3}{20} x_l + \left(\frac{1}{10} + \frac{\sqrt{3}}{8} \right) y_l = \frac{3\sqrt{3}}{80} \tag{19}$$

Решење задатка је приказано на слици 7

4.2. *Фундаментална матрица*

Матрица $\mathbf{E}$ садржи информације о положају једне камере у односу на другу, али не садржи никакве информације о самим камерама. У пракси нас више интересују координате у пикселима. Да бисмо одредили везу између позиције пиксела на једној слици и епиполарне линије на другој, увешћемо у матрицу и унутрашње особине две камере. Да бисмо то урадили, физичке координате $\mathbf{p}$ морамо пребацити у координате на слици $\mathbf{q}$. Сетимо се да је $\mathbf{q} = \mathbf{M}\mathbf{p}$ одакле се добија да је $\mathbf{p} = \mathbf{M}^{-1}\mathbf{q}$. Па једначина 13 постаје:

$$\mathbf{q}_r^T \left(\mathbf{M}_r^{-1} \right)^T \mathbf{E} \mathbf{M}_l^{-1} \mathbf{q}_l = 0 \tag{20}$$

Одакле добијамо дефиницију фундаменталне матрице $\mathbf{F}$:

$$\mathbf{F} = \left(\mathbf{M}_r^{-1} \right)^T \mathbf{E} \mathbf{M}_l^{-1} \tag{21}$$

па је:

$$\mathbf{q}_r^T \mathbf{F} \mathbf{q}_l = 0 \tag{22}$$

Фундаментална матрица **F** је слична као и есенцијална матрица **E** само што матрица **F** ради са пикселима, док матрица **E** ради са физичким координатама. Исто као и матрица **E**, матрица **F** је ранга 2. Фундаментална матрица има 7 параметара, два за сваки епипол и три за хомографију која повезује две слике.

Вежба 3 Фундаментална матрица

Одредити фундаменталну матрицу за камере постављене у конфигурацији као у претходном задатку. Одредити на којим позицијама на левој слици се може наћи пиксел са десне слике са координатама $x = 12$ и $y = 7$. Параметри леве камере су:$f_{xl} = 8$, $f_{yl} = 6$, $c_{xl} = 17$ и $c_{yl} = 11$; док су параметри десне камере:$f_{xr} = 8$, $f_{yr} = 6$, $c_{xr} = 14$ и $c_{yr} = 13$.

Решење вежбе 3

Унутрашње матрице камера су:

$$\mathbf{M}_l = \begin{bmatrix} 8 & 0 & 17 \\ 0 & 6 & 11 \\ 0 & 0 & 1 \end{bmatrix}; \ \mathbf{M}_r = \begin{bmatrix} 8 & 0 & 14 \\ 0 & 6 & 13 \\ 0 & 0 & 1 \end{bmatrix}; \tag{23}$$

па су њихове инверзне:

$$\mathbf{M}_l^{-1} = \frac{1}{48}\begin{bmatrix} 6 & 0 & -102 \\ 0 & 8 & -88 \\ 0 & 0 & 48 \end{bmatrix}; \ \mathbf{M}_r^{-1} = \frac{1}{48}\begin{bmatrix} 6 & 0 & -84 \\ 0 & 8 & -104 \\ 0 & 0 & 48 \end{bmatrix}; \tag{24}$$

Одакле добијамо фундаменталну матрицу **F**:

$$\mathbf{F} = \left(\mathbf{M}_r^{-1}\right)^T \mathbf{E}\mathbf{M}_l^{-1} = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 8 & 0 \\ -84 & -104 & 48 \end{bmatrix}\begin{bmatrix} 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & -\frac{\sqrt{3}}{2} \\ 0 & \frac{\sqrt{3}}{2} & 0 \end{bmatrix}\begin{bmatrix} 6 & 0 & -102 \\ 0 & 8 & -88 \\ 0 & 0 & 48 \end{bmatrix} = \tag{25}$$

$$= \begin{bmatrix} 0 & 3 & 0 \\ 4 & 0 & -4\sqrt{3} \\ -52 & -42+24\sqrt{3} & 52\sqrt{3} \end{bmatrix}\begin{bmatrix} 3 & 0 & -51 \\ 0 & 4 & -44 \\ 0 & 0 & 24 \end{bmatrix} = \frac{1}{6}\begin{bmatrix} 0 & 2 & -22 \\ 2 & 0 & -34-16\sqrt{3} \\ -26 & -28+16\sqrt{3} & 750+32\sqrt{3} \end{bmatrix} \tag{26}$$

Коефицијенти извучени испред матрице су избачени, јер је фундаментална матрица инваријантна у одосу на скалирање, тј. матрице **F** и $s\mathbf{F}$ исте. На основу једначине 22 се добија:

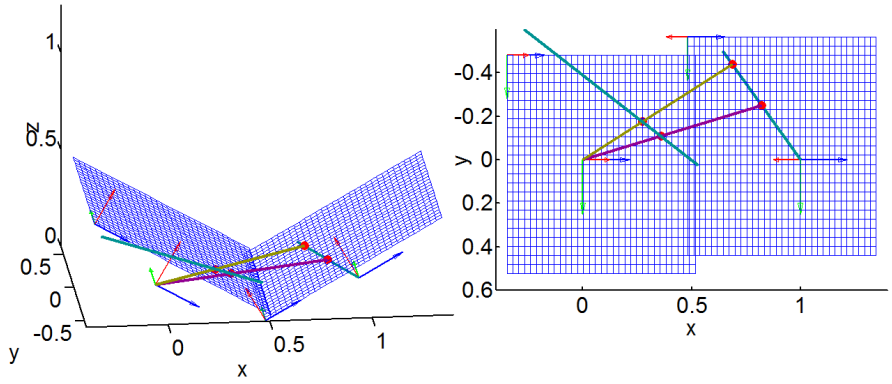
$$\begin{bmatrix} 12 & 7 & 1 \end{bmatrix}\begin{bmatrix} 0 & 2 & -22 \\ 2 & 0 & -34-16\sqrt{3} \\ -26 & -28+16\sqrt{3} & 750+32\sqrt{3} \end{bmatrix}\begin{bmatrix} x_l \\ y_l \\ 1 \end{bmatrix} = \begin{bmatrix} -12 & -4+16\sqrt{3} & 248-80\sqrt{3} \end{bmatrix}\begin{bmatrix} x_l \\ y_l \\ 1 \end{bmatrix} = 0 \tag{27}$$

Одакле се добија једначина праве:

$$248 - 80\sqrt{3} - 12x_l + \left(-4 + 16\sqrt{3}\right) y_l = 0 \tag{28}$$

Решење је приказано на слици 8.

Фундаменталну матрицу **F** можемо израчунати на сличан начин на који смо израчунали хомографију у претходном одељку. Потребно је обезбедити довољан број парова одговарајућих тачака на основу чега ћемо израчунати **F**. Унутрашње матрице камера не морају да нам буду познате јер су оне имплицитно укључене у



Slika 8. Приказ решења вежбе 3.На десној камери је приказан зрак који пролази пиксел на слици $\mathbf{q}_r$ заједно са још једном тачком на њему. Приказани су зраци од те две тачке ка левој камери заједо са местима где они продиру раван леве слике. Спајањем те две тачке је добијена одговарајућа епиполарна права.

фундаменталну матрицу, док изобличење мора бити уклоњено са слике. Минималан број парова тачака је 7, док се може рачунати и са више од 7 тачака када се минимизује разлика квадрата.

5. Стерео калибрација

Изградили смо доста теорије и математичких алата иза модела камера и тродимензионалног простора који ћемо сада искористити. Овај део ће се бавити калибрацијом стерео пара док ће се следећи бавити стерео ректификацијом. Стерео калибрација је поступак рачунања геометријког распореда две камере у простору. Насупрот томе, стерео ректификација је процес кориговања индивидуалних слика да би изгледале као да су усликане са две камере са поравнатим редовима (фронтално паралелна конфигурација, слика 1). Са таквом ректификацијом оптичке осе две камере су паралелне и самим тиме кажемо да се секу у бесконачности. Можемо наравно калибрисати камере да буду у било којој другој конфигурацији, али фокусираћемо се на много учесталији и једноставнији случај. Стерео калибрација налази матрицу ротације $\mathbf{R}$ и вектор транслације $\mathbf{T}$ између две камере (слика 6).

Поступак приликом калибрације стерео пара је готово исти као и приликом калибрације једне камере. Узимају се сукцесивне слике познатог објекта на основу чега се врши рачунање потребних параметара. Приликом стерео калибрације могуће је извршити калибрацију обе камере понаособ, уколико њихови параметри нису познати. Уколико су познати унутрашњи параметри камера пре стерео калибрације, ту информацију можемо искористити како бисмо смањили време рачунања, јер ће се приликом калибрације рачунати само матрица $\mathbf{R}$ и вектор $\mathbf{T}$. Већ смо показали како израчунати есенцијалну и фундаменталну матрицу, али како рачунамо $\mathbf{R}$ и вектор $\mathbf{T}$? Претпоставимо да имамо познат објекат са координатом у простору $\mathbf{P}$.

Такође претпоставимо да можемо одредити његов положај[c] у односу на леву и десну камеру, где за сваку тачку на објекту важи $\mathbf{P}_l = \mathbf{R}_l\mathbf{P} + \mathbf{T}_l$ и $\mathbf{P}_r = \mathbf{R}_r\mathbf{P} + \mathbf{T}_r$. Раније је наведено да координате из координатног система леве камере можемо пребацити у координатни систем десне камере једначином $\mathbf{P}_r = \mathbf{R}(\mathbf{P}_l - \mathbf{T})$. Одатле добијамо израз за рачунање матрице ротације $\mathbf{R}$ и вектора транслације $\mathbf{T}$ између камера:

$$\mathbf{R} = \mathbf{R}_r (\mathbf{R}_l)^T \tag{29}$$

$$\mathbf{T} = \mathbf{T}_l - \mathbf{R}^T\mathbf{T}_r \tag{30}$$

Уколико имамо само један поглед познатог предмета за који смо одредили позицију и орјентацију, у могућности смо да израчунамо матрицу ротације и вектор транслације. Међутим, због шума на слици и грешака приликом заокруживања, за сваки поглед добићемо мало другачије вредности $\mathbf{R}$ и $\mathbf{T}$. Из тог разлога је пожељно узети више различитих слика познатог објекта како би се неутралисао шум и смањила грешка.

Када смо одредили ротацију и транслацију или фундаменталну матрицу $\mathbf{F}$, можемо искористити ове резултате да ректификујемо две стерео слике тако да су епиполарне линије поређане дуж редова слике и да су правци скенирања приликом упаривања исти за обе слике. Иако $\mathbf{R}$ и $\mathbf{T}$ не одређују једнозначну ректификацију, видећемо како да их искористимо (заједно са осталим ограничењима) за ректификацију слике.

6. Стерео ректификација

Најлакше је одредити диспаритет када су две равни слике у фронталној паралелној конфигурацији као на слици 1. Нажалост, као што је раније дискутовано, идеалан распоред камера је скоро немогуће постићи код стерео система, јер камере нису никад компланарне и поравнате по редовима. Слика 3 представља циљ ректификације: Желимо да трансформишемо слике са наше две камере тако да изгледа ју као да су их усликале две камере директно поравнате у франоталној паралелној конфигурацији. Начин одабира равни на коју математички поравнавамо камере зависи од алгоритма који се користи.

Желимо да редови слике буду поравнати након рактификације да би поступак стерео кореспонденције (одређивања слике исте тачке на две различите камере) био поузданији и стабилнији. Запазимо да су поузданост и рачунска ефикасност побољшане ако приликом претраге претражујемо само један ред слике. Резултат поравнавања редова са обе слике са заједничком равни је да су епиполови у бесконачности. Пошто постоји бесконачан број фронталних паралелних ранви које можемо одабрати, морамо увести додатна ограничења. Она могу бити максимизовање преклапања у погледима и/или минимизовање изобличења. Као резултат ректификације, за сваку камеру ћемо добити матрицу ротације камере $\mathbf{R}_{rect}$ и нову унутрашњу матрицу $\mathbf{M}_{rect}$ спаковану у матрицу пројекције $\mathbf{P}$ након ректификације. На основу ових параметара можемо направити мапирање које пресликава позиције пиксела са оргиналне слике на ректификовану слику.

[c]За одређивање позијије познатог предмета потребно је дао познајемо параметре камере и да можемо да одредимо положај на слици минимум 4 тачке са предмета. О овоме је било речи у претходним предавањима

6.1. *Бугеов алгоритам*

Уколико су нам дати матрица ротације и вектор транслације $\mathbf{R}$ и $\mathbf{T}$, Бугеов алгоритам за стерео ректификацију покушава да минимизује изобличење након ректификације док максимизује површину које виде обе камере. Да би се минимизовало изобличење, матрица $\mathbf{R}$ је подељена на пола између две камере. Резултујуће матрице ротације називамо $\mathbf{r}_l$ и $\mathbf{r}_r$ за леву и десну камеру респективно. Свака од камера се ротира за пола ротације, да би њихови основни зраци били паралелни са векторским збиром оргиналних основних зрака. Као што је речено, ова ротација ставља камере да буду компланарне, али не и да редови буду поравнати. Да бисмо израчунали $\mathbf{R}_{rect}$ која ће одвести епипол леве камере у бесконачност и поравнати епиполарне линије хоризонтално, креирамо матрицу ротације почевши од правца епипола $\mathbf{e}_1$. Узећемо основну тачку (c_x, c_y) као координатни почетак. Правац епипола се поклапа директно са вектором транслације између две камере:

$$\mathbf{e}_1 = \frac{\mathbf{T}}{\|\mathbf{T}\|} \tag{31}$$

Следећи вектор $\mathbf{e}_2$ мора бити нормалан на $\mathbf{e}_1$. Добар избор за $\mathbf{e}_2$ је правац ортогоналан на основни зрак. Он се добија рачунањем векторског прозивода између основног зрака (оптичке осе) и вектора $\mathbf{e}_1$ након чега се нормализује вектор и добијамо:

$$\mathbf{e}_2 = \frac{\begin{bmatrix} -T_y & T_x & 0 \end{bmatrix}^T}{\sqrt{T_x^2 + T_y^2}} \tag{32}$$

Трећи вектор једноставно добијамо као векторски производ претходна два:

$$\mathbf{e}_3 = \mathbf{e}_1 \times \mathbf{e}_2 \tag{33}$$

Матрица која одводи епипол леве камере у бесконачност је

$$\mathbf{R}_{rect} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \end{bmatrix}^T \tag{34}$$

Онда се поравнавање по редовима добија ротирањем камера, где су ротације дате матрицама:

$$\mathbf{R}_l = \mathbf{R}_{rect}\mathbf{r}_l \tag{35}$$

$$\mathbf{R}_r = \mathbf{R}_{rect}\mathbf{r}_r \tag{36}$$

Матрице камера након ректификације $\mathbf{M}_{rect\_l}$ и $\mathbf{M}_{rect\_r}$ су одређене тако да слике имају исту резолуцију и померене су тако да редови одговарају једни другима. Ротацијом смо добили само да су епиполарне линије хоризонталне, међутим нису подешене резолуције камера и није поравнат први ред леве слике са првим редом друге слике. Оне су дате кроз матрице пројекције $\mathbf{P}_l$ и $\mathbf{P}_r$:

$$\mathbf{P}_l = \mathbf{M}_{rect\_l}\mathbf{P}'_l = \begin{bmatrix} f_x & 0 & c_{x\_l} \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{37}$$

$$\mathbf{P}_r = \mathbf{M}_{rect\_r}\mathbf{P}'_r = \begin{bmatrix} f_x & 0 & c_{x\_r} \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -T_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{38}$$

Битно је приметити да су за обе камере параметри f_x и f_y исти јер је потребно да је иста резолуција на обе камере такође, параметар c_y је исти јер су поравнати по редовима. У матрици $\mathbf{P}_r$ је додата транслација за $-T_x$ у правцу x, јер се за толико разликује координата тачке у односу на леву камеру. Матрице пројекције узимају тачке из тродимензионалног простора и пројектују их на слику:

$$\mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \tag{39}$$

Па се координате у пикселима могу добити дељењем са w (x/w, y/w). Уколико се изврши множење за леву и десну слику, добијају се координате тачке на левој и десној слици:

$$w \begin{bmatrix} x_l \\ y_l \\ 1 \end{bmatrix} = \mathbf{P}_l \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_{x\_l} & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = Z \begin{bmatrix} \frac{f_x}{Z} X + c_{x\_l} \\ \frac{f_y}{Z} Y + c_y \\ 1 \end{bmatrix} \tag{40}$$

$$w \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix} = \mathbf{P}_r \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_{x\_r} & -f_x T_x \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = Z \begin{bmatrix} \frac{f_x}{Z} X + c_{x\_r} - \frac{f_x T_x}{Z} \\ \frac{f_y}{Z} Y + c_y \\ 1 \end{bmatrix} \tag{41}$$

Приметите да се y координате пиксела поклапају што је последица поравнавања слике по редовима. Подсетимо се да је диспаритет $d = x_l - x_r = c_{x\_l} - c_{x\_r} + \frac{f_x T_x}{Z}$. На основу ових зависности можемо написати формуле за рачунање позиције тачке у простору уколико знамо позицију тачке на левој слици и диспаритет:

$$\frac{1}{Z} = \frac{1}{f_x T_x} \left(d + c_{x\_r} - c_{x\_l} \right) \tag{42}$$

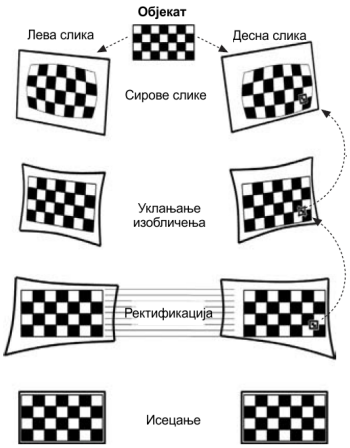
$$X = \frac{Z}{f_x} \left(x_l - c_{x\_l} \right) \tag{43}$$

$$Y = \frac{Z}{f_y} \left(y_l - c_y \right) \tag{44}$$

Што се може компактније написати коришћењем хомогених координата:

$$\mathbf{Q} \begin{bmatrix} x_l \\ y_l \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{f_x} & 0 & 0 & -\frac{c_{x\_l}}{f_x} \\ 0 & \frac{1}{f_y} & 0 & -\frac{c_y}{f_y} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{1}{f_x T_x} & \frac{c_{x\_r} - c_{x\_l}}{f_x T_x} \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ d \\ 1 \end{bmatrix} = w \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{45}$$

Матрица $\mathbf{Q}$ се назива репројективна матрица и она служи да на основу позиције тачке на левој слици и диспаритета одредимо позицију те тачке у простору у односу на координатни систем ректификоване камере. Да бисмо добили позицију тачака у односу на оргиналну позицију леве камере, морамо координате помножити са матрицом $\mathbf{R}_l^T$.



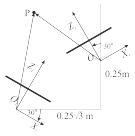
Slika 9. Процес ректификације

6.2. *Ректификациона мапа*

Када смо обавили стерео калибрацију можемо извршити ректификацију. Како бисмо то урадили ефикасно, можемо унапред израчунати мапе за десну и леву камеру понаособ. Као и са сваким мапирањем слике на слику, директно мапирање (приликом кога рачунамо где ће пиксели са оригиналне слике бити на коначној слици) неће попунити све пикселе на финалној слици, јер локација циља неће бити цео број. Због тога се и овде уместо директне користи инверзно мапирање: за сваку тачку са коначне слике израчунамо са које нецелобројне локације са оргиналне слике тај пиксел долази и онда интерполирамо вредност тог пиксела на основу целобројних пиксела који га окружују. Најчешће се за ово користи билинеарна интерполација. Приликом креирања ректификационе мапе узимају се у обзир параметри изобличења камере, почетна матрица камере, матрица ротације и матрице камере након ректификације. Процес ректификације је приказан на слици 9. Након узимања слике и рачунања мапе ректификације може се са сирове слике директно скочити на ректификовану. Након ректификације углавном се врши исецање слике као би се издвојио само део који се преклапа, односно види са обе камере.

Вежба 4 Ректификација и одређивање дубине

Нека лева камера има параметре $f_{xl} = 8$, $f_{yl} = 6$, $c_{xl} = 17$ и $c_{yl} = 11$ и нека десна камера има параметре $f_{xr} = 8$, $f_{yr} = 8$, $c_{xr} = 14$ и $c_{yr} = 13$. Камере су постављене као на слици 10. Прво ректификовати стерео пар према Бугеовом алгоритму, тако да заједнички параметри нових матрица камера буду као они са леве камере пре ректификације, а нека c_{xl} и c_{xr} остану непромењени. Затим одредити позицију тачака у односу на леву камеру пре ректификације чије су слике на оргиналној левој и десној слици:$x_l^1 = 14.997$, $y_l^1 = 7.602$, $x_l^2 = 12.381$, $y_l^2 = 7.536$,$x_l^3 = 11.249$, $y_l^3 = 14.676$ и $x_r^1 = 14.367$, $y_r^1 = 6.679$, $x_r^2 = 7.072$, $y_r^2 = 8.381$,$x_r^3 = 11.243$, $y_r^3 = 12.367$.



Slika 10. Позиције камера на почетку

Решење вежбе 4

Прво је битно заротирати камере за исти угао тако да им равни постану паралелне. Обратите пажњу да матрице ротације камера $\mathbf{r}_r$ и $\mathbf{r}_l$ представљају матрицу ротације из координатног система након ротације у оргинални координатни систем камере па је:

$$\mathbf{r}_l = Rot_{y,30^\circ} = \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \\ -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \end{bmatrix} \tag{46}$$

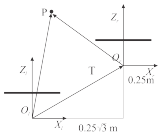
$$\mathbf{r}_r = Rot_{y,-30^\circ} = \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & -\frac{1}{2} \\ 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \end{bmatrix} \tag{47}$$

$$\tag{48}$$

Након те ротације, нови положај камера је приказан на слици 11. Треба запазити да је у овој конфигурацији $\mathbf{T} = \begin{bmatrix} 0.25\sqrt{3} & 0 & 0.25 \end{bmatrix}^T$. Приметимо да су равни слике паралелне и да су оптичке осе паралелне, међутим не ради се о фронталној паралелној конфигурацији јер се епипол леве слике не налази у бесконачности и самим тиме се епиполарне линије секу. Сада треба да заротирамо слике тако да епипол оде у бесконачност:

$$\mathbf{e}_1 = \frac{\mathbf{T}}{\|\mathbf{T}\|} = \frac{\begin{bmatrix} 0.25\sqrt{3} & 0 & 0.25 \end{bmatrix}^T}{0.5} = \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{1}{2} \end{bmatrix}^T \tag{49}$$

$$\mathbf{e}_2 = \frac{\begin{bmatrix} -T_y & T_x & 0 \end{bmatrix}^T}{\sqrt{T_x^2 + T_y^2}} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T \; \mathbf{e}_3 = \mathbf{e}_1 \times \mathbf{e}_2 = \begin{bmatrix} -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \end{bmatrix}^T \tag{50}$$



Slika 11. Позиције камера након прве ротације

Одакле се добија матрица $\mathbf{R}_{rect}$:

$$\mathbf{R}_{rect} = \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \\ -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \end{bmatrix} \tag{51}$$

И сада можемо израчунати матрице ротације:

$$\mathbf{R}_l = \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \\ -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \\ -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \\ 0 & 1 & 0 \\ -\frac{\sqrt{3}}{2} & 0 & \frac{1}{2} \end{bmatrix} \tag{52}$$

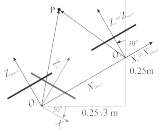
$$\mathbf{R}_r = \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \\ -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & -\frac{1}{2} \\ 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{53}$$

Позиције координатних система пре и после ректификације су приказане на слици 12. Матрице леве камере (коју ректификација не мења) и десне камере пре и после ректификације су:

$$\mathbf{M}_l = \mathbf{M}_{lrect} = \begin{bmatrix} 8 & 0 & 17 \\ 0 & 6 & 11 \\ 0 & 0 & 1 \end{bmatrix}; \tag{54}$$

$$\mathbf{M}_r = \begin{bmatrix} 8 & 0 & 14 \\ 0 & 8 & 13 \\ 0 & 0 & 1 \end{bmatrix}; \mathbf{M}_{rrect} = \begin{bmatrix} 8 & 0 & 14 \\ 0 & 6 & 11 \\ 0 & 0 & 1 \end{bmatrix}; \tag{55}$$

Претпоставимо да на десној камери имамо слику неке тачке са координатама x_r и y_r. Тачке које се пресликавају на ту тачку се налазе на правој која пролази кроз



Slika 12. Положаји камера пре и после ректификације

координатни почетак O_r и има правац задат као:

$$\mathbf{v}^r = s\mathbf{M_r}^{-1} \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix} \tag{56}$$

Тај вектор ћемо сада пребацити у координатни систем камере након ректификације

$$\mathbf{v}^{rrect} = \mathbf{R}_r\mathbf{v}^r = s\mathbf{R}_r\mathbf{M}_r^{-1} \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix} \tag{57}$$

Пројекција било које тачке са те праве на ректификовану слику је:

$$\begin{bmatrix} x_{rrect} \\ y_{rrect} \\ 1 \end{bmatrix} = s\mathbf{M}_{rrect}\mathbf{v}^{rrect} = s\mathbf{M}_{rrect}\mathbf{R}_r\mathbf{M}_r^{-1} \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix} \tag{58}$$

Обратите пажњу да је ова трансформација хомографија, са $\mathbf{H}_r = \mathbf{M}_{rrect}\mathbf{R}_r\mathbf{M}_r^{-1}$. Она представља пресликавање равни оргиналне слике на раван ректификоване слике. Сличан израз се може записати и за леву слику:

$$\begin{bmatrix} x_{lrect} \\ y_{lrect} \\ 1 \end{bmatrix} = s\mathbf{M}_{lrect}\mathbf{R}_l\mathbf{M}_l^{-1} \begin{bmatrix} x_l \\ y_l \\ 1 \end{bmatrix} = \mathbf{H}_l \begin{bmatrix} x_l \\ y_l \\ 1 \end{bmatrix} \tag{59}$$

Прво ћемо одредити хомографије за десну слику и леву слику:

$$\mathbf{H}_r = \mathbf{M}_{rrect}\mathbf{R}_r\mathbf{M}_r^{-1} = \begin{bmatrix}8&0&14\\0&6&11\\0&0&1\end{bmatrix}\begin{bmatrix}8&0&14\\0&8&13\\0&0&1\end{bmatrix}^{-1} = \begin{bmatrix}8&0&14\\0&6&11\\0&0&1\end{bmatrix}\frac{1}{8}\begin{bmatrix}1&0&-14\\0&1&-13\\0&0&8\end{bmatrix} = \begin{bmatrix}1&0&0\\0&\frac{3}{4}&\frac{5}{4}\\0&0&1\end{bmatrix} \tag{60}$$

$$\mathbf{H}_l = \mathbf{M}_{lrect}\mathbf{R}_l\mathbf{M}_l^{-1} = \begin{bmatrix}8&0&17\\0&6&11\\0&0&1\end{bmatrix}\begin{bmatrix}\frac{1}{2}&0&\frac{\sqrt{3}}{2}\\0&1&0\\-\frac{\sqrt{3}}{2}&0&\frac{1}{2}\end{bmatrix}\begin{bmatrix}8&0&17\\0&6&11\\0&0&1\end{bmatrix}^{-1} = \tag{61}$$

$$= \frac{1}{16}\begin{bmatrix}8-17\sqrt{3}&0&353\sqrt{3}\\-11\sqrt{3}&16&-88+187\sqrt{3}\\-\sqrt{3}&0&8+17\sqrt{3}\end{bmatrix} = \begin{bmatrix}-1.34&0&38.213\\-1.191&1&14.743\\-0.108&0&2.34\end{bmatrix}$$

Сада ћемо трансформисати тачке на десној слици из координата датих у односу на оргиналну слику у координате дате у односу на ректификовану слику.

$$\begin{bmatrix}x^1_{rrect}&x^2_{rrect}&x^3_{rrect}\\y^1_{rrect}&y^2_{rrect}&y^3_{rrect}\\w^1&w^2&w^3\end{bmatrix} = \mathbf{H}_r\begin{bmatrix}x^1_r&x^2_r&x^3_r\\y^1_r&y^2_r&y^3_r\\1&1&1\end{bmatrix} = \begin{bmatrix}1&0&0\\0&\frac{3}{4}&\frac{5}{4}\\0&0&1\end{bmatrix}\begin{bmatrix}14.367&7.072&11.243\\6.679&8.381&12.367\\1&1&1\end{bmatrix} = \begin{bmatrix}14.367&7.072&11.234\\6.259&7.536&10.525\\1&1&1\end{bmatrix} \tag{62}$$

Исту ствар ћемо урадити и за леву камеру:

$$\begin{bmatrix}x^1_{lrect}&x^2_{lrect}&x^3_{lrect}\\y^1_{lrect}&y^2_{lrect}&y^3_{lrect}\\w^1&w^2&w^3\end{bmatrix} = \mathbf{H}_l\begin{bmatrix}x^1_l&x^2_l&x^3_l\\y^1_l&y^2_l&y^3_l\\1&1&1\end{bmatrix} = \begin{bmatrix}-1.34&0&38.213\\-1.191&1&14.743\\-0.108&0&2.34\end{bmatrix}\begin{bmatrix}14.997&12.381&11.249\\7.602&7.536&14.676\\1&1&1\end{bmatrix} = \tag{63}$$

$$= \begin{bmatrix}18.112&21.619&23.136\\4.486&7.536&16.024\\0.717&1&1.122\end{bmatrix} = \begin{bmatrix}0.717&\begin{bmatrix}25.268\\6.259\\1\end{bmatrix}&\begin{bmatrix}21.618\\7.536\\1\end{bmatrix}&1.122&\begin{bmatrix}20.61\\14.275\\1\end{bmatrix}\end{bmatrix}$$

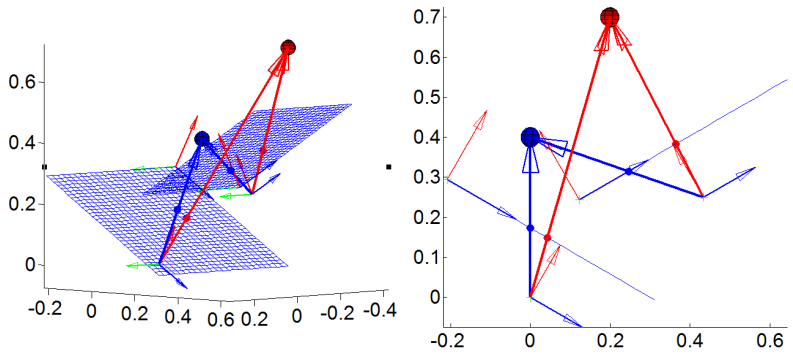
Приметите да y координата треће тачке није иста на левој и десној камери након ректификације. То практично значи да нисмо добро упарили тачке. Уколико бисте израчунали фундаменталну матрицу и проверили дат услов видели бисте да задати подаци за ову тачку не испуњавају потребан услов. Из тог разлога ћемо је избацити из даљег разматрања.

Сада нам је T_x практично једнак дужини вектора $\mathbf{T}$ јер се цело транслирање обавља у само једном правцу. Репројективна матрица $\mathbf{Q}$ је:

$$\mathbf{Q} = \frac{1}{24}\begin{bmatrix}3&0&0&-51\\0&4&0&-44\\0&0&0&24\\0&0&6&-18\end{bmatrix} \tag{64}$$

Диспаритет за прву тачку је $d^1 = x^1_{lrect} - x^1_{rrect} = 10.901$ док је за другу $d^2 = x^2_{lrect} - x^2_{rrect} = 14.546$. На крају добијамо координате у простору у односу на координатни систем везан за леву камеру:

$$\begin{bmatrix}X^1&X^2\\Y^1&Y^2\\Z^1&Z^2\\w^1&w^2\end{bmatrix} = \mathbf{Q}\begin{bmatrix}x^1_{lrect}&x^2_{lrect}\\y^1_{lrect}&y^2_{lrect}\\d^1&d^2\\1&1\end{bmatrix} = \frac{1}{24}\begin{bmatrix}3&0&0&-51\\0&4&0&-44\\0&0&0&24\\0&0&6&-18\end{bmatrix}\begin{bmatrix}25.268&21.618\\6.259&7.536\\10.901&14.5460\\1&1\end{bmatrix} = \begin{bmatrix}1.0336&0.5774\\-0.7902&-0.5774\\1.0000&1.0000\\1.9754&2.8868\end{bmatrix} \tag{65}$$



Slika 13. Израчунат положај тачака у простору.

Одатле се добијају позиције прве и друге тачке у односу на први координатни систем након ректификације:

$$\mathbf{P}_1 = \begin{bmatrix} 0.523 \\ -0.4 \\ 0.526 \end{bmatrix}; \ \mathbf{P}_2 = \begin{bmatrix} 0.2 \\ -0.2 \\ 0.346 \end{bmatrix}; \tag{66}$$

Када пребацимо те координате у координатни систем добијен после ротације која доводи камере у паралелан положај (слика 11) се добије:

$$\mathbf{P}_1^p = \mathbf{R}_{rect}^T \mathbf{P}_1 = \begin{bmatrix} 0.2 \\ -0.4 \\ 0.7 \end{bmatrix}; \ \mathbf{P}_2^p = \mathbf{R}_{rect}^T \mathbf{P}_2 = \begin{bmatrix} 0 \\ -0.2 \\ 0.7 \end{bmatrix}; \tag{67}$$

Позиција ових тачака у односу на координатни систем везан за леву камеру је:

$$\mathbf{P}_1^l = \mathbf{R}_l^T \mathbf{P}_1 = \begin{bmatrix} -0.177 \\ -0.4 \\ 0.706 \end{bmatrix}; \ \mathbf{P}_2^l = \mathbf{R}_l^T \mathbf{P}_2 = \begin{bmatrix} -0.2 \\ -0.2 \\ 0.346 \end{bmatrix}; \tag{68}$$

Графички приказ овог решења је дат на слици 13.

7. Упаривање тачака

Упаривање тачака се односи на налажење тачке из простора на два различита погледа. Оно се може вршити само на деловима сцене који се појављују на обе слике. То је један од разлога зашто добијамо боље резутате ако поравнамо камере у приближно фронтално паралено конфигурацију. Када знамо физичке координате и однос имеђу камера можемо, израчунати диспаритет. Без информације о диспаритету нисмо у могућности да израчунамо растојање до објекта.

Говорићемо укратко о једном брзом и ефикасном алгоритму за упаривање тачака који користи мале *SAD* (сума апсолутне разлике) прозоре како би нашао парове тачака између леве и десне ректификоване слике. Овај алгоритам налази само тачке са јаким сличностима између две слике. Као последица тога у сцени са пуно текстуре

сваки пиксел ће имати израчунату дубину. У сценама без тексутре само неколико тачака ће бити упарено. Постоје три фазе у овом алгоритму који ради са паром ректификованих слика:

(1) Префилтрирање ради побољшања текстуре
(2) Тражење парова тачака дуж хоризонталних епиполарних линија коришћењем прозора
(3) Постфилтеровање да би се уколонило нетачна упаривања

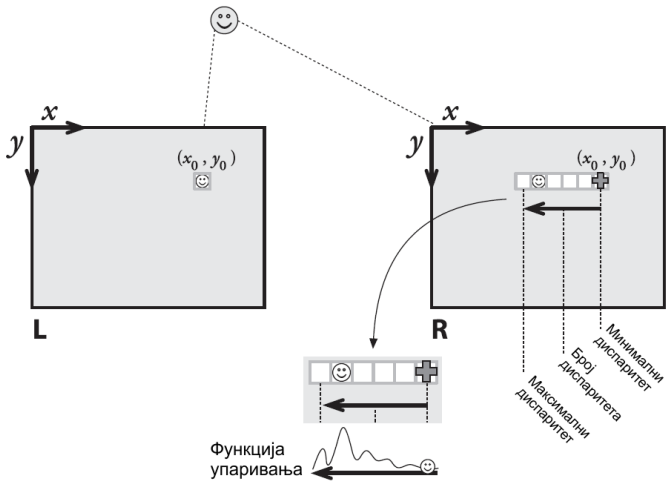
У префилтрирању улазне слике су нормализоване како би се смањила разлика у осветљају и побољшала текстура. Ово се ради превлачењем прозора димензија $5 \times 5, 7 \times 7, \ldots, 21 \times 21$. Ценрални пиксел у слици је замењен са $\min\left[\max\left(I_C - \bar{I}, -I_{CAP}\right), I_{CAP}\right]$.Где је $\bar{I}$ средња вредност у прозору, I_{CAP} је позитиван број чија је уобичајена вредност 30.

Упаривање се врши померањем *SAD* прозора. За сваку тачку у левој слици, претражује се одговарајући ред у десној слици. Након ректификације сваки ред је епиполарна линија тако да је позиција тачке на левој слици у истом реду као и на десној слици. Упаривање може бити урађено ако постоји довољно текстуре да би се детектовала тачка на десној слици и ако се та тачка уопште види на десној слици. Ако се пиксел на левој слици налази на координатама (x_0, y_0) онда његов парњак мора бити у истом реду (слика 14). Ако су камере у фронталној паралелној конфигурацији парњак са десне слике се мора наћи лево од позиције на левој слици. Ако су камере на почетку биле под углом, може се добити упаривање и са негативним диспаритетом (односно парњак се налази десно). Приликом упаривања битно је одредити минимални диспаритет и број диспаритета као би се извршило претраживање. Смањивање броја диспаритета убрзава претраживање, јер се смањује број покушаја упаривања приликом претраге епиполарне линије. Обратите пажњу да велики диспаритет одговара мањем растојању.

Одређивањем минималног диспаритета и броја диспаритета успоставља се *хороптер*, односно запремина која је покривена претрагом приликом стерео упаривања. Слика 15 приказује како параметри претраге ограничавају простор у коме мора бити тачка, приказано је за минимални диспаритет 12, 13 и 15 са бројем диспаритета 5. Сваки диспаритет одређује раван на којој се налази тачка која је паралелна са равни слике, задајући минимални и максимални диспаритет (минимални плус број диспаритета минус 1) одређујемо регион у коме морају се налазити тачке које упарујемо. Ако се налазе тачке ван овог опсега неће бити упарена, и биће 'рупа' на мапи диспаритета. Хороптер може бити повећан смањивањем растојања T између камера, смањивањем жижне даљине, увећавањем броја диспаритета или повећавањем ширине пиксела.

Упаривање унутар хороптера има једно ограничење, које се назива ограничење редоследа, које каже да је редослед тачака на левој слици и десној слици исти. Може се десити да нема упарентих тачака услед оклузије (појава да једна камера види неки део предмета који друга камера не види) али је редослед тачака које су пронађене остао исти. Слично томе, може бити много тачака на десној слици које не видимо на левој али оне не мењају редослед тачака које се виде на обе слике.

За најмањи инкремент у диспаритету можео одредити најмању резлуцију у



Slika 14. Свака тачка са леве сике се мора појавити у истом реду и на десној слици на истој позицији или лево од ње. На доњем делу слике је приказана функција упаривања.

дубини:

$$\Delta Z = \frac{Z^2}{f_x T_x} \Delta d \tag{69}$$

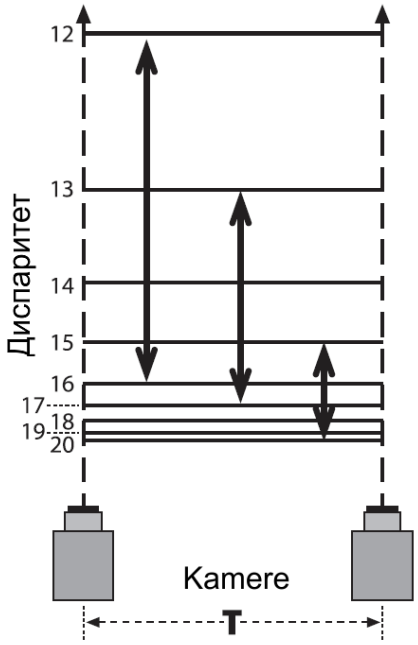
Корисно је имати ову формулу на уму, како бисмо знали да одредимо резолуцију коју очекујемо од стерео пара.

Након упаривања ред је на постфилтерисанње. Доњи део слике 14 приказује типичан одзив функције упаривања када се прозор помера од минималног до максималног дипаритета. Приметите да врло често имамо јак централни пик окружен мањим врховима. Када имамо кандидата за упаривање, постфилтеровање се користи да би се избегло нетачно упаривање. Води се рачуна о јединствености максималне вредности фунцкије упаривања, тако што однос $\frac{max\_val - min\_val}{min\_val}$ мора бити изнад задате вредности. max_val и min_val су максималне и минималне вредности функције упаривања за задату тачку са леве слике.

Такође, води се рачуна о томе да постоји довољно текстуре како би се могло извршити упаривање. Коначно, упаривање блокова може имати проблем близу ивица објеката. То је зато што прозор хвата позадину слике са једне стране ивице и предњу страну објекта са друге стране ивице. То доводи до региона са великим и малим диспаритетима. Приликом постфилтрирања води се рачуна да унутар одређеног прозора разлика у диспаритетима не прелази неку задату.

Вежба 5 Хороптер

Одредити број диспаритета и минимални диспаритет тако да приликом упаривања радимо само са тачкама на растојању између 0.25 и 1 метара. Камере



Slika 15. Свака хоризонтална линија представља константан диспаритет. Са бројем од 5 диспаритета добијамо различите опсеге хороптера.

су на међусобном растојању од 0.25 метара $f_x = 8$. Такође одредити резолуцију на најмањем и највећем растојању ако диспаритет можемо одредити до 1/4 пиксела. Претпоставити да су c_{x_l} и c_{x_r} једнаки.

Решење вежбе 5

Знамо да је:

$$\frac{1}{Z} = \frac{d}{f_x T_x} \tag{70}$$

Одакле добијамо минималну и максималну вредност диспаритета:

$$d_{min} = \frac{f_x T_x}{Z_{max}} = 2 \tag{71}$$

$$d_{max} = \frac{f_x T_x}{Z_{min}} = 8 \tag{72}$$

Одатле се види да је минималан диспаритет 2 и број диспаритета 7. Обратите пажњу да број диспаритета није 6, јер су могући целобројни диспаритети 2, 3, 4, 5, 6, 7, 8.

Slika 16. Оригиналне слике са камера

Што се тиче резолуције добијамо следеће:

$$\Delta Z_{max} = \frac{Z_{max}^2}{f_x T_x} \Delta d = 0.125 \tag{73}$$

$$\Delta Z_{min} = \frac{Z_{min}^2}{f_x T_x} \Delta d = 0.015625 \tag{74}$$

Добија се да стерео пар има 16 пута бољу (мању) резолуцију када је предмет на најближој позицији у односу на ситуацију када је на најдаљој позицији.

8. Програм за одређивање дубине

Коришћењем библиотеке *OpenCV* је имплементиран програм за одређивање дубине са слике. Програм прво чита слику са камера (16), притиском на тастер *'c'* започиње калибрација камера. Потребно је усликати 15 слика шаховске табле димензија 8×6 где се слике узимају притиском на тастер *'.'*. Након калибрације параметри се чувају у фајлу *'Stereopair.yml'*. Након калибрације, приликом поновног покретања програма не мора се вршити нова калибрација, већ се ишчитава напоменути фајл. Када се капритиском на тастер *'t'* добија се слика након исправљање изобличења (слика 17). Након притиска на тастер *'r'* добија се ректификована слика (18). Притиском на тастер *'d'* се добија мапа диспаритета (слика 19), док се притиском на тастер *'3'* врши репројекција и добијене координате тачака се памте у фајл под именом *'fajl.m'*. Листинг програма је дат у додатку.

Додатак А - C++ програм

Главни програм

```
#include "StereoPair.h"
#include "CalibratedCam.h"



#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <opencv2/nonfree/features2d.hpp>


#include <conio.h>



using namespace std;
using namespace cv;
```

Slika 17. Слике након уклањања изобличења

Slika 18. Ректификована слика

Slika 19. Мапа диспаритета

```
int clickX;
int clickY;
int clickedImage;
int imageClicked;
void calibratePair(CStereoPair& c2);
```

```
void CallBackFunc(int event, int x, int y, int
     flags, void* userdata)
{
     if  ( event == EVENT_LBUTTONDOWN )
     {
```

```
      clickX=x;
      clickedImage=0;
           if (x>=*(int*)userdata)
       {
         clickX=x-*(int*)userdata;
         clickedImage=1;
       }
       clickY=y;
       imageClicked=1;
      }

}


void saveXYZ(const char* filename, const Mat& mat
    );

int main()
{
  CStereoPair c2("http://10.1.62.89/mjpg/video.
       mjpg","http://10.1.62.88/mjpg/video.mjpg","
       StereoPair.yml");
  char c;

  Mat imgs[2];
  c2.grab();
  c2.retrieve(imgs[0],imgs[1]);
  int numsaves=0;
  int nrows=imgs[0].rows;
  int ncols=imgs[0].cols;
  Mat slika(imgs[0].rows,imgs[0].cols*2,imgs[0].
       type());
  imshow("slika",slika);
  Mat Fund;
  Mat depth;
  setMouseCallback("slika", CallBackFunc, (void*)
       &ncols);
  clickX=0;
  clickY=0;
  clickedImage=0;
  imgs[0]=slika(Rect(0,0,ncols,nrows));
  imgs[1]=slika(Rect(ncols,0,ncols,nrows));
  int retrieveCalib=0;
  int retrieveRect=0;
  int retrievedepth=0;
  bool depthcalculated=false;
  Rect TRoi;
  Rect Roi1,Roi2;
  Mat tmp;
  if (c2.isCalibrated())
  {
    c2.rectify(0,-1,Roi1,Roi2);
    Fund=c2.getFundamental();
    TRoi=Rect(MIN(Roi1.x,Roi2.x),MIN(Roi1.y,Roi2.
         y), MAX(Roi1.x+Roi1.width, Roi2.x+Roi2.
         width)-MIN(Roi1.x,Roi2.x),  MAX(Roi1.y+
         Roi1.height, Roi2.y+Roi2.height)-MIN(Roi1
         .y,Roi2.y));
  }

  int SADWindowSize=4,numDisparities=128;
  int numberOfImageChannels=3;
  int minDisparity=-100;
  c2.initBlockMatcher( minDisparity,
           numDisparities, //Num Disparities
           SADWindowSize , //SadSize
           32*numberOfImageChannels*SADWindowSize*
                SADWindowSize, //P1
           64*numberOfImageChannels*SADWindowSize*
                SADWindowSize, //P2
           -1, //disp12MaxDiff
           50, //PreFilterCap
           12, //UniguenessRatio
           10, // SpeckleWindowSize
           8, //Speckle Range
           true); // fullDP


  while ((c=waitKey(15))!=27)
  {
    c2.grab();
    if (retrieveCalib)
    {
      if (retrieveRect)
      {
        c2.retrieveRectified(imgs[0],imgs[1]);
        circle(imgs[clickedImage], Point(clickX,
             clickY), 5,Scalar(0,255,0), 2);
        line(imgs[1-clickedImage], Point(0,clickY
             ), Point(640,clickY), Scalar(0,255,0)
             ,2);
        if (retrievedepth)
        {
          c2.retrieveDepthMap(depth);
          depth.convertTo(tmp,CV_8UC1,255./(
               numDisparities*16),-minDisparity
               *255./(numDisparities));
          imshow("depth", tmp);
          retrievedepth=false;
          depthcalculated=true;
        }
      }
      else
      {
        c2.retrieveCalibrated(imgs[0],imgs[1]);
        if (imageClicked)
        {
          circle(imgs[clickedImage], Point(clickX
               ,clickY), 5,Scalar(0,255,0), 2);
          Mat v = (Mat_<double>(3,1) << clickX,
               clickY,1);
          Mat res;
          if (!clickedImage)
          {
            res=Fund*v;
            line(imgs[1], Point(0, -res.at<double
                 >(2,0)/res.at<double>(1,0)),
                 Point(640, -(res.at<double>(0,0)
                 *640+res.at<double>(2,0))/res.at<
                 double>(1,0)),Scalar(0,255,0),2);

          }
          else
          {
            res=v.t()*Fund;
            line(imgs[0], Point(0, -res.at<double
                 >(0,2)/res.at<double>(0,1)),
                 Point(640, -(res.at<double>(0,0)
                 *640+res.at<double>(0,2))/res.at<
                 double>(0,1)),Scalar(0,255,0),2);
          }
        }
      }
    }
    else
    {
      c2.retrieve(imgs[0],imgs[1]);
    }

    imshow("slika",slika);
    if (c=='c')
    {
      calibratePair(c2);
      c2.SaveCalibData("StereoPair.yml");
      c2.rectify(0,0.75,Roi1,Roi2);
      TRoi=Rect(MIN(Roi1.x,Roi2.x),MIN(Roi1.y,
           Roi2.y), MAX(Roi1.x+Roi1.width, Roi2.x+
           Roi2.width)-MIN(Roi1.x,Roi2.x),  MAX(
           Roi1.y+Roi1.height, Roi2.y+Roi2.height)
           -MIN(Roi1.y,Roi2.y));
    }
    else if (c=='t')
    {
      retrieveCalib=!retrieveCalib;
    }
    else if (c=='r')
    {
      retrieveRect=!retrieveRect;
    }
    else if (c=='d')
```

```
    {
      retrievedepth=!retrievedepth;
    }
    else if (c=='s')
    {
      if (depthcalculated)
      {
        imwrite("depth"+std::to_string((
            _ULONGLONG) numsaves++)+".png",tmp);
      }
      imwrite("image"+std::to_string((_ULONGLONG)
          numsaves++)+".png",slika);
    }
    else if (c=='3')
    {
      if (depthcalculated)
      {
        Mat xyz;
        depth=depth/16;
        reprojectImageTo3D(depth, xyz, c2.getQ(),
            false);
        saveXYZ("fajl.m", xyz);
      }
    }
  }


}

void calibratePair(CStereoPair& c2)
{
  printf("radi Kalibraciju");
    vector<Point2f> image1Corners;
    vector<Point2f> image2Corners;
    vector<Point3f> objectCorners;
    Size boardsize(8,6);
    int success=0;
    Mat img1, img2;
    for (int i=0;i<boardsize.height;i++)
      for (int j=0;j<boardsize.width;j++)
        objectCorners.push_back(Point3f(25.0f*i,
            25.0f*j,0.0f));
    namedWindow("Image 1");
    namedWindow("Image 2");
    namedWindow("Corners1");
    namedWindow("Corners2");
    while (waitKey(5)!=27)
    {
      c2.grab();
      c2.retrieve(img1,img2);
      imshow("Image 1", img1);
      imshow("Image 2", img2);


      if (waitKey(15)=='.')
      {

        Mat tmp1=img1.clone();
        Mat tmp2=img2.clone();
        cvtColor(img1,img1,CV_RGB2GRAY);
        cvtColor(img2,img2,CV_RGB2GRAY);
        bool f1=findChessboardCorners(img1,
            boardsize,image1Corners);
        bool f2=findChessboardCorners(img2,
            boardsize,image2Corners);
        if ((f1&&f2) && (image1Corners.size()==
            image2Corners.size()) &&(
            image1Corners.size()==objectCorners.
            size()))
        {
          stringstream  ss,ss2;
          cornerSubPix(img1,image1Corners,Size
              (3,3),Size(-1,-1),TermCriteria(
              CV_TERMCRIT_EPS+CV_TERMCRIT_ITER,
              300,1e-6));
          cornerSubPix(img2,image2Corners,Size
              (3,3),Size(-1,-1),TermCriteria(
              CV_TERMCRIT_EPS+CV_TERMCRIT_ITER,
              300,1e-6));
          drawChessboardCorners(tmp1,boardsize,
              image1Corners,true);
          drawChessboardCorners(tmp2,boardsize,
              image2Corners,true);
          char str[30];

          sprintf(str,"Pokusaj %d\n",success+1);
          putText(tmp1,str,Point(50,50),
              FONT_HERSHEY_COMPLEX,1,Scalar
              (0,0,255),2);
          imshow("Corners1",tmp1);
          imshow("Corners2",tmp2);
          c2.addCalibrationPoints(objectCorners,
              image1Corners,image2Corners);

          waitKey(200);
          if (++success==15) break;
        }
      }
    }
    destroyWindow("Image 1");
    destroyWindow("Image 2");
    destroyWindow("Corners1");
    destroyWindow("Corners2");
    c2.calibrate(false,0);
}


void saveXYZ(const char* filename, const Mat& mat
    )
{
    const double max_z = 1.0e3;
    FILE* fp = fopen(filename, "wt");

    for(int y = 0; y < mat.rows; y++)
    {
        for(int x = 0; x < mat.cols; x++)
        {
            Vec3f point = mat.at<Vec3f>(y, x);

            //if(fabs(point[2])<10 ||fabs(point
                [2] - max_z) < FLT_EPSILON ||
                fabs(point[2]) > max_z   )
                continue;
            fprintf(fp, "%d %d %f %f %f\n", y+1,
                x+1,point[0], point[1], point[2])
                ;
        }
    }
    fclose(fp);
}
```

Класа Камере

Фајл *'CalibratedCam.h'*:

```
#ifndef _CalibratedCam_
#define _CalibratedCam_


#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/calib3d/calib3d.hpp>

using namespace std;
using namespace cv;
class CCalibratedCapture
{
    VideoCapture cap;
    bool    captureCalibrated;
    bool    undistortIntitialized;
    Mat     camMatrix;
    Mat     distCoeffs;
    int     widht;
    int     heigth;
    Mat     mapX,mapY;
```

```
    int     intMethod;
  public:
    CCalibratedCapture():captureCalibrated(false)
        ,undistortIntitialized(false), intMethod(
        INTER_LINEAR) {};
    CCalibratedCapture(int i, String filename=
        String());
    CCalibratedCapture(String file, String
        filename=String());

    virtual CCalibratedCapture& operator >> (
        CV_OUT Mat& image);
    virtual CCalibratedCapture& operator >>= (
        CV_OUT Mat& image);
    void Calibrate(vector<vector<Point3f>>
        allobjectCorners,vector<vector<Point2f>>
        allimageCorners,int flags=0);
    void SetInerpMehhod(int method)   { intMethod
        =method; }

    bool isCalibrated()   {return
        captureCalibrated;};
    bool isRational()   {return distCoeffs.cols
        ==8;};
    int SaveCalibData(String filename);

    bool grab()   {return cap.grab(); }
    bool retrieve(Mat& img)   { return cap.
        retrieve(img);}
    bool retrieveCalibrated(Mat& img);
    Mat& getCamMatrix(){return camMatrix;};
    Mat& getDistCoeffs(){return distCoeffs;};
    Size size(){return Size(widht,heigth);};

};
#endif
```

Фахл *'CalibratedCam.cpp'*;

```
#include "CalibratedCam.h"


#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/calib3d/calib3d.hpp>

using namespace std;
using namespace cv;

CCalibratedCapture::CCalibratedCapture(String
    file, String filename)
{
  cap=VideoCapture(file);
  intMethod=INTER_LINEAR;
  widht=cap.get(CV_CAP_PROP_FRAME_WIDTH);
  heigth=cap.get(CV_CAP_PROP_FRAME_HEIGHT);
  undistortIntitialized=false;
  intMethod=INTER_LINEAR;
  camMatrix=Mat::eye(3,3,CV_64F);
  if (filename.empty())
    captureCalibrated=false;
  else
  {
    FileStorage fs(filename,FileStorage::READ);
    if (!fs.isOpened())
      captureCalibrated=false;
    else
    {
      fs["CamMatrix"]>>camMatrix;
      fs["distCoeffs"]>>distCoeffs;
      captureCalibrated=true;
    }
    fs.release();
  }
}


CCalibratedCapture::CCalibratedCapture(int i,
    String filename)
{
  cap=VideoCapture(i);
  intMethod=INTER_LINEAR;
  widht=cap.get(CV_CAP_PROP_FRAME_WIDTH);
  heigth=cap.get(CV_CAP_PROP_FRAME_HEIGHT);
  undistortIntitialized=false;
  intMethod=INTER_LINEAR;
  camMatrix=Mat::eye(3,3,CV_64F);
  if (filename.empty())
    captureCalibrated=false;
  else
  {
    FileStorage fs(filename,FileStorage::READ);
    if (!fs.isOpened())
      captureCalibrated=false;
    else
    {
      fs["CamMatrix"]>>camMatrix;
      fs["distCoeffs"]>>distCoeffs;
      captureCalibrated=true;
    }
    fs.release();
  }
}


CCalibratedCapture& CCalibratedCapture::operator
    >>(CV_OUT Mat& image)
{
  cap>>image;
  return *this;
}

CCalibratedCapture& CCalibratedCapture::operator
    >>= (CV_OUT Mat& image)
{
  assert(captureCalibrated);
  Mat tmp;
  cap>>tmp;
  if (!undistortIntitialized)
  {
    initUndistortRectifyMap(camMatrix,distCoeffs,
        Mat(),Mat(),tmp.size(),CV_32FC1,mapX,mapY
        );
    undistortIntitialized=true;
  }
  Mat t2;
  remap(tmp,t2,mapX,mapY,intMethod);
  image=t2;
  return *this;
}

void CCalibratedCapture::Calibrate(vector<vector<
    Point3f>> allobjectCorners,vector<vector<
    Point2f>> allimageCorners,int flags)
{
  vector<Mat> rvecs, tvecs;
  calibrateCamera(allobjectCorners,
      allimageCorners,Size(widht,heigth),
      camMatrix,distCoeffs,rvecs, tvecs ,flags);
  captureCalibrated=true;
  undistortIntitialized=false;
}

int CCalibratedCapture::SaveCalibData(String
    filename)
{
  if (captureCalibrated)
  {
    FileStorage fs(filename,FileStorage::WRITE);
    if (!fs.isOpened())
      return 0;
    fs<<"CamMatrix"<<camMatrix;
    fs<<"distCoeffs"<<distCoeffs;
    fs.release();
    return 1;
  }
  else return 0;
}
```

```
bool CCalibratedCapture::retrieveCalibrated(cv::
    Mat &img)
{
  if (!captureCalibrated)
    return false;
  Mat tmp;
  cap.retrieve(tmp);
  if (!undistortIntitialized)
  {
    initUndistortRectifyMap(camMatrix,distCoeffs,
        Mat(),camMatrix,tmp.size(),CV_32FC1,mapX,
        mapY);
    undistortIntitialized=true;
  }
  remap(tmp,img,mapX,mapY,intMethod);
  return true;
}
```

Класа стере пара

Фајл *'StereoPair.h'*:

```
#ifndef _StereoPair_
#define _StereoPair_

#include "CalibratedCam.h"


#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/calib3d/calib3d.hpp>




using namespace std;
using namespace cv;

class CStereoPair
{
    CCalibratedCapture    cap[2];
    bool          pairCalibrated;
    bool          pairRectified;
    bool          rectMapInitialized;
    bool            arbMapInitialized;

    bool          retrievedRectified;
    bool          retrievedRemaped;
    bool          rmRectMapInitialized;

  //  Mat           img;
    // privremene matrice
    Mat           RectImg[2];
    Mat           RotImg[2];
    vector<vector<Point3f>> allObjectPoints;
    vector<vector<Point2f>> allImagePoints[2];
    Mat           Rot;
    Mat           Trans;
    Mat           Essential;
    Mat           Fundamental;
    String          fCalibData[2];

    Mat           RRot[2];
    Mat           RP[2];
    Mat           RmapX[2],RmapY[2];
    Rect          RRoi[2];

    Mat           Q;
    StereoSGBM        blockMatcher;

    Mat           ARot[2]; // Proizvoljna
        Rotacija
    Mat           nCam[2];
    Mat           AmapX[2],AmapY[2];
    Mat           RMRMapX, RMRMapY;
    bool   retrieveRectified();
  public:
    CStereoPair():pairCalibrated(false){};
    CStereoPair(cv::String file1, cv::String
        file2,cv::String fcalibdata1,cv::String
        fcalibdata2);
    CStereoPair(cv::String file1, cv::String
        file2,cv::String fstereodata=String());
    virtual CStereoPair& operator >> (CV_OUT Mat&
         image);
    bool isCalibrated(){return pairCalibrated;}
    bool grab();
    bool retrieve(Mat& img1, Mat& img2);
    bool retrieveCalibrated(Mat& img1, Mat& img2)
        ;
    bool retrieveCalibrated(Mat img[]);

    bool retrieveRectified(Mat& img1, Mat& img2,
        Size size1=Size(), Size size2=Size());
    bool retrieveRectified(Mat img[],Size sz[]=
        NULL);

    bool retrieveRemaped(Mat& img1=Mat(), Mat&
        img2=Mat());
    bool retrieveRemaped(Mat img[]);
    bool RemapRectified(const Mat& src, Mat& dst)
        ;
    void addCalibrationPoints(vector<Point3f>
        objectPoints, vector<Point2f>
        image1Points, vector<Point2f>
        image2Points);
    double calibrate(bool clear=0, int flags=
        CV_CALIB_FIX_INTRINSIC, TermCriteria
        termCrit=TermCriteria(CV_TERMCRIT_EPS +
        CV_TERMCRIT_ITER, 100, 1e-5));

    void  rectify(int flags=0,double alpha=0,
        Rect& roi1=Rect(),Rect& roi2=Rect(),Size
        newImgSize=Size());

    Mat getCamMatrix(int i=1){ return ( cap[i-1].
        getCamMatrix());};
    Mat getDistCoeffs(int i=1){return ( cap[i-1].
        getDistCoeffs());};
    Mat getRRot(int i=1){return RRot[i-1];};
    Mat getRP(int i=1){return RP[i-1];};
    Mat getQ(){return Q;};
    Mat getFundamental(){return Fundamental;};
    int SaveCalibData(String filename, String
        fcalibdata1=String(), String fcalibdata2=
        String() );

    void initRotated(Mat& rvec1, Mat& rvec2);
    void initRotated(Mat Rot[], Mat newCam[]);
    void initBlockMatcher(int minDisparity, int
        numDisparities, int SADWindowSize, int P1
        =0, int P2=0, int disp12MaxDiff=0, int
        preFilterCap=0, int uniquenessRatio=0,
        int speckleWindowSize=0, int speckleRange
        =0, bool fullDP=false);
    void retrieveDepthMap(Mat& depthMap);

    CCalibratedCapture* capPtr(int i=1) {return (
        cap+(i-1));};
};

#endif
```

Фахл *'StereoPair.cpp'*;

```
#include "StereoPair.h"



#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/calib3d/calib3d.hpp>
```

```
using namespace std;
using namespace cv;

CStereoPair::CStereoPair(cv::String file1, cv::
    String file2,cv::String fcalibdata1,cv::
    String fcalibdata2)
{
  fCalibData[0]=fcalibdata1;
  fCalibData[1]=fcalibdata2;
  cap[0]=CCalibratedCapture(file1,fCalibData[0]);

  cap[1]=CCalibratedCapture(file2,fCalibData[1]);

  pairCalibrated=false;
  pairRectified=false;
  rectMapInitialized=false;
  arbMapInitialized=false;

  retrievedRectified=false;
  retrievedRemaped=false;
  rmRectMapInitialized=false;
}

CStereoPair::CStereoPair(cv::String file1, cv::
    String file2,cv::String fstereocalib)
{

  rectMapInitialized=false;
  arbMapInitialized=false;
  rmRectMapInitialized=false;

  if (fstereocalib.empty())
  {
    pairCalibrated=false;
    pairRectified=false;

    cap[0]=CCalibratedCapture(file1,"Cam1.yml");

    cap[1]=CCalibratedCapture(file2,"Cam2.yml");

  }
  else
  {
    FileStorage fs(fstereocalib,FileStorage::READ
        );
    if (!fs.isOpened())
    {
      pairCalibrated=false;
      pairRectified=false;
      fCalibData[0]="Cam1.yml";
      fCalibData[1]="Cam2.yml";
      cap[0]=CCalibratedCapture(file1,"Cam1.yml")
          ;
      cap[1]=CCalibratedCapture(file2,"Cam2.yml")
          ;
    }
    else
    {
      fs["Rot"]>>Rot;
      fs["Trans"]>>Trans;
      fs["Essential"]>>Essential;
      fs["Fundamental"]>>Fundamental;
      fs["fcalibdata1"]>>fCalibData[0];
      cap[0]=CCalibratedCapture(file1,fCalibData
          [0]);
      fs["fcalibdata2"]>>fCalibData[1];
      cap[1]=CCalibratedCapture(file2,fCalibData
          [1]);
      pairCalibrated=cap[0].isCalibrated() && cap
          [1].isCalibrated();
      pairRectified=false;

    }
    fs.release();

  }
}
CStereoPair& CStereoPair::operator >>(CV_OUT Mat&
     image)
{
  Mat i1;
  Mat i2;
  cap[0]>>i1;
  cap[1]>>i2;
  image=Mat(i1.rows,i1.cols*2,i1.type());
  Mat ih=image(Rect(0,0,i1.cols,i1.rows));
  i1.copyTo(ih);
  ih=image(Rect(i1.cols,0,i1.cols,i1.rows));
  i2.copyTo(ih);
  return *this;
}

bool CStereoPair::grab()
{
  retrievedRectified=false;
  retrievedRemaped  =false;
  bool res=cap[0].grab();
  return (res && cap[1].grab());

}

bool CStereoPair::retrieve(Mat& img1, Mat& img2)
{

  bool res=cap[0].retrieve(img1);
  return (res && cap[1].retrieve(img2));
}

bool CStereoPair::retrieveCalibrated(Mat& img1,
    Mat& img2)
{
  bool res=cap[0].retrieveCalibrated(img1);
  return (res && cap[1].retrieveCalibrated(img2))
      ;
}

bool CStereoPair::retrieveCalibrated(Mat img[])
{
  bool res[2];

  for (int i=0;i<2; i++)
  {
    res[i]=cap[i].retrieveCalibrated(img[i]);
  };
  return res[0]&& res[1];
}

void CStereoPair::addCalibrationPoints(vector<
    Point3f> objectPoints, vector<Point2f>
    image1Points, vector<Point2f> image2Points)
{
  allObjectPoints.push_back(objectPoints);
  allImagePoints[0].push_back(image1Points);
  allImagePoints[1].push_back(image2Points);
}

double CStereoPair::calibrate(bool clear, int
    flags,TermCriteria termCrit)
{
  int f2=(flags&CV_CALIB_RATIONAL_MODEL);
  if (!cap[0].isCalibrated()) cap[0].Calibrate(
      allObjectPoints,allImagePoints[0],f2);
  if (!cap[1].isCalibrated()) cap[1].Calibrate(
      allObjectPoints,allImagePoints[1],f2);
  if (cap[0].isRational())
    flags|=CV_CALIB_RATIONAL_MODEL;
  stereoCalibrate(allObjectPoints,allImagePoints
      [0],allImagePoints[1],
              cap[0].getCamMatrix(),cap[0].
                  getDistCoeffs(),
              cap[1].getCamMatrix(),cap[1].
                  getDistCoeffs(),cap[0].size(),
              Rot,Trans,Essential,Fundamental,
                  termCrit,flags);
  // CALIBRATION QUALITY CHECK
// because the output fundamental matrix
```

```
    implicitly
// includes all the output information,
// we can check the quality of calibration using
    the
// epipolar geometry constraint: m2^t*F*m1=0
    double err = 0;
    int npoints = 0;
    vector<Vec3f> lines1,lines2;
    for( int i = 0; i < allObjectPoints.size(); i
        ++ )
    {
        int npt = (int)allImagePoints[0][i].size
            ();
    Mat img1pt=Mat(allImagePoints[0][i]);
    Mat img2pt=Mat(allImagePoints[1][i]);
    undistortPoints(img1pt,img1pt,cap[0].
        getCamMatrix(),cap[0].getDistCoeffs(),Mat
        (),cap[0].getCamMatrix());
    undistortPoints(img2pt,img2pt,cap[1].
        getCamMatrix(),cap[1].getDistCoeffs(),Mat
        (),cap[1].getCamMatrix());
    computeCorrespondEpilines(img1pt,1,
        Fundamental,lines1);
    computeCorrespondEpilines(img2pt,2,
        Fundamental,lines2);
        for(int  j = 0; j < npt; j++ )
        {
            double errij = fabs(allImagePoints
                [0][i][j].x*lines2[j][0] +
                                    allImagePoints
                                        [0][i][j].y*
                                        lines2[j][1]
                                        + lines2[j
                                        ][2]) +
                                fabs(allImagePoints
                                    [1][i][j].x*lines1
                                    [j][0] +
                                    allImagePoints
                                        [1][i][j].y*
                                        lines1[j][1]
                                        + lines1[j
                                        ][2]);
            err += errij;
        }
        npoints += npt;
    }
    cout << "average reprojection err = " <<  err
        /npoints << endl;
  pairCalibrated=true;
  waitKey();
  return 0;
}

int CStereoPair::SaveCalibData(String filename,
    String fcalibdata1, String fcalibdata2 )
{
  if (pairCalibrated)
  {
    if (!fcalibdata1.empty())
      fCalibData[0]=fcalibdata1;
      fcalibdata1="Cam1.yml";
    if (fcalibdata2.empty())
      fcalibdata2="Cam2.yml";
    fCalibData[0]=fcalibdata1;
    fCalibData[1]=fcalibdata2;
    if (!cap[0].SaveCalibData(fCalibData[0]))
        return 0;
    if (!cap[1].SaveCalibData(fCalibData[1]))
        return 0;
    FileStorage fs(filename,FileStorage::WRITE);
    if (!fs.isOpened())
      return 0;
    fs<<"Rot"<<Rot;
    fs<<"Trans"<<Trans;
    fs<<"Essential"<<Essential;
    fs<<"Fundamental"<<Fundamental;
    fs<<"fcalibdata1"<<fCalibData[0];
    fs<<"fcalibdata2"<<fCalibData[1];
    return 1;
  }
  else return 0;


}

void CStereoPair::rectify(int flags, double alpha
    , Rect& roi1, Rect& roi2,Size newImgSize)
{
  stereoRectify(cap[0].getCamMatrix(),cap[0].
      getDistCoeffs(),
              cap[1].getCamMatrix(),cap[1].
                  getDistCoeffs(),
          cap[0].size(),Rot, Trans, RRot[0], RRot
              [1], RP[0], RP[1], Q,flags ,alpha,
              cap[0].size(),RRoi, RRoi+1);

  cout<<"cammatrix 1"<<cap[0].getCamMatrix()<<
      endl;
  cout<<"Trans"<<Trans<<endl;
  cout<<"Trans"<<RP[1]<<endl;
  roi1=Rect(RRoi[0]);
  roi2=Rect(RRoi[1]);
  pairRectified=true;
}

bool CStereoPair::retrieveRectified(cv::Mat &img1
    , cv::Mat &img2,Size size1, Size size2)
{
  if (!pairRectified)
    return false;
  retrieveRectified();

  RectImg[0].copyTo(img1);
  RectImg[1].copyTo(img2);


  return true;
}

bool CStereoPair::retrieveRectified(Mat img[],
    Size sz[])
{
  if (!pairRectified)
    return false;
  retrieveRectified();

  RectImg[0].copyTo(img[0]);
  RectImg[1].copyTo(img[1]);

  return true;
}


void CStereoPair::initBlockMatcher(int
    minDisparity, int numDisparities, int
    SADWindowSize, int P1, int P2, int
    disp12MaxDiff, int preFilterCap, int
    uniquenessRatio , int speckleWindowSize, int
    speckleRange, bool fullDP )
{
  blockMatcher=StereoSGBM(minDisparity,
      numDisparities, SADWindowSize,  P1 ,  P2,
      disp12MaxDiff ,  preFilterCap ,
      uniquenessRatio,  speckleWindowSize ,
      speckleRange ,  fullDP );
}

void CStereoPair::retrieveDepthMap(cv::Mat &
    depthMap)
{

  retrieveRectified();
  Rect TRoi=Rect(MIN(RRoi[0].x,RRoi[1].x),MIN(
      RRoi[0].y,RRoi[1].y), MAX(RRoi[0].x+RRoi
      [0].width, RRoi[1].x+RRoi[1].width)-MIN(
      RRoi[0].x,RRoi[1].x),  MAX(RRoi[0].y+RRoi
      [0].height, RRoi[1].y+RRoi[1].height)-MIN(
      RRoi[0].y,RRoi[1].y));
  depthMap= Mat(RectImg[0].rows,RectImg[0].cols,
      CV_32F,Scalar((blockMatcher.minDisparity-1)
```

```
      *16));
  Mat tmp;
  blockMatcher(RectImg[0](TRoi),RectImg[1](TRoi)
      ,tmp);
  tmp.copyTo(depthMap(TRoi));
}

void CStereoPair::initRotated(cv::Mat &hom1, cv::
    Mat &hom2)
{
  nCam[0]=cap[0].getCamMatrix();
  nCam[1]=cap[1].getCamMatrix();
  ARot[0]=nCam[0].inv()*hom1*nCam[0];
  ARot[1]=nCam[1].inv()*hom2*nCam[1];
  initUndistortRectifyMap(cap[0].getCamMatrix(),
      cap[0].getDistCoeffs(),ARot[0],nCam[0],cap
      [0].size(),CV_32FC1,AmapX[0],AmapY[0]);
  initUndistortRectifyMap(cap[1].getCamMatrix(),
      cap[1].getDistCoeffs(),ARot[1],nCam[1],cap
      [1].size(),CV_32FC1,AmapX[1],AmapY[1]);
  arbMapInitialized=true;

}

void CStereoPair::initRotated(Mat Rot[], Mat
    newCam[])
{
  for (int i=0;i<2;i++)
  {
    ARot[i]=Rot[i];
    nCam[i]=newCam[i];
    initUndistortRectifyMap(cap[i].getCamMatrix()
        ,cap[i].getDistCoeffs(),ARot[i],nCam[i],
        cap[i].size(),CV_32FC1,AmapX[i],AmapY[i])
        ;
  }
  arbMapInitialized=true;
}


bool CStereoPair::retrieveRemaped(cv::Mat &img1,
    cv::Mat &img2)
{
  if (!arbMapInitialized)
    return false;

  Mat tmp1, tmp2;
  cap[0].retrieve(tmp1);
  cap[1].retrieve(tmp2);

  remap(tmp1,img1,AmapX[0],AmapY[0],
      CV_INTER_LINEAR  );
  remap(tmp2,img2,AmapX[1],AmapY[1],
      CV_INTER_LINEAR);
  return true;
}

bool CStereoPair::retrieveRemaped(Mat img[])
{
  if (!arbMapInitialized)
    return false;
  if (!retrievedRemaped)
  {
    Mat tmp[2];
    CCalibratedCapture* cp=cap;
    Mat* RCI=RotImg;
    Mat* RMX=AmapX;
    Mat* RMY=AmapY;
    for (int i=0; i<2; i++)
    {
      cp[i].retrieve(tmp[i]);
      remap(tmp[i],RCI[i],RMX[i],RMY[i],
          CV_INTER_LINEAR);
    }
    retrievedRemaped=true;
  }

  img[0]=RotImg[0];
  img[1]=RotImg[1];
  return true;
}

bool CStereoPair::RemapRectified(const Mat& src,
    Mat& dst)
{
  if (!(arbMapInitialized && pairRectified))
    return false;
  if (!rmRectMapInitialized)
    initUndistortRectifyMap(RP[0](Range(0,3),
        Range(0,3)),Mat::zeros(5,1,CV_64FC1),ARot
        [0]*RRot[0].inv(), nCam[0], src.size(),
        CV_32FC1,RMRMapX,RMRMapY);
  remap(src,dst,RMRMapX,RMRMapY,CV_INTER_NN);
  return true;
}

bool CStereoPair::retrieveRectified()
{
  if (!retrievedRectified)
  {
    if (!rectMapInitialized)
    {
      initUndistortRectifyMap(cap[0].getCamMatrix
          (),cap[0].getDistCoeffs(),RRot[0],RP
          [0],cap[0].size(),CV_32FC1,RmapX[0],
          RmapY[0]);
      initUndistortRectifyMap(cap[1].getCamMatrix
          (),cap[1].getDistCoeffs(),RRot[1],RP
          [1],cap[1].size(),CV_32FC1,RmapX[1],
          RmapY[1]);
      rectMapInitialized=true;
    }
    Mat tmp[2];
    CCalibratedCapture* cp=cap;
    Mat* RCI=RectImg;
    Mat* RMX=RmapX;
    Mat* RMY=RmapY;
    for (int i=0; i<2; i++)
    {
      cp[i].retrieve(tmp[i]);
      remap(tmp[i],RCI[i],RMX[i],RM Y[i],
          CV_INTER_CUBIC);
    }
  }
  retrievedRectified=true;
  return true;
}
```