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

Модел камере и калибрација

Мастер студије
Летњи семестар 2021.

ванр. проф. др Милутин Николић
milutinn@uns.ac.rs

Садржај

Машинска визија почиње детекцијом светлости која долази из окружења. Светлост настаје као зрак који емитује неки извор светлости (нпр. сијалица или сунце), који затим путује кроз простор док не удари у неки предмет. Када светлост удари објекат, већи део светлости се апсорбује, а оно што се не апсорбује, већ се рефлектује, опажамо као боју. Рефлектовано светло прави свој пут до нашег ока (или камере) и скупља се на нашој мрежњачи (или на слици). Геометрија овог процеса - нарочито путовања зрака од објекта кроз објектив у нашем оку или камери до ретине или сензора - је од посебног значаја за машинску визију.

Једноставан али користан модел овог процеса јесте камера са отвором. Она представља имагинарни зид са малом рупом у центру који блокира све зраке осим оних који пролазе кроз мали отвор у центру. Почећемо од камере са отвором како би разјаснили основе пројективне геометрије. Нажалост, стварна камера са отвором није добар начин за фотографисање због мале количине светлости која пада на филм током кратке експозиције. Зато се у нашим очима и камерама налазе сочива која сакупљају више светлости него што би падало у једну тачку. Међутим, увођење сочива нас одаљава од једноставне геометрије камере са рупом и уводи дисторзију због несавршености сочива.

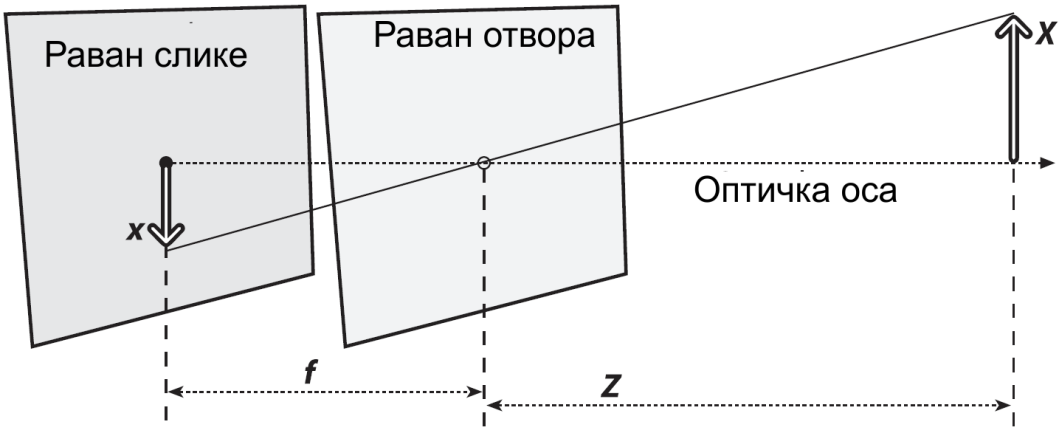
На почетку ћемо научити како калибрацијом камере да исправимо одступања од модела камере са рупом која уводи коришћење сочива. Калибрација камере је важна како би се нашли односи између мерења са слике и димензија у стварном свету. Дакле, однос између јединица фотоапарата (пиксела) и јединица стварног света (нпр. метара) је критичан у сваком покушају да се реконструише тродимензионална сцена.

1. Модел камере

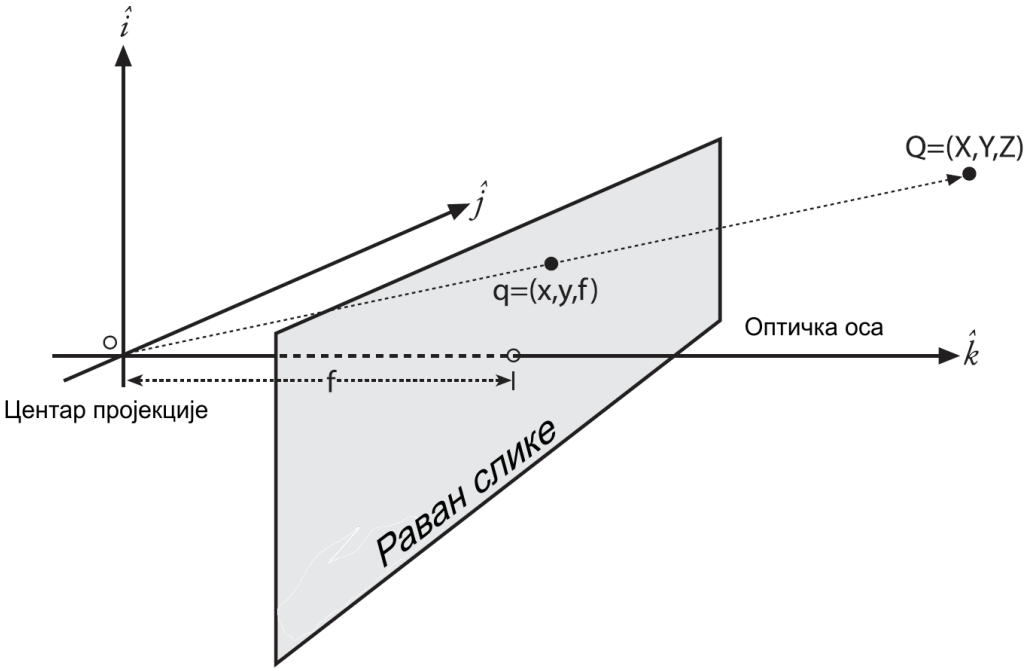
Дискусију ћемо почети од најједноставнијег модела камере са отвором. Код овог модела, светлост са сцене или удаљеног објекта пролази кроз отвор тако да из сваке тачке само један зрак улази у камеру, чиме се та тачка пројектује на раван слике (раван пројектовања). Однос величине слике и релативне величине удаљеног предмета се назива жижна даљина камере. За иделну камеру са отвором управо растојање од отвора до равни пројектовања представља жижну даљину. Модел ове камере је приказан на слици 1, где f предсавља жижну даљину камере, Z је растојање од камере до објекта, X је висина објекта и x је висина слике објекта на равни пројектовања. На основу сличности троуглова добија се:

$$-x = f\frac{X}{Z}. \tag{1}$$

Сада ћемо преуредити модел камере у еквивалентни модел, али са једноставнијом рачуницом. На слици 2 смо заменили позиције отвора и равни слике. Главна разлика је да објекат више није наопачке. Отвор сада представља центар пројекције. Слика тачке на пројективној равни јесте место где је продире зрак који путује од тачке на објекту ка центру пројекције. Растојање између центра пројекције и пројективне равни је жижна даљина f. Место у којој нормала из центра пројекције на раван слике продире ту раван назива се основна тачка. Оса која пролази кроз центар пројекције и основну тачку назива се оптичка оса. Поново се на основу сличности троуглова



Слика 1. Модел камере са отвором



Слика 2. Модел камере са обрнутим положајима равни слике и отвора

може добити:

$$x = f\frac{X}{Z} \tag{2}$$

где се види да је минус нестао пошто слика више није наопачке.

Може се помислити да је основна тачка еквивалентна центру слике, што би значило да је приликом склапања камере сензор монтиран са микронском прецизношћу. У реалном случају центар чипа обично није на оптичкој оси. Да би моделовали то одступање уведена су два нова параметра c_x и c_y. Резултат је релативно једноставан модел у коме се тачка Q из глобалног координатног система са координатама (X, Y, Z)

прелсикава на пиксел на слици q са координатама (x_s, y_s) према следећој зависности:

$$x_s = f_x \frac{X}{Z} + c_x; y_s = f_y \frac{Y}{Z} + c_y \tag{3}$$

Треба имати на уму да су уведене две различите жижне даљине. Разлог за то је то што су појединачни пиксели на типичном јефтином сензору правоугаоног а не квадратног облика. Жижна даљина f_x је заправо производ стварне жижне даљине f и броја пиксела по дужини s_x. Важно је имати на уму да се s_x и s_y не могу одредити било каквом методом калибрације, као ни стварна жижна даљина f (растојање центра пројекције од равни слике). Једино се могу одредити комбинације $f_x = fs_x$ и $f_y = fs_y$ калибрацином без директног мерења тих компоненти.

2. Матрица хомогене трансформације

Мапирање тачке Q у физичком свету са координатама (X_i, Y_i, Z_i) на тачку на пројективној равни са координатама (x_i, y_i) се назива пројективна трансформација. Погодан начин за представљање пројективне трансформације јесу хомогене координате. У посматраном случају, пројектовање се врши из тродимензионалног простора који види камера на дводимензионалну раван слике. Положај тачке у равни је одређен помоћу две координате, међутим како би искористили нотацију хомогених координата и трансформација додајемо и трећу координату на две већ постојеће. Тачке у равни слике ће бити представљене као тродимензионални вектори $\mathbf{q} = (q_1, q_2, q_3)$ где су све тачке чије су вредности пропорционалне еквивалентне. Тачан пиксел на који се пресликава тачка из простора можемо добити дељењем хомогених координата са $Z = q_3$. То нам омогућава да параметре који дефинишу камеру сместимо у једну матрицу трансформације коју ћемо називати унутрашња матрица камере. Коначно, трансформација тачке из простора на тачку на слици се може представити као:

$$\mathbf{q} = \mathbf{MQ}; \quad \mathbf{q} = \begin{bmatrix} wx_s \\ wy_s \\ w \end{bmatrix}; \mathbf{M} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}; \mathbf{Q} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}. \tag{4}$$

Вежба 1 Директна трансформација

Одредити на који пиксел на слици димензија 32 пута 24 пиксела ће се пресликати тачка у простору са координатама $X = 0.8$, $Y = -0.9$, $Z = 2$ уколико је $f_x = 8$, $f_y = 6$, $c_x = 17$ и $c_y = 11$.

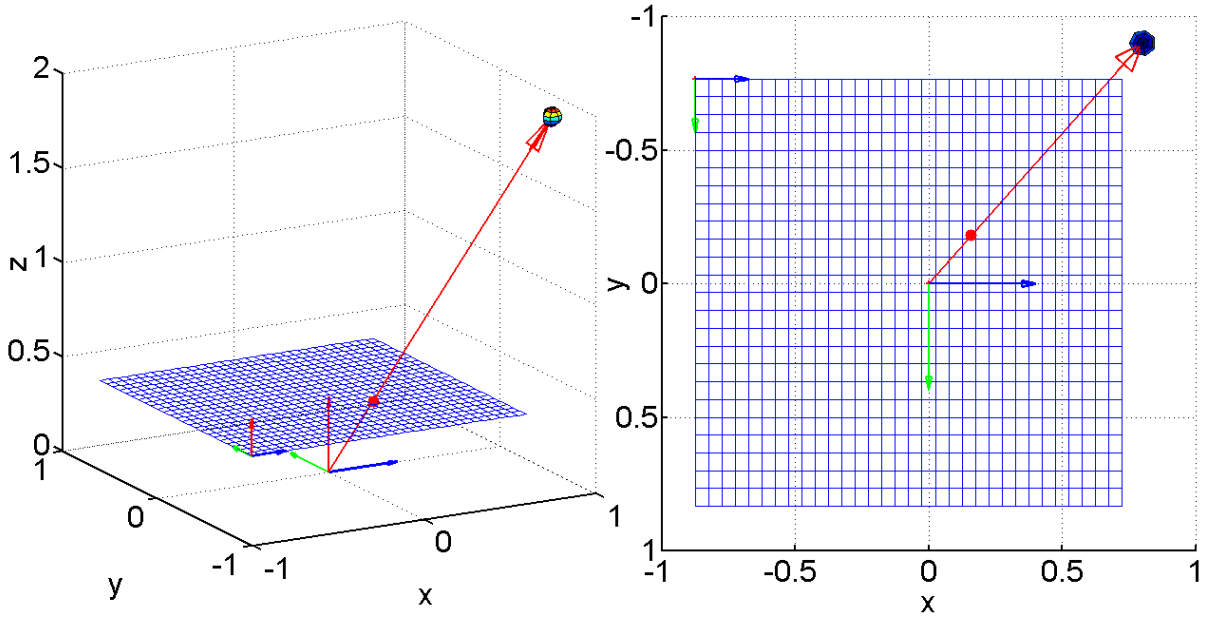
Решење вежбе 1

Унутрашња матрица камере за дати случај је:

$$\mathbf{M} = \begin{bmatrix} 8 & 0 & 17 \\ 0 & 6 & 11 \\ 0 & 0 & 1 \end{bmatrix}. \tag{5}$$

Множењем те матрице са координатама тачке у простору $\mathbf{Q}$ се добија позиција пиксела на слици у хомогеним координатама $\mathbf{q}$:

$$\mathbf{q} = \mathbf{MQ} = \mathbf{M} = \begin{bmatrix} 8 & 0 & 17 \\ 0 & 6 & 11 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.8 \\ -0.9 \\ 2 \end{bmatrix} = \begin{bmatrix} 40.4 \\ 16.6 \\ 2 \end{bmatrix}. \tag{6}$$



Слика 3. Графички приказ решења прве вежбе

Као што је претходно наведено да би смо добили координате на слици морамо добијене хомогене координате да поделимо са q_3 па се добија:

$$x = \frac{q_1}{q_3} = \frac{40.4}{2} = 20.2;\ y = \frac{q_2}{q_3} = \frac{16.6}{2} = 8.3. \tag{7}$$

Графички приказ решења је дат на слици 3.

Вежба 2 Инверзна трансформација

Одредити где се у простору налази тачка која се пресликава на пиксел на слици са координатама $x = 12$ $y = 17$. Димензије слике су 32 пута 24 пиксела и $f_x = 8$, $f_y = 6$, $c_x = 15$ и $c_y = 14$.

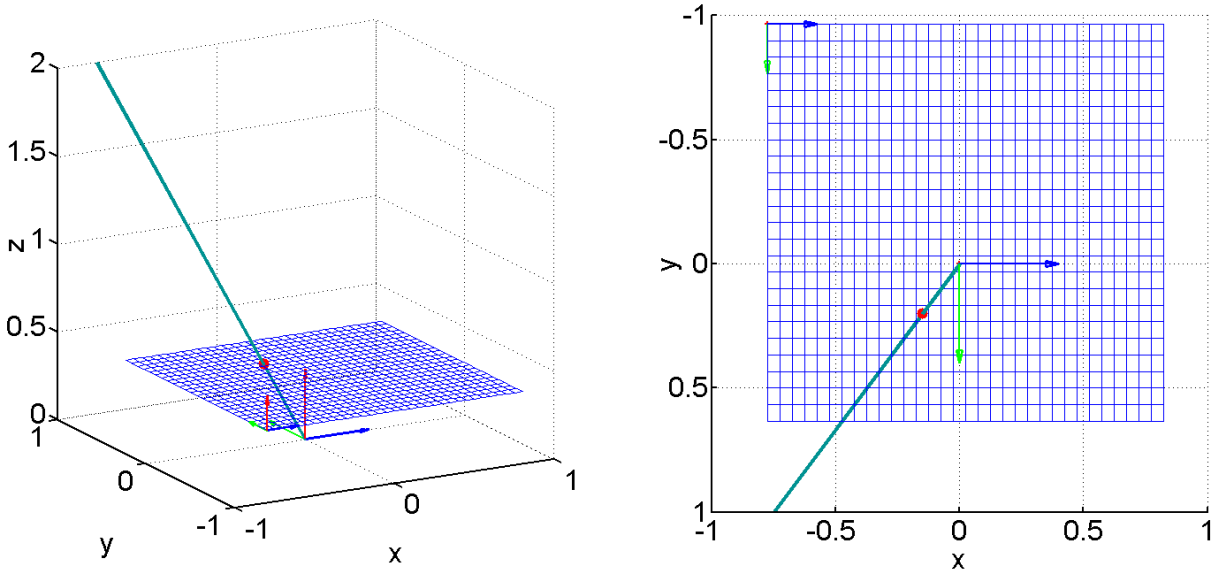
Решење вежбе 2

Унутрашња матрица камере за дати случај је:

$$\mathbf{M} = \begin{bmatrix} 8 & 0 & 15 \\ 0 & 6 & 14 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

На основу једначине 4 делује да једнозначно можемо одредити координату тачке у простору $\mathbf{Q}$, међутим треба узети у обзир да је $\mathbf{q}$ хомогена координата, што значи да су тачке чије су координате пропорционалне су еквивалентне. На примеру то занчи да је $\begin{bmatrix} x & y & 1 \end{bmatrix}^T$ еквивалентно са $\begin{bmatrix} wx & wy & w \end{bmatrix}^T$ за $w \neq 0$. Онда је закон пројекције:

$$w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{MQ} \tag{9}$$



Слика 4. Графички приказ решења прве вежбе

Одакле следи:

$$\mathbf{Q} = w\mathbf{M}^{-1}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = w\begin{bmatrix} 8 & 0 & 15 \\ 0 & 6 & 14 \\ 0 & 0 & 1 \end{bmatrix}^{-1}\begin{bmatrix} 12 \\ 17 \\ 1 \end{bmatrix} = \frac{w}{48}\begin{bmatrix} 6 & 0 & -90 \\ 0 & 8 & -112 \\ 0 & 0 & 48 \end{bmatrix}\begin{bmatrix} 12 \\ 17 \\ 1 \end{bmatrix} = \frac{w}{8}\begin{bmatrix} -3 \\ 4 \\ 8 \end{bmatrix} \tag{10}$$

Овим је дата једначина праве у простору, где се свака тачка на тој правој пресликава на пиксел $(12, 17)$. Права која представља решење овог проблема је приказана на слици 4.

Као што се види из претходног примера уколико детектујемо тачку на само једној камери, нисмо у стању да одредимо где се та тачка налази у простору, већ само полуправу на којој се она налази. Међутим код великог броја роботских система, користи се само једна камера за одређивање позиције објекта. Наравно, у том случају потребна нам је додатна претпоставка како би нам била довољна само једна камера. Један типичан пример ће бити илустрован у вежби 3

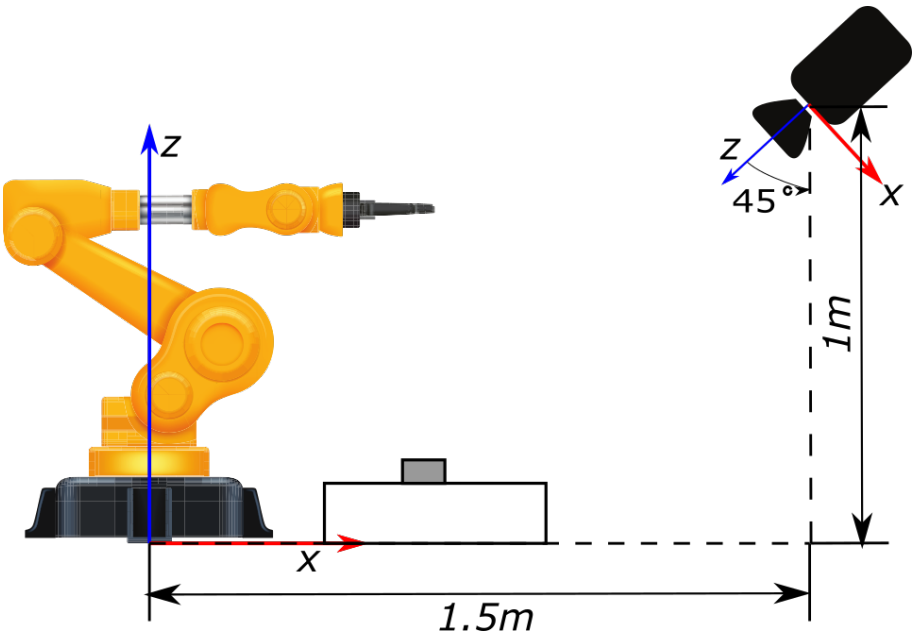
Вежба 3 Робот са камером

Претпоставимо да се на столу испред робота на висини $z = 0.1m$ налази предмет. Тај предмет је детектован камером, на координатама у пикселима на слици $x = 14$, $y = 15$. Позиција камере у односу на робота је приказана на слици 5. Одредити где се у односу на робота налази детектован објекат уколико је су димензије слике су 32 пута 24 пиксела и $f_x = 8$, $f_y = 6$, $c_x = 15$ и $c_y = 14$.

Решење вежбе 3

Као и у претходној вежби унутрашња матрица камере за дати случај је:

$$\mathbf{M} = \begin{bmatrix} 8 & 0 & 15 \\ 0 & 6 & 14 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$



Слика 5. Графички приказ поставке робота и камера

Како је матрица камере иста као и у претходној вежби, можемо искористити резултат из једначине (10), па лако добијамо:

$$\mathbf{Q} = w \begin{bmatrix} -3 \\ 4 \\ 24 \end{bmatrix} \tag{12}$$

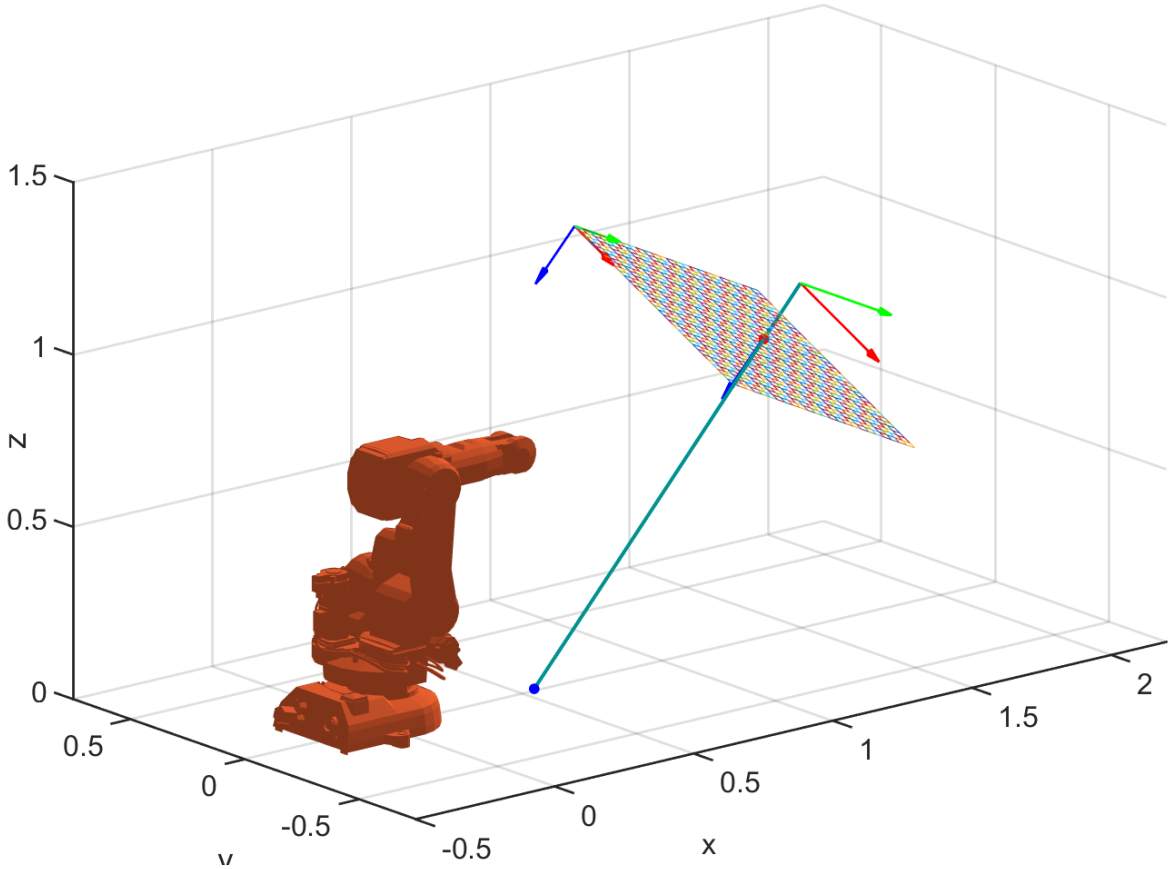
Овде треба обратити пажњу, да је $\mathbf{Q}$ задато у односу на координатни систем камере. Да би смо одредили позицију у односу на робота потребно је прво одредити матрицу хомогене трансформације између робота и камере:

$$\mathbf{H}_R^K = Trans_{x,1.5} \cdot Trans_{z,1} \cdot Rot_{y,\frac{\pi}{4}} Rot_{x,\pi} = \tag{13}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 1.5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

$$= \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} & 1.5 \\ 0 & -1 & 0 & 0 \\ -\frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15}$$

Познавајући матрицу хомогене трансформације, можемо одредити једначину дате



Слика 6. Графички приказ решења прве вежбе

полуправе у односу на координатни систем робота:

$$\mathbf{Q}_R = \mathbf{R}_R^K * \mathbf{Q} + \mathbf{d}_R^K = w \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} \\ 0 & -1 & 0 \\ -\frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} -3 \\ 4 \\ 24 \end{bmatrix} + \begin{bmatrix} 1.5 \\ 0 \\ 1 \end{bmatrix} \tag{16}$$

$$= w \begin{bmatrix} -27\frac{\sqrt{2}}{2} \\ -4 \\ -21\frac{\sqrt{2}}{2} \end{bmatrix} + \begin{bmatrix} 1.5 \\ 0 \\ 1 \end{bmatrix} \tag{17}$$

Како знамо да се предмет налази на равни $z = 0.1$, можемо одредити w:

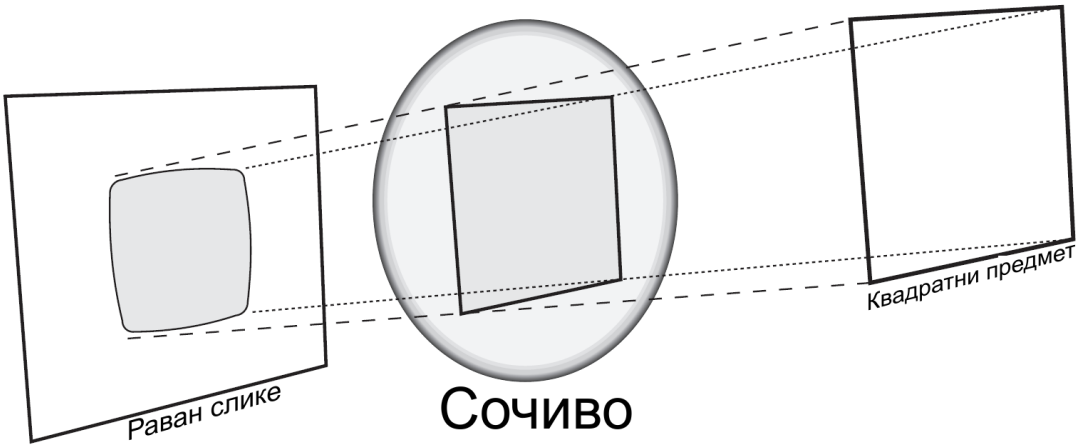
$$w = \frac{3}{70}\sqrt{2} \tag{18}$$

На основу тога, можемо одредити и остале координате:

$$x = \frac{12}{35} \tag{19}$$

$$y = -\frac{6}{35}\sqrt{2} \tag{20}$$

Приказ решења је дат на слици 6.



Слика 7. Графички приказ радијалног изобличења

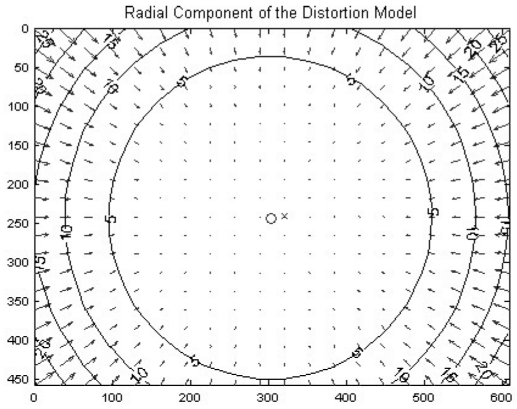
Са идеалном камером са отвором, имамо користан модел за разумевање тродимензионалне геометрије визије. Међутим, запапмтите да врло мало светлости пролази кроз отвор, па би у пракси таква изведба проузроковала врло споро фотографисање јер би морало да се сачека да се довољно светлости сакупи на сензору/филму. За камеру која слика брже, морамо се скупити много светлости са веће површине и фокусирати је, да би светлост конвергирала тачки пројекције. Како би то постигли, користимо сочива. Сочиво може да фокусира велику количину светлости на тачку што омогућава брзо сликање, али се тиме уводе изобиличења на слику.

3. Изобличење слике

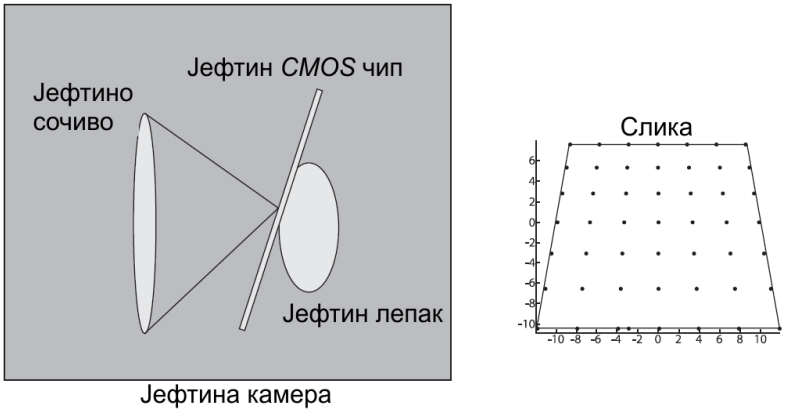
Теоретски, могуће је дефинисати сочиво које неће уносити изобилчења у слику. Међутим, у пракси ниједно сочиво није савршено. Главни разлог за то је производња, јер је знатно једноставније направити сферно сочиво него математички идеално параболично сочиво. Такође, веома је тешко прецизно поравнати сочиво са сензором. У овом поглављу ће бити описана два главна типа изобличења која уводе сочива и како моделовати сваки од њих. Радијално изобличење настаје као резултат неидеалности облика сочива, док тангенцијално изобличење настаје као последица процеса склапања целе камере.

Почећемо од радијалног изобличења. Сочива постојећих камера врло често значајно изобличавају пикселе који се налазе на ивицама слике. Овај феномен се често назива буричасти ефекат или *"fish-eye"* ефекат. Слика 7 даје назнаку зашто долази до радијалног изобличења. Код неких сочива, зраци који се преламају даље од центра се преламају више него зраци који су ближе центру сочива. На типичном јефитином сочиву је то преламање веће него што би требало, што се више одаљавамо од центра сочива. Буричасто изобличење је веома уочљиво у јефтиним веб камерама, али мање присутно код врхунских камера, где се улаже пуно труда у сложена сочива како би се умањило радијално изобличење.

Код радијалног изобличења, на (оптичком) центру слике изобличење је 0 (непостојеће) и расте како се удаљавамо ка периферији. У пракси, изобличења нису велика и функција изобличења није позната. Из тог разлога, функцију изобличења



Слика 8. Приказ радијалног изобличења, стрелице показују где се налазе приказане тачке на радијално изобличеној слици



Слика 9. Графички приказ узрока тангенционалног изобличења

ћемо представити са првих неколико елемената развоја функције у Тејлоров ред око тачке $r = 0$. Калибрацијом ће се одредити коефицијенти Тејлоровог реда чиме ће се добити апроксимирана функција изобличења. За јефтине веб камере се углавном користе прва два елемента тејлоровог реда, чији су коефицијенти по конвенцији k_1 и k_2. За камере са великим изобличењем, као што су на пример *fish-eye* сочива, се додаје и трећи параметар k_3. Радијално изобличење се може окарактерисати следећом једначином:

$$\begin{aligned} x_{kor} &= x\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \\ y_{kor} &= y\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \end{aligned} \tag{21}$$

где је (x, y) оргинална локација тачке на слици мерена у односу на центар слике и (x_{kor}, y_{kor}) нова локација у односу на центар слике која је настала као резултат корекције, док r представља растојање од центра слике. Треба приметити да су чланови r, r^3 и r^5 изостављени да би се одржала симетрија у односу на r. Слика 8 приказује изобличење мреже тачака као последицу радијалног изобличења. Што се више иде ка спољашњости то је јаче изражено померање пиксела ка центру слике. Други

најчешћи тип изобличења је тангенцијално изобличење. Оно настаје као резултат производње где сочиво није потпуно паралелно са сензором што је графички приказано на слици 9. Тангенцијално изобличење карактеризују два параметра p_1 и p_2, а закон корекције је дат следећим изразом:

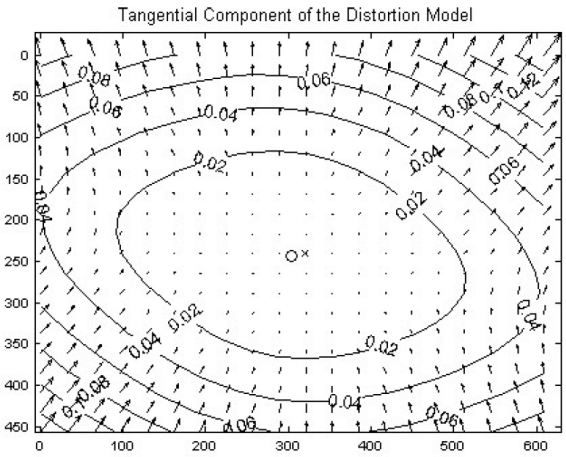
$$\begin{aligned} x_{kor} &= x + \left(2p_1 y + p_2\left(r^2 + 2x^2\right)\right) \\ y_{kor} &= y + \left(p_1\left(r^2 + 2y^2\right) + 2p_2 x\right) \end{aligned} \tag{22}$$

Тиме смо добили укупно пет коефицијената који карактеришу камеру и који су потребни за корекцију изобличења које уводи сочиво. Слика 10 приказује ефекте тангенцијалног изобличења на мрежи тачака. Постоје и други облици изобличења слике али они углавном имају много мањи утицај на слику од радијалног и тангенцијалног, због чега се нећемо бавити њима. Уведене су две корекције, али се поставља питање коју прву урадити од њих и зашто. Прави одговор је да је редослед уствари небитан. Једино је битно да се увек врши истим редоследом и да се врши редоследом као и приликом калибрације. Уколико се прво врши елиминација тангенцијалног па радијалног изобличења, приликом калибрације ће се добити различити параметри у односу на случај када се коригигује прво радијално па тангенцијално изобличење.

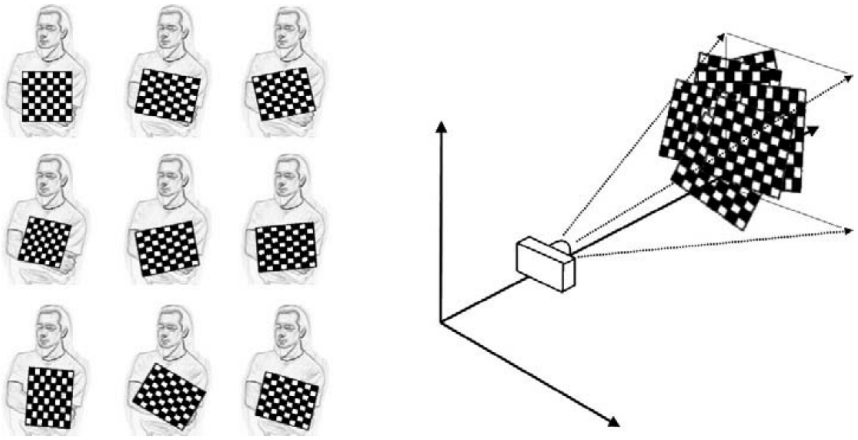
4. Калибрација

Сада када знамо како се моделира камера и који су то унутрашњи математички параметри камере и параметри изобличења, питање које се јаваља је како одредити те параметре. Један од метода калибрације је да сликамо познату структуру која има више различитихи тачака лаких за идентификацију. Посматрајући ту структуру из различитих углова, могуће је израчунати релативну позицијиу и орјентацију структуре у тренутку узимања слике као и унутрашње параметре камере. За сваку слику, позицију објекта у координатном систему везаног за камеру можемо описати помоћу вектора транслације и матрице ротације.

Знајући позицију тачака на објекту и на слици добијамо скуп једначина чијим решавањем се добијају унутрашњи (унутрашња матрица камере) и спољашњи (по-



Слика 10. Приказ тангенцијалног изобличења, стрелице показују где се налазе приказане тачке на тангенцијално изобличеној слици



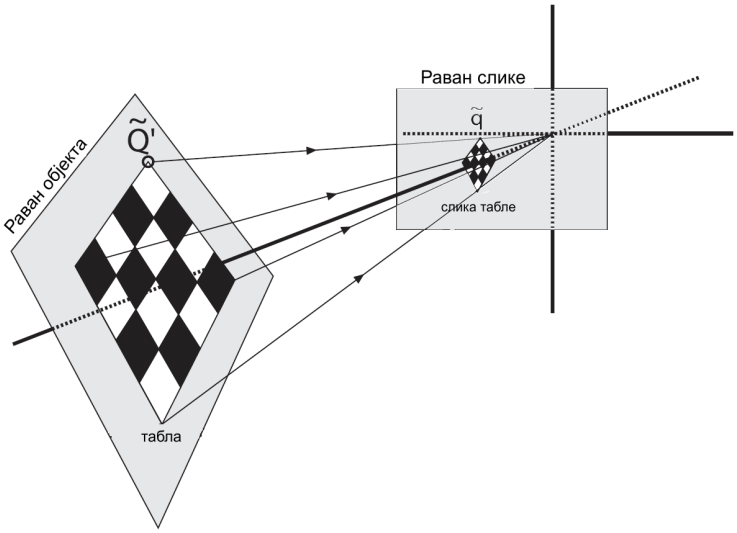
Слика 11. Шаховска табла усликана са различитим позицијама и орјентацијама

зиција и орјентација предмета) параметри као и параметри изобличења. Позиција и орјентација предмета се може приказати помоћу вектора транслације $[x, y, z]^T$ и три угла $[\phi, \theta, \psi]^T$, што нам даје 6 спољашњих параметара за сваку слику. Поред тога, унутрашња матрица камере има 4 пареметра (f_x, f_y, c_x и c_y) који су исти за све слике. Видећемо да уколико користимо планаран објекат за калибрацију са сваке слике можемо добити 8 параметара, одакле следи да су нам потребне бар две слике за одређивање геометријских параметара камере.

Прво ћемо се позабавити калибрационим објектом. У нашем случају ће то бити равна шаховска табла (не мора бити 8x8) на којој се налазе црна и бела поља наизменично (слика 11). Тачке од интереса су унутрашњи ћошкови на шаховској табли, односно места у којима се додирују 4 квадрата. У пракси, било који добро описан објекат може бити коришћен за калибрацију камере, али практичан избор је правилна шара која се понавља. Неке методе су засноване на тродимензионалним објектима, али раванска шара је много једноставнија за употребу, јер је тешко направити прецизне тродимензионалне објекте за калибрацију. У пракси је погодније користити шаховску таблу која је асиметрична и има једну парну а другу непарну димензију. Коришћењем такве асиметрије добија се шаховска табла која има само једну осу симетрије, па се њена орјентација увек може једнозначно одредити.

4.1. *Хомографија*

Сада ћемо разматрати шта можемо да добијемо са раванског предмета. Тачке на равни се трансформишу на раван на слици, и параметри ове трансформације се налазе у матрици хомографије димензија 3×3 коју ћемо описати. У машинској визији, раванску хомографију дефинишемо као пројективно мапирање једне равни на другу. Због тога је пресликавање позиција са дводимензионалне равне површи на сензор наше камере пример планарне хомографије. Ову трансформацију је могуће представити као множење матрица уколико користимо хомогене координате. Нека је позиција тачке на објекту за калибрацију задата са $\tilde{\mathbf{Q}}$ а њена позиција на слици са $\tilde{\mathbf{q}}$. Коришћењем добро познате релације из роботике и једначине 4 добијамо следећи



Слика 12. Приказ пресликавања табле на раван слике

израз:

$$\tilde{\mathbf{q}} = s\mathbf{M}\mathbf{W}\tilde{\mathbf{Q}} \tag{23}$$

Где је $\mathbf{W}$ матрица хомогене трансформације којом је задата позиција шаховске табле у координатном систему камере. Она се састоји из ротације и транслације $\mathbf{W} = \begin{bmatrix}\mathbf{R}\ \mathbf{t}\end{bmatrix}$. $\mathbf{R}$ представља матрицу ротације док $\mathbf{t}$ представља вектор транслације. Такође је уведен параметар s који је фактор скалирања, чија је намена да се експлицитно назначи да су све хомогене координате које су пропорционалне еквивалентне. Без губљења општости, пошто је посматрани предмет равански узећемо за њега да је $Z = 0$. Па се добија следећи израз:

$$\tilde{\mathbf{q}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = s\mathbf{M}\begin{bmatrix}\mathbf{r_1}\ \mathbf{r_2}\ \mathbf{r_3}\ \mathbf{t}\end{bmatrix}\begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = s\mathbf{M}\begin{bmatrix}\mathbf{r_1}\ \mathbf{r_2}\ \mathbf{t}\end{bmatrix}\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}. \tag{24}$$

Матрица хомографије која пресликава тачке на раванском предмету на слику описана матрицом

$$\mathbf{H} = \mathbf{M}\begin{bmatrix}\mathbf{r_1}\ \mathbf{r_2}\ \mathbf{t}\end{bmatrix} \tag{25}$$

одакле добијамо коначни израз за пресликавање:

$$\tilde{\mathbf{q}} = s\mathbf{H}\tilde{\mathbf{Q}}' \tag{26}$$

Матрица хомографије даје везу између позиција тачака на слици и тачака на објекту према следећем изразу:

$$\mathbf{p}_{slika} = s\mathbf{H}\mathbf{p}_{ravan};\ \ \mathbf{p}_{ravan} = s\mathbf{H}^{-1}\mathbf{p}_{slika} \tag{27}$$

$$\mathbf{p}_{slika} = \begin{bmatrix} x_{slika} \\ y_{slika} \\ 1 \end{bmatrix};\ \ \mathbf{p}_{ravan} = \begin{bmatrix} x_{ravan} \\ y_{ravan} \\ 1 \end{bmatrix} \tag{28}$$

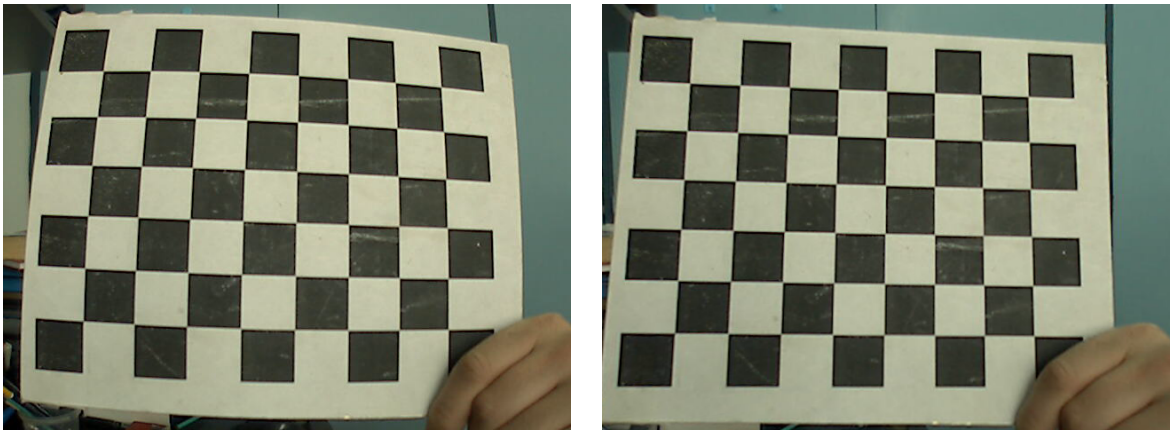
Да би одредили матрицу хомографије између две слике потребна су нам минимум 4 пара тачака. Уколико се користи већи број од четири тачке, решење ће бити добијено на основу минимизације квадратне грешке што је погодно, јер се тиме избацују утицаји шума на слику и других ефеката који нису узети у обзир. Матрица $\mathbf{H}$ иако је димензија 3×3 има 8 слободних параметара јер се узима $h_{33} = 1$ пошто је хомогена трансформација инваријантна на скалирање. Битно је напоменути да за рачунање матрице хомографије не морамо ништа да знамо о унутрашњој матрици камере, која је њен саставни део према једначини 23. Управо се на основу рачунања хомографија за различите погледе на калибрациони предмет рачунају унутрашњи параметри камере током процеса калибрације.

4.2. *Калибрација камере*

Коначно смо дошли до поступка калибрације камере и одређивања њених унутрашњих параметара и параметара изобличења. Прво ћемо обратити пажњу на број слика шаховске табле потребних да се одреде унутрашњи параметри и изобличење. Пре него што почнемо, прво морамо да одредимо број непознатих, односно, колико параметара покушавамо да одредимо током калибрације. Постоје четири унутрашња параметра (f_x, f_y, c_x и c_y) и пет параметара изобличења: три радијална (k_1, k_2 и k_3) и два тангенцијална (p_1 и p_2). Унутрашњи параметри су директно везани за тродимензоионалну геометрију (и тиме за спољашње параметре) и где се шаховска табла налази у простору, док су параметри изобличења везани за дводимензионалну геометрију начина на који су позиције тачака померене. Из тог разлога радимо са две класе параметара одвојено. Три тачке нам дају шест једначина и то је теоретки довољно да се одреде свих пет параметара изобличења, што говори да је једна слика шаховске табле довољна за одређивање параметара изобличења. Наравно, користи се много више од три тачке због прецизности и робусности.

За одређивање спољашњих параметара потребно је одредити три параметра ротације и три параметара транслације што је укупно шест по погледу. Претпоставимо да имамо N тачака на свакој слици и K слика што нам даје укупно $2NK$ услова. Уколико игноришемо изобличење имамо 4 параметара камере и $6K$ спољашњих параметара. Да би тај систем могао да се реши број услова мора бити већи од броја непознатих односно $2NK \geq 6K + 4$ (односно $(N-3)K \geq 2$). Делује да уколико је $N = 5$ само једна слика ће бити довољна за калибрацију. Међутим, $K > 1$ јер током калибрације рачунамо матрицу хомографије за сваки од K погледа. Као што је речено из хомографије се може добити највише 8 параметара из 4 тачке. Из тог разлога $(4-3)K \geq 2$ одакле следи $K \geq 2$. То значи да су две слике шаховске табле димензија 3×3 (броје се само унутрашњи ћошкови) минимум за вршење калибрације. Узимајући у обзир шум и нумеричку стабилност, углавном је потребно око 10 слика шаховске табле или више како би се добила квалитетна калибрација.

Приликом одређивања унутрашњих и спољашњих параметара претпоставља се да изобличење не постоји. Приликом одређивања параметара изобличења користе се већ одређени унутрашњи и спољашњи параметри. Након процене параметара изобличења, унутрашњи и спољашњи параметри се поново одређују. Поступак се понавља итеративно док цео алгоритам не исконвергира.



(а) (б)

Слика 13. Слика са камере пре и након уклањања изобличења

5. Исправљање изобиличења

Као што се већ наслућује, најчешће се раде две ствари са калибрисаном камером. Прва је да се неутралишу изобличења која уноси камера, а друга је конструисање тродимензионалне репрезентације околине са слике. Најефикаснији начин за исправљање изобличења је да се на основу параметара креира мапа изобличења, а затим се на основу ње врши ремапирање ради уклањање изобличења. Мапа изобличења је матрица (слика), где је у свако поље уписано са које локације оргиналне слике се пресликава пиксел у посматрано поље. Ово је врло погодан начин за отклањање изобличења, јер је мапу потребно само једном иницијализовати. Сам процес ремапирања који се понавља сваки пут када се узме слика је веома брз.

Тест програм, у коме је имплементиран цео поступак калибрације камере и исправљања дисторзије је дат у Додатку А. Коришћен је програмски језик *C++* заједно са библиотеком готових функција за машинску визију *OpenCV 2.4.8*. Сирова слика са камере и слика након исправљања изобличења су дате су на слици 13.

6. Одређивање позиције познатог објекта

У случају да имамо већ израчунату унутрашњу матрицу камере можемо израчунати позицију и орјентацију објекта који се посматра. Веома је битно да су димензије објекта који се посматра познате. Да бисмо израчунали његову позицију и орјентацију у простору морамо иденитификовати минимум четити тачке на слици које се налазе на посматраном објекту. Уз то, морамо знати где се те четири тачке налазе на објекту, релативно у односу на координатни систем везан за објекат. На основу ова два низа тачака и унутрашње матрице камере (и параметара изобличења, ако оно већ није неутралисано) може се израчунати позиција и орјентација жељеног објекта у простору.

Предпоставимо да је предмет равански и да смо препознали најмање четири тачке са тог раванског објекта и да знамо њихове координате на слици и на самом објекту. У том случају у стању смо да одредимо матрицу хомографије између равни предмета и равни слике **H**. Како је матрица камере **M** позната на основу израза (25) можемо

написати:

$$\left[\mathbf{r_1}\ \mathbf{r_2}\ \mathbf{t}\right] = s\mathbf{M}^{-1}\mathbf{H}. \tag{29}$$

Овде је узето у обзир да матрице $\mathbf{H}$ и $s\mathbf{H}$ престављају исту хомографију. Скалар s се одређује тако да дужине вектора r_1 и r_2 буду 1. Коначна матрица ротације се добија као:

$$\mathbf{R} = \left[\mathbf{r_1}\ \mathbf{r_2}\ \mathbf{r_1} \times \mathbf{r_2}\right]. \tag{30}$$

Илустрацију поступка приказаћемо на следећем промеру.

Вежба 4 Одређивање позиције

Претпоставимо да имамо камеру димензија 32 пута 24 пиксела са параметрима $f_x = 8$, $f_y = 8$, $c_x = 17$ и $c_y = 11$. Одредити позицију објекта у простору чије се тачке $x_1 = 0.25, y_1 = 0.25$, $x_2 = -0.25, y_2 = 0.25$, $x_3 = -0.25, y_3 = -0.25$ и $x_4 = 0.25, y_4 = -0.25$, пресликавају пикселе на слици $x_1^s = 25, y_1^s = 11$, $x_2^s = 21, y_2^s = 7$, $x_3^s = 17, y_3^s = 11$ и $x_4^s = 21, y_4^s = 15$ респективно.

Решење вежбе 4

Прво што је потребно јесте да одредимо матрицу хомографије. Уколико једначину (27) напишемо детаљно добија се:

$$s_1 \begin{bmatrix} x_1^s \\ y_1^s \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \tag{31}$$

Ову једначину ћемо написати у нешто другачијој форми:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & 0 & 0 & -x_1^s \\ 0 & 0 & 0 & x_1 & y_1 & 1 & 0 & 0 & -y_1^s \\ 0 & 0 & 0 & 0 & 0 & 0 & x_1 & y_1 & -1 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ s_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \tag{32}$$

Када узмемо у обзир сва четири пара тачака добићемо систем 12 једначина са 12 непознатих:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & 0 & 0 & -x_1^s & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & 0 & 0 & -y_1^s & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_1 & y_1 & -1 & 0 & 0 & 0 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -x_2^s & 0 & \\ 0 & 0 & 0 & x_2 & y_2 & 1 & 0 & 0 & 0 & -y_2^s & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_2 & y_2 & 0 & -1 & 0 & 0 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x_3^s & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & 0 & 0 & 0 & 0 & -y_3^s & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_3 & y_3 & 0 & 0 & -1 & 0 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x_4^s \\ 0 & 0 & 0 & x_4 & y_4 & 1 & 0 & 0 & 0 & 0 & 0 & -y_4^s \\ 0 & 0 & 0 & 0 & 0 & 0 & x_4 & y_4 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ -1 \end{bmatrix}. \tag{33}$$

У нашем случају тај систем једначина изгледа:

$$\begin{bmatrix} 0.25 & 0.25 & 1 & 0 & 0 & 0 & 0 & 0 & -25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.25 & 0.25 & 1 & 0 & 0 & -11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.25 & -1 & 0 & 0 & 0 \\ -0.25 & 0.25 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -21 & 0 & \\ 0 & 0 & 0 & -0.25 & 0.25 & 1 & 0 & 0 & 0 & -7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.25 & 0.25 & 0 & -1 & 0 & 0 \\ -0.25 & -0.25 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -17 & 0 \\ 0 & 0 & 0 & -0.25 & -0.25 & 1 & 0 & 0 & 0 & 0 & -11 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.25 & -0.25 & 0 & 0 & -1 & 0 \\ 0.25 & -0.25 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -21 \\ 0 & 0 & 0 & 0.25 & -0.25 & 1 & 0 & 0 & 0 & 0 & 0 & -15 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & -0.25 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ -1 \end{bmatrix}. \tag{34}$$

.

На основу тога се добија матрица хомографије:

$$\mathbf{H} = \begin{bmatrix} 8 & 8 & 21 \\ 8 & -8 & 11 \\ 0 & 0 & 1 \end{bmatrix}. \tag{35}$$

Даље добијамо:

$$\begin{bmatrix} \mathbf{r_1} & \mathbf{r_2} & \mathbf{t} \end{bmatrix} = s\mathbf{M}^{-1}\mathbf{H} = s \begin{bmatrix} 8 & 0 & 17 \\ 0 & 8 & 11 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 8 & 8 & 21 \\ 8 & -8 & 11 \\ 0 & 0 & 1 \end{bmatrix} = s \begin{bmatrix} 1 & 1 & 0.5 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{4} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} \end{bmatrix}. \tag{36}$$

Одатле коначно добијамо матрицу ротације и вектор транслације:

$$\mathbf{R} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & -1 \end{bmatrix}. \tag{37}$$

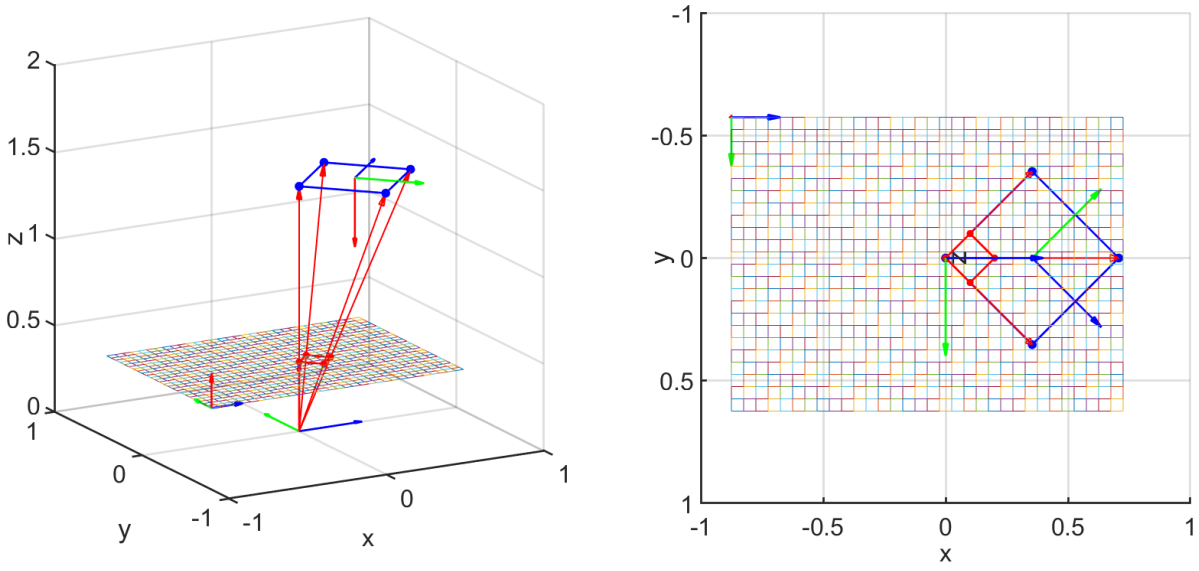
$$\mathbf{t} = \begin{bmatrix} \frac{\sqrt{2}}{4} \\ 0 \\ \frac{\sqrt{2}}{2} \end{bmatrix}, \tag{38}$$

чиме је овај проблем решен. Графички приказ решења је дат на слици 14.

Програм који проналази маркере познатих димензија и одређује њихово растојање од камере је излистан у додатку. На слици 15 се може видети идентификован предмет и његово растојање од камере у два различита случаја.

7. Дискусија

Видели смо у претходним одељцима да се на основу дводимензионалне слике могу извући неки закључци о удаљености објеката од камере. Такву информацију робот може да користи за дохватање. Иако можда не делује тако, одређивање параметара камере је једноставан поступак. Процедура је добро утврђена и потребно ју је урадити само једном за дату камеру. Међутим већи проблем је детекција познатог предмета и одређивања тачака од интереса на њему. Ова процедура није унапред одређена и сваки објекат се детектује на свој начин. Проблем представља променљива позиција

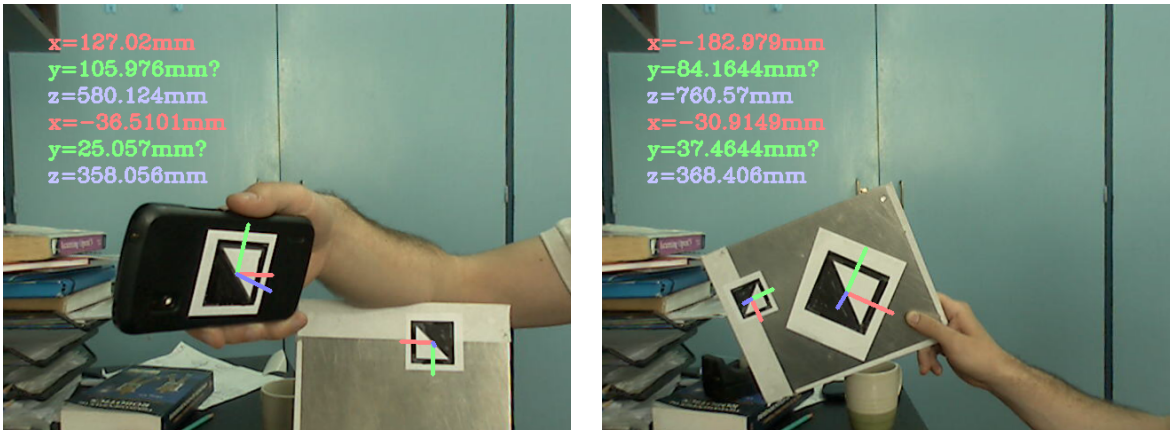


Слика 14. Графички приказ решења треће вежбе

(а) (б)

Слика 15. Детекција познатог објекта и одређивање растојања од њега

и орјентација због чега одређени предмет може имати практично бесконачан број различитих слика. Осим тога, промењљиво осветљење, одсјај и слично може додатно закомпликовати препознавање објеката. Такође, објетки сличног облика али различитих димензија могу направити проблем при процени удаљености. Овај ефекат је илустрован на слици 16, где су на левој слици два објекта истих димензија док су на десној слици два објекта истог облика али различитих димензија. На десној слици се добије да је већи објекат дупло ближи камери него што заиста јесте. Овај проблем је готово немогуће заобићи коришћењем једне камере, због чега се прелази на стереовизију. Она нам омогућава да одредимо удаљеност од камера готово свих тачака са слике без претходног познавања било каквог облика или димензија. У наставку курса ће бити више приче о стереовизији.



(а) (б)

Слика 16. Детекција више објеката

Додатак А - C++ програм

Главни програм

```
#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 <sstream>
#include <string>
#include <math.h>
#include <omp.h>
using namespace std;
using namespace cv;

RNG rng(12345);
void calibrateCam(CCalibratedCapture& cam);
bool CheckContour(int ind,  vector<vector<Point
    >>& contours, const vector<Vec4i>& hierarchy
    , Mat& img, vector<Point2f>& pts);

void main()
{
 CCalibratedCapture webcam("http://10.1.62.89/
     mjpg/video.mjpg","parametri.yml");
 Mat slika,dst,tmp,d2,down;
 char c;
 int retrieveCalib=0;
 bool trackobject=false;
 bool showtext=true;
 vector<vector<Point> > contours;

 vector<Vec4i> hierarchy;
 vector<Vec4i> lines;
 vector<Point3f> objpoints;
 vector<Point3f> coordframe;
 vector<Point2f> coordimage;
 Mat rvec=Mat::zeros(3,1,cv::DataType<double>::
     type);
 Mat tvec=Mat::zeros(3,1,cv::DataType<double>::
     type);

 objpoints.push_back(Point3f(-20.,-20,0));
 objpoints.push_back(Point3f(-20., 20,0));
 objpoints.push_back(Point3f( 20., 20,0));
 objpoints.push_back(Point3f(20.,-20,0));

 coordframe.push_back(Point3f(-0,-0,0));
 coordframe.push_back(Point3f(30, 0,0));
 coordframe.push_back(Point3f( 0, 30,0));
 coordframe.push_back(Point3f(0,-0,30));
 Mat dist=Mat::zeros(5,1, CV_32F);
 int numsaves=0;
 while ( (c=waitKey(1))!=27)
 {
   if (c=='c')
   {
     calibrateCam(webcam);
     webcam.SaveCalibData("parametri.yml");
   }
   else if (c=='t')
   {
     retrieveCalib=!retrieveCalib;
     if (!retrieveCalib) trackobject=false;
   }
   else if (c=='o')
   {
     if (retrieveCalib) trackobject=!trackobject
         ;
   }
   else if (c=='s')
     imwrite("image"+std::to_string((_ULONGLONG)
         numsaves++)+".png",slika);
   else if (c=='x')
     if (trackobject) showtext=!showtext;

   webcam.grab();
   if (retrieveCalib)
     webcam.retrieveCalibrated(slika);
   else
     webcam.retrieve(slika);


   if (trackobject)
   {
     cvtColor(slika, tmp, CV_BGR2GRAY);
     threshold(tmp,dst,128,255,CV_THRESH_BINARY)
         ;
     pyrDown(dst,tmp);
```

```
      threshold(tmp,tmp,128,255,CV_THRESH_BINARY)
          ;
      tmp.copyTo(down);
      findContours( tmp, contours, hierarchy,
           CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE,
           Point(0, 0) );
      Mat drawing = Mat::zeros( tmp.size(),
           CV_8UC3 );


      vector<Point2f> pts;
      int ncontours=0;
      for( int i = 0; i< contours.size(); i++ )
      {
        if (CheckContour(i, contours, hierarchy,
             down, pts)    )
        {
          cornerSubPix(dst,pts,Size(3,3),Size
               (-1,-1),TermCriteria(
               CV_TERMCRIT_EPS+CV_TERMCRIT_ITER,
               50,1e-6));

          solvePnP(objpoints, pts, webcam.
               getCamMatrix(),dist,rvec,tvec,
               false);
          projectPoints(coordframe,rvec,tvec,
               webcam.getCamMatrix(),dist,
               coordimage);

          line(slika,coordimage[0], coordimage
               [1], Scalar(128,128,255),4);
          line(slika,coordimage[0], coordimage
               [2], Scalar(129,255,128),4);
          line(slika,coordimage[0], coordimage
               [3], Scalar(255,128,128),4);

          if (showtext)
           {
             std::cout << "tvec:␣" << tvec << std
                  ::endl;
            std::string str="x="+std::to_string((
                 long double)tvec.at<double>(0,0))
                 +"mm";
            putText(slika,str,Point(50,50+
                 ncontours*90),
                 FONT_HERSHEY_COMPLEX,0.75, Scalar
                 (128,128,255),2);
            str="y="+std::to_string((long double)
                 tvec.at<double>(1,0))+"mm\n";
            putText(slika,str,Point(50,80+
                 ncontours*90),
                 FONT_HERSHEY_COMPLEX,0.75,Scalar
                 (129,255,128),2);
            str="z="+std::to_string((long double)
                 tvec.at<double>(2,0))+"mm";
            putText(slika,str,Point(50,110+
                 ncontours*90),
                 FONT_HERSHEY_COMPLEX,0.75,Scalar
                 (255,196,196),2);
          }
          ncontours++;
        }

      }
    }

    imshow("slika", slika);

  }
}

double Distance(Point pt1, Point pt2,double xsc
     =1, double ysc=1)
{
 return sqrt(pow((double)pt1.x-pt2.x,2)/xsc/xsc+
      pow((double)pt1.y-pt2.y,2)/ysc/ysc);
}
bool CheckContour(int ind,  vector<vector<Point
     >>& contours, const vector<Vec4i>& hierarchy
     , Mat& img, vector<Point2f>& output)
{
 double ca=contourArea(contours[ind]);
 if (ca<25 || ca>5000) return false ;
 if (hierarchy[ind].val[2]==-1) return false;
 Mat mask = Mat::zeros( img.size(), CV_8UC1 );

 drawContours( mask, contours, ind, Scalar(255),
       -1, 8, hierarchy, 0, Point() );
 Scalar immean=mean(img,mask);

 if (immean.val[0]>110 || (immean.val[0]<60) ||
      (contours[ind].size()<4)) return false;
 vector<Point> pts;
 pts.clear();
 output.clear();
 int top=img.rows;
 int bottom=0;
 int left=img.cols;
 int right=0;
 for (int j=0; j<contours[ind].size(); j++)
 {
   if  (contours[ind][j].y>bottom)
     bottom=contours[ind][j].y;
   if  (contours[ind][j].y<top)
     top=contours[ind][j].y;
   if  (contours[ind][j].x<left)
     left=contours[ind][j].x;
   if  (contours[ind][j].x>right)
     right=contours[ind][j].x;
 }
 Mat roi=mask(Rect(Point(left,top),Point(right,
      bottom)));
 Moments mom=moments(roi);
 Point2f center(mom.m10/mom.m00+left, mom.m01/
      mom.m00+top);
 vector<double> dist;
 dist.resize(contours[ind].size());
 for (int j=0;j<contours[ind].size(); j++)
 {
   dist[j]=Distance(contours[ind][j], center);
 }
 double disttresh=sqrt((double)(bottom-top)*(
      right-left))/2;
 for (int j=0;j<4; j++)
 {
   bool ok=true;
   for (int k=j;k<dist.size(); k++)
   {
     ok=true;
     for (int l=0; l<j; l++)
       ok=ok&& (Distance(contours[ind][k],
            contours[ind][l], right-left, bottom-
            top)>0.5);
     if (ok)
     {
       double tmp=dist[k];
       dist[k]=dist[j];
       dist[j]=tmp;
       Point tmppon=contours[ind][k];
       contours[ind][k]=contours[ind][j];
       contours[ind][j]=tmppon;
       break;
     }
   }

   if (!ok) return false;

   for (int k=j+1; k<dist.size(); k++)
     if (dist[k]>dist[j])
     {
       bool ok=true;
       for (int l=0; l<j; l++)
         ok=ok&&(Distance(contours[ind][k],
              contours[ind][l], right-left,
              bottom-top)>0.5);
       if (ok)
       {
         double tmp=dist[k];
         dist[k]=dist[j];
         dist[j]=tmp;
```

```
          Point tmppon=contours[ind][k];
          contours[ind][k]=contours[ind][j];
          contours[ind][j]=tmppon;

        }

      }

  pts.push_back(contours[ind][j]);
 }
 roi.setTo(Scalar(0));
 drawContours( mask, contours, hierarchy[ind].
      val[2], Scalar(255), -1, 8, hierarchy, 0,
      Point() );
 mom=moments(roi);
 center=Point(mom.m10/mom.m00+left, mom.m01/mom.
      m00+top);
 vector<int> hull;
 convexHull(pts, hull);
 int start=0;
 if (hull.size()!=4) return false;


 for (int i=1; i<4; i++)
   if (Distance(pts[hull[i]],center, right-left,
        bottom-top)>Distance(pts[hull[start]],
       center, right-left, bottom-top))
     start=i;

 for (int i=0; i<4; i++)
   output.push_back(pts[hull[(start+i)%4]]);

 cornerSubPix(img,output,Size(2,2),Size(-1,-1),
      TermCriteria(CV_TERMCRIT_EPS+
      CV_TERMCRIT_ITER, 50,1e-6));
 for (int i=0; i<4; i++)
 {
   output[i].x*=2;
   output[i].y*=2;
 }
 return true;
}
void calibrateCam(CCalibratedCapture& cam)
{
 printf("radi␣Kalibraciju");
 vector<Point2f> image1Corners;
 vector<Point3f> objectCorners;
 vector<vector<Point3f>> allObjectPoints;

 vector<vector<Point2f>> allImage1Points;

 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");
 namedWindow("Corners");


 while (waitKey(5)!=27)
 {
   cam.grab();
   cam.retrieve(img1);
   imshow("Image", img1);

   if (waitKey(15)=='.')
   {
     Mat tmp1=img1.clone();
     cvtColor(img1,img1,CV_RGB2GRAY);
     bool f1=findChessboardCorners(img1,
          boardsize,image1Corners);
     if ((f1) &&(image1Corners.size()==
          objectCorners.size()))
     {

       cornerSubPix(img1,image1Corners,Size
            (17,17),Size(-1,-1),TermCriteria(
            CV_TERMCRIT_EPS+CV_TERMCRIT_ITER,
            300,1e-6));
       drawChessboardCorners(tmp1,boardsize,
            image1Corners,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("Corners",tmp1);

       allObjectPoints.push_back(objectCorners);

       allImage1Points.push_back(image1Corners);
       if (++success==20) break;
     }
   }
 }
 cam.Calibrate(allObjectPoints, allImage1Points)
     ;
 destroyWindow("Image");
 destroyWindow("Corners");

}
```

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;
}
```