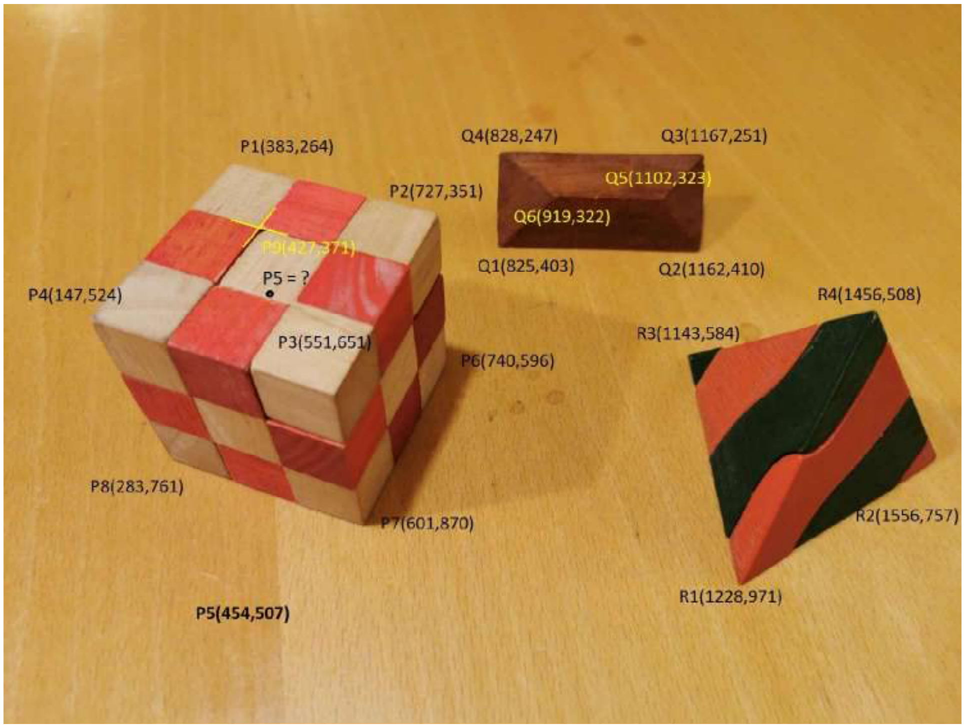
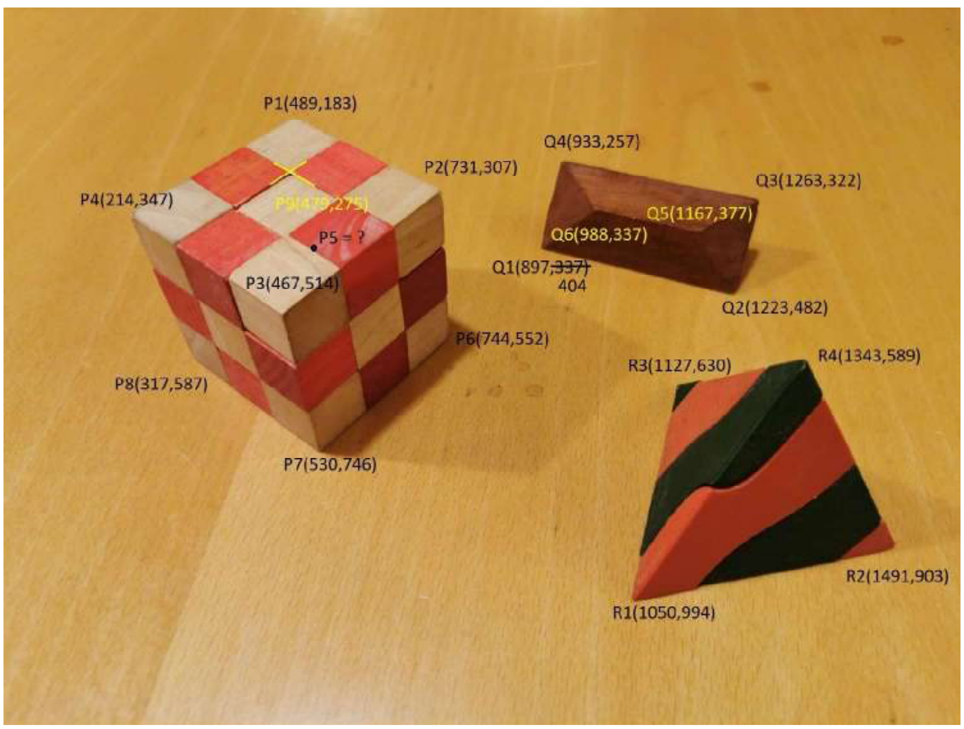
3 D rekonstrukcija iz ravanskih projekcija

Srdjan Vukmirovic, decembar 2023.

(pratece slike su leva i desna slika (kocke, tetraedra i “piramide”)





"Podesavanje" piksel koordinata

Ovo je osam proizvoljnih tacaka u piksel koordinatama P1, P2, P3, P4, R1, R2, R3, R4, ocitanih u Paint - u. To mogu biti bilo koje tacke koje identifikujemo na obe fotografije. Uzimamo 4 temena kocke (gornja pljosan) i 4 temena tetraedra. Mogli smo uzeti i vise odgovarajucih tacaka.

```
p1l = {383, 264, 1};
p2l = {727, 351, 1};
p3l = {551, 651, 1};
p4l = {147, 524, 1};
r1l = {1228, 971, 1};
r2l = {1556, 757, 1};
r3l = {1143, 587, 1};
r4l = {1456, 508, 1};
p1r = {489, 183, 1};
p2r = {731, 307, 1};
p3r = {467, 514, 1};
p4r = {214, 347, 1};
r1r = {1050, 994, 1};
r2r = {1491, 903, 1};
r3r = {1127, 630, 1};
r4r = {1343, 589, 1};
```

Teoretski je dovoljno i 7, ali je onda postupak odredjivanja nesto drugaciji)

```
leve8 = {p1l, p2l, p3l, p4l, r1l, r2l, r3l, r4l};
desne8 = {p1r, p2r, p3r, p4r, r1r, r2r, r3r, r4r};
```

Sada jos menjamo malo koordinate tako da koordinatni sistem bude u desnom gornjem uglu fotografija. Moje fotografije su 1600 x1200 piksela.
y - koordinata je dobra, x - koordinatu treba oduzeti od 1600. To radi funkcija piksel.

```
piksel[{x_, y_, z_}] := {1600 - x, y, 1}
```

Primenjujemo tu funkciju na sve tacke

```
leve8 = piksel /@ leve8
desne8 = piksel /@ desne8
```

```
{{1217, 264, 1}, {873, 351, 1}, {1049, 651, 1}, {1453, 524, 1},
 {372, 971, 1}, {44, 757, 1}, {457, 587, 1}, {144, 508, 1}}
```

```
{{1111, 183, 1}, {869, 307, 1}, {1133, 514, 1}, {1386, 347, 1},
 {550, 994, 1}, {109, 903, 1}, {473, 630, 1}, {257, 589, 1}}
```

Od sada radimo sa tim koordinatama

```
{p1l, p2l, p3l, p4l, r1l, r2l, r3l, r4l} = leve8 ;
{p1r, p2r, p3r, p4r, r1r, r2r, r3r, r4r} = desne8;
```

```
p1l
```

```
{1217, 264, 1}
```

Dakle, to su nase piksel koordinate.

Odredjivanje fundamentalne matrice F

Radi jednostavnosti, nastavljamo bez normalizacije.
Sledeca funkcija vraca jednu jednacinu gde su nepoznate koeficenti {f11, f12, f13, f21, f22, f23, f31, f32, f33} fundamental matrice F, slozeni po vrstama .

```
jed[{a1_, a2_, a3_}, {b1_, b2_, b3_}] :=
 {a1 b1, a2 b1, a3 b1, a1 b2, a2 b2, a3 b2, a1 b3, a2 b3, a3 b3}
```

Ovo je jedna jednacina $M'2^T F M'1 = 0$
koja se dobija iz odgovarajucih leve tacke $M'1 = (a1, a2, a3)$ i desne tacke $M'2 = (b1, b2, b3)$. Sada pravimo 8 jednacina jer imamo osam odgovarajucih tacaka.

```
jed8 = MapThread[jed, {leve8 , desne8}];
```

Ovo je matrica formata 8 x9 koja predstavlja 8 jednacina dobijenih iz korespodencija

```
MatrixForm[jed8];
```

$$\begin{pmatrix} 893814 & 35454 & 933 & 31614 & 1254 & 33 & 958 & 38 & 1 \\ 1147159 & 113997 & 1027 & 147444 & 14652 & 132 & 1117 & 111 & 1 \\ 604808 & 197220 & 692 & 194902 & 63555 & 223 & 874 & 285 & 1 \\ 420665 & 129710 & 595 & 86961 & 26814 & 123 & 707 & 218 & 1 \\ 79424 & 154768 & 272 & 105120 & 204840 & 360 & 292 & 569 & 1 \\ 332640 & 418608 & 432 & 626780 & 788766 & 814 & 770 & 969 & 1 \\ 318780 & 606510 & 414 & 988680 & 1881060 & 1284 & 770 & 1465 & 1 \\ 81786 & 272706 & 258 & 259306 & 864626 & 818 & 317 & 1057 & 1 \end{pmatrix}$$

```
SVDJed8 = SingularValueDecomposition[N[jed8]];
```

DLT algoritam. Radimo SVD dekompoziciju matrice jednacina formata 8 x9

```
MatrixForm /@ SVDJed8 (* evo kako izgledaju U,D,V takve da jed8 = UDV^T *)
```

{

−0.419003	0.257689	−0.476931	−0.236008	0.0692244	0.479028	−0.444516	0.2
−0.268706	0.0442007	0.0772879	−0.419503	0.283849	0.323873	0.611581	−0.
−0.471708	−0.106402	0.525205	0.199583	0.212719	−0.0434181	−0.489172	−0.
−0.679397	0.230958	−0.110433	0.256473	−0.246791	−0.424099	0.354907	0.2
−0.208296	−0.72625	0.0637833	0.227285	−0.268823	0.464693	0.159291	0.
−0.0563907	−0.463585	−0.64487	−0.0253177	0.0275066	−0.341462	−0.10145	−0.
−0.137816	−0.274989	0.235	−0.77662	−0.188946	−0.344454	−0.145057	0.2
−0.0508465	−0.221385	−0.0581109	0.0852465	0.836517	−0.180856	0.0801211	0.

3.21554×10^{6}	0.	0.	0.	0.	0.	0.	0.	0.
0.	1.33915×10^{6}	0.	0.	0.	0.	0.	0.	0.
0.	0.	371 503.	0.	0.	0.	0.	0.	0.
0.	0.	0.	79 951.2	0.	0.	0.	0.	0.
0.	0.	0.	0.	405.001	0.	0.	0.	0.
0.	0.	0.	0.	0.	319.283	0.	0.	0.
0.	0.	0.	0.	0.	0.	6.1727	0.	0.
0.	0.	0.	0.	0.	0.	0.	3.03387	0.

,

−0.862617	0.374983	−0.338618	−0.0248156	−0.000778497	−0.00068
−0.375364	−0.263631	0.617442	0.639025	0.00332173	0.00118
−0.000738312	−0.0000841406	0.000108476	−0.00335628	0.501949	0.5813
−0.275372	−0.191614	0.545603	−0.76795	−0.00456529	−0.0020
−0.197895	−0.867853	−0.454336	−0.0352801	−0.00159532	0.000707
−0.000314758	−0.00101406	−0.000637979	−0.0027066	0.629161	−0.4771
−0.000739149	0.0000956199	−0.0000756204	−0.00413544	0.325725	0.5585
−0.000379311	−0.00089214	−0.000356792	−0.00195461	0.496053	−0.349
-7.12839×10^{-7}	-9.40721×10^{-7}	-1.04729×10^{-6}	-8.61601×10^{-6}	0.00179075	−0.0002

Koeficenti matrice {f11, f12, f13, f21, f22, f23, f31, f32, f33} su poslednja kolona matrice V (**u Pythonu je to vrsta** ako koristite **np.linalg.svd**)

```
Fvector = (Transpose[SVDJed8[[3]]])[[9]]
```

{3.94733×10^{-7}, -3.9882×10^{-6}, −0.000812281, 2.23599×10^{-6}, 6.77347×10^{-7}, −0.0105414, −0.00082096, 0.0111215, 0.999882}

```
FF = Partition[Fvector, 3]; (* od tog vektora pravimo matricu F, zovemo je FF *)
```

[[0.00000039 − 0.00000399 − 0.00081228] [0.00000224 × 0.00000068 − 0.01054141] [−0.00082096 × 0.01112154 × 0.99988192]]

```
MatrixForm[FF]
```

$$\begin{pmatrix} 3.94733\times10^{-7} & -3.9882\times10^{-6} & -0.000812281 \\ 2.23599\times10^{-6} & 6.77347\times10^{-7} & -0.0105414 \\ -0.00082096 & 0.0111215 & 0.999882 \end{pmatrix}$$

```
test[x_, y_] := y.FF.x;
(* funkcije za proveru da li je y^T F x = 0 za odgovarajuce tacke *)
```

```
MapThread[test, {leve8, desne8}]
(* proveravamo da li je y^T F x = 0  zadovoljeno za svih 8 odgovarajucih tacaka *)
```

$\{4.44089\times10^{-16}, 8.43769\times10^{-15}, -2.13163\times10^{-14}, 2.70894\times10^{-14}, -6.03961\times10^{-14}, -2.4869\times10^{-14}, -3.37508\times10^{-14}, -1.86517\times10^{-14}\}$

Vidimo da su brojevi bliski nuli.

```
Det[FF] (* determinanta od FF bi trebala biti (bliska) nuli *).
```

2.9568×10^{-13}

Odredjivanje epipolova

Odredjivanje epipolova ne treba za rekonstrukciju, pa ovo poglavlje mozete da preskocite. Da bi mogli da nadjemo epipol e1 treba da resimo sistem Fe1 = 0. To mozemo da uradimo SVD dekompozicijom matrice FF.

```
SVDFF = SingularValueDecomposition[FF];
```

```
{U, DD, V} = SVDFF;
```

```
MatrixForm /@ SVDFF
```

$$\left\{\begin{pmatrix} -0.000812275 & -0.0427481 & 0.999086 \\ -0.0105408 & -0.99903 & -0.0427543 \\ 0.999944 & -0.0105658 & 0.00036089 \end{pmatrix}, \begin{pmatrix} 1. & 0. & 0. \\ 0. & 0.000118197 & 0. \\ 0. & 0. & 2.5016\times10^{-9} \end{pmatrix}, \begin{pmatrix} -0.000820938 & 0.0543453 & 0.998522 \\ 0.0111209 & -0.99846 & 0.0543511 \\ 0.999938 & 0.0111491 & 0.000215304 \end{pmatrix}\right\}$$

```
e1 = Transpose[V][[3]] (* treca kolona matrice V je trazeni epipol. Ta
  trece kolona odgovara najmanjoj sopstvenoj vrednosti matrice *)
```

{0.998522, 0.0543511, 0.000215304}

```
e1 = (1 / e1[[3]]) e1 (*  afine koordinate  epipola e1 *)
```

{4637.72, 252.438, 1.}

Da bi nasli epipol e2, treba da resimo $F^T e2 = 0$. Medjutim,
primetimo da je $F^T = (UDV^T)^T = VDU^T$. To znaci, ako je (U, D, V) SVD dekompozicija od F,

onda je (V, D, U) SVD dekompozicija od F^T. Zato je e2 treca kolona matrice U.

```
e2 = Transpose[U][[3]]
```

{0.999086, -0.0427543, 0.00036089}

```
e2 = (1 / e2[[3]]) e2 (*  afine koordinate  epipola e2 *)
```

{2768.39, -118.469, 1.}

Imajte u vidu da su ovo koordinate u "popravljenim" piksel koordinatama.

"Popravka" fundamentalne matrice (nije obavezno)

Postizanje uslova Det (FF) = 0, Singularity constraint, poglavlje 11.1.1, Zissermann, strana 281 (nije obavezno):

```
DD1 = DiagonalMatrix[{1, 1, 0}].DD
```

{{1., 0., 0.}, {0., 0.000118197, 0.}, {0., 0., 0.}}

```
MatrixForm[DD1]
```

$$\begin{pmatrix} 1. & 0. & 0. \\ 0. & 0.000118197 & 0. \\ 0. & 0. & 0. \end{pmatrix}$$

```
FF1 = U.DD1.Transpose[V];
```

```
MatrixForm /@ {FF, FF1}
```

$$\left\{\begin{pmatrix} 3.94733\times 10^{-7} & -3.9882\times 10^{-6} & -0.000812281 \\ 2.23599\times 10^{-6} & 6.77347\times 10^{-7} & -0.0105414 \\ -0.00082096 & 0.0111215 & 0.999882 \end{pmatrix}, \begin{pmatrix} 3.92238\times 10^{-7} & -3.98834\times 10^{-6} & -0.000812281 \\ 2.2361\times 10^{-6} & 6.77353\times 10^{-7} & -0.0105414 \\ -0.00082096 & 0.0111215 & 0.999882 \end{pmatrix}\right\}$$

```
Det /@ {FF, FF1}
```

$\{2.9568\times 10^{-13}, 2.30897\times 10^{-27}\}$

Vidimo da je determinanta matrice FF1 dosta manja, sto je bolje, mada je i FF bila dosta dobra. Primetimo da FF i FF1 imaju iste matrice U i V u SVD dekompoziciji. Zato imaju i iste epipolove. Nadalje koristimo FF1, tj. "popravljenu" fundamentalnu matricu

Odredjivanje osnovne matrice E

Posto su obe fotografije slikane istom kamerom, matrice kalibracije kamere su iste K1 = K2.

```
K1 = {{1300, 0, 800}, {0, 1300, 600}, {0, 0, 1}};
```

$$K1 = \begin{pmatrix} 1300 & 0 & 800 \\ 0 & 1300 & 600 \\ 0 & 0 & 1 \end{pmatrix};$$

Uzeli smo da je dx = dy = 1300 , x0 = 800, y0 = 600.
Naime x0, y0 su polovine dimenzija fotografije (nismo kropovali, vec samo reskalirali, nakon slikanja). Zizna daljina 1300 je (otprilike) odredjena kroz kalibraciju kamere - prethodni domaci. Ta zizna daljina je u pikselima, pa zavisi od dimenzija fotografije.

Osnovna matrica je E = K2^T .F.K1, gde je F fundamentalna matrica.

```
EE = Transpose[K1].FF1.K1;
```

```
EE // MatrixForm
```

$$\begin{pmatrix} 0.662882 & -6.74029 & -3.75894 \\ 3.77901 & 1.14473 & -10.85 \\ 1.08484 & 10.8385 & -0.30483 \end{pmatrix}$$

Dekompozicija osnovne matrice E

Postoje 4 dekompozicije osnovne matrice EE (slajdovi).
Trebaju nam matrice

```
Q0 = {{0, -1, 0}, {1, 0, 0}, {0, 0, 1}};
```

```
E0 = {{0, 1, 0}, {-1, 0, 0}, {0, 0, 0}};
```

```
MatrixForm /@ {Q0, E0}
```

$$\left\{\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}\right\}$$

Prvo radimo SVD dekompoziciju matrice E

```
{U, SS, V} = SingularValueDecomposition[EE];
```

Mathematica vraca dekompoziciju tako da je EE = U.SS.Transpose[V], Python obicno E = U.SS.V. Dakle treba zamena V-- > Transpose[V]. Takodje je moguce da kolone matrice U imaju drugaciji znak.

```
MatrixForm /@ {U, SS, V}
```

$$\left\{\begin{pmatrix} -0.581302 & -0.157807 & 0.798238 \\ -0.143329 & -0.945815 & -0.291359 \\ 0.800965 & -0.283779 & 0.527186 \end{pmatrix}, \begin{pmatrix} 12.9175 & 0. & 0. \\ 0. & 12.0496 & 0. \\ 0. & 0. & 0. \end{pmatrix}, \begin{pmatrix} -0.00449473 & -0.330858 & 0.94367 \\ 0.962669 & -0.256835 & -0.085463 \\ 0.270643 & 0.908058 & 0.319661 \end{pmatrix}\right\}$$

Primetite da su sopstvene vrednosti matrice EE priblizno jednake (oko 12), a tako i mora biti. Nisu savrseno jednake zbog raznih akumuliranih gresaka (ocitavanje koordinata piksela, socivo kamere nije savrseno geoemtrijsko ...). Naravno za druge slike nece biti 12. Na kraju, citava matrica EE je do na skaliranje odredjena, tako da smo i ovde mogli drugacije dobiti.

```
Det /@ {U, V}
{1., 1.}
```

Moguce da ce Python ili C++ vratiti (u zavisnosti od biblioteke) matrice U, V koje se razlikuju do na znak, ali to mozete zanemariti. U, V, su ortogonalne pa su im determinante svakako priblizno jednake 1 ili - 1. I sledece najverovatnije mozete zanemariti :

Ukoliko su Det[U], Det[V] razlicitog znaka, uzmite - EE umesto EE, i znak ce biti isti. Ukoliko su Det[U], Det[V] obe - 1, uzmite - U, -V, umesto U, V.

Sve ovo jer mi zelimo da determinanta U da bude + 1, ali to nije sustinsko za rekonstrukciju.

Koristimo sledece resenje (vidi Skripte ili Zisserman, strana 258, Result 9.18)

```
EC = U.E0.Transpose[U];
AA = U.Transpose[Q0].Transpose[V];
```

Vi ovde koristite V a ne Transpose[V]. Verovatno koristite i Q0 umesto Transpose[Q0].

U opstem slucaju (kada budete radili svoj primer, druge fotografije), koji zavisi najvise od biblioteke koju koristite, menjajuci Q0 < --> Q0^T, EC <-> -EC dobijate 4 moguca resenja. Kada probate 3 D rekonstrukciju (vidi ispod), jedno ce da radi dobro, verovatno bas ovo.

```
MatrixForm /@ {EC, AA}
```

$$\left\{\begin{pmatrix} 0. & 0.527186 & 0.291359 \\ -0.527186 & 0. & 0.798238 \\ -0.291359 & -0.798238 & 0. \end{pmatrix}, \begin{pmatrix} 0.944893 & 0.232995 & -0.229981 \\ -0.231776 & 0.97222 & 0.032691 \\ 0.231209 & 0.0224146 & 0.972646 \end{pmatrix}\right\}$$

```
MatrixForm /@ {EC.AA, EE}
```

$$\left\{\begin{pmatrix} -0.0548245 & 0.519071 & 0.300624 \\ -0.313575 & -0.104939 & 0.897646 \\ -0.0902903 & -0.843948 & 0.0409118 \end{pmatrix}, \begin{pmatrix} 0.662882 & -6.74029 & -3.75894 \\ 3.77901 & 1.14473 & -10.85 \\ 1.08484 & 10.8385 & -0.30483 \end{pmatrix}\right\}$$

Primetimo da je EE = λ EC.AA, tj. ovo razlaganje je do na skalar.

```
(EE[[1, 1]] / (EC.AA)[[1, 1]]) EC.AA // MatrixForm
```

$$\begin{pmatrix} 0.662882 & -6.27608 & -3.63483 \\ 3.79142 & 1.26882 & -10.8534 \\ 1.0917 & 10.2042 & -0.494663 \end{pmatrix}$$

Vazi $M_C = AA.M + CC$ gde su M koordinate tacke u sistemu druge kamere,
a M_C koordinate u sistemu prve kamere. Obrnuto $M = AA^{-1} M_C - AA^{-1} CC = AA^T M_C - AA^T CC$.
Matrica AA^T govori kako je prva kamera zarotirana u sistemu druge kamere. Vektor $-AA^T CC$ predstavlja poziciju prve kamere u sistemu druge kamere.
Da bi odredili vektor CC, primetite da je matrica EC kososimetricna matrica,

tj. ona predstavlja skalarni proizvod vektorom CC. Pravimo pomocnu funkciju koja izdvaja vektor iz kososimetricne matrice.

```
koso2v[A_] := {A[[3, 2]], A[[1, 3]], A[[2, 1]]};
```

Evo trazenog vektora

```
CC = koso2v[EC]
```

```
{-0.798238, 0.291359, -0.527186}
```

Matrice kamera (u koordinatnom sistemu druge kamere)

Druga kamera je u sopstvenom koordinatnom sistemu. Zato je ona jednaka [K2 | 0], tj. odredjena samo unutrasnjom matricom kamere (sposljasna matrica kamere je trivijalna).

$$T2 = \begin{pmatrix} 1300 & 0 & 800 & 0 \\ 0 & 1300 & 600 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix};$$

Matrica koja sledi (na osnovuu prethodno ispricanog) je orijentacija prve kamere u sistemu druge kamere.

```
Transpose[AA] // MatrixForm
```

$$\begin{pmatrix} 0.944893 & -0.231776 & 0.231209 \\ 0.232995 & 0.97222 & 0.0224146 \\ -0.229981 & 0.032691 & 0.972646 \end{pmatrix}$$

Sledeci vektor je pozicija prve kamere u sistemu druge kamere.

```
CC1 = -Transpose[AA].CC
```

```
{0.94367, -0.085463, 0.319661}
```

Matrica prve kamere je $T1 = \left[K1\, AA^T \,\middle|\, K1.CC1\right]$, tj. dodajemo K1.CC1 kao cetvrtu kolonu (transponujemo, pa je dodamo kao vrstu, pa sve transponujemo)

```
T1 = Transpose [Append[Transpose[K1.Transpose[AA]], K1.CC1]];
```

```
MatrixForm[T1]
```

$$\begin{pmatrix} 1044.38 & -275.157 & 1078.69 & 1482.5 \\ 164.905 & 1283.5 & 612.727 & 80.6947 \\ -0.229981 & 0.032691 & 0.972646 & 0.319661 \end{pmatrix}$$

Sada, kada imamo 3 x4 matrice kamera, mozemo da pristupimo triangulaciji.

Triangulacija (opsti slucaj)

Ako su $M'_1 = T1\,M$, $M'_2 = T2\,M$,

piksel projekcije jedne iste tacke prostora M na levu i desnu sliku, to nam omogucava da izracunamo (4 koordinate) tacke M.

(* Ovo je Python test primer (sa casa) za proveru funkcije triangulisi. Provericemo na njemu.

```
T1=np.array([[-2,-1,0,2],[-3,0,1,0],[-1,0,0,0]])
T2=np.array([[2,-2,0,-2],[0,-3,2,-2],[0,-1,0,0]])
M1=np.array([5,3,1])
M2=np.array([-2,1,1])
print(triangulisi(T1,T2,M1,M2))
```

*)

```
T1p = {{-2, -1, 0, 2}, {-3, 0, 1, 0}, {-1, 0, 0, 0}};
T2p = {{2, -2, 0, -2}, {0, -3, 2, -2}, {0, -1, 0, 0}};
M1 = {5, 3, 1};
M2 = {-2, 1, 1};
```

Za svaku tacku dobijamo sistem od 4. jednacine sa 4 homogene nepoznate. Mogli bi uzeti i samo 3 jednacine

```
jednacine[T1_, T2_, m1_, m2_] :=
 {m1[[2]] T1[[3]] - m1[[3]] T1[[2]], -m1[[1]] T1[[3]] + m1[[3]] T1[[1]],
  m2[[2]] T2[[3]] - m2[[3]] T2[[2]], -m2[[1]] T2[[3]] + m2[[3]] T2[[1]] }
```

```
jedM = jednacine[T1p, T2p, M1, M2]
```

Iz tih jednacina, koje zavise od leve i desne projekcije tacke, se rekonstuise odgovarajuca tacka prostora. Naravno, koristimo u funkciji "jednacine" matrice kamera T1 i T2. Evo kako to izgleda na primeru prve tacke. Koristimo SVD dekompoziciju.

```
V = SingularValueDecomposition[N[jedM]][[3]]; (* N je za numericko racunanje *)
```

```
MatrixForm[V]
```

$$\begin{pmatrix} 0.41618 & 0.693901 & -0.109369 & -0.57735 \\ -0.803866 & -0.0205571 & -0.141577 & -0.57735 \\ 0.174052 & -0.254334 & -0.951326 & 0. \\ -0.387686 & 0.673344 & -0.250946 & 0.57735 \end{pmatrix}$$

Uzimamo poslednju kolonu (u Matematici, u Python-u vrsta)

```
M = Transpose[V][[4]]
```

```
{-0.57735, -0.57735, 0., 0.57735}
```

Funkcija za afine koordinate

```
UAfine[XX_] := Drop[XX / Last[XX], -1];
```

Ovo su afine koordinate trazene tacke prostora

```
UAfine[M]
```

```
{-1., -1., 0.}
```

A ovo su njene projekcije (sto se poklapa sa postavkom primera)

```
UAfine[T1p.M]
```

```
{5., 3.}
```

```
UAfine[T2p.M]
```

```
{-2., 1.}
```

Dakle, funkcija za triangulaciju jedne tacke radi.

```
triang[T1_, T2_, M1_, M2_] := Module[{jedM, V},
  jedM = jednacine[T1, T2, M1, M2];
  V = SingularValueDecomposition[N[jedM]][[3]];
  UAfine[Transpose[V][[4]]]
 ] (* Module *)
```

```
triang[T1p, T2p, M1, M2]
```

```
{-1., -1., 0.}
```

Triangulacija tacaka sa fotografije

Dakle, imamo funkciju za triangulaciju triang[T1_, T2_, M1_, M2_]. Setimo se da smo odredili matrice kamera nase scene. To su T1, T2.
 Njih smo odredili sa samo 8 tacaka. Sada cemo da rekonstruisemo 3 D koordinate svih 19 tacaka na sceni, na osnovu njihovih levih i desnih piksel koordinata.

To su svih temena kocke P1 - P8, tacku P9 na gornjoj pljosni kocke, 6 temena Q1 - Q6 "piramide" i 4 temena R1 - R4 tetraedra. Dodajemo nove tacke, a zatim rekonstruisemo sve kamerama koje su odredjene samo na osnovu onih 8. Piksel koordinate nevidljivog temena kocke smo odredili na poznati nacin.

```
p5l = piksel[{454, 507, 1}]; (* preostale tacke kocke, leva slika *)
p6l = piksel[{740, 596, 1}];
p7l = piksel[{601, 870, 1}];
p8l = piksel[{283, 761, 1}];
p9l = piksel[{427, 371, 1}];
q1l = piksel[{825, 403, 1}];  (* tacke piramide, leva slika *)
q2l = piksel[{1162, 410, 1}];
q3l = piksel[{1167, 251, 1}];
q4l = piksel[{828, 247, 1}];
q5l = piksel[{1102, 323, 1}];
q6l = piksel[{919, 322, 1}];

p5r = piksel[{537, 420, 1}];  (* isto, desna slika *)
p6r = piksel[{744, 552, 1}];
p7r = piksel[{530, 746, 1}];
p8r = piksel[{317, 587, 1}];
p9r = piksel[{479, 275, 1}];
q1r = piksel[{897, 404, 1}];
q2r = piksel[{1223, 482, 1}];
q3r = piksel[{1263, 322, 1}];
q4r = piksel[{933, 257, 1}];
q5r = piksel[{1167, 377, 1}];
q6r = piksel[{988, 337, 1}];

MatrixForm /@ {T1, T2}
```

$$\left\{\begin{pmatrix} 1044.38 & -275.157 & 1078.69 & 1482.5 \\ 164.905 & 1283.5 & 612.727 & 80.6947 \\ -0.229981 & 0.032691 & 0.972646 & 0.319661 \end{pmatrix}, \begin{pmatrix} 1300 & 0 & 800 & 0 \\ 0 & 1300 & 600 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}\right\}$$

```
leve19 = {p1l, p2l, p3l, p4l, p5l, p6l, p7l,
   p8l, p9l, q1l, q2l, q3l, q4l, q5l, q6l, r1l, r2l, r3l, r4l};
desne19 = {p1r, p2r, p3r, p4r, p5r, p6r, p7r, p8r, p9r, q1r,
   q2r, q3r, q4r, q5r, q6r, r1r, r2r, r3r, r4r};
```

Primenjujemo funkciju za triangulaciju na sve parove leva - desna tacka

```
tacke3D = MapThread[triang[T1, T2, #1 , #2] &, {leve19, desne19}]
{{-0.83564, 1.12381, -3.49506},
 {-0.17348, 0.753889, -3.3052}, {-0.71606, 0.209322, -2.82695},
 {-1.31209, 0.584777, -2.92207}, {-0.82496, 0.602838, -4.12409},
 {-0.15778, 0.185678, -3.96299}, {-0.709326, -0.347049, -3.52554},
 {-1.31419, 0.0763032, -3.58449}, {-0.800943, 0.822163, -3.25221},
 {0.338922, 0.702702, -4.43113}, {1.43763, 0.44055, -4.38677},
 {1.53631, 0.946502, -4.30036}, {0.451138, 1.17543, -4.37389},
 {1.18291, 0.743683, -4.16952}, {0.60785, 0.864715, -4.17487},
 {0.670238, -0.970561, -3.36304}, {1.93191, -0.794925, -3.60269},
 {1.00819, -0.0587544, -3.96688}, {1.22909, 0.0444798, -2.92958}}
```

Da bi prikazali objekte povezacemo linijama odredjena rekonstruisana 3 D temena. Slede parovi temena koja odredjuju ivice.

```
ivice = {{1, 2}, {1, 4}, {1, 5}, {3, 4}, {3, 2}, {3, 7}, {6, 7}, {6, 5}, {6, 2}, {8, 7},
   {8, 4}, {8, 5}, {10, 11}, {10, 13}, {10, 15}, {12, 13}, {12, 11}, {12, 14}, {14, 15},
   {11, 14}, {15, 13}, {16, 17}, {16, 18}, {16, 19}, {17, 18}, {17, 19}, {18, 19}};

Show[Graphics3D[{Blue, Thickness[0.005], Line[Part[tacke3D, #]]} & /@ivice],
 Boxed → False]
```