```
NormalizeCoordinates[PC1_, PC2_] :=
  Module[{CentroidPC1, CentroidPC2(*,projectedPointsK1,projectedPointsK2*)},
  Modul

    Print["PC1 = ", MatrixForm[PC1]];
    gib aus           Matritzenform
    Print["PC2 = ", MatrixForm[PC2]];
    gib aus           Matritzenform

    (*Normalize Image coordinates____________________________________ *)
      normalisiere Bild

    CentroidPC1 = Mean[PC1];
                  arithmetisches Mittel
    CentroidPC2 = Mean[PC2];
                  arithmetisches Mittel
    Print["CemtroidPC1 = ", N[CentroidPC1]];
    gib aus                 numerischer Wert
    Print["CemtroidPC2 = ", N[CentroidPC2]];
    gib aus                 numerischer Wert

    (*move centroid and Points to origin*)
    IntermediatePC1 = Map[# - CentroidPC1[[1 ;; 2]] &, PC1[[All, 1 ;; 2]]];
                      wende an                              alle
    IntermediatePC2  = Map[# - CentroidPC2[[1 ;; 2]] &, PC2[[All, 1 ;; 2]]];
                       wende an                             alle
    Print["Mean[IntermediatePC1] = ", N[Mean[IntermediatePC1]]];
    gib aus arithmetisches Mittel     ·· arithmetisches Mittel
    Print["Mean[IntermediatePC2] = ", N[ Mean[IntermediatePC2]]];
    gib aus arithmetisches Mittel     ··· arithmetisches Mittel
    (*Get Men Distance*)
      erhalte
    DistanceVectorPC1 = Map[{0, 0} - # &, IntermediatePC1];
                        wende an
    DistancesPC1  = Map[Sqrt[#[[1]]^2 + #[[2]]^2] &, DistanceVectorPC1];
                    w··· Quadratwurzel
    MeanDistPC1  = Mean[DistancesPC1];
                   arithmetisches Mittel
    Print["MeanDistPC1 = ", N[MeanDistPC1]];
    gib aus                 numerischer Wert
    DistanceVectorPC2 = Map[{0, 0} - # &, IntermediatePC2];
                        wende an
    DistancesPC2  = Map[Sqrt[#[[1]]^2 + #[[2]]^2] &, DistanceVectorPC2];
                    w··· Quadratwurzel
    MeanDistPC2  = Mean[DistancesPC2];
                   arithmetisches Mittel
    Print["MeanDistPC2 = ", N[MeanDistPC2]];
    gib aus                 numerischer Wert

    (*Create Matrix out of Results
     to Move and Scale all Points and get RMS to Sqrt(2)*)
                 skaliere                         Quadratwurzel

    T = {{Sqrt[2] / MeanDistPC1, 0, (Sqrt[2] / MeanDistPC1) * -CentroidPC1[[1]]}, {0,
          Quadratwurzel              Quadratwurzel
```

```
  Sqrt[2] / MeanDistPC1, (Sqrt[2] / MeanDistPC1) * -CentroidPC1[[2]]}, {0, 0, 1}};
  Quadratwurzel            Quadratwurzel

Print[MatrixForm[N[T]]];
gib aus Matritzenform numerischer Wert
PC1Norm = Map[T.# &, PC1];
          wende an
PC1Norm = Map[{#[[1]], #[[2]]} &, PC1Norm];
          wende an
Print["Matrix normalization PC1= ", N[PC1Norm]];
gib aus                             numerischer Wert
Print["RMS distance=", N[Mean[Map[Norm[#] &, PC1Norm]]]];
gib aus                 ·· arit··· w··· Norm

TPrime = {{Sqrt[2] / MeanDistPC2, 0, (Sqrt[2] / MeanDistPC2) * -CentroidPC2[[1]]}, {0,
           Quadratwurzel            Quadratwurzel
  Sqrt[2] / MeanDistPC2, (Sqrt[2] / MeanDistPC2) * -CentroidPC2[[2]]}, {0, 0, 1}};
  Quadratwurzel            Quadratwurzel

Print[MatrixForm[N[TPrime]]];
gib aus Matritzenform numerischer Wert

PC2Norm = Map[TPrime.# &, PC2];
          wende an
PC2Norm = Map[{#[[1]], #[[2]]} &, PC2Norm];
          wende an
Print["Matrix normalization PC2 = ", N[PC2Norm]];
gib aus                              numerischer Wert
Print["RMS distance=", N[Mean[Map[Norm[#] &, PC2Norm]]]];
gib aus                 ·· arit··· w··· Norm

 (*Compute Fundamentalmatrix with normalized pixel points PC1Norm and PC2Norm*)
 FundamentalMtxBestimmung[PC1, PC2, PC1Norm, PC2Norm, T, TPrime(*,K1,K2*)];
];


(*Compute Fundamentalmatrix with normalized-8-Point_Algorithm________*)

FundamentalMtxBestimmung[PC1_, PC2_, PC1Norm_, PC2Norm_, T_, TPrime_(*,K1_,K2_*)] :=
 Module[{CoefficientMatx = {}, UDVErgebnis, LängeV , FinF11 , FinF12 ,
 Modul
   FinF13 , FinF21 , FinF22 , FinF23 , FinF31 , FinF32, FinF33, FFin , SVDF},

  Print["
  gib aus

Begin Computation of
beginne Kontext
        Fundamentalmatrix____________________________________________________

"];
```

```
For[qq = 1, qq ≤ Length[PC1Norm], qq++,
For−Schleife      Länge
 AppendTo[CoefficientMatx, {PC2Norm[[qq, 1]] * PC1Norm[[qq, 1]],
 hänge an bei
    PC2Norm[[qq, 1]] * PC1Norm[[qq, 2]], PC2Norm[[qq, 1]],
    PC2Norm[[qq, 2]] * PC1Norm[[qq, 1]], PC2Norm[[qq, 2]] * PC1Norm[[qq, 2]],
    PC2Norm[[qq, 2]], PC1Norm[[qq, 1]], PC1Norm[[qq, 2]], 1}];
];
Print["CoefficientMtx = ", MatrixForm[N[CoefficientMatx]]];
gib aus                    Matritzenform numerischer Wert
UDVErgebnis = SingularValueDecomposition[N[CoefficientMatx]];
              Singulärwertzerlegung     numerischer Wert
Print["Rank CoefficientMatx = ", MatrixRank[CoefficientMatx]];
gib aus                          Rang der Matrix

LängeV = Length[UDVErgebnis[[3]]];
         Länge
Print["LängeV ", LängeV];
gib aus
FinF11 = UDVErgebnis[[3, 1, LängeV]];
FinF12 = UDVErgebnis[[3, 2, LängeV]];
FinF13 = UDVErgebnis[[3, 3, LängeV]];
FinF21 = UDVErgebnis[[3, 4, LängeV]];
FinF22 = UDVErgebnis[[3, 5, LängeV]];
FinF23 = UDVErgebnis[[3, 6, LängeV]];
FinF31 = UDVErgebnis[[3, 7, LängeV]];
FinF32 = UDVErgebnis[[3, 8, LängeV]];
FinF33 = UDVErgebnis[[3, 9, LängeV]];

FFin =
 {{FinF11, FinF12, FinF13}, {FinF21, FinF22, FinF23}, {FinF31, FinF32, FinF33}};

Print["FFin = ", MatrixForm[N[Chop[FFin]]]];
gib aus          Matritzenform ·· ersetze kleine Zahlen mit 0
Print["FFin Rank = ", MatrixRank[FFin]];
                      Rang der Matrix

PC2NormTest = Map[{#[[1]], #[[2]], 1} &, PC2Norm];
              wende an
PC1NormTest = Map[{#[[1]], #[[2]], 1} &, PC1Norm];
              wende an
Print["PC2NormTest = ", N[PC2NormTest]];
                        numerischer Wert
Print["PC1NormTest = ", N[PC1NormTest]];
gib aus                 numerischer Wert
l = {};
lPrime = {};

AppendTo[l, Map[FFin.# &, PC1NormTest]];
hänge an bei wende an
AppendTo[lPrime, Map[Transpose[FFin].# &, PC2NormTest]];
hänge an bei     w··· transponiere
```

```
Print[
gib aus
  Show[ContourPlot[l.{x, y, 1} == 0, {x, -50, 50}, {y, -10, 10}], PlotRange -> All]];
        Konturgraphik                                          Koordinatenbe··· alle

Print[Show[
gib ··· zeige an
   ContourPlot[lPrime.{x, y, 1} == 0, {x, -50, 50}, {y, -10, 5}], PlotRange -> All]];
   Konturgraphik                                               Koordinatenbe··· alle

SVDF = SingularValueDecomposition[FFin];
       Singulärwertzerlegung
SVDF[[2]] = {SVDF[[2, 1]], SVDF[[2, 2]], {0, 0, 0}};

FPrime = SVDF[[1]].SVDF[[2]].Transpose[SVDF[[3]]];
                            transponiere
Print["FPrime MatrixRank = ", MatrixRank[FPrime]];
gib aus          Rang der Matrix  Rang der Matrix
Print["FPrime = ", MatrixForm[N[FPrime]]];
gib aus            Matritzenform numerischer Wert
Print[MatrixRank[FPrime]];
gib aus Rang der Matrix

lRank2 = {};
lPrimeRank2 = {};

AppendTo[lRank2, Map[FPrime.# &, PC1NormTest]];
hänge an bei     wende an
AppendTo[lPrimeRank2, Map[Transpose[FPrime].# &, PC2NormTest]];
                      w··· transponiere



Print[Show[
gib aus zeige an
   ContourPlot[lRank2.{x, y, 1} == 0, {x, -50, 50}, {y, -10, 10}], PlotRange -> All]];
   Konturgraphik                                                 Koordinatenbe··· alle

Print[Show[ContourPlot[lPrimeRank2.{x, y, 1} == 0,
gib aus zeig··· Konturgraphik
    {x, -50, 20}, {y, -10, 4}], PlotRange -> All]];
                                Koordinatenbe··· alle


(*Denormalize F*)
F = Transpose[TPrime].FPrime.T;
    transponiere
Print["Demnormalized FPrime -> F =", MatrixForm[N[F]]];
gib aus                              Matritzenform numerischer Wert
Print["F Rank = ", MatrixRank[F]];
gib aus            Rang der Matrix
For[tt = 1, tt ≤ Length[PC1], tt++,
For-Schleife      Länge
 Print[N[PC2[[tt]].F.PC1[[tt]]]];
 gib aus numerischer Wert
```

```
    gib aus  numerischer Wert
    ];

    lRank2De = {};
    lPrimeRank2De = {};

    AppendTo[lRank2De, Map[F.# &, PC1]];
    hänge an bei       wende an
    AppendTo[lPrimeRank2De, Map[Transpose[F].# &, PC2]];
                               w···  transponiere

    Print[Show[ContourPlot[lRank2De.{x, y, 1} == 0,
    gib ···  zeig··· Konturgraphik
        {x, -4000, -2000}, {y, -1280, 1280}], PlotRange -> All]];
                                              Koordinatenbe··· alle
    Print[Show[ContourPlot[lPrimeRank2.{x, y, 1} == 0, {x, -2000, 3000},
              zeig··· Konturgraphik
        {y, -3000, 600}], PlotRange -> All]];
                          Koordinatenbe··· alle



    epipole = Flatten[NullSpace[F]];
              ebne ein  Nullraum
    epipolePrime = Flatten[NullSpace[Transpose[F]]];
                   ebne ein  Nullraum   transponiere
    testEpipolPrime = Transpose[F].epipolePrime;
                      transponiere
    testEpipol = F.epipole;

    Print["epipole =", Chop[epipole]];
    gib aus            ersetze kleine Zahlen mit 0
    Print["epipolePrime = ", Chop[epipolePrime]];
    gib aus                  ersetze kleine Zahlen mit 0
    Print["Test F^T.e' = ", Chop[testEpipolPrime]];
    gib aus                 ersetze kleine Zahlen mit 0
    Print["Test F.e = ", Chop[testEpipol]];
    gib aus              ersetze kleine Zahlen mit 0

    NewRectification[F, epipole, epipolePrime, images];
    Print["
    gib aus

End Computation of
beende Kontext
      Fundamentalmatrix____________________________________________________


"];


    (*EssentialMtxAlgorithm[F,PC1Norm,
      PC2Norm,T,TPrime,projectedPointsK1,projectedPointsK2];*)
    EssentialMtx[F, PC1, PC2];
```

```
  ];

(*1. Method: Compute essential Matrix with 8-Pint-Algorithm*)
     Methode

EssentialMtxAlgorithm[F_, PC1Norm_, PC2Norm_,
   T_, TPrime_, projectedPointsK1_, projectedPointsK2_] :=
  Module[{normPC1, normPC2, K1, K2, UDVErgebnis, LängeV , FinE11 , FinE12 , FinE13 ,
  Modul
    FinE21 , FinE22 , FinE23 , FinE31 , FinE32, FinE33, EFin, SVDE, CoefficientMtx = {}},

   Print["
   gib aus

Begin Computation of Essential Matrix
beginne Kontext
      with 8-Point-Algorithm______________________________
           Punkt


"];

   (*Different Cameramatrices from different set ups*)
   (*K1={{17.3158028,0,6.146589863},{0,17.31981867,4.600615527},{0,0,1}};
   K2={{18.60732121,0,4.145650968},{0,18.58796099,3.22706539},{0,0,1}};*)

   K1 = {{18.530, 0, 6.094}, {0, 18.537, 3.994}, {0, 0, 1}};
   K2 = K1;

   (*px to mm for both imagePoints*)

   (*PC1mm = Map[(#*6.5)/1000 &,PC1];
             wende an
   PC2mm = Map[(#*4.29)/1000 &,PC2];*)
           wende an

   PC1mm = Map[(# * 6.5) / 1000 &, PC1];
           wende an
   PC2mm = Map[(# * 4.29) / 1000 &, PC2];
           wende an

   (*PC1mm = Map[{#[[1]],#[[2]],1}&,PC1mm];
             wende an
   PC2mm = Map[{#[[1]],#[[2]],1}&,PC2mm];*)
           wende an

   Print["PC1 in mm = ", MatrixForm[N[PC1mm]]];
   gib aus                Matritzenform numerischer Wert
   Print["PC2 in mm = ", MatrixForm[N[PC2mm]]];
   gib aus                Matritzenform numerischer Wert

   normPC1 = Map[Inverse[K1].# &, PC1mm];
             w··· inverse Matrix
```

```
              w···  inverse Matrix
normPC2 = Map[Inverse[K2].# &, PC2mm];
              w···  inverse Matrix
normPC1 = Map[{#[[1]], #[[2]], 1} &, normPC1 ];
          wende an
normPC2 = Map[{#[[1]], #[[2]], 1} &, normPC2 ];
          wende an

Print["normalized Coordinates K1 = ", MatrixForm[N[normPC1]]];
gib aus                                Matritzenform numerischer Wert
Print["normalized Coordinates K2 = ", MatrixForm[N[normPC2]]];
gib aus                                Matritzenform numerischer Wert

For[qq = 1, qq ≤ Length[normPC1], qq++,
For−Schleife         Länge
 AppendTo[CoefficientMtx, {normPC2[[qq, 1]] * normPC1[[qq, 1]],
 hänge an bei
    normPC2[[qq, 1]] * normPC1[[qq, 2]], normPC2[[qq, 1]],
    normPC2[[qq, 2]] * normPC1[[qq, 1]], normPC2[[qq, 2]] * normPC1[[qq, 2]],
    normPC2[[qq, 2]], normPC1[[qq, 1]], normPC1[[qq, 2]], 1}];
];
Print["CoefficientMtx = ", MatrixForm[N[CoefficientMtx]]];
gib aus                    Matritzenform numerischer Wert
UDVErgebnis = SingularValueDecomposition[N[CoefficientMtx]];
              Singulärwertzerlegung      numerischer Wert
Print["Rank CoefficientMatx = ", MatrixRank[CoefficientMtx]];
gib aus                          Rang der Matrix

LängeV = Length[UDVErgebnis[[3]]];
         Länge
Print[LängeV];
gib aus
FinE11 = UDVErgebnis[[3, 1, LängeV]];
FinE12 = UDVErgebnis[[3, 2, LängeV]];
FinE13 = UDVErgebnis[[3, 3, LängeV]];
FinE21 = UDVErgebnis[[3, 4, LängeV]];
FinE22 = UDVErgebnis[[3, 5, LängeV]];
FinE23 = UDVErgebnis[[3, 6, LängeV]];
FinE31 = UDVErgebnis[[3, 7, LängeV]];
FinE32 = UDVErgebnis[[3, 8, LängeV]];
FinE33 = UDVErgebnis[[3, 9, LängeV]];

EFin =
 {{FinE11, FinE12, FinE13}, {FinE21, FinE22, FinE23}, {FinE31, FinE32, FinE33}};
Print["EFin = ", N[Chop[EFin]]];
gib aus          ·· ersetze kleine Zahlen mit 0
Print["EFin = ", MatrixRank[EFin]];
gib aus          Rang der Matrix

SVDE = SingularValueDecomposition[EFin];
       Singulärwertzerlegung
Print["SVDE = ", SVDE];
gib aus
SVD = SingularValueDecomposition[EFin];
      Singulärwertzerlegung
```

```
        ⌊Singulärwertzerlegung
    Print["SVD of EMtx =", SVD];
    ⌊gib aus
    SVD[[2]] = DiagonalMatrix[
                ⌊Diagonalmatrix
      {(SVD[[2, 1, 1]] + SVD[[2, 2, 2]]) / 2, (SVD[[2, 1, 1]] + SVD[[2, 2, 2]]) / 2, 0}];
    Print["new SVD = ", SVD];
    ⌊gib aus

    EPrime = SVD[[1]].SVD[[2]].Transpose[SVD[[3]]];
                               ⌊transponiere
    Print["EPrime = ", EPrime];
    ⌊gib aus
    Print["EPrime = ", MatrixRank[EPrime]];
    ⌊gib aus            ⌊Rang der Matrix

    Print["
    ⌊gib aus

End Computation of Essential Matrix
⌊beende Kontext
      with 8-Point-Algorithm______________________________
             ⌊Punkt


"];
    ExtractRotationAndTranslation[EFin, PC1, PC2];

  ];


(*2. Method: Compute essential matrix with Fundamentalmatrix*)
     ⌊Methode
EssentialMtx[F_, PC1_, PC2_] :=
  Module[{K1, K2, EsMtx, SVD, normPC1, normPC2(*,PC1mm,PC2mm*)},
  ⌊Modul

    Print["
    ⌊gib aus

Begin Computation of Essential Matrix
⌊beginne Kontext
      with K1 and K2______________________________________


"];

    (*Different Cameramatrices from different set ups*)
    (*60 D and 6D First Try*)
       ⌊leite ab ⌊·· ⌊erstes Element
    (*K1={{17.3158028,0,6.146589863},{0,17.31981867,4.600615527},{0,0,1}};
    K2={{18.60732121,0,4.145650968},{0,18.58796099,3.22706539},{0,0,1}};*)
    (*K1={{2663.969662, 0, 945.6292097},{0, 2664.587487, 707.7870042},{0, 0, 1}};
    K2={{4337.370912, 0, 966.352207},{0,4332.858039, 752.2296946},{0, 0, 1}};*)
```

```
(*same Cameras 6D*)
                     leite ab
(*K1={{18.530,0,6.094},{0,18.537,3.994},{0,0,1}};
K2=K1;
K1={{2850.913,0,937.6861},{0,2851.852,614.5411},{0,0,1}};
K2=K1;*)

(*60D and 6D with Mathematic
   leite ab  leite ab
 Correspondong detection different Cameras same resolution*)
(*K1={{18.17617,0,6.1583},{0,18.18014,4.568642},{0,0,1}};
K2={{19.146558,0,4.229572},{0,19.00778,3.125653},{0,0,1}};*)
K1 = {{4436.0715295011, 0, 985.9143}, {0, 4430.717, 728.5904}, {0, 0, 1}};
K2 = A.{{2796.335, 0, 947.4308}, {0, 2796.944, 702.8681}, {0, 0, 1}};

(*60D and 6D with Mathematic
   leite ab  leite ab
 Correspondong detection and different Camera resolutions*)
(*K1={{2825.059,0,927.4028},{0,2809.878,599.5711},{0,0,1}};
K2={{6423.957,0,1108.976},{0,6394.091,657.8209},{0,0,1}};*)

Print["PC1 = ", PC1];
gib aus
PC1mm = PC1;

PC2mm = PC2;

Print["PC1mm = ", MatrixForm[PC1mm]];
gib aus              Matritzenform
Print["PC2mm = ", MatrixForm[PC2mm]];
gib aus              Matritzenform
normPC1 = Map[Inverse[K1].# &, PC1mm];
          w···  inverse Matrix
normPC2 =  Map[Inverse[K2].# &, PC2mm];
           w···  inverse Matrix
normPC1 = Map[{#[[1]], #[[2]], 1} &, normPC1 ];
          wende an
normPC2 =  Map[{#[[1]], #[[2]], 1} &, normPC2 ];
           wende an

Print["normalized Coordinates K1 = ", MatrixForm[N[normPC1]]];
gib aus                                Matritzenform  numerischer Wert
Print["normalized Coordinates K2 = ", MatrixForm[N[normPC2]]];
gib aus                                Matritzenform  numerischer Wert

EsMtx = Transpose[K2].F.K1;
        transponiere
Print["EsMtx = ", EsMtx];
gib aus
Print[MatrixRank[EsMtx]];
gib aus Rang der Matrix

SVD = SingularValueDecomposition[EsMtx];
      Singulärwertzerlegung
```

```
    Print["SVD of EMtx =", SVD];
    gib aus
    SVD[[2]] = DiagonalMatrix[
                Diagonalmatrix
       {(SVD[[2, 1, 1]] + SVD[[2, 2, 2]]) / 2, (SVD[[2, 1, 1]] + SVD[[2, 2, 2]]) / 2, 0}];

    Print["new SVD = ", SVD];
    gib aus

    EPrime = SVD[[1]].SVD[[2]].Transpose[SVD[[3]]];
                               transponiere
    Print["EPrime = ", EPrime];
    gib aus
    Print[MatrixRank[EPrime]];
    gib aus Rang der Matrix

    For[zz = 1, zz ≤ Length[normPC1], zz++,
    For–Schleife     Länge
     Print[normPC2[[zz]].EPrime.normPC1[[zz]]];
     gib aus
    ];

    Print["
    gib aus

End Computation of Essential Matrix
beende Kontext
      with K1 and K2___________________________________


"];


    Print["
    gib aus

Begin Computation of extern Cameraparameters
beginne Kontext
      with K1 and K2___________________________________


"];


    ExtractRotationAndTranslation[EPrime, F, PC1mm , PC2mm , K1, K2];

   ];

(*Compute extrinsic Cameraparameters______________________________*)


ExtractRotationAndTranslation[EPrime_, F_, PC1mm_, PC2mm_, K1_, K2_ ] :=
  Module[{W, Z, U, V, Sigma, S1, S2, R1, R2, i, t, P21, P22, P23, P24, NewE},
  Modul
```

```
 Modul
    Print["
    gib aus

Begin Reconstruction of Rotation and
beginne Kontext
        Translation_____________________________________________________


"];


    Print["E = ", MatrixForm[EPrime]];
    gib aus   Expo···  Matritzenform

    {U, Sigma, V} = SingularValueDecomposition[EPrime];
                    Singulärwertzerlegung
    Print["U of E = ", U];
    gib aus         Exponentialkonstante E
    Print["Sigma of E = ", Sigma];
    gib aus             Exponentialkonstante E
    Print["V of E = ", V];
    gib aus         Exponentialkonstante E

    Sigma = {{1, 0, 0}, {0, 1, 0}, {0, 0, 0}};


    ETest1 = U.Sigma.Transpose[V];
                     transponiere
    Print["ETest1 = ", ETest1];
    gib aus
    {U, Sigma, V} = SingularValueDecomposition[ETest1];
                    Singulärwertzerlegung
    Print["U of E = ", U];
    gib aus         Exponentialkonstante E
    Print["Sigma of E = ", Sigma];
    gib aus             Exponentialkonstante E
    Print["V of E = ", V];
    gib aus         Exponentialkonstante E



    If[Det[U] == -1,
    ···  Determinante
     U = U * -1;
    ];

    If[Det[V] == -1,
    ···  Determinante
     V = V * -1;
    ];

    ETest2 = U.Sigma.Transpose[V];
                     transponiere
    Print["ETest2 = ", ETest2];
    gib aus
```

```
gib aus

W = {{0, -1, 0}, {1, 0, 0}, {0, 0, 1}};
Z = {{0, 1, 0}, {-1, 0, 0}, {0, 0, 0}};

S1 = -U.Z.Transpose[U];
              transponiere
S2 = U.Z.Transpose[U];
            transponiere
R2 = U.Transpose[W].Transpose[V];
         transponiere   transponiere
R1 = U.W.Transpose[V];
            transponiere
Print["S1 = ", MatrixForm[S1]];
gib aus        Matritzenform
Print["S2 = ", MatrixForm[S2]];
gib aus        Matritzenform
Print["R1 = ", MatrixForm[R1]];
gib aus        Matritzenform
Print["R2 = ", MatrixForm[R2]];
gib aus        Matritzenform

Print[Det[U]];
gib aus Determinante
Print[Det[V]];
gib aus Determinante
Print["Det E =", Det[EPrime]];
gib aus  De·· Exp··· Determinante
Print["Det F =", Det[F]];
gib aus  Determina··· Determinante

Print["Check if t of S1, S2 is equal = ", Map[NullSpace[#] &, {S1, S2}]];
gib aus  prüfe                            w··· Nullraum
Print["R1 is Rotation = ", Chop[Transpose[R1].R1]];
gib aus                    erse·· transponiere
Print["R2 is Rotation =", Chop[Transpose[R2].R2]];
gib aus                   erse·· transponiere

Print["Determinant of R1 = ", Det[R1]];
gib aus                       Determinante
Print["Determinant of R2 = ", Det[R2]];
gib aus                       Determinante

LeftEprime = NullSpace[Transpose[EPrime]];
             Nullraum  transponiere
t = Flatten[NullSpace[S1]];
    ebne ein Nullraum

(*t={U[[3,1]],U[[3,2]],U[[3,3]]};*)

Print["t =", t];
gib aus
LeftS2 = NullSpace[Transpose[S2]];
         Nullraum  transponiere
```

```
Print["Left E und S = ", LeftEprime, " , ", LeftS2];
gib aus   links   Exponentialkonstante E

(*P1=R1;
P2=R1;
P3=R2;
P4=R2;*)

P1 = U.W.Transpose[V];
         transponiere
P2 = U.W.Transpose[V];
         transponiere
P3 = U.Transpose[W].Transpose[V];
       transponiere   transponiere
P4 = U.Transpose[W].Transpose[V];
       transponiere   transponiere

P1 = {{P1[[1, 1]], P1[[1, 2]], P1[[1, 3]], 1*t[[1]]}, {P1[[2, 1]], P1[[2, 2]],
    P1[[2, 3]], 1*t[[2]]}, {P1[[3, 1]], P1[[3, 2]], P1[[3, 3]], 1*t[[3]]}};
P2 = {{P2[[1, 1]], P2[[1, 2]], P2[[1, 3]], -1*t[[1]]}, {P2[[2, 1]], P2[[2, 2]],
    P2[[2, 3]], -1*t[[2]]}, {P2[[3, 1]], P2[[3, 2]], P2[[3, 3]], -1*t[[3]]}};
P3 = {{P3[[1, 1]], P3[[1, 2]], P3[[1, 3]], 1*t[[1]]}, {P3[[2, 1]], P3[[2, 2]],
    P3[[2, 3]], 1*t[[2]]}, {P3[[3, 1]], P3[[3, 2]], P3[[3, 3]], 1*t[[3]]}};
P4 = {{P4 [[1, 1]], P4 [[1, 2]], P4 [[1, 3]], -1*t[[1]]}, {P4 [[2, 1]], P4 [[2, 2]],
    P4 [[2, 3]], -1*t[[2]]}, {P4 [[3, 1]], P4 [[3, 2]], P4 [[3, 3]], -1*t[[3]]}};

Print["T1 = ", MatrixForm[P1]];
gib aus         Matritzenform
Print["T2 = ", MatrixForm[P2]];
gib aus        Matritzenform
Print["T3 = ", MatrixForm[P3]];
gib aus         Matritzenform
Print["T4 = ", MatrixForm[P4]];
gib aus         Matritzenform

(*R2=R2*-1;
Test = {{0,-t[[3]],t[[2]]},{t[[3]],0,-t[[1]]},{-t[[2]],t[[1]],0}}.R2;
Test = S2.R2;
Print[MatrixForm[Test]];*)
gib aus Matritzenform

PList = {};

AppendTo[PList, P1];
hänge an bei
AppendTo[PList, P2];
hänge an bei
AppendTo[PList, P3];
hänge an bei
AppendTo[PList, P4];
hänge an bei

Print["PList = ", PList];
gib aus
```

```
    gib aus
    Print["Length PList = ", Length[PList]];
    gib aus  Länge               Länge

    Print["
    gib aus

End Reconstruction of Rotation and
beende Kontext
      Translation_______________________________________________


"];

    For[uu = 1, uu ≤ Length[PList], uu++,
    For-Schleife      Länge
     RecMtx = PList[[uu]];

     RForOK2 = {{RecMtx[[1, 1]], RecMtx[[1, 2]], RecMtx[[1, 3]]},
       {RecMtx[[2, 1]], RecMtx[[2, 2]], RecMtx[[2, 3]]},
       {RecMtx[[3, 1]], RecMtx[[3, 2]], RecMtx[[3, 3]]}};
     RForOK2 = Transpose[RForOK2];
               transponiere

     tForOK2 = {RecMtx[[1, 4]], RecMtx[[2, 4]], RecMtx[[3, 4]]};

     StructureComputation[F, PList[[uu]], PC1, PC2, K1, K2, RForOK2, tForOK2];
    ];

   ];
```