```
(*Computation essential Matrix (2 different ways) and extrinsic Cameraparameters*)
(*_______________________________________________________________________*)


(*1 Method:
    Methode
  Compute essential matrix with 8-Point-Algorithm and normalized Image coordinates*)
                                  Punkt                        Bild
ComputeEssentialMtx[PC1_, PC2_] :=
  Module[{EMtx, normC1, normC2, K1, K2, n, CoefficientMtx, SVDE},
  Modul

   Print["Begin Computing essential
   gib aus  beginne Kontext
      Matrix________________________________________________________"];
   Print["PC1 = ", PC1];
   gib aus
   K1 = {{zeta1, 0, 0}, {0, zeta1, 0}, {0, 0, 1}};
   K2 = AK.{{zeta2, 0, 0}, {0, zeta2, 0}, {0, 0, 1}};

   normC1 = Map[Inverse[K1].# &, PC1];
            w⋯  inverse Matrix
   normC2 =  Map[Inverse[K2].# &, PC2];
             w⋯  inverse Matrix

   Print["normalized Coordinates K1 = ", MatrixForm[normC1]];
   gib aus                               Matritzenform
   Print["normalized Coordinates K2 = ", MatrixForm[N[normC2]]];
   gib aus                               Matritzenform numerischer Wert

   CoefficientMtx = ConstantArray[0, {9, 9}];
                    konstantes Array
   For[i = 1, i ≤ 9, i++,
   For–Schleife
    CoefficientMtx[[i]] =
      {normC2[[i, 1]] * normC1[[i, 1]], normC2[[i, 1]] * normC1[[i, 2]], normC2[[i, 1]],
       normC2[[i, 2]] * normC1[[i, 1]], normC2[[i, 2]] * normC1[[i, 2]],
       normC2[[i, 2]], normC1[[i, 1]], normC1[[i, 2]], 1};
   ];
   Print["CoefficientMtx = ", MatrixForm[N[CoefficientMtx]]];
   gib aus                    Matritzenform numerischer Wert
   n = NullSpace[N[CoefficientMtx]];
       Nullraum  numerischer Wert
   Print["ns =", n];
   gib aus
   EMtx = {{n[[1, 1]], n[[1, 2]], n[[1, 3]]},
     {n[[1, 4]], n[[1, 5]], n[[1, 6]]}, {n[[1, 7]], n[[1, 8]], n[[1, 9]]}};
   Print["EMtx = ", MatrixForm[Simplify[EMtx]]];
   gib aus          Matritzenform vereinfache

   Print["SVD E = ", SingularValueDecomposition[EMtx]];
   gib aus      Expo⋯ Singulärwertzerlegung
   EMtx = EMtx / EMtx[[2, 3]];
   Print["EMtx = ", MatrixForm[Simplify[EMtx]]];
   gib aus          Matritzenform vereinfache
```

```
  gib aus                    Matritzenform  vereinfache

    Print["EigenValue for testing", Eigenvalues[EMtx.Transpose[EMtx]]];
    gib aus                                  Eigenwerte              transponiere
    ComputingCamerasFromE[EMtx, PC1, PC2];
  ];


(*2.Method: Compute essential matrix from Fundamentalmatrix_____*)
   Methode
ComputeEssentialMtxFromFormular[F_, PC1_, PC2_] := Module[{K1, K2, EMtx},
                                                   Modul

    Print["Begin Computing essential
    gib aus  beginne Kontext
       Matrix_____________________________________________________"];
    (*Compute EssentialMatrix with EMtx = Transpose[K2].F.K1;*)
                                          transponiere
    K1 = {{zeta1, 0, 0}, {0, zeta1, 0}, {0, 0, 1}};
    K2 = AK.{{zeta2, 0, 0}, {0, zeta2, 0}, {0, 0, 1}};

    EMtx = Transpose[K2].F.K1;
           transponiere
    Print["EMtx = ", MatrixForm[N[EMtx]]];
    gib aus              Matritzenform  numerischer Wert
    ComputingCamerasFromE[EMtx, PC1, PC2, F, K1, K2];
  ];



(*Compute extrinsic Cameraparameters___________________________*)
ComputingCamerasFromE[EMtx_, PC1_, PC2_, F_, K1_, K2_] :=
  Module[{W, Z, U, V, Sigma, S1, S2, R1, R2, i, t, P21, P22, P23, P24},
  Modul


    {U, Sigma, V} = SingularValueDecomposition[EMtx];
                    Singulärwertzerlegung
    Print["U of E = ", N[U]];
    gib aus         Expo··· numerischer Wert
    Print["Sigma of E = ", N[Sigma]];
    gib aus             Expo··· numerischer Wert
    Print["V of E = ", N[V]];
    gib aus         Expo··· numerischer Wert

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

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

```
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
R1 = U.Transpose[W].Transpose[V];
         transponiere   transponiere
R2 = U.W.Transpose[V];
             transponiere
Print["S1 = ", MatrixForm[N[S1]]];
gib aus           Matritzenform numerischer Wert
Print["S2 = ", MatrixForm[N[S2]]];
gib aus           Matritzenform numerischer Wert
Print["R1 = ", MatrixForm[N[R1]]];
gib aus           Matritzenform numerischer Wert
Print["R2 = ", MatrixForm[N[R2]]];
gib aus           Matritzenform numerischer Wert
(*Proof if R1 and R2 are possible rotations*)
i = IdentityMatrix[3];
    Einheitsmatrix
Print["Test R1 is rotation =", Transpose[N[R1]].N[R1]];
gib aus                        transponiere numeri··· numerischer Wert
Print["Test R2 is rotation =", Transpose[N[R2]].N[R2]];
gib aus                        transponiere numeri··· numerischer Wert
(*Map[If[SameQ[Transpose[#].#,i],
  w··· ··· identi··· transponiere
     Print[#," is Rotation"],Print[N[#]," is no Rotation"]]&,{R1,R2}];*)
     gib aus                  gib aus numerischer Wert
(*Map[Print[Det[#]]&,{R1,R1}];*)
  w··· gib aus Determinante
(*Compute t from S1&S2*)

Print["Check if t of S1, S2 is equal = ", Map[NullSpace[#] &, {S1, S2}]];
gib aus prüfe                             w··· Nullraum
t = Flatten[NullSpace[S1]];
    ebne ein Nullraum
Print["t = ", N[t] ];
              numerischer Wert
(*t is missing a scale factor of lamda set lamda to -
 1 and 1 and you get four different solutions*)

P21 = U.W.Transpose[V];
          transponiere
P21 = {{P21[[1, 1]], P21[[1, 2]], P21[[1, 3]], -1 * t[[1]]},
   {P21[[2, 1]], P21[[2, 2]], P21[[2, 3]], -1 * t[[2]]},
   {P21[[3, 1]], P21[[3, 2]], P21[[3, 3]], -1 * t[[3]]}};

Print["P21  = ", MatrixForm[N[P21]]];
gib aus            Matritzenform numerischer Wert

P22 = U.Transpose[W].Transpose[V];
          transponiere   transponiere
P22 = {{P22[[1, 1]], P22[[1, 2]], P22[[1, 3]], -1 * t[[1]]},
```

```
      {P22[[2, 1]], P22[[2, 2]], P22[[2, 3]], -1*t[[2]]},
      {P22[[3, 1]], P22[[3, 2]], P22[[3, 3]], -1*t[[3]]}};

    Print["P22  = ", MatrixForm[N[P22]]];
    gib aus              Matritzenform numerischer Wert

    P23 = U.W.Transpose[V];
                transponiere
    P23 = {{P23[[1, 1]], P23[[1, 2]], P23[[1, 3]], 1*t[[1]]},
      {P23[[2, 1]], P23[[2, 2]], P23[[2, 3]], 1*t[[2]]},
      {P23[[3, 1]], P23[[3, 2]], P23[[3, 3]], 1*t[[3]]}};

    Print["P23  = ", MatrixForm[N[P23]]];
    gib aus              Matritzenform numerischer Wert

    P24 = U.Transpose[W].Transpose[V];
              transponiere  transponiere
    P24 = {{P24[[1, 1]], P24[[1, 2]], P24[[1, 3]], 1*t[[1]]},
      {P24[[2, 1]], P24[[2, 2]], P24[[2, 3]], 1*t[[2]]},
      {P24[[3, 1]], P24[[3, 2]], P24[[3, 3]], 1*t[[3]]}};

    Print["P24  = ", MatrixForm[N[P24]]];
    gib aus              Matritzenform numerischer Wert

    Print["End Computing essential
    gib aus  beende Kontext
       Matrix________________________________________________________"];


    PList = {};

    AppendTo[PList, P21];
    hänge an bei
    AppendTo[PList, P22];
    hänge an bei
    AppendTo[PList, P23];
    hänge an bei
    AppendTo[PList, P24];
    hänge an bei

    Print["PList = ", PList];
    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];


  (*If[beta== 0,
    wenn
   CreateTriangulation[PC1,PC2,PList[[uu]]];
  ];

  If[beta≠  0,
  wenn
   CreateTriangulation[PC1,PC2,PList[[uu]]]
  ];*)


 ];

 Clear[PList];
 lösche
];
```