```
(*Rektificationalgorithm Charles Loop & Zhengyou Zhang______________*)
(*____________________________________________________________*)


NewRectification[F_, e_, ePrime_, ImagePlaneC1Points_, ImagePlaneC2Points_] :=
  Module[{Hp, HpPrime, HrPrime, Hr, w, wPrime, eInf, ePrimeInf, Vc,
  Modul
    ePrimeHorizontal, eHorizontal, RecPointsC2, RecPointsC1 , RecGraphicPointsC1,
    RecGraphicPointsC2, G2, G1, P, PPrime, pc, pcPrime, piWidth, piHeight,
    pjWidth, pjHeight, pi = {}, pj = {}, n, PP, PPPrime, pcpc, pcpcPrime,
    A, APrime, B, BPrime, ex, ePrimex, z, z1, z2, piA, piB, piC, piD, piSA,
    piSB, Hs, HsPrime, pjA, pjB, pjC, pjD, pjSA, pjSB, eL, eLPrime, zGuess},

   Print["
   gib aus
Begin New Rectification with Disortion minimization
beginne Kontext
      criterion_______________________________________________
"];

   For[i = 1, i ≤ Length[ImagePlaneC1Points], i++,
   For-Schleife  Länge
    AppendTo[pi, ImagePlaneC1Points[[i]]];
    hänge an bei
    AppendTo[pj, ImagePlaneC2Points[[i]]];
    hänge an bei
   ];

   (*Disortion minimization Criterion for finding z and w and w' *)

   n = Length[pi];
      Länge
   Print["pi = ", N[pi]];
   gib aus         numerischer Wert
   Print["pj = ", N[pj]];
   gib aus          numerischer Wert
   Print["n = ", n];
   gib aus

   minXPi  = Min[Map[#[[1]] &, pi]];
             kle·· wende an
   maxXPi  = Max[Map[#[[1]] &, pi]];
             gr··· wende an
   minYPi  = Min[Map[#[[2]] &, pi]];
             kle·· wende an
   maxYPi  = Max[Map[#[[2]] &, pi]];
             gr··· wende an

   minXPj  = Min[Map[#[[1]] &, pj]];
             kle·· wende an
   maxXPj  = Max[Map[#[[1]] &, pj]];
             gr··· wende an
   minYPj  = Min[Map[#[[2]] &, pj]];
             kle·· wende an
```

```
                  ⌊kle··· ⌊wende an
maxYPj = Max[Map[#[[2]] &, pj]];
                  ⌊gr··· ⌊wende an

Print["minXpi = ", N[minXPi]];
⌊gib aus            ⌊numerischer Wert
Print["maxXPi = ", N[maxXPi]];
⌊gib aus            ⌊numerischer Wert
Print["minYpi = ", N[minYPi]];
⌊gib aus            ⌊numerischer Wert
Print["maxYPi = ", N[maxYPi]];
⌊gib aus            ⌊numerischer Wert
Print["minXPj = ", N[minXPj]];
⌊gib aus            ⌊numerischer Wert
Print["maxXPj = ", N[maxXPj]];
⌊gib aus            ⌊numerischer Wert
Print["minYPj = ", N[minYPj]];
⌊gib aus            ⌊numerischer Wert
Print["maxYPj = ", N[maxYPj]];
⌊gib aus            ⌊numerischer Wert


piWidth = EuclideanDistance[Wxmax, Wxmin];
          ⌊euklidischer Abstand
piHeight = EuclideanDistance[Hymax, Hymin];
           ⌊euklidischer Abstand
pjWidth = EuclideanDistance[Wxmax, Wxmin];
          ⌊euklidischer Abstand
pjHeight = EuclideanDistance[Hymax, Hymin];
           ⌊euklidischer Abstand

Print["piWidth = ", N[piWidth]];
⌊gib aus             ⌊numerischer Wert
Print["piHeight = ", N[piHeight]];
⌊gib aus              ⌊numerischer Wert
Print["pjWidth = ", N[pjWidth]];
⌊gib aus             ⌊numerischer Wert
Print["pjHeight = ", N[pjHeight]];
⌊gib aus              ⌊numerischer Wert

(*projective transformation H_p and H_p'___________________*)
(*Find w  and w' by minimizing z___________________________*)
  ⌊finde

P = ConstantArray[0, {3, n}];
    ⌊konstantes Array
PPrime = ConstantArray[0, {3, n}];
         ⌊konstantes Array


pc = {0, 0};
pcPrime = {0, 0};
For[i = 1, i ≤ Length[pi] - 1, i++,
⌊For-Schleife    ⌊Länge

  pc = Total[pi];
       ⌊Gesamtsumme
```

```
             Gesamtsumme
 pcPrime = Total[pj];
           Gesamtsumme
];

pc = pc / n;
pcPrime = pcPrime / n;

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

For[i = 1, i ≤ Length[pi], i++,
For-Schleife        Länge
 P[[1, i]] = (pi[[i, 1]] - pc[[1]]);
 P[[2, i]] = (pi[[i, 2]] - pc[[2]]);

 PPrime[[1, i]] = (pj[[i, 1]] - pcPrime[[1]]);
 PPrime[[2, i]] = (pj[[i, 2]] - pcPrime[[2]]);
];
Print["P = ", N[P]];
gib aus       numerischer Wert
Print["PPrime = ", N[PPrime]];
gib aus            numerischer Wert
PP = P.Transpose[P];
       transponiere
Print["PP = ", N[PP]];
gib aus        numerischer Wert
PPPrime = PPrime.Transpose[PPrime];
                 transponiere
Print["PPPrime = ", N[PPPrime]];
gib aus             numerischer Wert
Print["pc = ", N[pc]];
gib aus        numerischer Wert
pcpc = {{pc[[1]]}, {pc[[2]]}, {pc[[3]]}}.{{pc[[1]], pc[[2]], pc[[3]]}};

Print["pcpc = ", N[pcpc]];
gib aus          numerischer Wert
pcpcPrime = {{pcPrime[[1]]}, {pcPrime[[2]]}, {pcPrime[[3]]}}.
  {{pcPrime[[1]], pcPrime[[2]], pcPrime[[3]]}};

Print["pcpcPrime = ", N[pcpcPrime]];
gib aus               numerischer Wert



ex = {{0, -e[[3]], e[[2]]}, {e[[3]], 0, -e[[1]]}, {-e[[2]], e[[1]], 0}};
ePrimex = {{0, -ePrime[[3]], ePrime[[2]]},
  {ePrime[[3]], 0, -ePrime[[1]]}, {-ePrime[[2]], ePrime[[1]], 0}};

A = Transpose[ex].PP.ex;
    transponiere
B = Transpose[F].pcpc.F;
    transponiere
```

```
transponiere

APrime = Transpose[ePrimex].PPPrime.ePrimex;
         transponiere
BPrime = Transpose[F] pcpcPrime.F;
         transponiere

Print["A = ", MatrixForm[N[A]]];
gib aus       Matritzenform numerischer Wert
Print["B = ", MatrixForm[N[B]]];
gib aus       Matritzenform numerischer Wert
Print["APrime = ", MatrixForm[N[APrime]]];
gib aus            Matritzenform numerischer Wert
Print["BPrime = ", MatrixForm[N[BPrime]]];
gib aus            Matritzenform numerischer Wert


Print["{A[[1,1;;2]],A[[2,1;;2]]}=", N[{A[[1, 1 ;; 2]], A[[2, 1 ;; 2]]}]];
gib aus                             numerischer Wert
Print["{APrime[[1,1;;2]],APrime[[2,1;;2]]}=",
gib aus
 N[{APrime[[1, 1 ;; 2]], APrime[[2, 1 ;; 2]]}]];
 numerischer Wert

DD = CholeskyDecomposition[{A[[1, 1 ;; 2]], A[[2, 1 ;; 2]]}];
     Cholesky-Zerlegung
DDPrime = CholeskyDecomposition[{APrime[[1, 1 ;; 2]], APrime[[2, 1 ;; 2]]}];
          Cholesky-Zerlegung

Print["DD =", N[DD]];
gib aus       numerischer Wert
Print["DDPrime =", N[DDPrime]];
gib aus            numerischer Wert

Print["{B[[1,1;;2]],B[[2,1;;2]]}=", N[{B[[1, 1 ;; 2]], B[[2, 1 ;; 2]]}]];
gib aus                             numerischer Wert
Print["{BPrime[[1,1;;2]],BPrime[[2,1;;2]]}=",
gib aus
 N[{BPrime[[1, 1 ;; 2]], BPrime[[2, 1 ;; 2]]}]];
 numerischer Wert

DTBD =
 Eigensystem[Transpose[Inverse[DD]].{B[[1, 1 ;; 2]], B[[2, 1 ;; 2]]}.Inverse[DD]];
 Eigen···    transponiere inverse Matrix                               inverse Matrix
DTBPrimeD = Eigensystem[Transpose[Inverse[DDPrime]].
            Eigensystem transponiere inverse Matrix
   {BPrime[[1, 1 ;; 2]], BPrime[[2, 1 ;; 2]]}.Inverse[DDPrime]];
                                              inverse Matrix

Print["DTBD[[2,1]] = ", N[DTBD[[2, 1]]]];
gib aus                 numerischer Wert
 Print["Eigensystem DTB1 = ", N[DTBD]];
 gib aus Eigensystem          numerischer Wert
 Print["Eigensystem DTBPrimeD = ", N[DTBPrimeD]];
 gib aus Eigensystem               numerischer Wert
```

```
gib aus   Eigensystem                    numerischer Wert
z1 = Inverse[DD].DTBD[[2, 1]];
       inverse Matrix
Print["z1 first = ", Inverse[DD].DTBD[[2, 1]]];
gib aus                  inverse Matrix
If[DTBD[[1, 2]] ≥ DTBD[[1, 1]],
wenn
 z1 = Inverse[DD].DTBD[[2, 2]];
        inverse Matrix
 Print["z1 second = ", z1];
 gib aus
];

 Print["z1= ", N[z1]];
 gib aus          numerischer Wert

z2 = Inverse[DDPrime].DTBPrimeD[[2, 1]];
       inverse Matrix
Print["z2 first = ", z2];
gib aus
If[DTBPrimeD[[1, 2]] ≥ DTBPrimeD[[1, 1]],
wenn
 z2 = Inverse[DDPrime].DTBPrimeD[[2, 2]];
        inverse Matrix
 Print["z2 second = ", z2];
 gib aus
];

 Print["z2= ", N[z2]];
 gib aus          numerischer Wert

z = (z1 / Normalize[z1] + z2 / Normalize[z2]);
          normalisiere          normalisiere

z = {z[[1]], z[[2]], 0};
Print["z =", N[z]];
gib aus         numerischer Wert


(*Similarity Transformation Hr and Hr' *)

w = {{0, -e[[3]], e[[2]]}, {e[[3]], 0, -e[[1]]}, {-e[[2]], e[[1]], 0}}.z;
wPrime = F.z;

Print["w = ", N[w]];
gib aus          numerischer Wert
Print["wPrime = ", N[wPrime]];
gib aus              numerischer Wert
wPrime = wPrime / wPrime[[3]];
Print["wPrime = ", N[wPrime]];
gib aus              numerischer Wert
w = w / w[[3]];
Print["w = ", N[w]];
gib aus          numerischer Wert
```

```
HpPrime = {{1, 0, 0}, {0, 1, 0}, {wPrime[[1]], wPrime[[2]], wPrime[[3]]}};
Print["HpPrime = ", MatrixForm[N[HpPrime]]];
gib aus                   Matritzenform  numerischer Wert

Hp = {{1, 0, 0}, {0, 1, 0}, {w[[1]], w[[2]], w[[3]]}};
Print["Hp = ", MatrixForm[N[Hp]]];
gib aus              Matritzenform  numerischer Wert
ePrimeInf = HpPrime.ePrime;
eInf = Hp.e;

Print["ePrime inf = ", N[ePrimeInf]];
gib aus                   numerischer Wert
Print["e inf = ", N[eInf]];
gib aus              numerischer Wert

Vc = 0.705; (*Wie bekomme ich Vc raus???*)
Hr = {{F[[3, 2]] - w[[2]] * F[[3, 3]], w[[1]] * F[[3, 3]] - F[[3, 1]], 0}, {F[[3, 1]] -
     w[[1]] * F[[3, 3]], F[[3, 2]] - w[[2]] * F[[3, 3]], F[[3, 3]] + Vc}, {0, 0, 1}};

HrPrime = {{wPrime[[2]] * F[[3, 3]] - F[[2, 3]],
    F[[1, 3]] - wPrime[[1]] * F[[3, 3]], 0}, {wPrime[[1]] * F[[3, 3]] - F[[1, 3]],
    wPrime[[2]] * F[[3, 3]] - F[[2, 3]], Vc}, {0, 0, 1}};

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

ePrimeHorizontal = HrPrime.ePrimeInf;
Print["ePrimeHorizontal = ", N[ ePrimeHorizontal]];
gib aus                             numerischer Wert
eHorizontal = Hr.eInf;
Print["eHorizontal = ", N[eHorizontal]];
gib aus                       numerischer Wert

(*Shearing Transformation H_s and H_s'*)

piA = {(piWidth) / 2, 0, 1};
piB = {piWidth, (piHeight) / 2, 1};
piC = {(piWidth) / 2, piHeight, 1};
piD = {0, (piHeight) / 2, 1};

piA = Hr.Hp.piA;
piB = Hr.Hp.piB;
piC = Hr.Hp.piC;
piD = Hr.Hp.piD;

Print["piA = ", piA];
gib aus
Print["piB = ", piB];
gib aus
Print["piC = ", piC];
gib aus
Print["piD = ", piD];
gib aus
```

```
gib aus

piX = piB - piD;
piY = piC - piA;

Print["piX = ", piX];
gib aus
Print["piY = ", piY];
gib aus


piSA = (piHeight^2 * piX[[2]]^2 + piWidth^2 * piY[[2]]^2) /
   ((piHeight * piWidth) * (piX[[2]] * piY[[1]] - piX[[1]] * piY[[2]]));
piSB = (piHeight^2 * piX[[1]] * piX[[2]] + piWidth^2 * piY[[1]] * piY[[2]]) /
   ((piHeight * piWidth) * (piX[[1]] * piY[[2]] - piX[[2]] * piY[[1]]));
Print["piSA = ", N[piSA]];
gib aus          numerischer Wert
Print["piSB = ", N[piSB]];
gib aus          numerischer Wert


Hs = {{piSA, piSB, 0}, {0, 1, 0}, {0, 0, 1}};


pjWidth = pjWidth * 1;
pjHeight = pjHeight * 1;


pjA = {(pjWidth) / 2, 0, 1};
pjB = {pjWidth, (pjHeight) / 2, 1};
pjC = {(pjWidth) / 2, pjHeight, 1};
pjD = {0, (pjHeight) / 2, 1};

pjA = HrPrime.HpPrime.pjA;
pjB = HrPrime.HpPrime.pjB;
pjC = HrPrime.HpPrime.pjC;
pjD = HrPrime.HpPrime.pjD;


Print["pjA = ", N[pjA]];
gib aus         numerischer Wert
Print["pjB = ", N[pjB]];
gib aus         numerischer Wert
Print["pjC = ", N[pjC]];
gib aus         numerischer Wert
Print["pjD = ", N[pjD]];
gib aus         numerischer Wert
pjX = pjB - pjD;
pjY = pjC - pjA;

Print["pjX = ", N[pjX]];
gib aus         numerischer Wert
Print["pjY = ", N[pjY]];
gib aus         numerischer Wert
```

```
pjSA = (pjHeight^2 * pjX[[2]]^2 + pjWidth^2 * pjY[[2]]^2) /
   ((pjHeight * pjWidth) * (pjX[[2]] * pjY[[1]] - pjX[[1]] * pjY[[2]]));

pjSB = (pjHeight^2 * pjX[[1]] * pjX[[2]] + pjWidth^2 * pjY[[1]] * pjY[[2]]) /
   ((pjHeight * pjWidth) * (pjX[[1]] * pjY[[2]] - pjX[[2]] * pjY[[1]]));
Print["pjSA = ", N[pjSA]];
gib aus                numerischer Wert
Print["pjSB = ", N[pjSB]];
gib aus                numerischer Wert


HsPrime = {{pjSA, pjSB, 0}, {0, 1, 0}, {0, 0, 1}};


eL = ConstantArray[0, {8, 3}];
     konstantes Array
eLPrime = ConstantArray[0, {8, 3}];
          konstantes Array

RecPointsC2 = ConstantArray[0, {8, 3}];
              konstantes Array
RecPointsC1 = ConstantArray[0, {8, 3}];
              konstantes Array

For[i = 1, i ≤ 8, i++,
For-Schleife

 RecPointsC1[[i]] = (*Hs.Hr.*)Hp.pi[[i]];
 eL[[i]] = N[Cross[eInf, RecPointsC1[[i, All]]]];
           ·· Kreuzprodukt                  alle
 RecPointsC1[[i]] = RecPointsC1[[i]] / RecPointsC1[[i, 3]];

 RecPointsC2[[i]] = (*HsPrime.HrPrime.*)HpPrime.pj[[i]];
 eLPrime[[i]] = N[Cross[ePrimeInf, RecPointsC2[[i, All]]]];
                ·· Kreuzprodukt                       alle
 RecPointsC2[[i]] = RecPointsC2[[i]] / RecPointsC2[[i, 3]];

];

Print["RecPointsC1 =", MatrixForm[N[RecPointsC1]]];
gib aus                Matritzenform numerischer Wert
Print["RecPointsC2 =", MatrixForm[N[RecPointsC2]]];
gib aus                Matritzenform numerischer Wert

RecGraphicPointsC1 = Map[{#[[1]], #[[2]]} &, RecPointsC1];
                     wende an
RecGraphicPointsC2 = Map[{#[[1]], #[[2]]} &, RecPointsC2];
                     wende an

G1 = Show[ListPlot[RecGraphicPointsC1 [[1 ;; 8]], PlotStyle → Darker[Green]],
     zeig··· listenbezogene Graphik                Darstellungsstil dunkler grün
  ListLinePlot[{RecGraphicPointsC1 [[4, All]],
  listenbezogene Liniengraphik          alle
```

```
listenbezogene Liniengraphik                          alle
    RecGraphicPointsC1 [[1, All]], RecGraphicPointsC1 [[2, All]],
                               alle                           alle
    RecGraphicPointsC1 [[3, All]], RecGraphicPointsC1 [[4, All]],
                               alle                           alle
    RecGraphicPointsC1 [[8, All]], RecGraphicPointsC1 [[7, All]],
                               alle                           alle
    RecGraphicPointsC1 [[6, All]], RecGraphicPointsC1 [[5
                               alle
      , All]], RecGraphicPointsC1 [[8, All]]}, PlotStyle → Darker[Green]],
        alle                          alle        Darstellungsstil dunkler  grün
  ListLinePlot[{RecGraphicPointsC1 [[1, All]], RecGraphicPointsC1 [[5, All]]},
  listenbezogene Liniengraphik                 alle                          alle
   PlotStyle → Darker[Green]],
   Darstellungsstil dunkler  grün
  ListLinePlot[{RecGraphicPointsC1 [[2, All]], RecGraphicPointsC1 [[6, All]]},
  listenbezogene Liniengraphik                 alle                          alle
   PlotStyle → Darker[Green]],
   Darstellungsstil dunkler  grün
  ListLinePlot[{RecGraphicPointsC1 [[3, All]], RecGraphicPointsC1 [[7, All]]},
  listenbezogene Liniengraphik                 alle                          alle
   PlotStyle → Darker[Green]]];
   Darstellungsstil dunkler  grün


G2 = Show[ListPlot[RecGraphicPointsC2[[1 ;; 8]], PlotStyle → Darker[Red]],
     zeig··· listenbezogene Graphik                 Darstellungsstil dunkler  rot
  ListLinePlot[{RecGraphicPointsC2[[4, All]],
  listenbezogene Liniengraphik                alle
    RecGraphicPointsC2[[1, All]], RecGraphicPointsC2[[2, All]],
                              alle                          alle
    RecGraphicPointsC2[[3, All]], RecGraphicPointsC2[[4, All]],
                              alle                          alle
    RecGraphicPointsC2[[8, All]], RecGraphicPointsC2[[7, All]],
                              alle                          alle
    RecGraphicPointsC2[[6, All]], RecGraphicPointsC2[[5
                              alle
      , All]], RecGraphicPointsC2[[8, All]]}, PlotStyle → Darker[Red]],
        alle                         alle        Darstellungsstil dunkler  rot
  ListLinePlot[{RecGraphicPointsC2[[1, All]],
  listenbezogene Liniengraphik                alle
    RecGraphicPointsC2[[5, All]]}, PlotStyle → Darker[Red]],
                              alle       Darstellungsstil dunkler  rot
  ListLinePlot[{RecGraphicPointsC2[[2, All]], RecGraphicPointsC2[[6, All]]},
  listenbezogene Liniengraphik                alle                         alle
   PlotStyle → Darker[Red]],
   Darstellungsstil dunkler  rot
  ListLinePlot[{RecGraphicPointsC2[[3, All]], RecGraphicPointsC2[[7, All]]},
  listenbezogene Liniengraphik                alle                         alle
   PlotStyle → Darker[Red]]];
   Darstellungsstil dunkler  rot
Print[Show[G1, G2, PlotRange → All]];
gib aus zeige an          Koordinatenb··· alle

yAx = 1.4;
```

```
xAx = -0.6;

Print[Show[G2, ContourPlot[eL[[1]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
gib aus zeige an   Konturgraphik
  ContourPlot[eL[[2]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eL[[3]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eL[[4]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eL[[5]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eL[[6]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eL[[7]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eL[[8]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], G1,
  Konturgraphik
  ContourPlot[eLPrime[[1]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eLPrime[[2]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eLPrime[[3]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eLPrime[[4]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eLPrime[[5]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eLPrime[[6]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eLPrime[[7]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  ContourPlot[eLPrime[[8]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  Konturgraphik
  PlotRange → Automatic]];
  Koordinatenb··· automatisch

Print["
gib aus
End New Rectification with Disortion minimization
beende Kontext
        criterion__________________________________________________
"];
  ];
```