Inhalt

[Hausaufgabe 0 2](#_Toc520213448)

[Dach überm Kopf 2](#_Toc520213449)

[Hausaufgabe 1 3](#_Toc520213450)

[1.1 Farbkonvertierung 3](#_Toc520213451)

[1.2 Bildgradient 3](#_Toc520213452)

[1.3 Harris Detector Eingabeparameter 4](#_Toc520213453)

[1.4 Harris Detector Bildvorbereitung 6](#_Toc520213454)

[1.5 Harris Detector Merkmalsextraktion 8](#_Toc520213455)

[1.6 Harris Detector Plot 9](#_Toc520213456)

[1.7 Harris Detector Upgrade 10](#_Toc520213457)

[1.8 Kuchen 11](#_Toc520213458)

[1.9 Harris Detector Merkmalsvorbereitung 12](#_Toc520213459)

[1.10 Harris Detector Kacheln 13](#_Toc520213460)

[1.11 Harris Detector 15](#_Toc520213461)

[Hausaufgabe 2 17](#_Toc520213462)

[2.1 Input Parser 17](#_Toc520213463)

[2.2 Merkmalsvorbereitung 18](#_Toc520213464)

[2.3 Normierung der Fenster 20](#_Toc520213465)

[2.4 Normierung der Fenster 22](#_Toc520213466)

[2.5 Korrespondezmatrix 23](#_Toc520213467)

[2.6 Plot 25](#_Toc520213468)

[Hausaufgabe 3 26](#_Toc520213469)

[3.1 Achtpunktalgorithmus Teil 1 26](#_Toc520213470)

[3.2 Achtpunktalgorithmus Teil 2 27](#_Toc520213471)

[3.3 RanSaC Eingabeparameter 28](#_Toc520213472)

[3.4 RanSac Vorbereitung 30](#_Toc520213473)

[3.5 Sampson Distanz 31](#_Toc520213474)

[3.6 RanSac Algorithmus 31](#_Toc520213475)

[3.7 Essentielle Matrix 33](#_Toc520213476)

[Hausaufgabe 4 35](#_Toc520213477)

[4.1 Euklidische Bewegung 35](#_Toc520213478)

[4.2 Rekonstruktion Vorbereitung 35](#_Toc520213479)

[4.3 Rekonstruktion Gleichungssysteme 37](#_Toc520213480)

[4.4 Darstellung 39](#_Toc520213481)

[4.5 Rückprojektion 41](#_Toc520213482)

# Hausaufgabe 0

## Dach überm Kopf

function W = dach(w)

% Diese Funktion implementiert den ^-Operator.

% Sie wandelt einen 3-Komponenten Vektor in eine

% schiefsymetrische Matrix um.

W = [0 -w(3) w(2);

w(3) 0 -w(1);

-w(2) w(1) 0];

end

# Hausaufgabe 1

## Farbkonvertierung

Schreiben Sie eine MATLAB-Funktion

gray\_image = rgb\_to\_gray(input\_image)

die ein uebergebenes Bild nach der Formel

C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\2003CCEB.tmpin ein Grauwertbild konvertiert.

Konvertieren Sie den Datentyp vor der Berechnung zu double. Der Rueckgabewert der Funktion soll jedoch wieder uint8 sein.

Falls der Funktion ein Grauwertbild (nur 1 Kanal) uebergeben wird, soll dieses sofort als Resultat zurueck gegeben werden.

function gray\_image = rgb\_to\_gray(input\_image)

% Diese Funktion soll ein RGB-Bild in ein Graustufenbild umwandeln. Falls

% das Bild bereits in Graustufen vorliegt, soll es direkt zurueckgegeben werden.

% Umwandlung in Double

d\_img = double(input\_image);

% Abfrage, ob RGB vorliegt

if size(d\_img, 3) == 3

% Intensitäten der einzelnen Farben abfragen.

redChannel = d\_img(:, :, 1);

greenChannel = d\_img(:, :, 2);

blueChannel = d\_img(:, :, 3);

% Berechnung nach Vorgabe

gray\_image = .299\*redChannel + .587\*greenChannel + .114\*blueChannel;

% Umwandlung in uint8

gray\_image = uint8(gray\_image);

else

% Falls kein RGB vorliegt, gib direkt das graue Bild zurück

gray\_image = input\_image;

end

end

## Bildgradient

1. Schreiben Sie eine MATLAB-Funktion
2. [Fx, Fy] = sobel\_xy(input\_image)
3. welche die erste Ableitung des uebergebenen Bildes sowohl in x- als auch in y-Richtung mittels Sobel-Filter approximiert.
4. function [Fx, Fy] = sobel\_xy(input\_image)
5. % In dieser Funktion soll das Sobel-Filter implementiert werden, welches
6. % ein Graustufenbild einliest und den Bildgradienten in x- sowie in
7. % y-Richtung zurueckgibt.
8. % Zeilen- und Spaltenanzahl des Bildes bestimmen.
9. [zeilen, spalten] = size(input\_image);
10. % 1-Pixel breiten Rahmen mit Intensität = 0 hinzufügen
11. % Oberen und unteren Teil des Rahmens hinzufügen
12. img = [zeros(1,spalten); input\_image; zeros(1,spalten)];
13. % Linken und und rechten Rahmen hinzufügen
14. img = [zeros(zeilen+2,1), img, zeros(zeilen+2,1)];
16. % Vorbelegung von Fx, Fy und deren Zwischenergebnisse mit Nullen
17. Fx = zeros(zeilen, spalten);
18. Fy = zeros(zeilen, spalten);
19. zw\_x = zeros(zeilen+2, spalten);
20. zw\_y = zeros(zeilen+2, spalten);
21. % Intensitäten der im betrachteten Fenster liegenden Pixel (Vorbelegung).
22. I\_mat = zeros(3,1);
23. % Ausnutzen der Separierbarkeit des Sobel-Operators (weniger Rechenaufwand)
24. % Zunächst Faltung mit der ersten Zeile der Sobel-Matrizen (horizontal und
25. % vertikal) berechnen. In diesem ersten Teil der Berechnung wird für jedes
26. % Pixel ein links und rechts benachbarte Pixel benötigt, daraus folgt die
27. % Begrenzung für die Laufvariable "y".
28. for x = 1:zeilen+2
29. for y = 2:spalten+1
30. % Intensitäten der im betrachteten Fenster (1x3 Pixel) liegenden
31. % Pixel bestimmen.
32. for i = -1:1
33. I\_mat(i+2) = img(x,y-i);
34. end
35. % Faltung der ersten Zeile der horizontalen Sobel-Matrix mit den
36. % Intensitäten des betrachteten Fensters.
37. zw\_x(x,y-1) = [1 0 -1]\*I\_mat;
38. % Faltung der ersten Zeile der vertikalen Sobel-Matrix mit den
39. % Intensitäten des betrachteten Fensters.
40. zw\_y(x,y-1) = [1 2 1]\*I\_mat;
41. end
42. end
43. % Zwischenergebnisse der Faltung "Fx" und "Fy" haben die Dimension
44. % "(zeilen+2) x spalten"
45. % Analog zum ersten Teil der Berechnung:
46. % Faltung mit der ersten Spalte der Sobel-Matrizen
47. for x = 2:zeilen+1
48. for y = 1:spalten
49. % Intensitäten der im betrachteten Fenster (3x1 Pixel) liegenden
50. % Pixel bestimmen. (für horizontale Ableitung)
51. for i = -1:1
52. I\_mat(i+2) = zw\_x(x-i,y);
53. end
54. % Faltung der ersten Spalte (in Zeilenvektor transformiert) der
55. % horizontalen Sobel-Matrix mit den Intensitäten des Fensters.
56. Fx(x-1,y) = [1 2 1]\*I\_mat;
57. % Intensitäten der im betrachteten Fenster (3x1 Pixel) liegenden
58. % Pixel bestimmen. (für vertikale Ableitung)
59. for i = -1:1
60. I\_mat(i+2) = zw\_y(x-i,y);
61. end
62. % Faltung der ersten Spalte (in Zeilenvektor transformiert) der
63. % vertikalen Sobel-Matrix mit den Intensitäten des Fensters.
64. Fy(x-1,y) = [1 0 -1]\*I\_mat;
65. end
66. end
67. % Nun haben "Fx" und "Fy" haben die gewünschte Dimension: zeilen x spalten
68. end

## 1.3 Harris Detector Eingabeparameter

Unter Verwendung der zuvor implementierten Funktionen, schreiben Sie eine MATLAB-Funktion

merkmale = harris\_detektor(input\_image, varargin)

die aus dem Grauwertbild input\_image Harris-Merkmale ueber das Kriterium

C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\70A82131.tmpextrahiert, deren x/y-Pixelkoordinaten in der Matrix merkmale spaltenweise speichert und abhaengig von einem optionalen Parameter do\_plot das Bild anzeigt und darin die detektierten Merkmale markiert.

Zuerst sollen Sie dem harris\_detektor die Faehigkeit geben, mit einer variierenden Anzahl von Parametern zurecht zu kommen. Neben dem Standardargument input\_image soll die Funktion folgende Parameter uebernehmen oder bei Abwesenheit eines Parameters diesen durch einen Standardwert ersetzen koennen:

* segment\_length (numerisch, ungerade, C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\74BCEB67.tmp): steuert die Groesse des Bildsegments (Standardwert: 15)
* k (numerisch, C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\B4F1D10D.tmp): gewichtet zwischen Ecken- und Kantenprioritaet (In der Literatur wird oftmals C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\40B513A3.tmpgesetzt)
* tau (numerisch, C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\BB59F0A9.tmp): legt den Schwellenwert zur Detektion einer Ecke fest (Standardwert: C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\D4EE019F.tmp)
* do\_plot (logical): bestimmt, ob das Bild angezeigt wird oder nicht (Standardwert: False)

Ueberpruefen Sie dabei auch etwaige Anforderungen an die Eingabeparameter wie Datentyp, Wertebereich oder Aehnliches.

Geben Sie zum Testen in dieser Aufgabe die verarbeiteten Parameter **statt** der Merkmale zurueck:

merkmale = {segment\_length, k, tau, do\_plot}

function merkmale = harris\_detektor(input\_image, varargin)

% In dieser Funktion soll der Harris-Detektor implementiert werden, der

% Merkmalspunkte aus dem Bild extrahiert

%% Input parser

p = inputParser;

% Standardwerte definieren.

defaultsegment\_length = 15;

defaultk = 0.05;

defaulttau = 10^6;

defaultdo\_plot = false;

% Bedingungen für gültige Inputs definieren.

validsegment\_length = @(x) isnumeric(x) && x > 0 && mod(x,2) == 1;

validk = @(x) isnumeric(x) && x <= 1 && x >= 0;

validtau = @(x) isnumeric(x) && x > 0;

validdo\_plot = @(x) islogical(x);

% Optionale Parameterübergabe definieren.

addOptional(p,'segment\_length',defaultsegment\_length, validsegment\_length);

addOptional(p,'k',defaultk, validk);

addOptional(p,'tau',defaulttau, validtau);

addOptional(p,'do\_plot',defaultdo\_plot, validdo\_plot);

% Parsen des Inputs durchführen.

parse(p, varargin{:});

% Rückgabe der verarbeiteten Parameter.

merkmale = {p.Results.segment\_length, p.Results.k, p.Results.tau, p.Results.do\_plot};

end

## 1.4 Harris Detector Bildvorbereitung

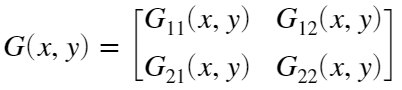
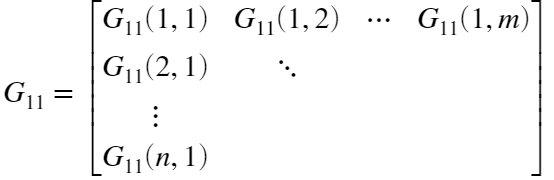
Unter Verwendung der zuvor implementierten Funktionen, schreiben Sie eine MATLAB-Funktion

merkmale = harris\_detektor(input\_image, varargin)

die aus dem Grauwertbild input\_image Harris-Merkmale ueber das Kriterium

C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\BD959C05.tmpextrahiert, deren x/y-Pixelkoordinaten in der Matrix merkmale spaltenweise speichert und abhaengig von einem optionalen Parameter do\_plot das Bild anzeigt und darin die detektierten Merkmale markiert.

Nachdem die Eingangsgroessen in der letzten Aufgabe ueberprueft wurden, implementieren Sie in diesem Teil bitte die unten aufgelisteten Prorammbestandteile. Gehen Sie davon aus, dass alle in der vorherigen Aufgabe geforderten Variablen (segment\_length, k, tau, do\_plot) weiterhin zur Verfueung stehen.

1. Ueberpruefen Sie zunaechst, ob es sich um ein Grauwertbild handelt. Ist dies nicht der Fall, beenden Sie das Programm mit der Fehlermeldung: *"Image format has to be NxMx1"*
2. Konvertieren Sie das Grauwertbild zum Datentyp double.
3. Danach ermitteln Sie den Bildgradienten in horizontaler Richtung Ix und vertikaler Richtung Iy mit Hilfe der in Aufgabe 1.2 erstellten Funktion.
4. Erstellen Sie einen symmetrischen Gewichtungsvektor w als Teil eines separablen Filters fuer das Segment W, um den Einfluss zentraler Pixel zu erhoehen. Achten Sie darauf Filterbreite an die Breite des Segments anzupassen.
5. Bestimmen Sie die Eintraege der Harris Matrix fuer jedes Pixel im Bild. Speichern Sie die Information in jeweils drei Matrizen, die die selbe Groesse haben wie das Bild selbst. Zum Beispiel 
6. Beachten Sie, dass durch **geschickte Filterung** das gesamte Bild auf einmal verarbeitet werden kann.

Geben Sie zum Testen in dieser Aufgabe die Zwischenergebnisse **statt** der Merkmale zurueck:

merkmale = {Ix, Iy, w, G11, G22, G12}

function merkmale = harris\_detektor(input\_image, varargin)

% In dieser Funktion soll der Harris-Detektor implementiert werden, der

% Merkmalspunkte aus dem Bild extrahiert

%% Input parser aus Aufgabe 1.3

input\_parser

seg\_length = segment\_length;

%% Vorbereitung zur Feature Detektion

% Pruefe ob es sich um ein Grauwertbild handelt

if(size(input\_image,3) ~= 1)

error('Image format has to be NxMx1');

end

% Intensitäten in Double-Werte umwandeln.

img = double(input\_image);

% Approximation des Bildgradienten

[Ix, Iy] = sobel\_xy(img);

% Gewichtung

% Hilfsvariable zur Bestimmung des Gauß-Filter.

% Für alle Werte x, für die gilt -s <= x <= s wird die Gaußverteilung

% bestimmt.

s = (seg\_length-1)/2;

% 2\*Standardabweichung soll gleich "s" sein bzw. Standard-

% abweichung = s/2. Daraus folgt für die Varianz:

var = (s/2)^2;

% Gauß-Verteilung ist symmetrisch zum Erwartungswert.

% Nur Werte links vom Erwartungswert müssen berechnet werden.

X = -s:1:-1;

w = exp(-(X.^2)/(2\*var^2));

% Summe für spätere Normierung berechnen mit exp(0) = 1

sm = 2\*sum(w)+1;

% Gewichte mit Normierung berechnen.

% (Wert für x = 0 => exp(0)=1 hinzufügen.)

w = (1/sm)\*[w,1,fliplr(w)];

% Quadrate der Gradienten berechnen und in einem Vektor zusammenfassen.

qu = {Ix.\*Ix, Ix.\*Iy, Iy.\*Iy};

% Nutzen der Separierbarkeit des Gauß-Filters.

% => 6 statt 9 Multiplikationen und 5 statt 8 Additionen pro

% berechnetem Pixel.

% Jeweils im ersten Teil der Berechnung Faltung mit dem Zeilenvektor

% der Gewichtung. Dann Faltung mit dem Spaltenvektor der Gewichtung.

G11 = conv2(qu{1},w,'same');

G11 = conv2(G11,w.','same');

G12 = conv2(qu{2},w,'same');

G12 = conv2(G12,w.','same');

G22 = conv2(qu{3},w,'same');

G22 = conv2(G22,w.','same');

% Ausgabe

merkmale = {Ix, Iy, w, G11, G22, G12};

## 1.5 Harris Detector Merkmalsextraktion

Unter Verwendung der zuvor implementierten Funktionen, schreiben Sie eine MATLAB-Funktion

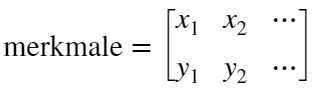
merkmale = harris\_detektor(input\_image, varargin)

die aus dem Grauwertbild input\_image Harris-Merkmale ueber das Kriterium

C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\4F39DED7.tmpextrahiert, deren x/y-Pixelkoordinaten in der Matrix merkmale spaltenweise speichert und abhaengig von einem optionalen Parameter do\_plot das Bild anzeigt und darin die detektierten Merkmale markiert.

Gehen Sie weiterhin davon aus, dass alle in den vorherigen Aufgaben geforderten Variablen weiterhin zur Verfuegung stehen.

Implementieren Sie nun die in der obigen Formel beschriebene Harrismessung. Finden Sie eine sinnvolle Behandlung der Merkmale die näher als ceil(segment\_lenght/2) am Rand des Bildes sind. Eleminieren Sie danach alle Merkmale, die kleiner als der Schwellwert tau sind, und speichern Sie die resultierende Matrix in einer variable corners C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\296AC5FD.tmp. Erstellen Sie aus den uebrig gebliebenen Eintraegen die x/y-Pixelkoordinaten-Matrix merkmale mit der Form

. Gute Dienste leistet heir die find() Funktion.

Geben Sie zum Testen in dieser Aufgabe die Zwischenergebnisse und die Merkmale zurueck:

merkmale = {H, corners, merkmale}

function merkmale = harris\_detektor(input\_image, varargin)

% In dieser Funktion soll der Harris-Detektor implementiert werden, der

% Merkmalspunkte aus dem Bild extrahiert

%% Input parser aus Aufgabe 1.3

input\_parser

%% Vorbereitung zur Feature Detektion aus Aufgabe 1.4

image\_preprocessing

%% Merkmalsextraktion ueber die Harrismessung

% Vorbelegung von corners

[zl, sp] = size(G11);

corners = zeros(zl,sp);

% Berechnung: H = det(G)-k\*tr(G)^2

H = G11.\*G22-G12.^2-k\*(G11+G22).^2;

% Am Rand des Bildes mit der Breite "s", soll keine Ecken detektiert werden,

% da hier die Harris-Matrix nicht aussagekräftig ist.

s = (segment\_length-1)/2;

% Dazu Rand mit Nullen belegen.

% Oberer und unterer Rand:

R = [zeros(s,sp); H(s+1:zl-s,:); zeros(s,sp)];

% Linker und rechter Rand:

R = [zeros(zl,s), R(:,s+1:sp-s), zeros(zl,s)];

% Prüfen, ob H >= Schwellwert tau und Koordinaten speichern.

[Y,X] = find(R >= tau);

% Merkmale speichern.

merkm = [X.';Y.'];

% Koordinaten von Merkmale in lineare Indizes umwandeln.

lin\_index = sub2ind([zl,sp],Y,X);

% H-Wert wird nur an den Koordinaten übernommen, die in merkm gespeichert sind.

corners(lin\_index) = R(lin\_index);

% Ausgabe

merkmale = {H, corners, merkm};

end

## 1.6 Harris Detector Plot

Unter Verwendung der zuvor implementierten Funktionen, schreiben Sie eine MATLAB-Funktion

merkmale = harris\_detektor(input\_image, varargin)

die aus dem Grauwertbild input\_image Harris-Merkmale ueber das Kriterium

C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\E513C99.tmpextrahiert, deren x/y-Pixelkoordinaten in der Matrix merkmale spaltenweise speichert und abhaengig von einem optionalen Parameter do\_plot das Bild anzeigt und darin die detektierten Merkmale markiert.

Gehen Sie weiterhin davon aus, dass alle in den vorherigen Aufgaben geforderten Variablen weiterhin zur Verfuegung stehen. Zeichnen Sie die gefundenen Merkmale in das Bild input\_image ein, wenn die Variable do\_plot dies vorschreibt. Um die Merkmale im Bild farbig einzuzeichnen, verwenden Sie punktuelle Plots (Square, Cross, Point, Circle, ...) keine Linienplots.

function merkmale = harris\_detektor(input\_image, varargin)

% In dieser Funktion soll der Harris-Detektor implementiert werden, der

% Merkmalspunkte aus dem Bild extrahiert

%% Input parser aus Aufgabe 1.3

input\_parser

%% Vorbereitung zur Feature Detektion aus Aufgabe 1.4

image\_preprocessing

%% Merkmalsextraktion ueber die Harrismessung aus Aufgabe 1.5

harrismessung

%% Plot

% Ecken im Bild mit roten Rechtecken markieren, falls "do\_plot" = true

if do\_plot

% Zeichne Eingabe-Bild

imshow(input\_image);

% Ecken im selben Plot markieren.

hold on;

% Plotte rote Punkte, an den Stellen, an denen Ecken sind.

plot(merkmale(1,:),merkmale(2,:), 'r.');

end

end

## 1.7 Harris Detector Upgrade

Der bisher erstellte Detektor ist nur eine suboptimale Loesung, da die Auswahl der Merkmale lediglich ueber einen **globalen** Schwellwert tau getroffen wird, die Auspraegung der Merkmale ueber das Bild allerdings stark variieren kann. Eine weitaus bessere Loesung ist es, zunaechst sehr schwache Merkmale, die beispielsweise durch Rauschen entstanden sind, ueber einen kleinen globalen Schwellwert im gesamten Bild zu eliminieren.

Kopieren Sie den Input Parser aus Aufgabe 1.3 und erweitern Sie ihn um folgende optionale Argumente:

* min\_dist (numerisch, C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\A5F9630F.tmp): ist der minimale Pixelabstand zweier Merkmale (Standardwert: 20)
* tile\_size (numerisch): definiert die Kachelgroesse, je nach Eingabe entweder die Seitenlaenge fuer eine quadratische Kachel oder ein Vektor mit zwei Eintraegen fuer Hoehe und Breite. (Standardwert: 200)
* N (numerisch, C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\359D2EF5.tmp): ist die maximale Anzahl an Merkmalen innerhalb einer Kachel (Standardwert: 5)

Stellen Sie sicher, dass unabhaengig von der Eingabe des Benutzers tile\_size immer ein Vektor

tile\_size = [height, width];

ist, der die Hoehe und die Breite der Kachel beinhaltet.

Geben Sie zum Testen in dieser Aufgabe die verarbeiteten Parameter **statt** der Merkmale zurueck:

merkmale = {min\_dist, tile\_size, N}

function merkmale = harris\_detektor(input\_image, varargin)

% In dieser Funktion soll der Harris-Detektor implementiert werden, der

% Merkmalspunkte aus dem Bild extrahiert

%% Input parser

% Standardwerte defnieren.

p = inputParser;

defaultsegment\_length = 15;

defaultk = 0.05;

defaulttau = 10^6;

defaultdo\_plot = false;

defaultmin\_dist = 20;

defaulttile\_size = 200;

defaultN = 5;

% Bedingungen für gültige Inputs definieren.

validsegment\_length = @(x) isnumeric(x) && x > 0 && mod(x,2) == 1;

validk = @(x) isnumeric(x) && x <= 1 && x >= 0;

validtau = @(x) isnumeric(x) && x > 0;

validdo\_plot = @(x) islogical(x);

validmin\_dist\_N = @(x) isnumeric(x) && x >= 1;

validtile\_size = @(x) isnumeric(x) && numel(x) <= 2;

% Optionale Parameterübergabe definieren.

addOptional(p,'segment\_length',defaultsegment\_length, validsegment\_length);

addOptional(p,'k',defaultk, validk);

addOptional(p,'tau',defaulttau, validtau);

addOptional(p,'do\_plot',defaultdo\_plot, validdo\_plot);

addOptional(p,'min\_dist',defaultmin\_dist, validmin\_dist\_N);

addOptional(p,'tile\_size',defaulttile\_size, validtile\_size);

addOptional(p,'N',defaultN, validmin\_dist\_N);

% Parsen durchführen.

parse(p, varargin{:});

tile\_size = p.Results.tile\_size;

% Falls Skalar für Kachelgröße übergeben wurde, Vektor erstellen.

if isscalar(tile\_size)

tile\_size = [tile\_size, tile\_size];

end

% Rückgabe

merkmale = {p.Results.min\_dist, tile\_size, p.Results.N};

end

## 1.8 Kuchen

Um zu verhindern, dass mehrere Merkmale ein und desselben realen Objektpunktes extrahiert werden, duerfen Merkmalspunkte einen minimalen Abstand (in Pixeln) zueinander nicht unterschreiten. Zusaetzlich soll das Bild in kleinere Kacheln unterteilt werden, wobei die Groesse der Kachel vom Benutzer festgelegt werden kann. Innerhalb einer Kachel werden dann maximal nur die N staerksten Merkmale gespeichert. Beachten Sie, dass der zuvor definierte Mindestabstand zwischen zwei Merkmalspunkten auch ueber Kachelgrenzen hinweg eingehalten werden soll.

Erstellen Sie dafuer die Funktion

Cake = cake(min\_dist)

welche eine Matrix zurueck gibt die eine kreisfoermige Anordnung von Nullen beinhaltet und den Rest der Matrix mit Einsen auffuellt.

Eine Null wird dann eingesetzt, wenn der Pixel-Abstand zum Mittelpunkt der Matrix kleiner oder gleich der min\_dist ist. Andersherum sind alle Eintraege Einsen, die einen groesseren Abstand haben als min\_dist.

function Cake = cake(min\_dist)

% Die Funktion cake erstellt eine "Kuchenmatrix", die eine kreisfoermige

% Anordnung von Nullen beinhaltet und den Rest der Matrix mit Einsen

% auffuellt. Damit koennen, ausgehend vom staerksten Merkmal, andere Punkte

% unterdrueckt werden, die den Mindestabstand hierzu nicht einhalten.

% Zunächst nur einen Viertelkreis berechnen, um Berechnungen/Vergleiche und somit auch Rechenaufwand zu reduzieren.

% Später dann gesamten Kreis aus den Viertelkreis zusammenstellen.

% Vektor mit aufsteigenden Werten, der Länge min\_dist+1 erstellen.

x = 0:min\_dist;

% Zwei Matrizen der Dimension (min\_dist+1) x (min\_dist+1) erstellen.

% x1 enthält die Abstände zum ersten Element der Matrix bzw. zum "Mittelpunkt des späteren Kreises" in x-Richtung.

% x2 enthält die Abstände zum ersten Element der Matrix bzw. zum "Mittelpunkt des späteren Kreises" in y-Richtung.

[x1, x2] = meshgrid(x,x);

% Vorbelegung des Viertel-Cakes mit Einsen.

Cake = ones(min\_dist+1);

% Null einsetzen, wenn der Pixel-Abstand zum Mittelpunkt der Matrix kleiner oder gleich der min\_dist ist.

Cake(x1.^2+x2.^2<=min\_dist^2)=0;

% Kuchen aus vier Vierteln des Kuchens zusammensetzen.

Cake = [rot90(rot90(Cake(2:min\_dist+1,2:min\_dist+1))), flipud(Cake(2:min\_dist+1,:));

fliplr(Cake(:,2:min\_dist+1)), Cake];

% Umwandeln des Typs double in logical.

Cake = logical(Cake);

end

## 1.9 Harris Detector Merkmalsvorbereitung

Um zu verhindern, dass mehrere Merkmale ein und desselben realen Objektpunktes extrahiert werden, duerfen Merkmalspunkte einen minimalen Abstand (in Pixeln) zueinander nicht unterschreiten. Zusaetzlich soll das Bild in kleinere Kacheln unterteilt werden, wobei die Groesse der Kachel vom Benutzer festgelegt werden kann. Innerhalb einer Kachel werden dann maximal nur die N staerksten Merkmale gespeichert. Beachten Sie, dass der zuvor definierte Mindestabstand zwischen zwei Merkmalspunkten auch ueber Kachelgrenzen hinweg eingehalten werden soll.

In dieser Aufgabe starten Sie mit der corners Matrix, die Sie aus Aufagbe 1.5 kennen. Hierbei handelte es sich um eine Matrix, die die selbe Groesse hat wie das Grauwertbild, jedoch die Eintraege den Werten der Harrismessung H entsprechen. Messwerte unterhalb des Schwellwerts tau, sowie fehlerhafte Ausschlaege am Rand wurden bereits zu 0 gesetzt.

Fuegen Sie einen Nullrand der Breite min\_dist um die Matrix corners hinzu.

Sortieren Sie die Inidizes aller Merkmale in corners, die nicht 0 sind in absteigender Reihenfolge der Merkmalsstaerke. (Der Index eines Elements einer Matrix ist die Nummer des Pixels, wenn man von links oben spaltenweise zu zaehlen beginnt.) Speichern Sie das Ergebnis im Array sorted\_index.

Geben Sie zum Testen in dieser Aufgabe die verarbeiteten Parameter **statt** der Merkmale zurueck:

merkmale = {corners, sorted\_index}

function merkmale = harris\_detektor(input\_image, varargin)

% In dieser Funktion soll der Harris-Detektor implementiert werden, der

% Merkmalspunkte aus dem Bild extrahiert

%% Input parser aus Aufgabe 1.7

input\_parser\_neu

%% Vorbereitung zur Feature Detektion aus Aufgabe 1.4

image\_preprocessing

%% Merkmalsextraktion ueber die Harrismessung aus Aufgabe 1.5

harrismessung\_neu

%% Merkmalsvorbereitung

% Nullrand der Breite min\_dist hinzufügen.

[zl, sp] = size(corners);

% Oberen und unteren Teil des Rahmens hinzufügen.

corners = [zeros(min\_dist,sp); corners; zeros(min\_dist,sp)];

% Linken und rechten Teil des Rahmens hinzufügen.

corners = [zeros(zl+2\*min\_dist,min\_dist), corners, zeros(zl+2\*min\_dist,min\_dist)];

% Bildindizes aller Merkmale in corners, die nicht 0 sind.

index = find(corners>0);

% Nach Merkmalsstärke sortieren.

% sort\_h\_corners: Merkmalsstärken in absteigender Reihenfolge;

% sort\_i: Indizes durch die aus aus dem unsortiereten Vektor der Merkmalsstärken

% C = "corners(index)" der sortierte Vektor V (hier durch die Tilde ersetzt, da nicht benötigt)

% entsteht, d.h. V = C(sort\_i)".

[~, sort\_i] = sort(corners(index),'descend');

% Sortierung mit Hilfe von "sort\_i" nun auch auf den Vektor, der die Bildindizes

% enthält, anwenden:

sorted\_index = index(sort\_i);

% Ausgabe

merkmale = {corners, sorted\_index};

end

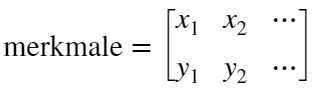
## 1.10 Harris Detector Kacheln

Um zu verhindern, dass mehrere Merkmale ein und desselben realen Objektpunktes extrahiert werden, duerfen Merkmalspunkte einen minimalen Abstand (in Pixeln) zueinander nicht unterschreiten. Zusaetzlich soll das Bild in kleinere Kacheln unterteilt werden, wobei die Groesse der Kachel vom Benutzer festgelegt werden kann. Innerhalb einer Kachel werden dann maximal nur die N staerksten Merkmale gespeichert. Beachten Sie, dass der zuvor definierte Mindestabstand zwischen zwei Merkmalspunkten auch ueber Kachelgrenzen hinweg eingehalten werden soll.

Legen Sie ein Akkumulatorfeld AKKA an (ein Eintrag pro Kachel), welches spaeter darueber Aufschluss geben soll, wie viele Merkmale pro Kachel schon gefunden wurden. Am Anfang ist das Akkumulatorfeld mit Nullen gefuellt.

Zur Erinnerung: Die Groesse der Kacheln wird vom Parameter tile\_size bestimmt.

Legen Sie ein mit Nullern gefuelltes 2D-Array merkmale an, das wie anfangs erklaert, spaeter x/y-Pixelkoordinaten der gefundenen Merkmale spaltenweise speichern soll.

Beachten Sie, dass sowohl die Anzahl der gewuenschten Merkmale pro Kachel, als auch die Anzahl der vorverarbeiteten Merkmale in sorted\_index, einen Einfluss auf die Groesse des Arrays merkmale haben koennen.

Geben Sie zum Testen in dieser Aufgabe die allozierten Arrays **statt** der eigentlichen Merkmalskoordinaten zurueck

merkmale = {AKKA, merkmale};

function merkmale = harris\_detektor(input\_image, varargin)

% In dieser Funktion soll der Harris-Detektor implementiert werden, der

% Merkmalspunkte aus dem Bild extrahiert

%% Input parser aus Aufgabe 1.7

input\_parser\_neu

%% Vorbereitung zur Feature Detektion aus Aufgabe 1.4

image\_preprocessing

%% Merkmalsextraktion ueber die Harrismessung aus Aufgabe 1.5

harrismessung\_neu

%% Merkmalsvorbereitung aus Aufgabe 1.9

feature\_preprocessing

%% Akkumulatorfeld

% Anzahl Zeilen und Spalten des Eingangsbildes bestimmen.

[zl,sp] = size(input\_image);

% Aufgerundete Anzahl der Kacheln die horizontal bzw. vertikal in das Bild passen.

anz\_x = ceil(zl/tile\_size(1));

anz\_y = ceil(sp/tile\_size(2));

% Entsprechendes Akkumulatorfeld erstellen.

AKKA = zeros(anz\_x,anz\_y);

% Merkmale mit Nullen vorbelegen:

% Zeilenanzahl = 2 (spaltenweise Speicherung der x- und y-Koordinate.)

% Spaltenanzahl entspricht maximal der Anzahl an Kacheln ("anz\_x\*anz\_y") mal der maximalen Anzahl an Merkmalen pro Kachel (N).

% Falls die Anzahl der Merkmale pro Kachel geringer ist als N, dann entspricht die Spaltenanzahl der Länge des Vektors sorted\_index.

merkmale = zeros(2,min([length(sorted\_index), anz\_x\*anz\_y\*N]));

% Was ist, wenn das Bild z.B. in 2 Kacheln unterteilt wird und N=2 gewählt wird. In Kachel 1 sind 10 Merkmale und in Kachel 2 sind

% 0 Merkmale?

% Die Spaltenanzahl von merkmale sollte eigt. 2 betragen, dieser Code erzeugt jedoch eine merkmale-Matrix mit 4 Spalten!?

% Ausgabe

merkmale = {AKKA, merkmale};

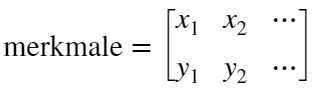
end

## 1.11 Harris Detector

Um zu verhindern, dass mehrere Merkmale ein und desselben realen Objektpunktes extrahiert werden, duerfen Merkmalspunkte einen minimalen Abstand (in Pixeln) zueinander nicht unterschreiten. Zusaetzlich soll das Bild in kleinere Kacheln unterteilt werden, wobei die Groesse der Kachel vom Benutzer festgelegt werden kann. Innerhalb einer Kachel werden dann maximal nur die N staerksten Merkmale gespeichert. Beachten Sie, dass der zuvor definierte Mindestabstand zwischen zwei Merkmalspunkten auch ueber Kachelgrenzen hinweg eingehalten werden soll.

Sie koennen nun die in den vorherigen Aufgaben definierten Variablen corners, sorted\_index, AKKA, merkmale, ... und die cake Funktion nutzen, um die oben genannten Anforderungen zu erfuellen. Durch die Einfuehrung des Abstands und des globalen Schwellwertes darf es vorkommen, dass innerhalb einer Kachel weniger als N Merkmale extrahiert werden.

Geben Sie die x/y-Pixelkoordinaten der gefundenen Merkmale spaltenweise zurueck:



function merkmale = harris\_detektor(input\_image, varargin)

% In dieser Funktion soll der Harris-Detektor implementiert werden, der

% Merkmalspunkte aus dem Bild extrahiert

%% Input parser aus Aufgabe 1.7

input\_parser\_neu

%% Vorbereitung zur Feature Detektion aus Aufgabe 1.4

image\_preprocessing

%% Merkmalsextraktion ueber die Harrismessung aus Aufgabe 1.5

harrismessung\_neu

%% Merkmalsvorbereitung aus Aufgabe 1.9

feature\_preprocessing

%% Akkumulatorfeld aus Aufgabe 1.10

akkumulatorfeld

%% Merkmalsbestimmung mit Mindestabstand und Maximalzahl pro Kachel

% Größe von corners-Matrix bestimmen.

[zl, sp] = size(corners);

% Laufvariable, die die Anzahl an geprüften Merkmalen zählt.

anz\_p = 0;

% Laufvariable, die die Anzahl an abgepeicherten Merkmalen zählt.

anz\_m = 0;

% MIND-Matrix analog zu AKKA aufstellen, die Aufschluss darüber gibt, ob in

% einer Kachel überhaupt noch "Platz" für ein weiteres Merkmal ist (Element

% = 0) oder ob durch die max. Anzahl Merkmale pro Kachel N oder den Mindestabstand

% "min\_dist" es unmöglich ist, dass ein weiteres Merkmal dieser Kachel

% gespeichert wird. (Element = 1)

MIND = zeros(size(AKKA));

% Kreis mit Nullen erstellen (zur Einhaltung von "min\_dist").

kreis = cake(min\_dist);

% While-Schleife ausführen...

% ...solange die Anzahl an abgespeicherten Merkmalen kleiner ist als die

% Anzahl erkannter Meerkmale, d.h. anz\_p < length(sorted\_index),

% ...und solange eine Kachel existiert, aus der möglicherweise noch ein

% Merkmal gespeichert werden könnte, d.h. "N" und "min\_dist" verbieten

% nicht weitere Speicherungen von Merkmalen => sum(MIND(:)) < numel(MIND)

% => Wenn alle Elemente in MIND = 1, dann kann kein weiteres Merkmal mehr

% gespeichert werden.

while anz\_p < length(sorted\_index) && sum(MIND(:)) < numel(MIND)

% Anzahl der geprüften Merkmale um 1 erhöhen.

anz\_p = anz\_p + 1;

% Linearen Index bestimmen, der potentiell den Merkmalen hinzugefügt wird.

p\_index = sorted\_index(anz\_p);

% Entsprechende Position in corners-Matrix bestimmen.

[p\_zl,p\_sp] = ind2sub([zl sp],p\_index);

% Entsprechende Bildposition bestimmen.

p\_zl\_B = p\_zl-min\_dist;

p\_sp\_B = p\_sp-min\_dist;

% Prüfen, ob Mindestabstand zu benachbarten Merkmalen eingehalten wird,

% d.h. corners(p\_index) ~= 0 und ob von der Kachel, in der sich das

% betrachtete Merkmal befindet, nicht bereits N-Merkmale gespeichert

% wurden, d.h. AKKA(kachel\_index(1),kachel\_index(2)) < N

kachel\_index = [ceil(p\_zl\_B/tile\_size(1)),ceil(p\_sp\_B/tile\_size(2))];

if corners(p\_index) ~= 0 && AKKA(kachel\_index(1),kachel\_index(2)) < N

% Anzahl gespeicherter Merkmal um 1 erhöhen.

anz\_m = anz\_m + 1;

% Merkmal speichern.

merkmale(:,anz\_m) = [p\_sp\_B,p\_zl\_B];

% Kreis mit Nullen, um die Position des Merkmals legen, um "min\_dist" einzuhalten.

% Kreis Matrix so in die corners-Matrix einfügen, dass der

% "Mittelpunkt" des Kreises mit der Position des Merkmals

% übereinstimmt.

corners(p\_zl-min\_dist:p\_zl+min\_dist,p\_sp-min\_dist:p\_sp+min\_dist)...

= kreis.\*corners(p\_zl-min\_dist:p\_zl+min\_dist,p\_sp-min\_dist:p\_sp+min\_dist);

% => corners-Matrix enthält nun Nullen im Umgebungsradius "min\_dist"

% Akkumulatorfeld aktualisieren / Element der entsprechenden Kachel

% um 1 erhöhen.

AKKA(kachel\_index(1),kachel\_index(2)) = AKKA(kachel\_index(1),kachel\_index(2)) + 1;

% Aktuelle Kachel aus corners-Matrix bestimmen. (Mit Hilfe von "min()" berücksichtigen,

% dass manche Kacheln kleiner sein können als durch tile\_size vorgegeben.)

kachel = corners((kachel\_index(1)-1)\*tile\_size(1)+1:min(kachel\_index(1)\*tile\_size(1),zl),...

(kachel\_index(2)-1)\*tile\_size(2)+1:min(kachel\_index(2)\*tile\_size(2),sp));

% Prüfen, ob es aufgrund von "N" oder "min\_dist" überhaupt noch möglich ist, ein

% weiteres Merkmal aus der aktuellen Kachel zu speichern. Falls nicht, setze entsprechendes

% MIND-Element auf 1.

MIND(kachel\_index(1),kachel\_index(2)) = AKKA(kachel\_index(1),kachel\_index(2)) >= N...

|| ~(any(kachel(:)~=0));

end

end

% Ausgabe

merkmale = merkmale(:,1:anz\_m);

% Plot entsprechend Aufgabe 1.6

if do\_plot

imshow(input\_image);

hold on;

plot(merkmale(1,:),merkmale(2,:), 'go');

end

end

# Hausaufgabe 2

## 2.1 Input Parser

Schreiben Sie eine MATLAB-Funktion

Korrespondenzen = punkt\_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)

der zwei Grauwert-Bilder I1, I2, zwei Merkmalspunktmatrizen Mpt1, Mpt2 sowie optional die Parameter window\_length, min\_corr und do\_plot uebergeben werden koennen. Die Funktion soll die Regionen um die mitgelieferten Merkmalspunkte miteinander abgleichen, um Korrespondenzpunktpaare fuer weitere Stereo-Algorithmen zu gewinnen. Die Merkmalspunkte sollen mit dem in Modul 1 entwickelten Harris-Detektor extrahiert werden. Eine funktionierende harris\_detektor Funktion wird Ihnen bereit gestellt und kann wie unten zu sehen aufgerufen werden.

Implementieren Sie in dieser Aufgabe den Input Parser, der die optionalen Parameter prueft:

* window\_length (numerisch, ungerade, C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\C6A16E47.tmp) Seitenlaenge des quadratischen Fensters um die Merkmalspunkte, welche untereinander verglichen werden (Standardwert = 25)
* min\_corr (numerisch, C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\A4D4B6ED.tmp) Unterer Schwellwert fuer die staerke der Korrelation zweier Merkmale (Standardwert = 0.95)
* do\_plot (logical) bestimmt, ob das Bild angezeigt wird oder nicht (Standardwert = False)

Stellen Sie sicher, dass die Eingabebilder I1, I2 als double werte in den Variablen Im1 und Im2 gespeichert werden.

Geben Sie zum Testen in dieser Aufgabe die verarbeiteten Parameter **statt** der Korrespondenzen zurueck:

Korrespondenzen = {window\_length, min\_corr, do\_plot, Im1, Im2}

function Korrespondenzen = punkt\_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)

% In dieser Funktion sollen die extrahierten Merkmalspunkte aus einer

% Stereo-Aufnahme mittels NCC verglichen werden um Korrespondenzpunktpaare

% zu ermitteln.

%% Input parser

p = inputParser;

% Standardwerte definieren

defaultwindow\_length = 25;

defaultmin\_corr = 0.95;

defaultdo\_plot = false;

% Bedingungen für gültige Inputs definieren:

validwindow\_length = @(x) isnumeric(x) && x>1 && mod(x,2)==1;

validmin\_corr = @(x) isnumeric(x) && x>0 && x<1;

validdo\_plot = @(x) islogical(x);

% Optionale Parameterübergabe definieren:

addOptional(p,'window\_length',defaultwindow\_length, validwindow\_length);

addOptional(p,'min\_corr',defaultmin\_corr, validmin\_corr);

addOptional(p,'do\_plot',defaultdo\_plot, validdo\_plot);

% Parsen durchführen.

parse(p,varargin{:});

% Eingabebilder I1, I2 als double werte in den Variablen Im1 und Im2 speichern.

Im1 = double(I1);

Im2 = double(I2);

Korrespondenzen = {p.Results.window\_length, p.Results.min\_corr, p.Results.do\_plot, Im1, Im2};

end

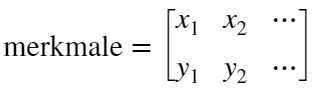
## 2.2 Merkmalsvorbereitung

Schreiben Sie eine MATLAB-Funktion

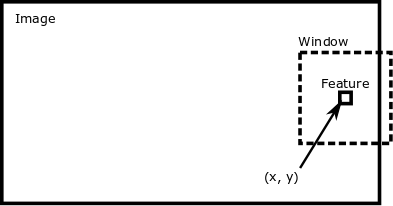
Korrespondenzen = punkt\_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)

der zwei Grauwert-Bilder I1, I2, zwei Merkmalspunktmatrizen Mpt1, Mpt2 sowie optional die Parameter window\_length, min\_corr und do\_plot uebergeben werden koennen. Die Funktion soll die Regionen um die mitgelieferten Merkmalspunkte miteinander abgleichen, um Korrespondenzpunktpaare fuer weitere Stereo-Algorithmen zu gewinnen. Die Merkmalspunkte sollen mit dem in Modul 1 entwickelten Harris-Detektor extrahiert werden. Eine funktionierende harris\_detektor Funktion wird Ihnen bereit gestellt und kann wie unten zu sehen aufgerufen werden.

Die Variablen Mpt1 und Mpt2 enthalten die x/y-Koordinaten der Merkmalspunkte, wie Sie sie vom letzten Mal kennen

.

Um diese spaeter auf Korrelation testen zu koennen, muss ein Fenster um diese herum definiert werden koennen. Liegen die Merkmale zu nahe am Rand des Bildes, fehlen eventuell Pixel, um das Fenster komplett fuellen zu koennen. Entfernen Sie daher alle Merkmalspunkte die zu nahe an den Raendern des Bildes liegen und erstellen Sie zwei Variablen no\_pts1 und no\_pts2, in denen Sie die Anzahl der verbleibenden Merkmalspunkte speichern.

Beachten Sie, dass die x/y-Koordinaten des Merkmalspunkts im Zentrum des Fensters liegen sollen.

Geben Sie zum Testen in dieser Aufgabe die Zwischenergebnisse **statt** der Korrespondenzen zurueck:

Korrespondenzen = {no\_pts1, no\_pts2, Mpt1, Mpt2};

function Korrespondenzen = punkt\_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)

% In dieser Funktion sollen die extrahierten Merkmalspunkte aus einer

% Stereo-Aufnahme mittels NCC verglichen werden um Korrespondenzpunktpaare

% zu ermitteln.

%% Input parser aus Aufgabe 2.1

input\_parser

%% Merkmalsvorbereitung

% Maße des ersten Bildes

[zl1,sp1] = size(I1);

% Maße des zweiten Bildes

[zl2,sp2] = size(I2);

% Erlaubte minimale Distanz zum Bildrand.

s = (window\_length-1)/2;

% X- und Y-Korrdinaten der Merkmale auslesen.

X1 = Mpt1(1,:);

X2 = Mpt2(1,:);

Y1 = Mpt1(2,:);

Y2 = Mpt2(2,:);

% Erlaubter Wertebereich der x-Korrdinate wird durch die Spaltenanzahl des Bildes bestimmt.

% Finde die Indizes, an denen der Wert der x-Koordinate gütlig ist.

X1\_linindex = find(s < X1 & X1 <= sp1-s);

X2\_linindex = find(s < X2 & X2 <= sp2-s);

% Übrig gebliene Koordinaten bestimmen.

X1 = X1(X1\_linindex);

X2 = X2(X2\_linindex);

Y1 = Y1(X1\_linindex);

Y2 = Y2(X2\_linindex);

% Erlaubter Wertebereich der y-Korrdinate wird durch die Zeilenanzahl des Bildes bestimmt.

Y1\_linindex = find(s < Y1 & Y1 <= zl1-s);

Y2\_linindex = find(s < Y2 & Y2 <= zl2-s);

% Übrig gebliene Koordinaten bestimmen.

X1 = X1(Y1\_linindex);

X2 = X2(Y2\_linindex);

Y1 = Y1(Y1\_linindex);

Y2 = Y2(Y2\_linindex);

Korrespondenzen = {size(X1,2), size(X2,2), [X1;Y1], [X2;Y2]};

end

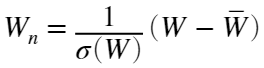
## 2.3 Normierung der Fenster

Schreiben Sie eine MATLAB-Funktion

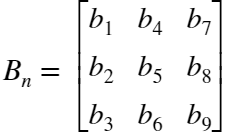
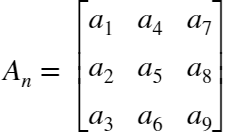
Korrespondenzen = punkt\_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)

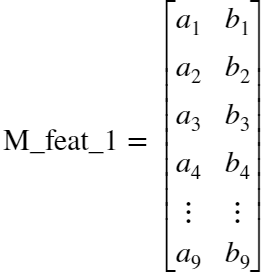
der zwei Grauwert-Bilder I1, I2, zwei Merkmalspunktmatrizen Mpt1, Mpt2 sowie optional die Parameter window\_length, min\_corr und do\_plot uebergeben werden koennen. Die Funktion soll die Regionen um die mitgelieferten Merkmalspunkte miteinander abgleichen, um Korrespondenzpunktpaare fuer weitere Stereo-Algorithmen zu gewinnen. Die Merkmalspunkte sollen mit dem in Modul 1 entwickelten Harris-Detektor extrahiert werden. Eine funktionierende harris\_detektor Funktion wird Ihnen bereit gestellt und kann wie unten zu sehen aufgerufen werden.

Da nach Aufagbe 2.2 nur noch Merkmale vorhanden sind, koennen nun problemlos Fensterbereiche aus den Ursprungsbildern heraus kopiert werden. Extrahieren Sie um jeden Merkmalspunkt in den Listen Mpt1, Mpt2 ein quadratisches Fenster mit der Seitenlaenge window\_length. Normieren Sie die Werte in diesem Fenster mit Hilfe der aus Vorlesung 1.4 Korrespondenzschaetzung (Seite 13) bekannter Methode:

Speichern Sie alle normierten Fensterdaten als stacked Vektor in einer Matrix Mat\_feat\_1 bzw. Mat\_feat\_2.

Angenommen in Mpt1 befinden sich zwei Merkmalspunkte, was in zwei normalisierten Fensterausschnitten C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6EB53DE5.tmpund C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\55158E3B.tmpresultiert. Die Seitenlaenge window\_length wird hier als 3 angenommen.

Somit wuerde die Matrix Mat\_feat\_1 wie folgt aussehen:

Geben Sie zum Testen in dieser Aufgabe die Zwischenergebnisse **statt** der Korrespondenzen zurueck:

Korrespondenzen = {Mat\_feat\_1, Mat\_feat\_2};

function Korrespondenzen = punkt\_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)

% In dieser Funktion sollen die extrahierten Merkmalspunkte aus einer

% Stereo-Aufnahme mittels NCC verglichen werden um Korrespondenzpunktpaare

% zu ermitteln.

%% Input parser aus Aufgabe 2.1

input\_parser

%% Merkmalsvorbereitung aus Aufabe 2.2

feature\_preprocessing

%% Normierung

% Erlaubte minimale Distanz zum Bildrand.

s = (window\_length-1)/2;

% Matrizen für stacked vectoren anlegen.

Mat\_feat\_1 = zeros(window\_length^2,no\_pts1);

Mat\_feat\_2 = zeros(window\_length^2,no\_pts2);

for i = 1:no\_pts1

% Quadratisches Fenster extrahieren.

W = double(I1(Mpt1(2,i)-s:Mpt1(2,i)+s,Mpt1(1,i)-s:Mpt1(1,i)+s));

% Mittelwert subtrahieren.

W = W - mean(W(:));

% Teilen durch die Standardabweichung

W = W/std(W(:));

% Normierte Matrix als Vektor speichern.

Mat\_feat\_1(:,i) = W(:);

end

% Analog zu oben: Normierung für Bild 2 durchführen.

for i = 1:no\_pts2

W = double(I2(Mpt2(2,i)-s:Mpt2(2,i)+s,Mpt2(1,i)-s:Mpt2(1,i)+s));

W = W - mean(W(:));

W = W/std(W(:));

Mat\_feat\_2(:,i) = W(:);

end

Korrespondenzen = {Mat\_feat\_1, Mat\_feat\_2};

end

## 2.4 Normierung der Fenster

Schreiben Sie eine MATLAB-Funktion

Korrespondenzen = punkt\_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)

der zwei Grauwert-Bilder I1, I2, zwei Merkmalspunktmatrizen Mpt1, Mpt2 sowie optional die Parameter window\_length, min\_corr und do\_plot uebergeben werden koennen. Die Funktion soll die Regionen um die mitgelieferten Merkmalspunkte miteinander abgleichen, um Korrespondenzpunktpaare fuer weitere Stereo-Algorithmen zu gewinnen. Die Merkmalspunkte sollen mit dem in Modul 1 entwickelten Harris-Detektor extrahiert werden. Eine funktionierende harris\_detektor Funktion wird Ihnen bereit gestellt und kann wie unten zu sehen aufgerufen werden.

Benutzen Sie nun die normierten Fensterausschnitte in Mat\_feat\_1 und Mat\_feat\_2, um die Normalized Cross Correlation Matrix NCC\_matrix zu berechnen. Die Formel fuer die Berechnung der NCC finden Sie in Vorlesung 1.4 Korrespondenzschaetzung auf Folie 14. Im Eintrag x,y der NCC\_matrix steht die Korrelation des Punktes X im zweiten Bild mit dem Punkt Y im ersten Bild. Bedenken Sie bei der Implementierung der NCC, dass sich das Skalarprodukt zweier MatrizenC:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\ABCEFEF3.tmpeffizient als vektorielles Skalarprodukt C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\69870879.tmprealisieren laesst.

Setzen Sie alle Eintraege in der NCC\_matrix zu 0, die unterhalb des Schwellwerts min\_corr liegen. Sortieren Sie die Indizes aller Korrespondenzwerte in NCC\_matrix, die nicht 0 sind, in absteigender Reihenfolge der Korrespondenzstaerke. (Der Index eines Elements einer Matrix ist die Nummer des Pixels, wenn man von links oben spaltenweise zu zaehlen beginnt.) Speichern Sie das Ergebnis im array sorted\_index.

Geben Sie zum Testen in dieser Aufgabe die Zwischenergebnisse **statt** der Korrespondenzen zurueck:

Korrespondenzen = {NCC\_matrix, sorted\_index};

function Korrespondenzen = punkt\_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)

% In dieser Funktion sollen die extrahierten Merkmalspunkte aus einer

% Stereo-Aufnahme mittels NCC verglichen werden um Korrespondenzpunktpaare

% zu ermitteln.

%% Input parser aus Aufgabe 2.1

input\_parser

%% Merkmalsvorbereitung aus Aufabe 2.2

feature\_preprocessing

%% Normierung aus Aufgabe 2.3

window\_normalization

%% NCC Brechnung

% NCC\_matrix:

% X-te Zeile entspricht X-tem Punkt in Bild2

% Y-te Zeile entspricht Y-tem Punkt in Bild1

NCC\_matrix = zeros(no\_pts2,no\_pts1);

for X = 1:no\_pts2

for Y = 1:no\_pts1

% Korrelation mittels Skalarprodukt berechnen.

NCC\_matrix(X,Y) = (1/(window\_length^2-1))\*(Mat\_feat\_2(:,X)'\*Mat\_feat\_1(:,Y));

end

end

% Eintraege in der NCC\_matrix zu 0 setzen, die unterhalb des Schwellwerts min\_corr liegen.

NCC\_matrix(find(NCC\_matrix < min\_corr)) = 0;

% Sortieren der Indizes aller Korrespondenzwerte in NCC\_matrix, die nicht 0 sind, in absteigender Reihenfolge nach der Korrespondenzstaerke.

% Dazu zunächst alle alle Indizes finden, bei denen die Korrespondenzstärke größer als 0 ist.

index = find(NCC\_matrix > 0);

% In absteigender Reihenfolge nach der Korrespondenzstaerke sortieren

[NCC\_matrix\_sort, sorted\_index] = sort(NCC\_matrix(index),'descend');

% Sortierung auf index-Vektor anwenden.

sorted\_index = index(sorted\_index);

Korrespondenzen = {NCC\_matrix, sorted\_index};

end

## 2.5 Korrespondezmatrix

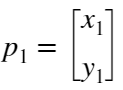
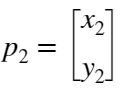
Schreiben Sie eine MATLAB-Funktion

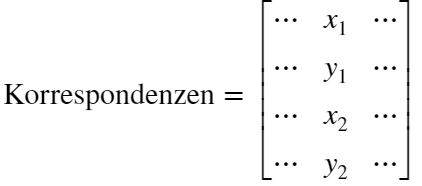
Korrespondenzen = punkt\_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)

der zwei Grauwert-Bilder I1, I2, zwei Merkmalspunktmatrizen Mpt1, Mpt2 sowie optional die Parameter window\_length, min\_corr und do\_plot uebergeben werden koennen. Die Funktion soll die Regionen um die mitgelieferten Merkmalspunkte miteinander abgleichen, um Korrespondenzpunktpaare fuer weitere Stereo-Algorithmen zu gewinnen. Die Merkmalspunkte sollen mit dem in Modul 1 entwickelten Harris-Detektor extrahiert werden. Eine funktionierende harris\_detektor Funktion wird Ihnen bereit gestellt und kann wie unten zu sehen aufgerufen werden.

Nutzen Sie nun sorted\_index und die NCC\_matrix, um die Korrespondenzpunktpaare zu bestimmen. Die Liste sorted\_index zeigt an, welche Punkte die staerksten Korrespondenzen aufweisen. Im Eintrag x,y der NCC\_matrix steht die Korrelation des Punktes X in Bild 2 mit dem Punkt Y im Bild 1. Wenn ein Korrespondenzpunkt in einer Spalte gefunden wurde, koennen Sie diese Spalte zu 0 setzen, damit ein Merkmalspunkt in Bild 1 nicht mehr als einem anderen Merkmalspunkt in Bild 2 zugewiesen werden kann. Somit koennen Sie auch pruefen, ob ein Eintrags in sorted\_index ueberhaupt noch zu bearbeiten ist.

Die Koordinaten der von zwei korrespondierenden Punkten speichern Sie in der Korrespondenzen Matrix C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\EAC179EF.tmp, wobei K die Anzahl der eindeutigen Korrespondenzen ist.

Seien und zwei korrespondierende Punkte, so tragen Sie diese in der Form

in die Korrespondenzen Matrix ein.

function Korrespondenzen = punkt\_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)

% In dieser Funktion sollen die extrahierten Merkmalspunkte aus einer

% Stereo-Aufnahme mittels NCC verglichen werden um Korrespondenzpunktpaare

% zu ermitteln.

%% Input parser aus Aufgabe 2.1

input\_parser

%% Merkmalsvorbereitung aus Aufabe 2.2

feature\_preprocessing

%% Normierung aus Aufgabe 2.3

window\_normalization

%% Normalized Cross Correlation aus Aufgabe 2.4

ncc\_calculation

%% Korrespondenz

% Leere Korrespondenzenmatrix erstellen. Sie kann nicht mehr Korrespondenzen enthalten als sorted\_index Indizes enthält.

Korrespondenzen = zeros(4,numel(sorted\_index));

% Anzahl gefundener Korrespondenzen

anz\_korr = 0;

% Laufvariable (Index des sorted\_index)

i = 0;

% Schleife nur solange auführen, wie...

% ...weitere Elemente in sorted\_index vorhanden sind,

% ...nicht jedem Merkmalspunkt in Bild 1 bereits ein Merkmalspunkt aus Bild 2 zugewiesen worden ist.

while i < numel(sorted\_index) && anz\_korr < no\_pts1

% Laufvariable / Index des sorted\_index um 1 erhöhen.

i = i+1;

% Prüfen, ob Korrespondenz nicht bereits auf 0 gesetzt werden ist bzw. Punkt Y nicht bereits einem Punkt X zugewiesen worden ist.

if NCC\_matrix(sorted\_index(i)) ~= 0

% Anzahl gefundener Korrespondenzen um 1 erhöhen

anz\_korr = anz\_korr + 1;

% Nummer der Merkmalspunkte auslesen.

[no\_merk\_B2, no\_merk\_B1] = ind2sub([no\_pts2,no\_pts1],sorted\_index(i));

% Korrespondezpunkt in die Matrix Korrespondenzen eintragen.

Korrespondenzen(:,anz\_korr) = [Mpt1(:,no\_merk\_B1);Mpt2(:,no\_merk\_B2)];

% Aktuelle Spalte der NCC\_matrix auf 0 setzen, damit ein Merkmalspunkt in Bild 1 nicht mehr als einem anderen Merkmalspunkt in Bild 2 zugewiesen werden kann.

NCC\_matrix(:,no\_merk\_B1) = 0;

end

end

% Überflüssige Nullspalten der Korrespondenzen-Matrix entfernen.

Korrespondenzen = Korrespondenzen(:,1:anz\_korr)

end

## 2.6 Plot

Schreiben Sie eine MATLAB-Funktion

Korrespondenzen = punkt\_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)

der zwei Grauwert-Bilder I1, I2, zwei Merkmalspunktmatrizen Mpt1, Mpt2 sowie optional die Parameter window\_length, min\_corr und do\_plot uebergeben werden koennen. Die Funktion soll die Regionen um die mitgelieferten Merkmalspunkte miteinander abgleichen, um Korrespondenzpunktpaare fuer weitere Stereo-Algorithmen zu gewinnen. Die Merkmalspunkte sollen mit dem in Modul 1 entwickelten Harris-Detektor extrahiert werden. Eine funktionierende harris\_detektor Funktion wird Ihnen bereit gestellt und kann wie unten zu sehen aufgerufen werden.

Wenn die Variable do\_plot den Wert true enthaelt, stellen Sie beide Bilder ueberlagert dar. Bild 2 liegt sozusagen mit einer Transparenz von 50% ueber Bild 1. Zeichnen Sie dann mit zwei verschiedenen Farben die gefunden korrespondierenden Punkte ein und verbinden Sie jedes zusammengehoerende Paar mit einer Linie.

function Korrespondenzen = punkt\_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)

% In dieser Funktion sollen die extrahierten Merkmalspunkte aus einer

% Stereo-Aufnahme mittels NCC verglichen werden um Korrespondenzpunktpaare

% zu ermitteln.

%% Input parser aus Aufgabe 2.1

input\_parser

%% Merkmalsvorbereitung aus Aufabe 2.2

feature\_preprocessing

%% Normierung aus Aufgabe 2.3

window\_normalization

%% Normalized Cross Correlation aus Aufgabe 2.4

ncc\_calculation

%% Korrespondenz aus Aufgabe 2.5

correspondence

%% Zeige die Korrespondenzpunktpaare an

if do\_plot

% Bild 1 darstellen

h = imshow(I1);

% Transparenz auf 50 % stellen.

set(h, 'AlphaData', 0.5);

% Alles in einer Figure darstellen:

hold on;

% Bild 2 darstellen;

h = imshow(I2);

% Transparenz auf 50 % stellen.

set(h, 'AlphaData', 0.5);

% Merkmalspunkte plotten:

plot(Korrespondenzen(1,:),Korrespondenzen(2,:), 'go');

plot(Korrespondenzen(3,:),Korrespondenzen(4,:), 'ro');

% Korrespondierende Merkmalspunkte mit Linie verbinden.

for i = 1:size(Korrespondenzen,2)

line([Korrespondenzen(1,i),Korrespondenzen(3,i)],[Korrespondenzen(2,i),Korrespondenzen(4,i)], 'LineWidth', 1);

end

end

end

# Hausaufgabe 3

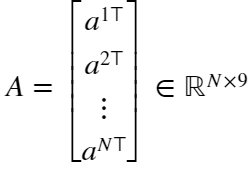
## 3.1 Achtpunktalgorithmus Teil 1

Schreiben Sie die Funktion

[EF] = achtpunktalgorithmus(Korrespondenzen, K)

welche aus einer Sammlung von Korrespondenzpunktpaaren ueber den Achtpunktalgorithmus die Essentielle Matrix bzw. die Fundamentalmatrix schaetzt. Welche Matrix tatsaechlich geschaetzt wird soll hierbei davon abhaengen, ob im Funktionsaufruf eine Kalibrierungsmatrix *K* mitgeliefert wird oder nicht. Beachten Sie bei der Implementierung die unterschiedlichen Eigenschaften von *E* und *F* und fuehren Sie wo noetig eine Fallunterscheidung durch.

Transfomrieren Sie die Korrespondenzpunkpaare in zwei Arrays mit homogenen kalibirierten Koordinaten x1 und x2. Beachten Sie den Zusammenhang zwischen kalibrierten Koordinaten C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\A24C6127.tmpund unkalibrierten Koordinaten C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\58F78CCD.tmp. Wenn eine Kalibrierungsmatrix *K*mituebergeben wird, muessen Sie die Koorindaten damit kalibirieren. Wenn kein K gegeben ist, dann muessen Sie die Koordinaten nur in homogene Koordinaten umwandeln.

Erstellen sie aus den homogenen kalibirierten Koordinaten nun die Matrix A, wobei gilt mit der SingulaerwertszerlegungC:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\4CC38869.tmp und C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\382C2F5F.tmp. Die Definition der Matrizen A und V finden Sie in Vorlesung 3-3 Achtpunktalgorithmus.

Geben Sie im ersten Teil der Implementierung die Matrix A, die Matrix V aus der Singulaerwertszerlegung von A und die homogenen kalibrierten Koordinaten x1 und x2 **statt** der Essentielen Matrix / Fundamentalmatrix zurueck

EF = {x1, x2, A, V};

function [EF] = achtpunktalgorithmus(Korrespondenzen, K)

% Diese Funktion berechnet die Essentielle Matrix oder Fundamentalmatrix

% mittels 8-Punkt-Algorithmus, je nachdem, ob die Kalibrierungsmatrix 'K'

% vorliegt oder nicht

% Umwandlung der Koordinaten der Korrespondenzpunktpaare in homogene Koordinaten.

Kor = [Korrespondenzen(1:2,:); ones(1,length(Korrespondenzen)); Korrespondenzen(3:4,:); ones(1,length(Korrespondenzen))];

% Bestimme Anzahl der Korrespondenzpunktpaare.

l\_Kor = length(Kor);

% Falls eine Kalibrierungsmatrix übergeben wurde: Berechne aus den unkalibrierten Koordinaten die kalibrierten Koordinaten.

if nargin == 2

for i = 1:l\_Kor

Kor(:,i) = [inv(K)\*Kor(1:3,i); inv(K)\*Kor(4:6,i)];

end

end

% Vorbelegung der Matrix A mit Nullen

A = zeros(l\_Kor,9);

% Kronecker-Produkt der Korrespondenzpunktpaare berechnen und zeilenweise der Matrix A hinzufügen.

for i = 1:l\_Kor;

A(i,:) = kron(Kor(1:3,i),Kor(4:6,i))';

end

% Berechne V der Singulärwertzerlegung von A.

[~,~,V] = svd(A);

EF ={Kor(1:3,:), Kor(4:6,:), A, V};

end

## 3.2 Achtpunktalgorithmus Teil 2

Schreiben Sie die Funktion

[EF] = achtpunktalgorithmus(Korrespondenzen, K)

welche aus einer Sammlung von Korrespondenzpunktpaaren ueber den Achtpunktalgorithmus die Essentielle Matrix bzw. die Fundamentalmatrix schaetzt. Welche Matrix tatsaechlich geschaetzt wird soll hierbei davon abhaengen, ob im Funktionsaufruf eine Kalibrierungsmatrix *K* mitgeliefert wird oder nicht. Beachten Sie bei der Implementierung die unterschiedlichen Eigenschaften von *E* und *F* und fuehren Sie wo noetig eine Fallunterscheidung durch.

Im zweiten Teil der Implementierung sollen sie aus der Singulaerwertszerlegung von A gewonnenen Matrix V bestehend aus den *Rechts-Singulaervektoren* die Essentielle Matrix bzw. die Fundamentalmatrix schaetzen. Wenn *K*angegeben ist soll die Essentielle Matrix zurueck gegeben werden, jedoch wenn kein *K*angegeben ist soll die Fundamentalmatrix zurueck gegeben werden.

function [EF] = achtpunktalgorithmus(Korrespondenzen, K)

% Diese Funktion berechnet die Essentielle Matrix oder Fundamentalmatrix

% mittels 8-Punkt-Algorithmus, je nachdem, ob die Kalibrierungsmatrix 'K'

% vorliegt oder nicht

%% Anfang Achtpunktalgorithmus aus Aufgabe 3.1

achtpunkt\_teil1

%% Schaetzung der Matrizen

% 9. Spalte von V der Singulärwertzerlegung von A liefer die Lösung zum Minimierungsproblem => Vorlesung 3\_3, Folie 8

% Bestimme Matrix G:

G = reshape(V(:,9),[3,3]);

% Projektion von G auf den Raum der essentiellen Matrizen.

% Bestimme U und V der Singulärwertzerlegung von G.

[U\_G,Z,V\_G] = svd(G);

% Fundamentalmatrix berechnen. (F-Matrix entspricht in dem Fall, bei dem eine K-Matrix übergeben wurde, der essentiellen Matrix.)

if nargin == 1

% Falls keine K-Matrix übergeben wurde, Fundamentalmatrix berechnen.

EF = U\_G\*diag([Z(1,1) Z(2,2) 0])\*V\_G';

else

% Falls eine K-Matrix übergeben wurde, essentielle Matrix berechnen.

EF = U\_G\*diag([1 1 0])\*V\_G';

end

end

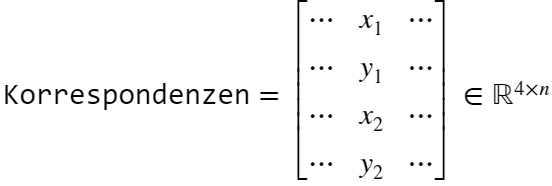
## 3.3 RanSaC Eingabeparameter

Implementieren Sie die Funktion

[Korrespondenzen\_robust] = F\_ransac(Korrespondenzen, varargin)

welche aus den mitgelieferten Korrespondenzpunktpaaren (in Pixelkoordinaten) einen Satz robuster Korrespondenzpunktpaare auswaehlt. Die Auswahl der geeigneten Untermenge von Korrespondenzpunktpaaren soll durch den **RanSaC**-Algorithmus realisiert werden, welcher die folgenden optional veraenderbaren Parameter verwendet:

* epsilon (numerisch, C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\DC12CFC5.tmp) geschaetzte Wahrscheinlichkeit, dass ein zufaellig gewaehltes Korrespondenzpunktpaar ein Ausreisser ist (Standardwert = 50,0%)
* p (numerisch, C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\31475B1B.tmp) gewuenschte Wahrscheinlichkeit, dass der Algorithmus einen Satz Korrespondenzpunktpaare liefert, in dem sich kein Ausreisser befindet (Standardwert = 50,0%)
* tolerance (numerisch) Toleranz, dass innerhalb dessen ein Korrespondenzpaar als zum Modell passend (Teil des Consensus-Sets) bewertet wird (Standardwert = 0.01)

Wandeln Sie danach die uebergebenen Korrespondenzpunktpaare aus der Varibale Korrespondenzen in zwei Variablen x1\_pixel und x2\_pixel C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\87DA3EE1.tmpum, welche die homogenen Pixelkoordinaten der Korrespondenzpunktpaare aus Bild 1 und Bild 2 enthalten sollen. Zur Erinnerung: 

Geben Sie die verarbeiteten Parameter und die homogenen Pixelkoordinaten zurueck:

Korrespondenzen\_robust = {epsilon, p, tolerance, x1\_pixel, x2\_pixel};

function [Korrespondenzen\_robust] = F\_ransac(Korrespondenzen, varargin)

% Diese Funktion implementiert den RANSAC-Algorithmus zur Bestimmung von

% robusten Korrespondenzpunktpaaren

% Input parser

par = inputParser;

% Standardwerte definieren

defaultepsilon = 0.5;

defaultp = 0.5;

defaulttolerance = 0.01;

% Bedingungen für gültige Inputs definieren:

validepsilon = @(x) isnumeric(x) && x<1 && x>0;

validp = @(x) isnumeric(x) && x<1 && x>0;

validtolerance = @(x) isnumeric(x);

% Optionale Parameterübergabe definieren:

addOptional(par,'epsilon',defaultepsilon, validepsilon);

addOptional(par,'p',defaultp, validp);

addOptional(par,'tolerance',defaulttolerance, validtolerance);

% Parsen durchführen.

parse(par,varargin{:});

x1\_pixel = [Korrespondenzen(1:2,:); ones(1,length(Korrespondenzen))];

x2\_pixel = [Korrespondenzen(3:4,:); ones(1,length(Korrespondenzen))];

Korrespondenzen\_robust = {par.Results.epsilon, par.Results.p, par.Results.tolerance, x1\_pixel, x2\_pixel};

end

## 3.4 RanSac Vorbereitung

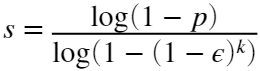
Implementieren Sie die Funktion

[Korrespondenzen\_robust] = F\_ransac(Korrespondenzen, varargin)

welche aus den mitgelieferten Korrespondenzpunktpaaren (in Pixelkoordinaten) einen Satz robuster Korrespondenzpunktpaare auswaehlt. Die Auswahl der geeigneten Untermenge von Korrespondenzpunktpaaren soll durch den **RanSaC**-Algorithmus realisiert werden.

Bereiten Sie folgende Variablen fuer den RanSaC Algorithmus vor:

Anzahl der benoetigten Puntke: C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\494E71BD.tmp

Die Iterationszahl: 

Eine Zustandsvariable, die die Anzahl der Korrespondenzen im bisher groessten Consensus-Set speichern soll. Initialisieren sie diese mit 0: largest\_set\_size

Eine Zustandsvariable, die die Sampson-Distanz des bisher groessten Consensus-Set speichern soll. Initialisieren sie diese mit ∞: largest\_set\_dist

Ein Zwischenspeicher für die Fundamentalmatrix mit der das bisher groesste Consensus-Set gefunden wurde. Initialisieren sie diese mit 3x3 Matrix gefuellt mit Nullern: largest\_set\_F

Geben Sie die folgende Variablen statt Korrespondenzen\_robust zurueck.

Korrespondenzen\_robust = {k, s, largest\_set\_size, largest\_set\_dist, largest\_set\_F};

function [Korrespondenzen\_robust] = F\_ransac(Korrespondenzen, varargin)

% Diese Funktion implementiert den RANSAC-Algorithmus zur Bestimmung von

% robusten Korrespondenzpunktpaaren

%% Input parser

input\_parser

%% RANSAC Algorithmus Vorbereitung

k = 8;

s = log(1-p)/log(1-(1-epsilon)^k);

largest\_set\_size = 0;

largest\_set\_dist = inf;

largest\_set\_F = zeros(3);

Korrespondenzen\_robust = {k, s, largest\_set\_size, largest\_set\_dist, largest\_set\_F};

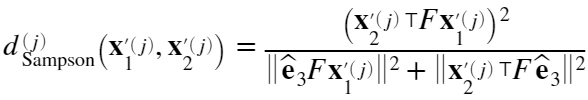
end

## 3.5 Sampson Distanz

Schreiben Sie die Funktion

sd = sampson\_dist(F, x1\_pixel, x2\_pixel)

welche die Sampson-Distanz fuer alle uebergebenen Korrespondanzpunktpaaren von Bild 1 und Bild 2 berechnet. Die Funktion soll die Sampson-Distanz sowohl fuer ein einzelnes Korrespondanzpunktpaar (C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\7BFFC459.tmp) als auch fuer eine ganze Reihe an Korrespondanzpunktpaaren (C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\B8FA00CF.tmp) ausgeben koennen.

Fuer das j-te Korrespondenzpunktpaar C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\BBB652B5.tmpberechnet sich die Sampson-Distanz zu wobei C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\120DF8D1.tmpdie schiefsymmetrische Matrix des Einheitsvektors C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\2575C407.tmpist.

Implementieren Sie diese Funktion effizient in Matlab und verwende Sie keine Schleifen.

function sd = sampson\_dist(F, x1\_pixel, x2\_pixel)

% Diese Funktion berechnet die Sampson Distanz basierend auf der

% Fundamentalmatrix F

% Vorher programmierten Dach-Operator verwenden.

e3\_dach = dach([0 0 1]);

f1 = e3\_dach\*F\*x1\_pixel;

f2 = (x2\_pixel'\*F\*e3\_dach)';

sd = (sum( (x2\_pixel'\*F)'.\*x1\_pixel ).^2) ./ (sum(f1.^2)+sum(f2.^2));

end

## 3.6 RanSac Algorithmus

Implementieren Sie die Funktion

[Korrespondenzen\_robust] = F\_ransac(Korrespondenzen, varargin)

welche aus den mitgelieferten Korrespondenzpunktpaaren (in Pixelkoordinaten) einen Satz robuster Korrespondenzpunktpaare auswaehlt. Die Auswahl der geeigneten Untermenge von Korrespondenzpunktpaaren soll durch den **RanSaC**-Algorithmus realisiert werden.

Führen Sie die folgenden Schritte in jeder Iteration C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\8FD252AD.tmpdurch:

1. Schaetzen Sie aus k zufaellig gewaelten Korrespondenzpunktpaaren die Fundamentalmatrix *F* mit Hilfe des zuvor implementierten Achtpunktalgorithmus.
2. Berechnen Sie die Sampson-Distanz effizient fuer **alle** Korrespondenzpunktpaaren, nicht nur fuer die k ausgwaelten. Diese Distanz erlaubt Aussagen ueber die Qualiteat des Consensus-Sets.
3. Ein Korrespondenzpunktpaar kann ins aktuelle Consensus-Set aufgenommen werden wenn gilt C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\28802643.tmp.
4. Bestimmen Sie für das aktuelle Consensus-Set die Anzahl der enthaltenen Paare und die absolute Set-Distanz indem sie die Sampson-Distanzen für das aktuelle Consensus-Set aufsummieren.
5. Vergleichen Sie diese mit der bisher groessten Set-Groesse gespeichert in largest\_set\_size. Ist das aktuelle Set groesser wird dieses als des neue groesste uebernommen. Sind beide gleich gross vergleichen Sie die Set-Distanz mit der des bisher groessten Sets gespeichter in largest\_set\_size. Das Set mit der kleineren absolut Sampson-Distanz soll das neue groesste Set werden. Speichern Sie auch die Fundamentalmatrix F mit der das neue groesste Set generiert wurde in largest\_set\_F.
6. Passen Sie die Werte fuer largest\_set\_size und largest\_set\_size nach jeder Iteration fuer das neue groesste Set an.

Wenn die Anzahl der Itterationen ausgeschoepft ist, geben Sie aus den Korrespondezen alle Korrespondenzpunktpaaren zurueck die zum besten Consensus-Set gehoeren:

Korrespondenzen\_robust C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\2E1ABC49.tmpwobei n die groesse des besten Consensus-Sets ist.

Geben Sie zu **testzwecken** nicht nur die robusten Korrespondenzen sondern auch die largest\_set\_F Matrix zurueck.

Korrespondenzen\_robust = {Korrespondenzen\_robust, largest\_set\_F}

Weitere Hilfestellung zum RanSaC Algorithmus finden Sie unter:

* An Invitation to 3-D Vision (S 388) <https://link-springer-com.eaccess.ub.tum.de/book/10.1007/978-0-387-21779-6>
* https://de.wikipedia.org/wiki/RANSAC-Algorithmus
* Moodle-Kurs

function [Korrespondenzen\_robust] = F\_ransac(Korrespondenzen, varargin)

% Diese Funktion implementiert den RANSAC-Algorithmus zur Bestimmung von

% robusten Korrespondenzpunktpaaren

%% Input parser

input\_parser

%% RANSAC Algorithmus Vorbereitung

ransac\_preparation

%% RANSAC Algorithmus

% Indizes des besten Consesus-Sets. (Spaltenindizes von Korrespondenzen)

best\_ind\_cons\_set = 1;

i = 1;

while i <= s

% 8 zufällige Korrespondenzpunktpaare auswählen.

kor\_rand = Korrespondenzen(:,randi(size(Korrespondenzen,2),[8,1]));

% Fundamentale Matrix schätzen.

F = achtpunktalgorithmus(kor\_rand);

% Sampson-Distanz für alle Korrespondenzpunktpaare berechnen.

sd = sampson\_dist(F, x1\_pixel, x2\_pixel);

% Indizes des aktuellen Consensus-Sets bestimmen.

ind\_cons\_set = find(sd < tolerance);

% Aktuelle Anzahl der im Consensus-Set enthaltenen Paare und akutelle Set-Distanz bestimmen.

anz\_cons = length(ind\_cons\_set);

set\_dist = sum(sd(ind\_cons\_set));

% Vergleichen Sie diese mit der bisher groessten Set-Groesse gespeichert in largest\_set\_size.

% Ist das aktuelle Set groesser wird dieses als des neue groesste uebernommen.

% Sind beide gleich gross vergleichen Sie die Set-Distanz mit der des bisher groessten Sets gespeichter in largest\_set\_size.

% Das Set mit der kleineren absolut Sampson-Distanz soll das neue groesste Set werden.

if anz\_cons > largest\_set\_size || (anz\_cons == largest\_set\_size && set\_dist < largest\_set\_dist)

largest\_set\_size = anz\_cons;

largest\_set\_dist = set\_dist;

% Indizes des besten Consesus-Sets abspeichern. (Spaltenindizes von Korrespondenzen)

best\_ind\_cons\_set = ind\_cons\_set;

% Speichern Sie auch die Fundamentalmatrix F mit der das neue groesste Set generiert wurde in largest\_set\_F.

largest\_set\_F = F;

end

i = i+1;

end

Korrespondenzen\_robust = Korrespondenzen(:,best\_ind\_cons\_set);

Korrespondenzen\_robust = {Korrespondenzen\_robust, largest\_set\_F};

end

## 3.7 Essentielle Matrix

In dieser Aufgabe schreiben Sie keine Funktion sondern Skript, dass folgende Aufgaben erfuellt:

* Extrahieren Sie zunaechst Korrespondenzpunkte aus dem vorgegebenen Bildpaar mit Hilfe der vorherigen Hausaufgaben.
* Finden Sie robuste Korrespondenzpunktpaare mit Hilfe der in dieser Hausaufgabe implementierten Funktionen. Achten Sie darauf, dass die Punkte tatsaechlich allgemein genug im Raum liegen und passen Sie gegebenenfalls die Parameter der Merkmalsextraktion und der Korrespondenzschaetzung an! Hierfuer empfiehlt es sich die robuten Korrespondenzen wie in Aufgabe 2.6 zu plotten.
* Laden Sie jetzt die gegebene Kameramatrix *K* und berechnen Sie die Essentielle Matrix *E* aus den zuvor bestimmten robusten Korrespondenzen.

%% Bilder laden

I1 = imread('szeneL.png');

Image1 = rgb\_to\_gray(I1);

I2 = imread('szeneR.png');

Image2 = rgb\_to\_gray(I2);

%% Harris-Merkmale berechnen

% Mindestabstand und Kachelgröße erhöhen, Anzahl Merkmale pro Kachel um 1 senken.

Merkmale1 = harris\_detektor(Image1, 'min\_dist', 40, 'tile\_size', 250, 'N', 4);

Merkmale2 = harris\_detektor(Image2, 'min\_dist', 40, 'tile\_size', 250, 'N', 4);

%% Korrespondenzschätzung

% Verkleinere den unteren Schwellwert fuer die Staerke der Korrelation zweier Merkmale, um mehr Korrepondenzen für die spätere Auswertung zu erhalten

Korrespondenzen = punkt\_korrespondenzen(Image1,Image2,Merkmale1,Merkmale2, 'min\_corr', 0.85, 'do\_plot', true);

%% Finde robuste Korrespondenzpunktpaare mit Hilfe des RANSAC-Algorithmus

% Geschaetzte Wahrscheinlichkeit, dass ein zufaellig gewaehltes Korrespondenzpunktpaar ein Ausreisser ist, ist niedriger als der Standarwert von 50%.

% Höhere Wahrscheinlichkeit, dass der Algorithmus einen Satz Korrespondenzpunktpaare liefert, in dem sich kein Ausreisser befindet, als Standardwert von 50% erwünscht.

% Toleranz anpassen.

Korrespondenzen\_robust = F\_ransac(Korrespondenzen, 'epsilon', 0.3, 'p', 0.95, 'tolerance', 2);

%% Zeige die robusten Korrespondenzpunktpaare

figure;

% Bild 1 darstellen

h = imshow(Image1);

% Transparenz auf 50 % stellen.

set(h, 'AlphaData', 0.5);

% Alles in einer Figure darstellen:

hold on;

% Bild 2 darstellen;

h = imshow(Image2);

% Transparenz auf 50 % stellen.

set(h, 'AlphaData', 0.5);

% Merkmalspunkte plotten:

plot(Korrespondenzen\_robust(1,:),Korrespondenzen\_robust(2,:), 'go');

plot(Korrespondenzen\_robust(3,:),Korrespondenzen\_robust(4,:), 'ro');

% Korrespondierende Merkmalspunkte mit Linie verbinden.

for i = 1:size(Korrespondenzen\_robust,2)

line([Korrespondenzen\_robust(1,i),Korrespondenzen\_robust(3,i)],[Korrespondenzen\_robust(2,i),Korrespondenzen\_robust(4,i)], 'LineWidth', 1);

end

size(Korrespondenzen\_robust,2)

%% Berechne die Essentielle Matrix

K = load('K.mat');

K = K.K;

E = achtpunktalgorithmus(Korrespondenzen\_robust, K);

disp(E)

# Hausaufgabe 4

## 4.1 Euklidische Bewegung

Schreiben Sie die Funktion

[T1,R1,T2,R2] = TR\_aus\_E(E)

die aus einer Essentiellen Matrix *E* die moeglichen Rotationen und Translationen C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\CA61EE3F.tmpund C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\932651A5.tmpberechnet und diese zurueck gibt.

Bestimmen Sie zunaechst die Singulaerwertszerlegung von *E* und stellen Sie sicher, dass *U* und *V* Rotationsmatrizen sind.

Berechnen Sie dann die beiden moeglichen Ergebnisse fuer R und T und geben sie beide (C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\98D497FB.tmp) zureuck.

Hinweis: Betrachten Sie den Nullraum von C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\16F7EEC1.tmpoder nutzen Sie die Formeln aus der Vorlesung.

Geben Sie zu testzwecken auch die Rotationsmatrizen *U* und *V* zurueck.

[T1,R1,T2,R2,U,V] = TR\_aus\_E(E)

function [T1, R1, T2, R2, U, V]=TR\_aus\_E(E)

% Diese Funktion berechnet die moeglichen Werte fuer T und R

% aus der Essentiellen Matrix

% Singulärwertzerlegung von E bestimmen.

[U1,Z,V1] = svd(E);

% Sicherstellen, dass U und V Rotationsmatrizen sind.

U = U1\*diag([1 1 det(U1)]);

V = V1\*diag([1 1 det(V1)]);

% R und T\_dach mit den Formeln von Folie 4 aus Vorlesung 3\_4 bestimmen.

Rz(:,:,1) = [0 -1 0; 1 0 0; 0 0 1];

Rz(:,:,2) = [0 1 0; -1 0 0; 0 0 1];

% Jeweils zwei Lösungen für R und T:

for i=1:2

R(:,:,i) = U\*Rz(:,:,i)'\*V';

T\_d = U\*Rz(:,:,i)\*Z\*U';

% T\_dach in T umwandeln:

T(:,i) = [T\_d(3,2); T\_d(1,3); T\_d(2,1)];

end

T1 = T(:,1);

R1 = R(:,:,1);

T2 = T(:,2);

R2 = R(:,:,2);

end

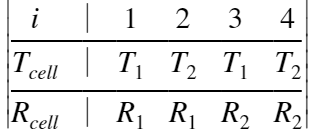
## 4.2 Rekonstruktion Vorbereitung

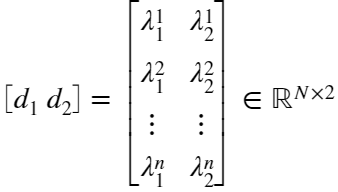
Über eine Tiefenschätzung kann die korrekte euklidische Transformation und somit eine 3D-Rekonstruktion der Szene bestimmt werden. Implementieren Sie eine Funktion

[T, R, lambda, P1] = rekonstruktion(T1, T2, R1, R2, Korrespondenzen, K)

welche die möglichen euklidischen Transformationen, die Korrespondenzpunktpaare sowie die Kalibrierungsmatrix einliest.

Es gibt vier Moeglichkeiten um C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\D4CB5F77.tmpund C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\23B22F9D.tmpzu kombinieren. Erstellen Sie daher zwei cell-arrays T und R um die Kombinationsmoeglichkeiten abzubilden:

Transfomrieren Sie die Korrespondenzpunkpaare C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\CDB37039.tmpin zwei Arrays mit homogenen kalibirierten Koordinaten Arrays x1 und x2 C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\F35AF7AF.tmpwobei N die Anzahl der Korrespondenzpunktpaare ist. Benutzen Sie dafür die Kalibrierungsmatrix K um von unkalibrierten Koordinaten C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\3F71CC95.tmpauf kalibrierte Koordinaten C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\EB57006B.tmpzum kommen.

Intitalisieren Sie zusaetzlich ein cell-array d\_cell mit vier Zellen, die jeweils eine Nx2 Matrix gefuellt mit Nullern beinhalten soll. Hier sollen speater die Werte fuer die Tiefeninformationen gespeichert werden.

Geben sie zu **Testzwecken** folgende Werte zurueck:

[T\_cell, R\_cell, d\_cell, x1, x2] = rekonstruktion(T1, T2, R1, R2, Korrespondenzen, K)

function [T\_cell, R\_cell, d\_cell, x1, x2] = rekonstruktion(T1, T2, R1, R2, Korrespondenzen, K)

%% Preparation

% Erstellen Sie daher zwei cell-arrays T und R um die Kombinationsmoeglichkeiten abzubilden

T\_cell = {T1, T2, T1 T2};

R\_cell = {R1, R1, R2, R2};

% Anzahl Korrespondenzpunktpaare

N = size(Korrespondenzen,2);

% Homogene kalibrierte Koordinaten berechnen:

x1 = inv(K)\*[Korrespondenzen(1:2,:); ones(1,N)];

x2 = inv(K)\*[Korrespondenzen(3:4,:); ones(1,N)];

% Cell\_array mit vier Zellen initialisieren:

d\_cell = cell(1,4);

for i=1:4

d\_cell{i} = zeros(N,2);

end

end

## 4.3 Rekonstruktion Gleichungssysteme

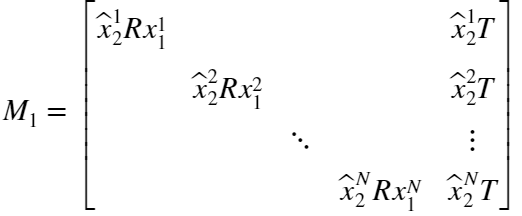
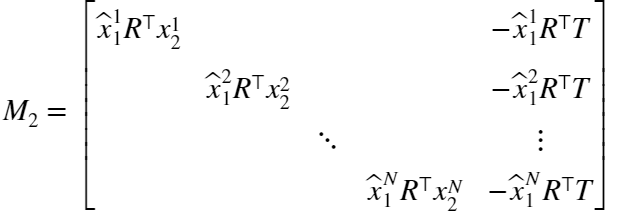
Über eine Tiefenschätzung kann die korrekte euklidische Transformation und somit eine 3D-Rekonstruktion der Szene bestimmt werden. Implementieren Sie eine Funktion

[T, R, lambda, P1] = rekonstruktion(T1, T2, R1, R2, Korrespondenzen, K)

welche die möglichen euklidischen Transformationen, die Korrespondenzpunktpaare sowie die Kalibrierungsmatrix einliest.

Durch die vier moeglichen Kombinationen aus C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\89D596E7.tmpund C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\83DD088D.tmpmuessen wir 4 mal das Gleichungssystem C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5EEB723.tmpaufstellen, um die Tiefeninformation lambda fuer die 3D Punkte in Cameraframe 1 zu erhalten. Gleiches gilt fuer das Gleichungssystem C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\E928E029.tmp, dass uns die Tiefeninformation fuer die 3D Punkte in Cameraframe 2 liefert.

Iterieren Sie druch alle Kombinationen T\_cell{i} und R\_cell{i}. Fuehren Sie fuer jede Kombination folgende Schritte durch:

1. Stellen Sie die Matrizen und .
2. Bestimmen Sie die Loesungen fuer die Gleichungssysteme C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6BF544DB.tmpund C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\26AA8EA1.tmpmittels Singulaerwertszerlegung und normieren sie die Vektoren C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\7C2B6A57.tmpsodass gilt C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\A701DD7D.tmp.
3. Kopieren Sie dann die Vektoren C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\9A458993.tmpin das cell-array d\_cell{i} C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\229A0C19.tmpohne die *γ* Werte.

Bestimmen Sie abschliessend die Kombination von R und T, die die meisten positiven Tiefeninformaitonen erzeugt hat. Hierfuer koennen Sie das cell-array d\_cell nutzen.

Geben Sie die besten gefundenen Werte fuer R C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\189C5E8F.tmp, T C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\F9973675.tmpund C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\8916654B.tmpzurueck. Um zu testen, ob das Aufstellen der Gleichungssysteme funtioniert, geben Sie auch die Matrizen C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\71B13891.tmpund C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\2523D9C7.tmpfuer die letzte Kombination C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\EB8FAE6D.tmpund C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\177AB803.tmpzurueck.

[T, R, lambda, M1, M2] = rekonstruktion(T1, T2, R1, R2, Korrespondenzen, K)

function [T, R, lambda, M1, M2] = rekonstruktion(T1, T2, R1, R2, Korrespondenzen, K)

%% Vorbereitung aus Aufgabe 4.2

preparation

% Anzahl Korrespondenzpunktpaare

N = size(Korrespondenzen,2);

% Vobelegung von M1 und M2 mit Nullen.

M1 = zeros(3\*N,N+1);

M2 = M1;

% Initialisierung: Anzahl positiver Tiefeneinträge.

anz\_pos = 0;

% Initialisierung: Index der Kombination mit den meisten posivite Tiefeneinträgen.

i\_max\_komb = 0;

% Iterieren druch alle Kombinationen T\_cell{i} und R\_cell{i}

for i = 1:length(T\_cell)

% Berechnung von x2\_dach\*R\*x1 = x2 x (R\*x1) für M1 und entsprechende Rechnung für M2

% Kreuzprodukt wird spaltenweise berechnet

KP1 = cross(x2,R\_cell{i}\*x1);

KP2 = cross(x1,R\_cell{i}'\*x2);

% Matrizen M1 und M2 erstellen:

for k = 1:N

M1(3\*k-2:3\*k,k) = KP1(:,k);

M2(3\*k-2:3\*k,k) = KP2(:,k);

M1(3\*k-2:3\*k,N+1) = dach(x2(:,k))\*T\_cell{i};

M2(3\*k-2:3\*k,N+1) = -dach(x1(:,k))\*R\_cell{i}'\*T\_cell{i};

end

[~,~,V1] = svd(M1);

[~,~,V2] = svd(M2);

% Lösung der Gleichungssysteme ist die rechtsseitigen Singulärvektor zum kleinsten Singulärwert (hier (N+1)-te Spalte von V)

% Normierung mit dem (N+1)ten Eintrag der Lösungsvektoren.

d1 = (1/V1(N+1,N+1))\*V1(:,N+1);

d2 = (1/V2(N+1,N+1))\*V2(:,N+1);

d\_cell{i} = [d1(1:N), d2(1:N)];

% Anzahl positiver Tiefeneinträge bestimmen.

anz = length(find(d\_cell{i}>0));

% Falls mehr positive Tiefeneinträge als bisher gefunden wurden, Anzahl und Index speichern.

if anz > anz\_pos

anz\_pos = anz;

i\_max\_komb = i;

end

end

% Translation, Rotation und Lambda der Kombination zurückgeben, die zuvor mit i\_max\_komb bestimmt worden ist.

T = T\_cell{i\_max\_komb};

R = R\_cell{i\_max\_komb};

lambda = d\_cell{i\_max\_komb};

end

## 4.4 Darstellung

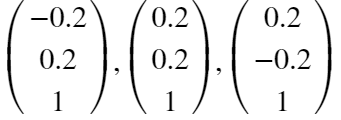
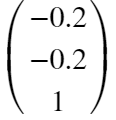
Über eine Tiefenschätzung kann die korrekte euklidische Transformation und somit eine 3D-Rekonstruktion der Szene bestimmt werden. Implementieren Sie eine Funktion

[T, R, lambda, P1] = rekonstruktion(T1, T2, R1, R2, Korrespondenzen, K)

welche die möglichen euklidischen Transformationen, die Korrespondenzpunktpaare sowie die Kalibrierungsmatrix einliest.

Berechnen Sie die Weltkoordinaten der Bildpunkte P1 aus den Tielfeninformationen lambda und den Bildkoordinaten x1.

Zeichnen Sie die Weltkoordinaten P1 mit Hilfe der Funktion scatter3() und numerieren Sie diese im Bild mit der text() Funktion.

Gehen Sie davon aus, dass der Cameraframe 1 in der Z=1 Ebene des Weltkoordinatensystem liegt. Die Ecken des Cameraframes 1 sollen bei und liegen. Speichern Sie diese in einer 3x4 Matrix camC1 und berechnen Sie mit Hilfer dieser Matrix und der euklidischen Bewegung die Weltkoordinaten des Cameraframes 2. Speichern Sie diese in der 3x4 Matrix camC2.

Zeichnen Sie nun die Quadrate fuer Cameraframe 1 (blau) und 2 (rot) mit der Funktion plot3 in die Szene. Beschriften Sie die Cameraframes in 3D mit "Cam1" bzw. "Cam2". Setzen Sie die Kamerapoisiton zu C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\4F2565.tmpund den Kameraaufpunkt zu C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\E6E161BB.tmp. Beschriften Sie die Achsen mit X,Y und Z und aktivieren Sie das Koordinatengitter.

Geben sie zu Testzwecken die Koordinatenmatrizen fuer beide Frames mit aus:

[T, R, lambda, P1, camC1, camC2] = rekonstruktion(T1, T2, R1, R2, Korrespondenzen, K)

function [T, R, lambda, P1, camC1, camC2] = rekonstruktion(T1, T2, R1, R2, Korrespondenzen, K)

% Diese Funktion schaetzt die Tiefeninformationen und bestimmt somit die korrekte

% euklidische Bewegung R und T. Zusaetzlich gibt sie die Weltkoordinaten der Bildpunkte

% bezueglich Bild 1 zurueck und deren Tiefeninformationen.

%% Vorbereitung aus Aufgabe 4.2

preparation

%% R, T und lambda aus Aufgabe 4.3

R\_T\_lambda

% EGENTLICHE AUFGABE

% Anzahl Korrespondenzpunktpaare

N = size(Korrespondenzen,2);

figure;

% Weltkoordinaten der Bildpunkte P1 aus den Tiefeninformationen lambda und den Bildkoordinaten x1 berechnen.

P1 = zeros(3,N);

for i = 1:N

P1(:,i) = lambda(i,1)\*x1(:,i);

scatter3(P1(1,i), P1(2,i), P1(3,i));

str = sprintf('%d', i);

text(P1(1,i),P1(2,i),P1(3,i), str);

hold on;

end

% Ecken des Cameraframes 1 speichern.

camC1 = [-0.2 0.2 0.2 -0.2; 0.2 0.2 -0.2 -0.2; ones(1,4)];

% Cameraframe 1 einzeichnen

plot3([camC1(1,:),camC1(1,1)], [camC1(2,:),camC1(2,1)], [camC1(3,:), camC1(3,1)], 'b');

text(camC1(1,1),camC1(2,1), camC1(3,1), 'Cam1');

hold on;

% Cameraframe mit Rotation und Translation berechnen: P1 = R'\*P2-R'\*T = R'\*(P2-T); P2 entspricht hier camC1 bzw. den Koordinaten der Eckpunkte eines Frames bzgl. des jeweiligen Kooridnantenursprungs.

camC2 = R' \* (camC1 - repmat(T, [1 4]));

% Cameraframe 2 einzeichnen

plot3([camC2(1,:),camC2(1,1)], [camC2(2,:),camC2(2,1)], [camC2(3,:), camC2(3,1)], 'r');

text(camC2(1,1),camC2(2,1), camC2(3,1), 'Cam2');

% Achsen beschriften

xlabel('X-Achse');

ylabel('Y-Achse');

zlabel('Z-Achse');

% Kameraposition setzen.

campos([43 -22 -87]);

% Kameraaufpunkt setzen.

camup([0 -1 0]);

% Koordinatengitter aktivieren:

grid on;

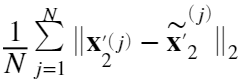
end

## 4.5 Rückprojektion

Abschließend soll die Qualität der 3D-Rekonstruktion quantitativ ueber den Rückprojektionsfehler in Kamera2 bestimmt werden. Implementieren Sie in der Funktion

[repro\_error] = rueckprojektion(Korrespondenzen, P1, Image2, T, R, K)

folgende Schritte:

* Berechnen Sie die homogenen Bildkoordinaten C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\97EA1E81.tmp, die Projektion der aus Bild 1 rekonstruierten 3D-Punkte C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\AFD5E537.tmpin Kamera 2. Bestimmen Sie hierzu erst die Weltkoordinaten von C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\7B57B5D.tmpim Cameraframe 2.
* Zeigen Sie Bild 2 an und zeichnen sie die gefundenen Merkmale, sowie die berechneten, rueckprojezierten Punkte so ein, dass die Herkunft des Punktes sowie die paarweise Zuordnung klar erkennbar ist.
* Berechnen Sie den mittleren Rueckprojektionsfehler und liefern Sie diesen als Funktionswert zurueck.

Geben Sie zu Testzwecken auf die rueckprojezierten Punkte C:\Users\Andre\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\7AAB97F9.tmpals x2\_repro zusaetzlich zum mittleren Rueckprojektionsfehler zurueck:

[repro\_error, x2\_repro] = rueckprojektion(Korrespondenzen, P1, Image2, T, R, K)

function [repro\_error, x2\_repro] = rueckprojektion(Korrespondenzen, P1, Image2, T, R, K)

% Diese Funktion berechnet den mittleren Rueckprojektionsfehler der

% Weltkooridnaten P1 aus Bild 1 im Cameraframe 2 und stellt die

% korrekten Merkmalskoordinaten sowie die rueckprojezierten grafisch dar.

% Weltkorrdinaten von P1 im Cameraframe 2 bestimmen:

P2 = R\*P1+T;

% Homogene Pixelkoordinaten x2\_hom durch Normierung von Z=1 bestimmen.

N = size(P2,2);

x2homp = zeros(3,N);

for i = 1:N

x2homp(:,i) = K\*(1/P2(3,i))\*P2(:,i);

end

figure;

% Bild 2 darstellen

imshow(Image2);

hold on;

% Originale Merkmalspunkte des Bildes 2 plotten:

plot(Korrespondenzen(3,:),Korrespondenzen(4,:), 'go');

% Rückprojezierte Merkmalspunkte plotten;

plot(x2homp(1,:),x2homp(2,:), 'yo');

% Orginigale und rückprojeziert Merkmalspunkte paarweise verbinden.

for i = 1:N

line([Korrespondenzen(3,i),x2homp(1,i)],[Korrespondenzen(4,i),x2homp(2,i)], 'LineWidth', 1);

% Linie beschriften

str = sprintf('%d', i);

t = text(Korrespondenzen(3,i),Korrespondenzen(4,i), str, 'Color', 'red');

end

% Mittleren Rueckprojektionsfehler berechnen: Differenz bilden, elementweise quadrieren, spaltenweise Summe berechnen, elementweise Quadratwurzel berechnen, Mittelwert bestimmen

repro\_error = mean(sqrt(sum((Korrespondenzen(3:4,:)-x2homp(1:2,:)).^2)));

x2\_repro = x2homp;

end