

BIV Projekt

Gruppe: 4

Mitglieder:

Pascal Egner (1626852)
Bastian Böhm (1630242)
Richard Musebrink (1625590)

Grundlagen	4
Feature	4
Feature Detection	4
Blob Detektoren	4
Corner Detektoren	4
Feature Description	4
Feature Matching	5
Funktionalität der Applikation	5
Unterstützte Blob Detektoren	5
Unterstützte Corner Detektoren	5
Anleitung der Applikation	6
Quelltext der Applikation - Erläuterungen	10
Code	10
Bild zuschneiden	10
Umwandlung in Graustufen	10
Auswahl des Feature Detection Verfahrens	11
Darstellung gefundener Features	11
Feature Extraction	12
Feature Matching	12
Darstellung von Feature-Paaren	12
Validierung gefundener Features	13
Evaluation	14
Testbild 1: Box (zwei verschiedene Bilder)	14
Verfahren	14
BRISK	14
ORB	14
SURF	15
Auswertung:	15
Testbild 2: Elephant (zwei verschiedene Bilder)	15
Verfahren	15
BRISK	15
FAST	16
Harris	16
KAZE	16
minEigen	17
ORB	17
SURF	17
Auswertung	18
Testbild 3: HS Mannheim Streetview (zwei verschiedene Perspektiven)	19

Verfahren	19
BRISK	19
FAST	19
Harris	19
KAZE	20
ORB	20
SURF	20
Auswertung	20
Testbild 4: Buchseite (zwei verschiedene Perspektiven)	21
Verfahren	21
BRISK	21
FAST	21
KAZE	21
minEigen	21
ORB	21
SURF	22
Auswertung	22
Testbild 5: Sonnenblumen (Ausschnitt aus dem Originalbild)	23
Verfahren	23
KAZE	23
ORB	23
SURF	23
Auswertung	23
Testbild 6: Buch (zwei verschiedene Szenen)	24
Verfahren	24
SURF	24
BRISK	24
Auswertung	24
Testbild 7: Stereo-Endoskopie (leicht versetzte Kamera)	25
Verfahren	25
KAZE	25
Orb	25
SURF	26
FAST	26
Harris	26
Auswertung	27
Fazit	27
Reflektion	28
Ausblick	28
Zusatz: Anwendungsbeispiel mit Puzzleteilen	29

Grundlagen

Feature

Features sind **spezifische Strukturen im Bild**, also beispielsweise Punkte, Kanten oder ganze Objekte. In der Regel sucht man nach möglichst beschreibenden Features, also solche, bei denen Verwechslungen sehr unwahrscheinlich sind.

Feature Detection

Feature Detection beschreibt das Identifizieren von Features in einem gegebenen Bild, wofür es eine Vielzahl an Verfahren gibt. Grob können die in unserer Arbeit verwendeten Detection Verfahren in zwei Kategorien unterteilt werden, die sogenannten *Blob*- und *Corner*-Detektoren.

Blob Detektoren

Blob-Erkennungsmethoden erkennen *Regionen* in einem digitalen Bild, die sich in ihren Eigenschaften, wie Helligkeit oder Farbe, von den umgebenden *Regionen* unterscheiden. Ein Blob ist also ein **Bereich eines Bildes**, in dem einige Eigenschaften konstant oder annähernd konstant sind; alle Bildpunkte in einem Blob können in gewissem Sinne als einander ähnlich betrachtet werden.

Corner Detektoren

Die Corner (Ecken) Detection versucht, Punkte in einem digitalen Bild zu identifizieren, an denen sich die Bildintensität stark ändert. Die Punkte, an denen sich die Bildintensität am stärksten ändert, sind typischerweise Ecken, welche charakterisch für ein Merkmal im Bild sein können.

Feature Description

Bei der Feature Description geht es darum gefundene Features zu beschreiben (eine solche Beschreibung nennt man Feature-Deskriptor). Die Anforderungen an Feature-Deskriptoren sind abhängig vom Anwendungszweck. Während eine möglichst genaue Beschreibung immer gewünscht ist, muss eventuell zwischen Genauigkeit und Kompaktheit abgewogen werden. Solche Beschreibungen können dann z.B. für das Vergleichen von Punkten genutzt werden (Feature Matching).

Die Form eines Feature-Deskriptors ist abhängig von dem verwendeten Feature Detection Verfahren. Ein Feature wird jedoch im allgemeinen bei Corner Detektoren durch die Kantenorientierung, die Gradientengröße und bei Blob Detektoren durch die Polarität und die Stärke des Blobs beschrieben.

Feature Matching

Der Abgleich von Merkmalen oder allgemein der Bildabgleich ist der Vorgang, Ähnlichkeiten zwischen zwei Bildern zu finden z.B. dieselbe Szene aus verschiedenen Blickwinkeln. Dabei werden mittels *Feature Description* und *Feature Extraction* Verfahren Feature-Deskriptoren berechnet und mit Feature-Deskriptoren anderer Bilder verglichen um Merkmalsübereinstimmungen zwischen Bildern zu finden.

Funktionalität der Applikation

Die entwickelte MATLAB Applikation ermöglicht den Vergleich verschiedener Feature Detection Verfahren und kann auch Feature Matching durchführen. Der Vergleich wird durch das Anwenden verschiedener Verfahren über eine hierfür optimierte Benutzeroberfläche ermöglicht. Die Ergebnisse von Feature Detection und Matching sind in der Applikation visualisiert.

Die nächsten beiden Abschnitte führen die unterstützten Verfahren, aufgeschlüsselt nach *Blob-* und *Corner*-Detektoren auf.

Unterstützte Blob Detektoren

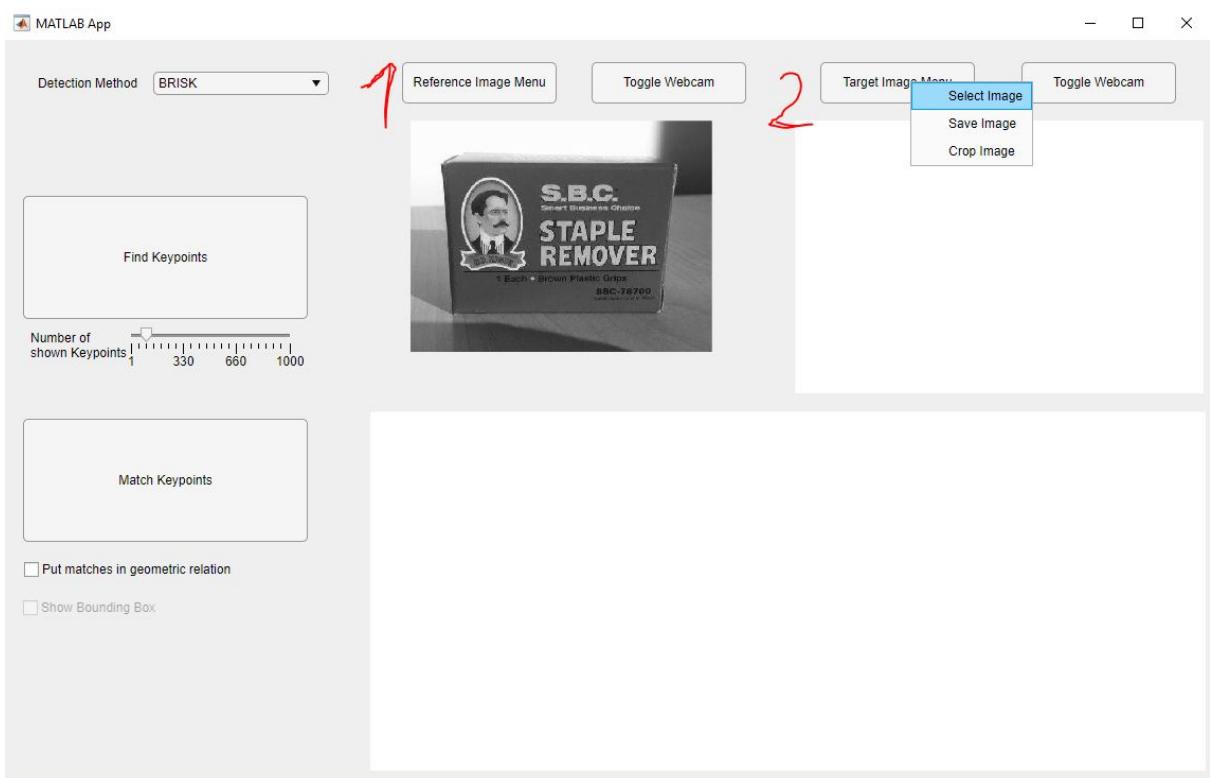
1. Speeded up robust features (SURF)
2. KAZE

Unterstützte Corner Detektoren

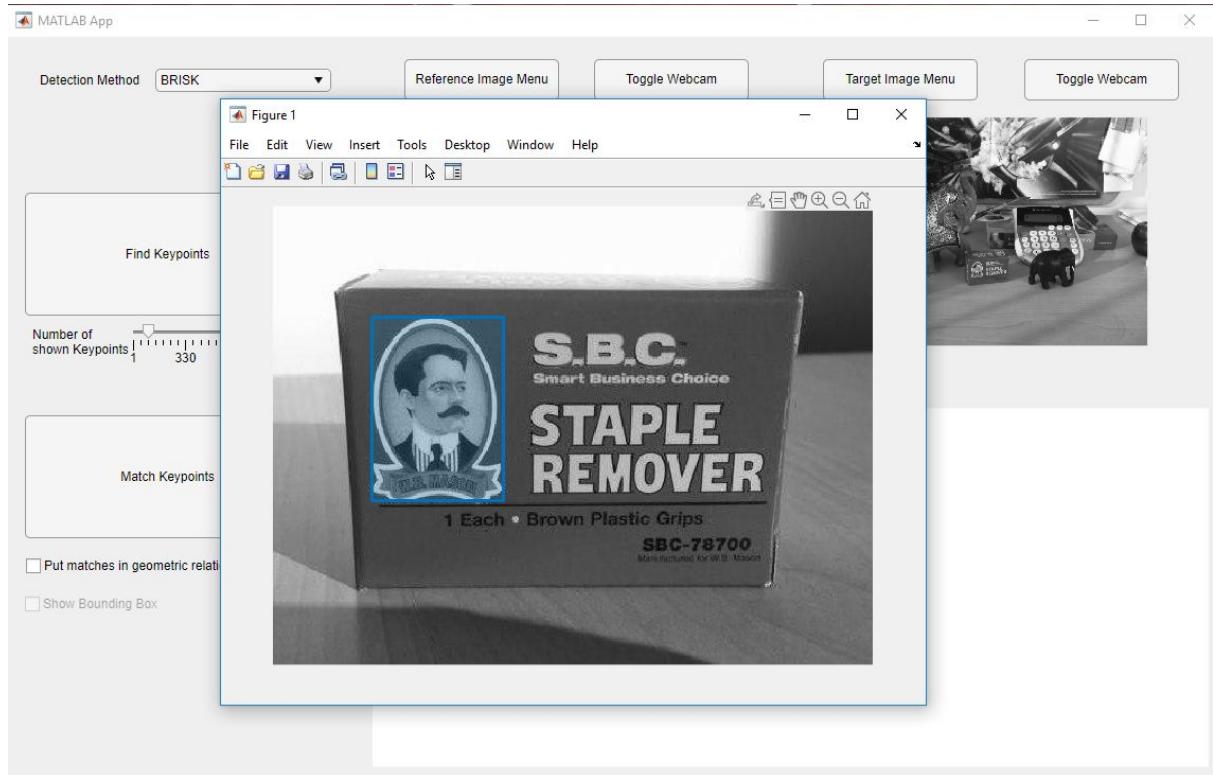
1. Minimum Eigenvalue Algorithmus (MinEigen)
2. Features from Accelerated Segment Test (FAST)
3. Oriented FAST and Rotated BRIEF (ORB)
4. Harris-Stephens-Algorithmus (Harris)
5. Binary Robust Invariant Scalable Keypoints (BRISK)

Anleitung der Applikation

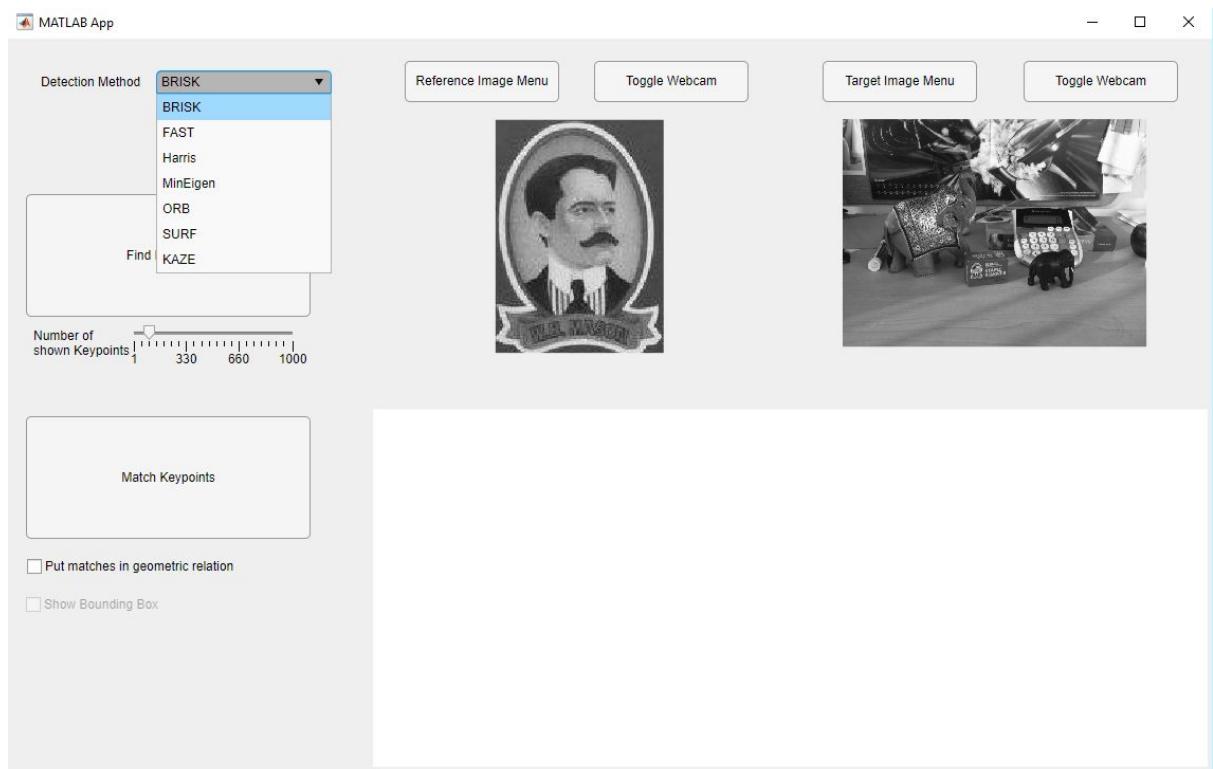
1. Zunächst müssen Bilder ausgewählt werden auf denen Features erkannt werden sollen. Dafür muss **Rechtsklick** auf “Reference Image Menu” gedrückt werden und unter dem Punkt “Select Image” ein Bild ausgewählt werden. Das “Reference Image (1)” ist dabei ein Bild der Szene die auf dem “Target Image (2)” gefunden werden soll. Alternativ kann auch die Webcam genutzt werden um Bilder zu machen. Die Kamera muss zu diesem Zweck erst aktiviert werden. Nachdem ein Ausschnitt gewählt wurde wird die Kamera wieder abgewählt, wodurch der aktuelle Ausschnitt übernommen wird. Alle Bilder können anschließend mit “Save Image” gespeichert werden. Dies ermöglicht es auch einen Bildausschnitt für einen späteren Zeitpunkt abzuspeichern.



2. Falls der Fokus des Bildes angepasst werden soll, kann unter dem Punkt "Crop Image", ein Teilbereich des Bilder zur weiteren Verarbeitung ausgewählt werden.

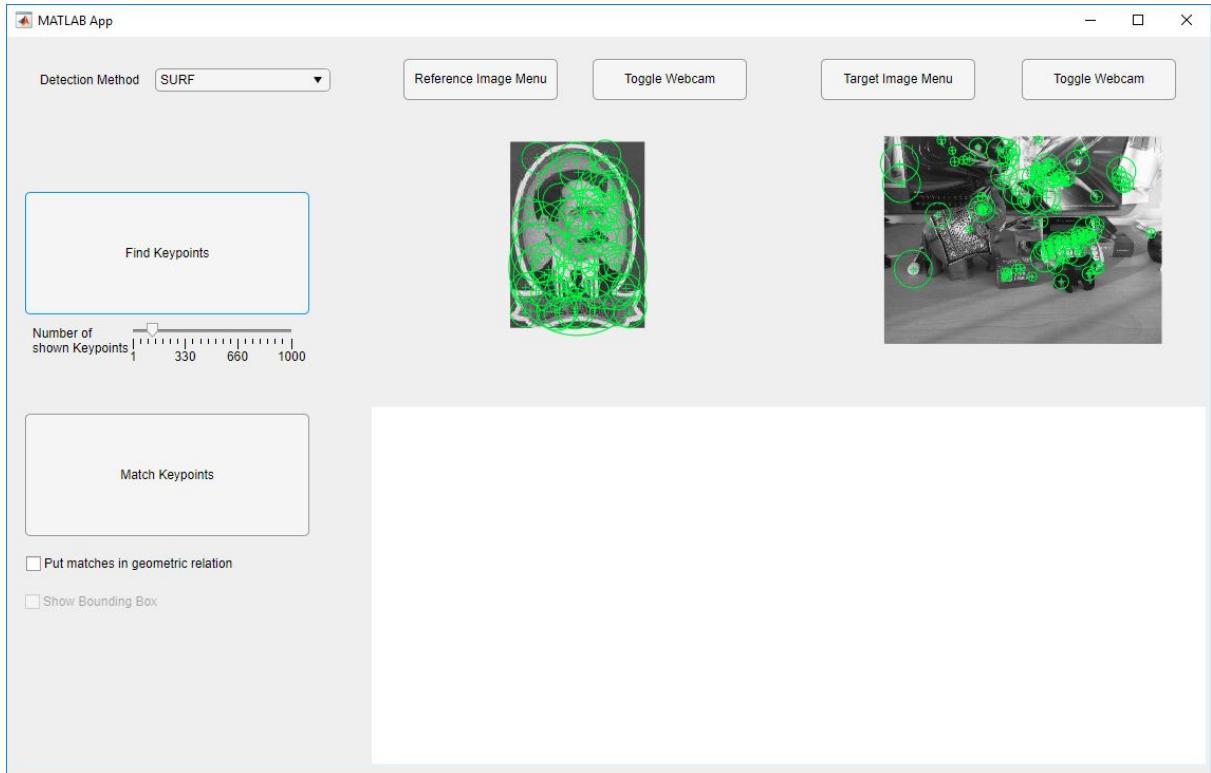


3. Über das Dropdown Menu "Detection Method", kann die gewünschte Methode zur Feature Detection ausgewählt werden.

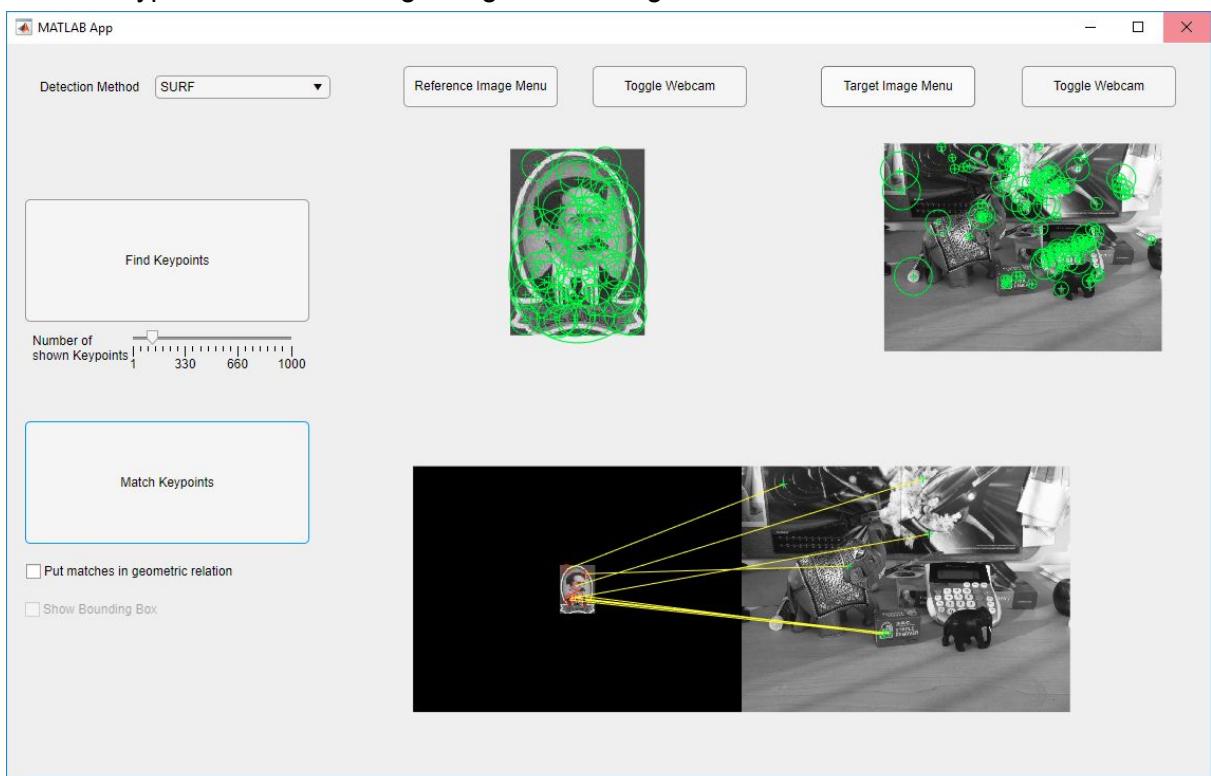


4. Wenn zwei Bilder ausgewählt wurden können über den Knopf "Find Keypoints" die Features der Bilder mit dem ausgewählten Verfahren gefunden werden. Dies

geschieht auch automatisch wenn ein Verfahren ausgewählt wird. Über den Slider "Number of shown Keypoints" kann eingestellt werden wie viele Keypoints (beginnend mit dem Stärksten) angezeigt werden sollen.

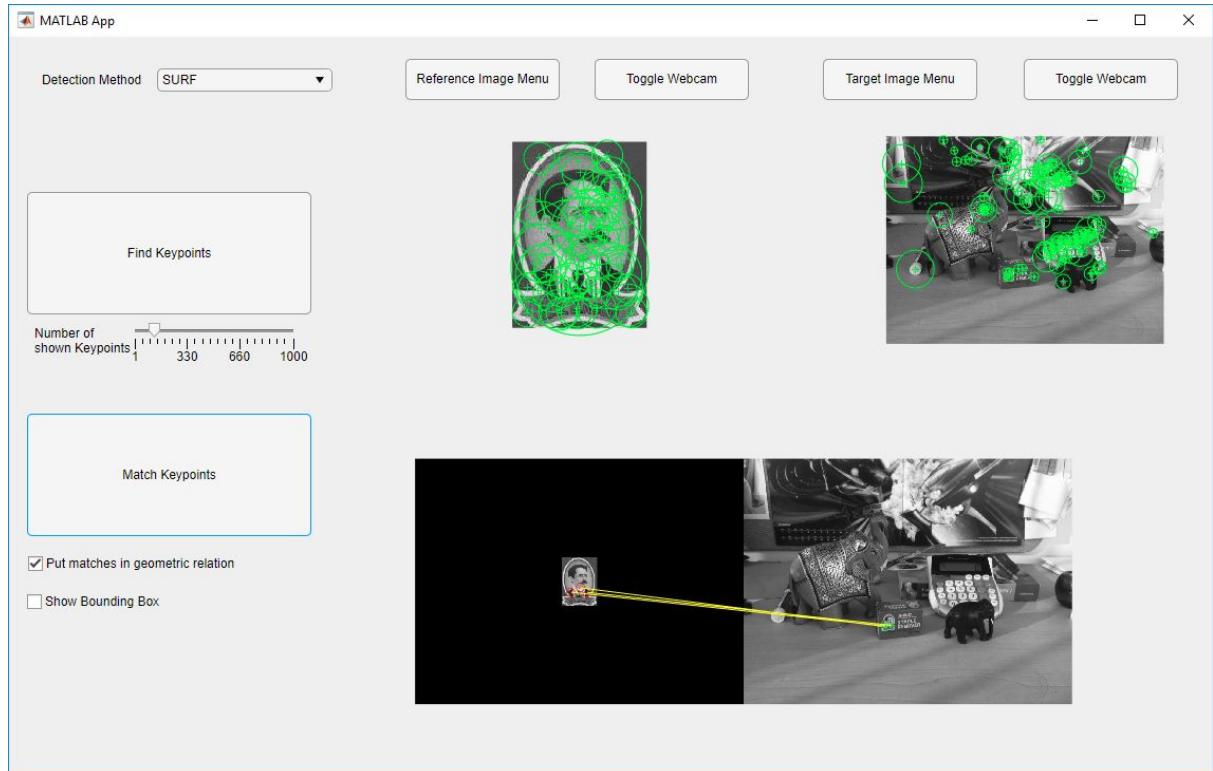


5. Erst wenn für beide Bilder, Features gefunden wurden, können über den Knopf "Match Keypoints" zueinander gehörige Features gefunden werden.

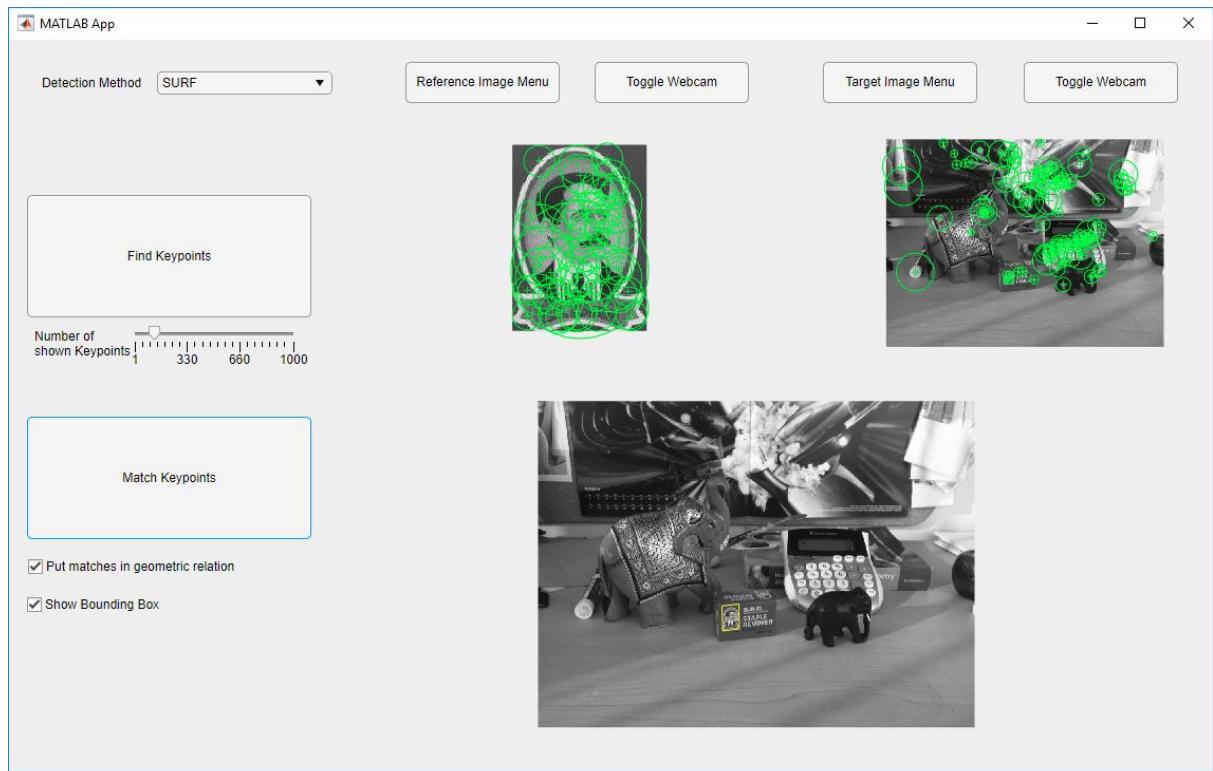


6. Bei Aktivierung der Checkbox "Put matches in geometric relation" werden nur noch solche Feature Paare angezeigt, welche in einem logischen Zusammenhang zu

benachbarten Paaren stehen (für die eine einheitliche Transformation zwischen den gegebenen Bildern existiert). Dies sortiert fehlerhafte Paare aus und erhöht die Genauigkeit des Feature Matching.



- Über die Checkbox "Show Bounding Box" kann eine Bounding Box um die gefundene Stelle im Zielbild eingezeichnet werden. Diese Option ist nur verfügbar wenn "Put matches in geometric relation" aktiviert wurde.



Quelltext der Applikation - Erläuterungen

Die Codestruktur einer MATLAB App ist streng vorgegeben und eignet sich nicht zur Erklärung, daher bilden wir hier die wichtigsten Abschnitte ab und geben Beschreibungen, wo nötig.

Code

Bild zuschneiden

Falls der Nutzer nur einen Ausschnitt vom Bild haben möchte, kann hiermit das Bild zurecht geschnitten werden.

```
%-Bild zurecht schneiden-----
preview = imshow(image);
roi = drawrectangle;
crop = imcrop(image,roi.Position);
```

Umwandlung in Graustufen

Da die Feature-Detection Verfahren nur mit Graustufenbildern funktionieren, müssen die Ausgewählten Bilder in ein solches umgewandelt werden.

```
%-Umwandeln in Graustufenbild, falls nicht schon geschehen-----
if(size(target_image_data, 3) == 3)
    target_image_data_grey = rgb2gray(target_image_data);
else
    target_image_data_grey = target_image_data;
end
```

Auswahl des Feature Detection Verfahrens

Das über ein Drop-Down ausgewählte Detektion Verfahren wird auf beide geladenen Bilder angewendet, wir erhalten dabei jeweils ein Set von Feature-Punkten.

Die Applikation bietet alle in MATLAB implementierten Feature Detection Verfahren, mit Ausnahme von MSER. MSER nutzt eine besondere Darstellung und ist daher für einen direkten Vergleich ungeeignet.

```
%--Das ausgewählte Detektionsverfahren anwenden-----

switch DetectionMethodDropDown.Value
    case "BRISK"
        referencePoints = detectBRISKFeatures(reference_image_data_grey);
        targetPoints = detectBRISKFeatures(target_image_data_grey);
    case "FAST"
        referencePoints = detectFASTFeatures(reference_image_data_grey);
        targetPoints = detectFASTFeatures(target_image_data_grey);
    case "Harris"
        referencePoints = detectHarrisFeatures(reference_image_data_grey);
        targetPoints = detectHarrisFeatures(target_image_data_grey);
    case "MinEigen"
        referencePoints = detectMinEigenFeatures(reference_image_data_grey);
        targetPoints = detectMinEigenFeatures(target_image_data_grey);
    case "ORB"
        referencePoints = detectORBFeatures(reference_image_data_grey);
        targetPoints = detectORBFeatures(target_image_data_grey);
    case "SURF"
        referencePoints = detectSURFFeatures(reference_image_data_grey);
        targetPoints = detectSURFFeatures(target_image_data_grey);
    case "KAZE"
        referencePoints = detectKAZEFeatures(reference_image_data_grey);
        targetPoints = detectKAZEFeatures(target_image_data_grey);
end
```

Darstellung gefundener Features

Die gefundenen Feature-Punkte werden über das jeweilige Bild gezeichnet. Die Anzahl der Punkte ist auf die X stärksten Punkte begrenzt, wobei die Anzahl X über einen Slider eingestellt werden kann.

```
%--Die gefundenen Feature Punkte anzeigen-----

hold(ReferenceImage, "on");
plot(referencePoints.selectStrongest(round(NumberOfShownKeypointsSlider.Value)), ReferenceImage);
hold(ReferenceImage, "off");

hold(TargetImage, "on");
plot(targetPoints.selectStrongest(round(NumberOfShownKeypointsSlider.Value)), TargetImage);
hold(TargetImage, "off");
```

Feature Extraction

Um später Feature Matching durchführen zu können, werden Feature-Deskriptoren für die gefundenen Feature-Punkte extrahiert.

```
%-Feature Deskriptoren extrahieren-----  
[referenceFeatures, referencePoints] = extractFeatures(reference_image_data_grey, referencePoints);  
[targetFeatures, targetPoints] = extractFeatures(target_image_data_grey, targetPoints);
```

Feature Matching

Durch Anwendung von Feature Matching erhalten wir Punkt Paare zwischen den beiden Bildern, welche ähnliche Deskriptoren aufweisen (Feature Paare).

```
%-Zueinander passende Punkte mithilfe der Deskriptoren finden-----  
pairs = matchFeatures(referenceFeatures, targetFeatures);
```

Darstellung von Feature-Paaren

Mit der *showMatchedFeatures* Funktion können wir einander zugeordnete Punkte anzeigen lassen, die Korrelation wird mit Verbindungslien zwischen den Punkten visualisiert.

```
%-Vermeintlich übereinstimmende Merkmale anzeigen-----  
matchedReferencePoints = referencePoints(pairs(:, 1), :);  
matchedTargetPoints = targetPoints(pairs(:, 2), :);  
  
showMatchedFeatures(reference_image_data, target_image_data, matchedReferencePoints, ...  
matchedTargetPoints, 'montage', "Parent", ResultImage);
```

Validierung gefundener Features

Hier wird mit Hilfe der [*estimateGeometricTransform*](#) Funktion ein logischer Zusammenhang zwischen den Feature-Punkten gesucht. Grob wird dabei eine Transformation zwischen den Feature-Punkten im ersten zu denen im zweiten Bild berechnet. Gleichzeitig werden Ausreißer entfernt, die mit der kalkulierten Transformation nicht in Einklang zu bringen sind. Dies ermöglicht die Erstellung einer Bounding Box, welche die Position des gesuchten Ausschnittes im Zielbild anzeigt.

```
%-Transformation aus den übereinstimmende Merkmale berechnen und als Bounding Box anzeigen-----

[tform, estimatedReferencePoints, estimatedTargetPoints] = ...
estimateGeometricTransform(matchedReferencePoints, matchedTargetPoints, 'affine');

foundPolygon = [1, 1;...                                % top-left
                size(reference_image_data, 2), 1;...    % top-right
                size(reference_image_data, 2), size(reference_image_data, 1);...
                1, size(reference_image_data, 1);...    % bottom-right
                1, 1];                                % bottom-left
                                                % top-left

newFoundPolygon = transformPointsForward(tform, foundPolygon);

imshow(target_image_data, 'Parent', ResultImage);
hold(ResultImage, "on");
line(ResultImage, newFoundPolygon(:, 1), newFoundPolygon(:, 2), 'Color', 'y');
hold(ResultImage, "off");
```

Evaluation

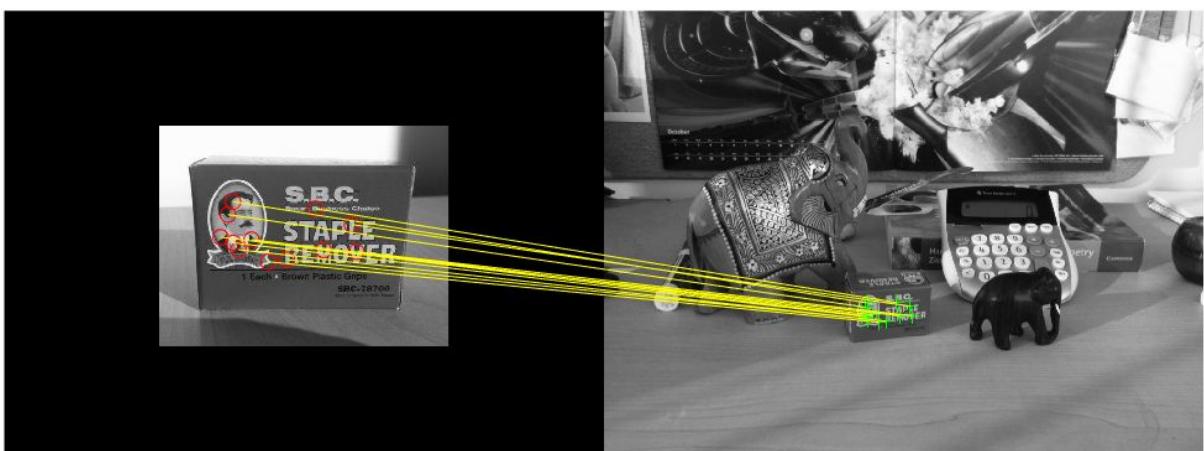
Die einzelnen Verfahren wurden auf eine Reihe von Testbildern angewendet und ihre Ergebnisse miteinander verglichen. Aufgeführt sind nur die Verfahren, bei denen das Feature Matching erfolgreich war.

Bei allen Beispielen, kam *estimateGeometricTransform* zum Einsatz um falsche Paare zu minimieren. Alle verwendeten Bilder sowie weitere (wie z.B. weniger blutige Stereo-Endoskopie Bilder) sind im beigefügten Bilder Ordner.

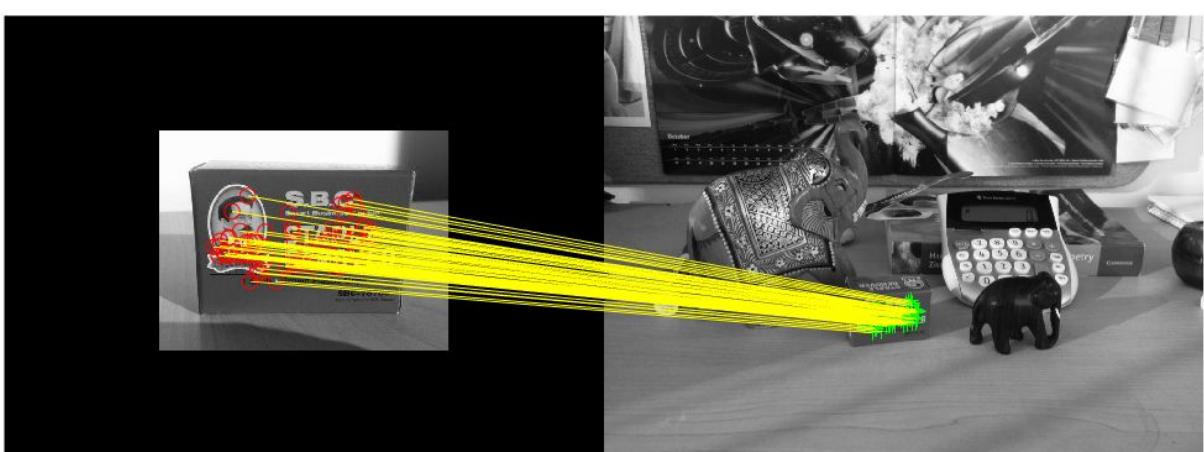
Testbild 1: Box (zwei verschiedene Bilder)

Verfahren

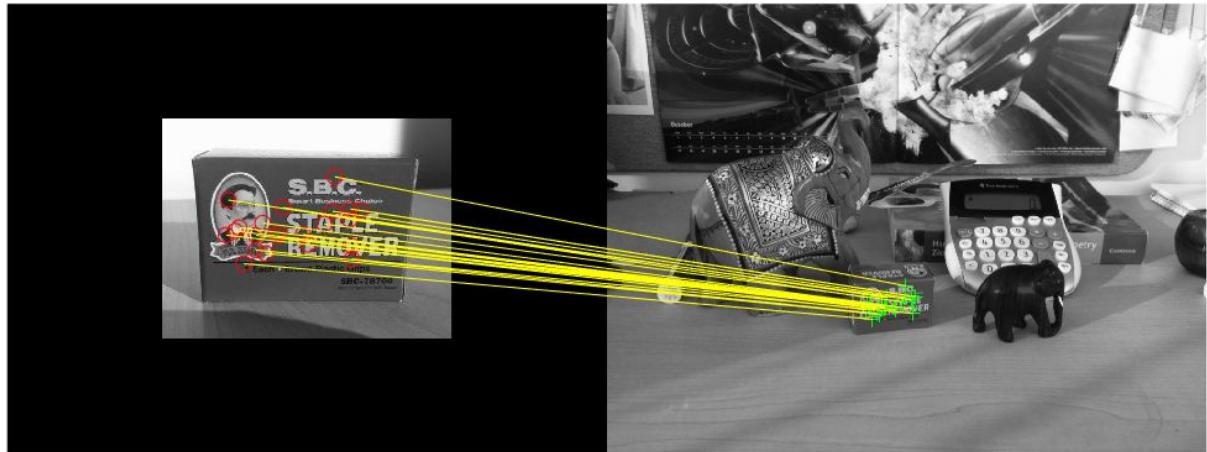
BRISK



ORB



SURF



Auswertung:

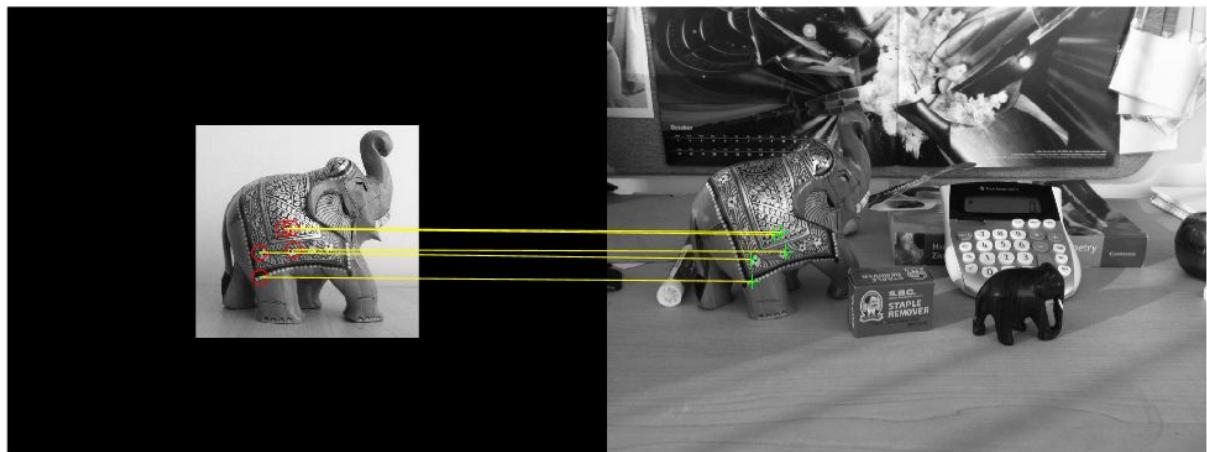
BRISK, ORB und SURF liefern mehr als annehmbare Ergebnisse. Alle anderen Verfahren, habe keine Paare gefunden.

Mit BRISK und ORB als Corner Detektoren und SURF als Blob Detektor funktionieren beide Verfahrensarten gut. Irritiert hat uns hier, dass Corner Detektoren wie Harris keine Paare gefunden haben, obwohl das Beispielbild aufgrund der Schrift eigentlich viele scharfe Ecken hat.

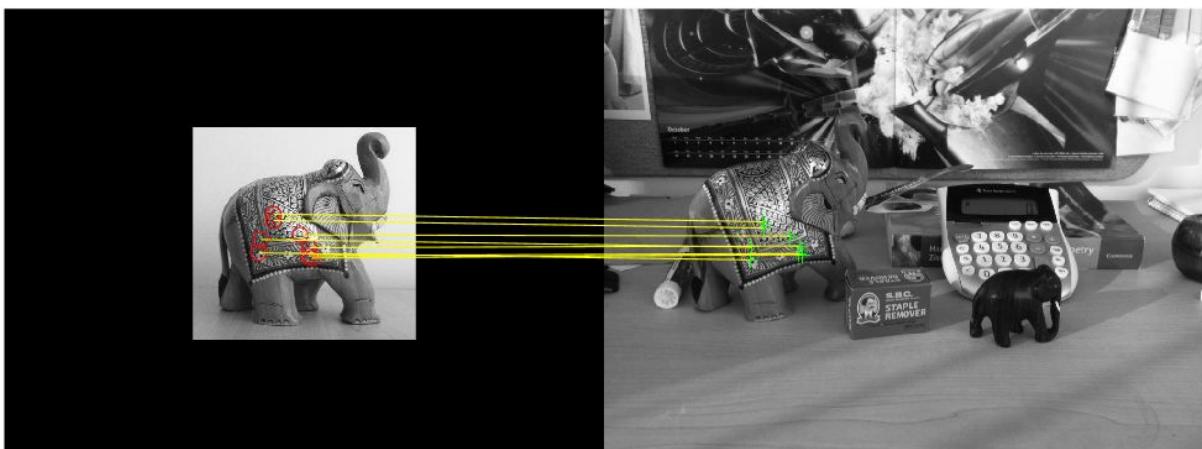
Testbild 2: Elephant (zwei verschiedene Bilder)

Verfahren

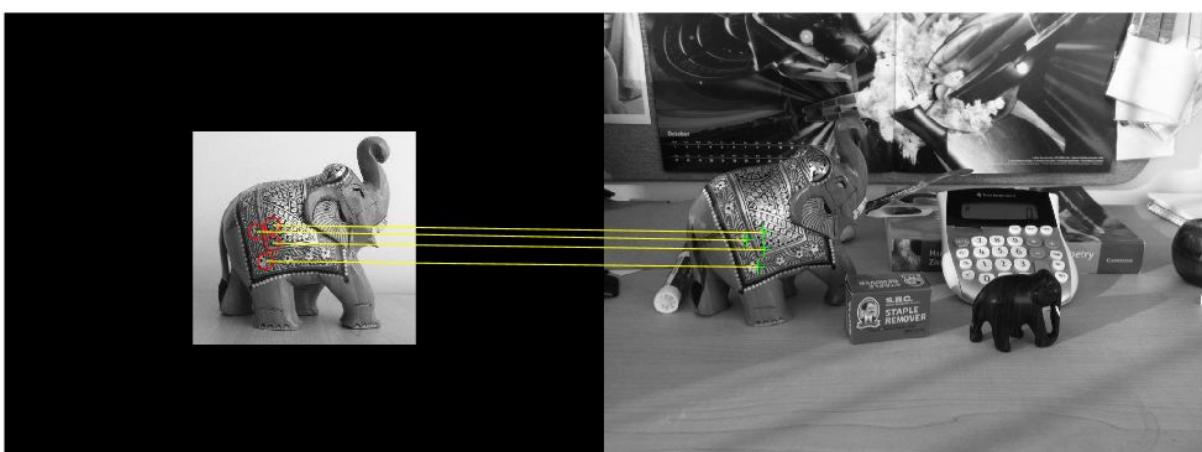
BRISK



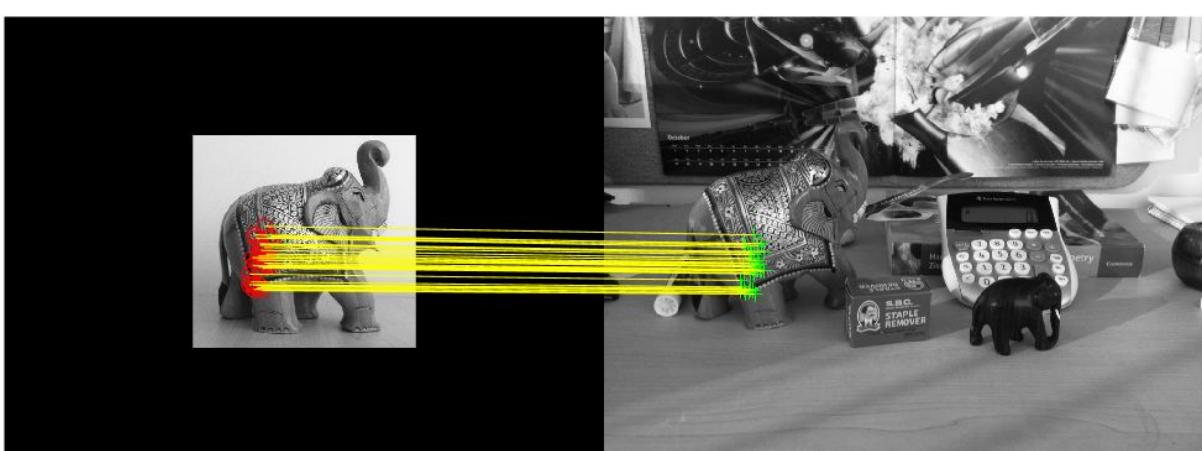
FAST



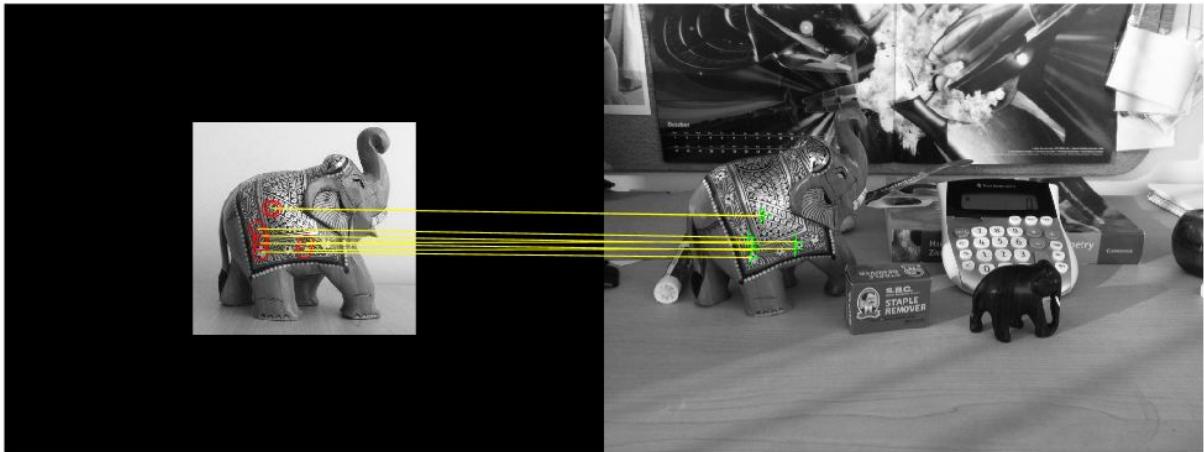
Harris



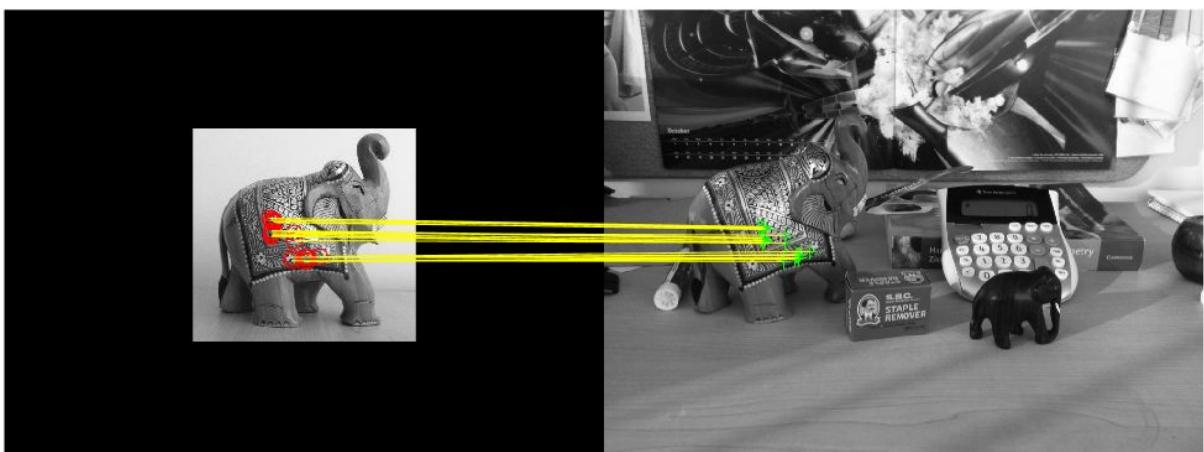
KAZE



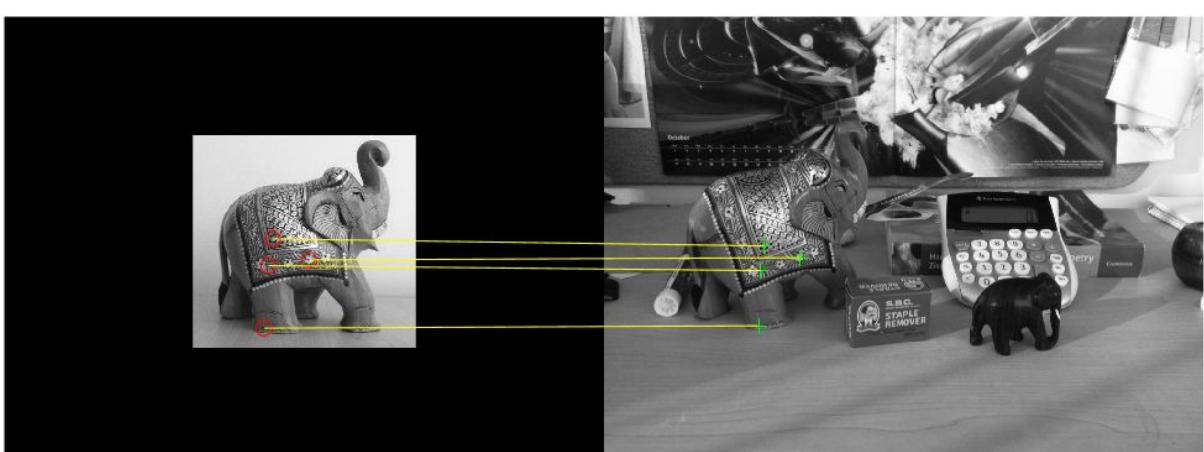
minEigen



ORB



SURF



Auswertung

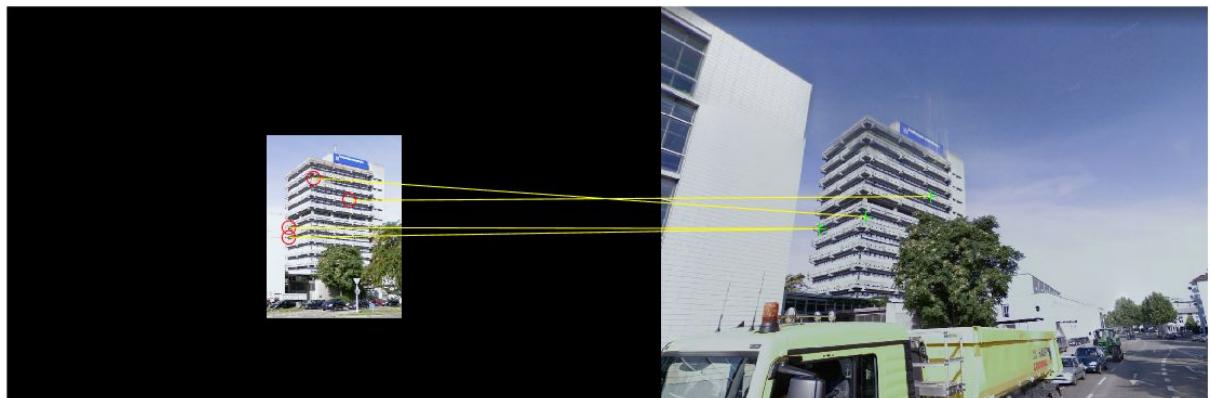
Alle Verfahren haben annehmbare Feature Paare gefunden. Es fällt auf, dass sich die gefundenen Paare der Corner Detektoren auf den Teppich auf dem Elefanten beschränken, wohingegen die Blob Detektoren auch Paare auf dem Elefanten selbst finden.

Dies ist vermutlich auf die feinen Strukturen des Teppichs zurückzuführen, welche für Corner Detektoren besser geeignet sind.

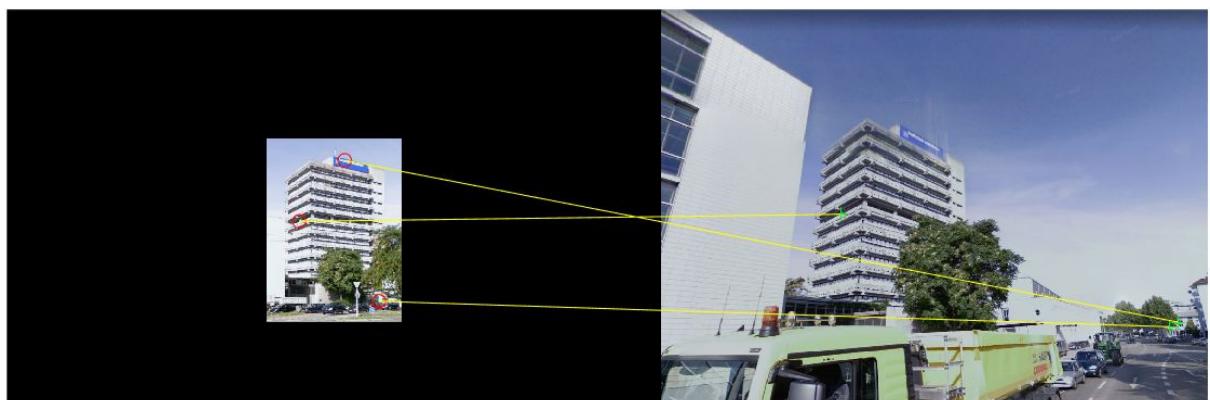
Testbild 3: HS Mannheim Streetview (zwei verschiedene Perspektiven)

Verfahren

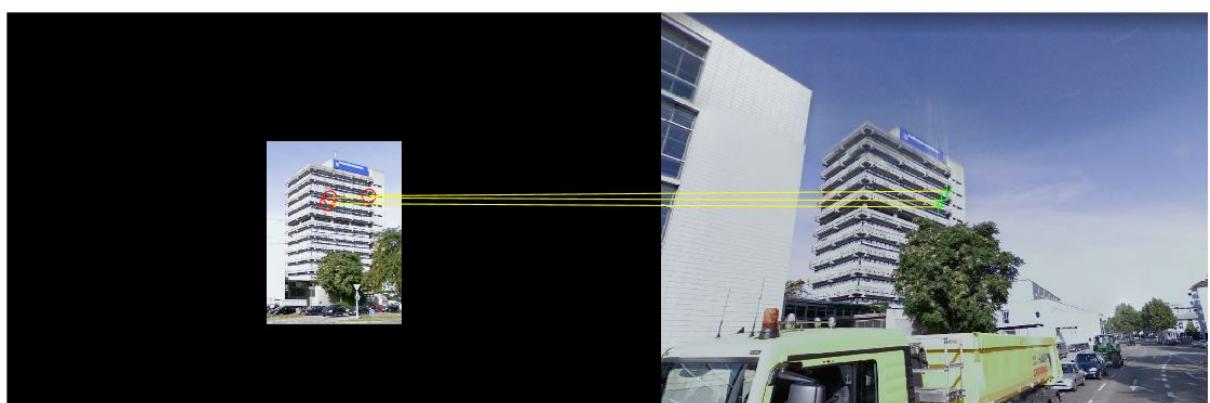
BRISK



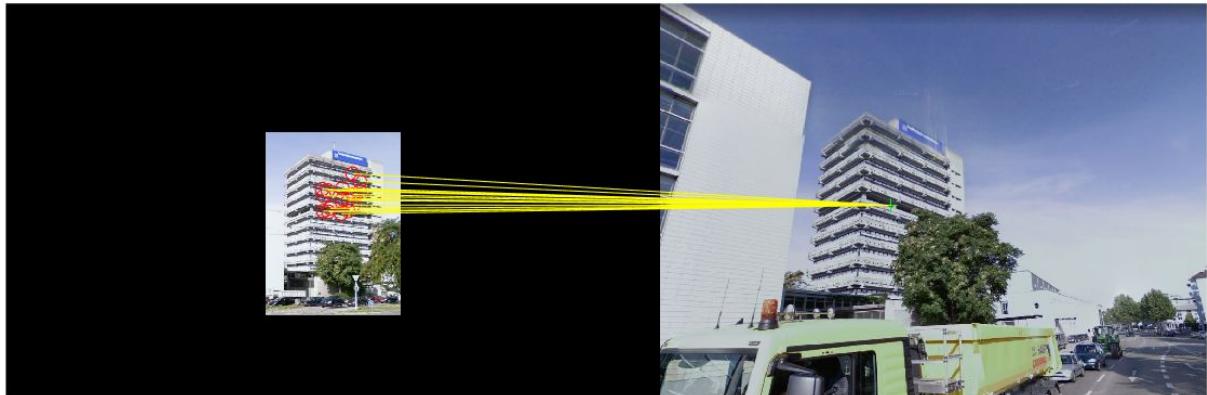
FAST



Harris



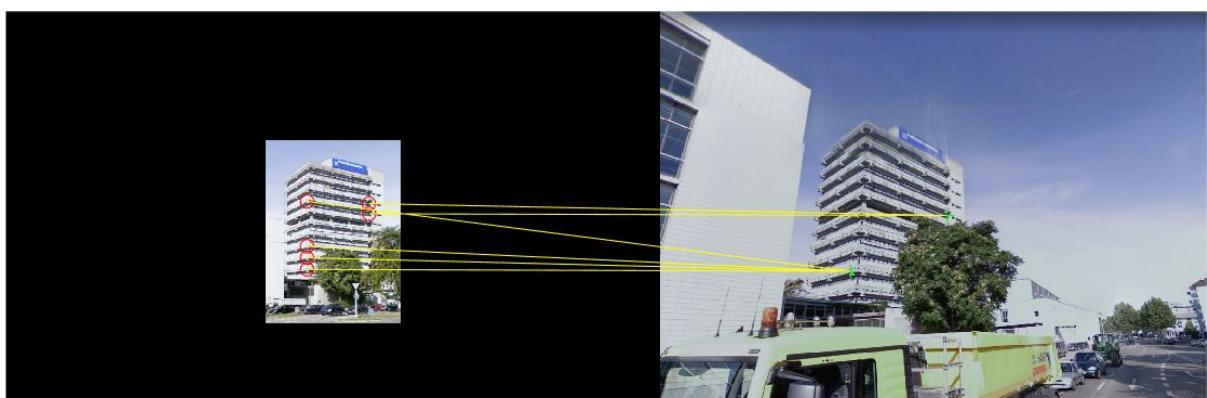
KAZE



ORB



SURF



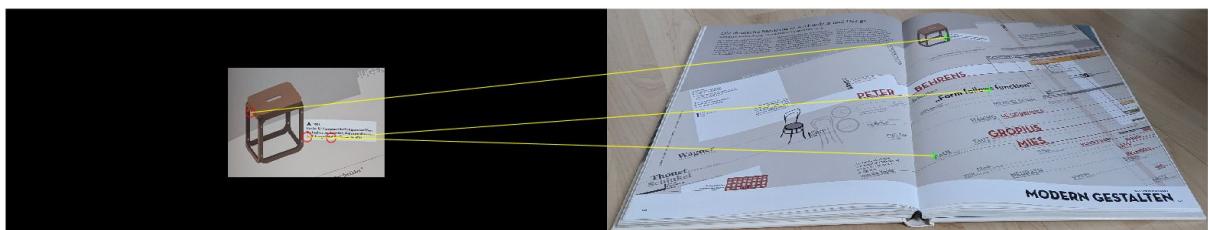
Auswertung

Die feinen Strukturen der Hausoberfläche wurden nur vom ORB (Corner Detektor) wirklich richtig erkannt. Mit ORB als Detektor konnte auch als einziges eine korrekte Bounding Box dargestellt werden. Die Blob Verfahren haben hier sehr inkonsistente Ergebnisse geliefert. Allgemein scheint KAZE die Tendenz zu haben mehrere Feature von Bild A auf ein einzelnes in Bild B zu matchen.

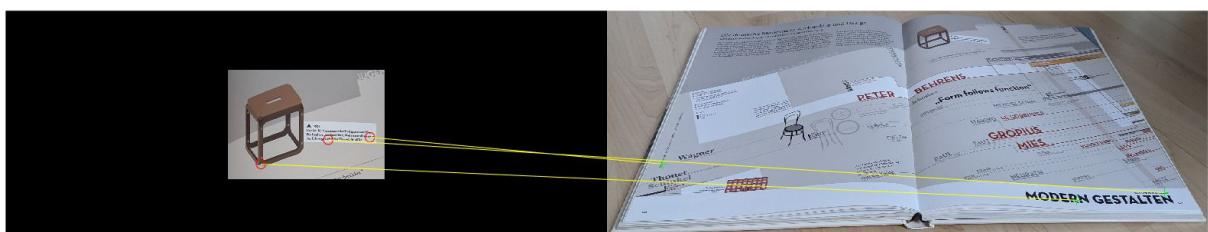
Testbild 4: Buchseite (zwei verschiedene Perspektiven)

Verfahren

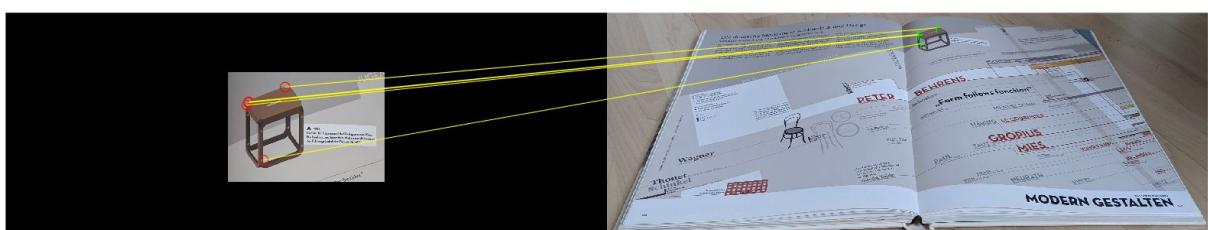
BRISK



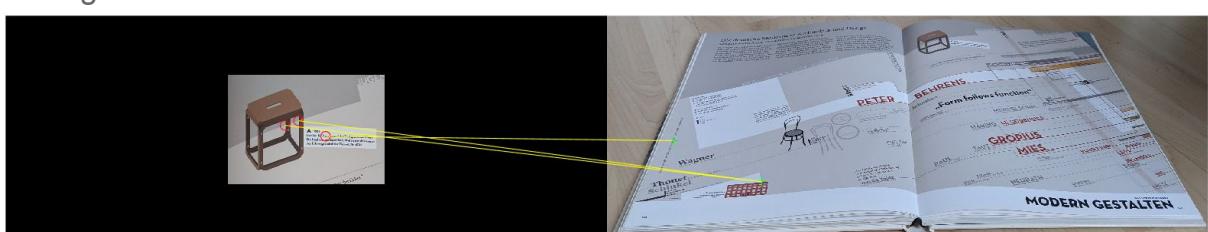
FAST



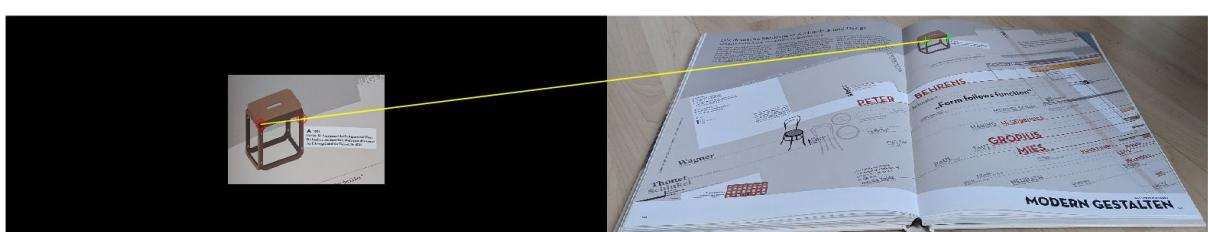
KAZE



minEigen



ORB



SURF



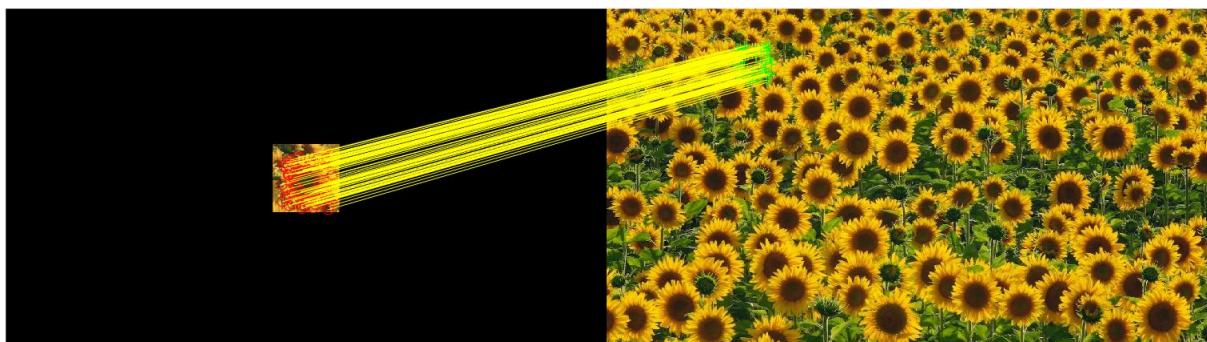
Auswertung

In diesem Beispiel wird die Erkennung von Feature-Paaren aufgrund der starken Verzerrung erschwert. Die Corner Detektoren haben hiermit scheinbar im allgemeinen ein Problem, wohingegen KAZE und SURF als Blob Detektoren ein annehmbares Ergebnis erzielen. Dies könnte an dem sehr robusten Feature Description Verfahren von SURF liegen.

Testbild 5: Sonnenblumen (Ausschnitt aus dem Originalbild)

Verfahren

KAZE



ORB



SURF



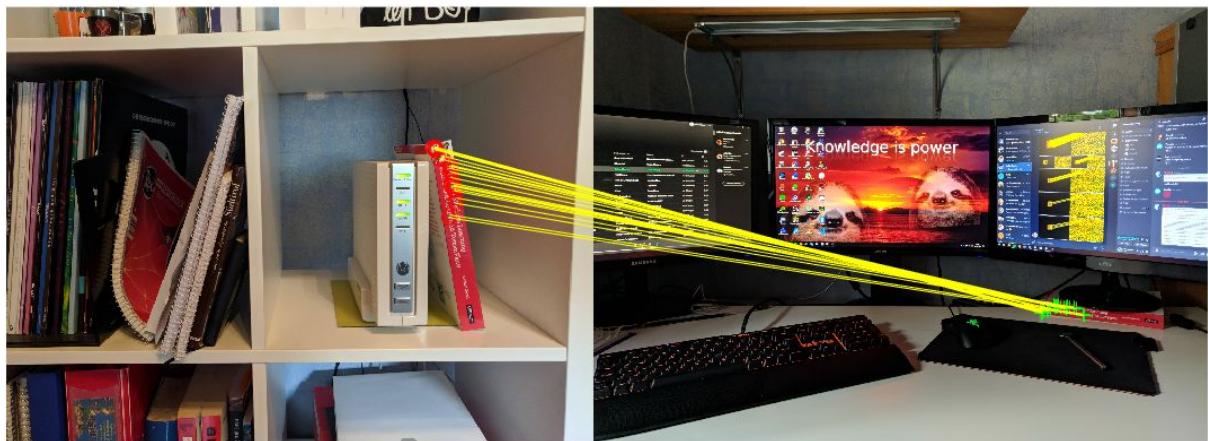
Auswertung

Die Blob Verfahren SURF und vor allem KAZE konnten viele Feature Paare finden. Die Blütenblätter wurden hierbei als Blobs erkannt und konnten gut gematcht werden. Auch ORB konnte als einziger Corner Detektor ausreichend Feature Paare finden. Dabei liegen die gefundenen Features zum Großteil am Übergang zwischen den hellen Blütenblättern und der dunklen Blüte.

Testbild 6: Buch (zwei verschiedene Szenen)

Verfahren

SURF



BRISK



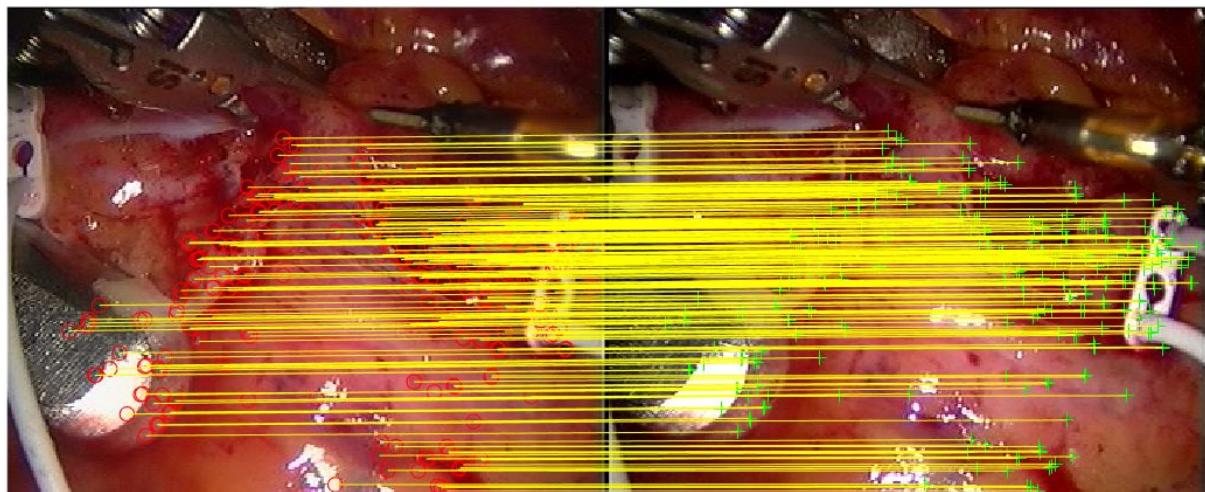
Auswertung

Nur die Verfahren BRISK und SURF konnten hier das Buch welches in beiden Szenen zu sehen ist wiederfinden. Wobei ORB und KAZE jeweils einen zu großen Speicherverbrauch hatten und die Anwendung abgestürzt ist (die lag vermutlich an der hohen Auflösung des Bildes). BRISK und SURF nutzen jeweils Feature Deskriptor Verfahren welche einen Invarianz gegenüber Rotation und Skalierung haben. Da das Buch eine deutlich andere Position in den Bildern einnimmt, konnten Verfahren ohne diese Invarianz vermutlich keine Feature-Paare finden.

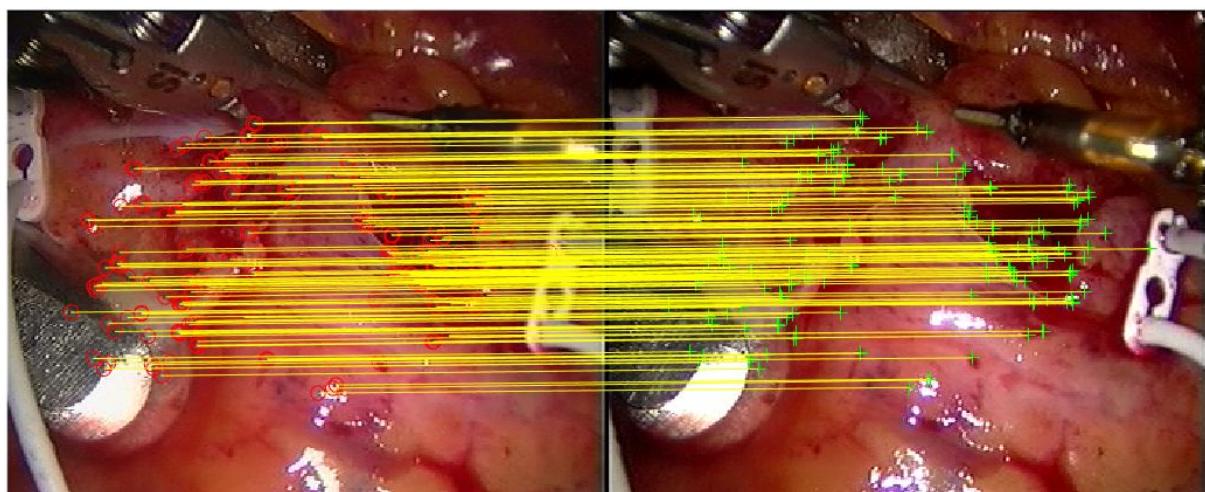
Testbild 7: Stereo-Endoskopie (leicht versetzte Kamera)

Verfahren

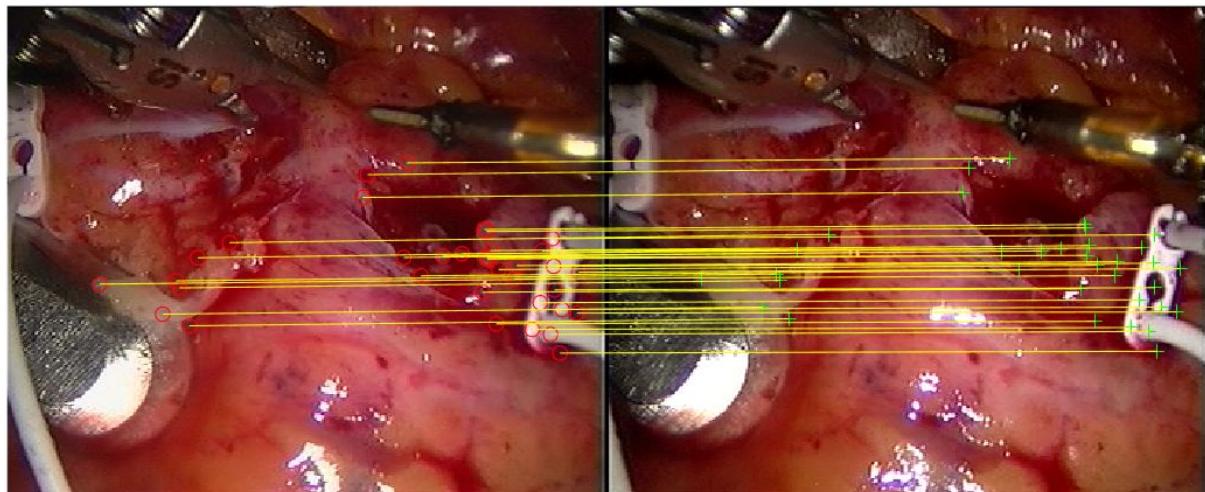
KAZE



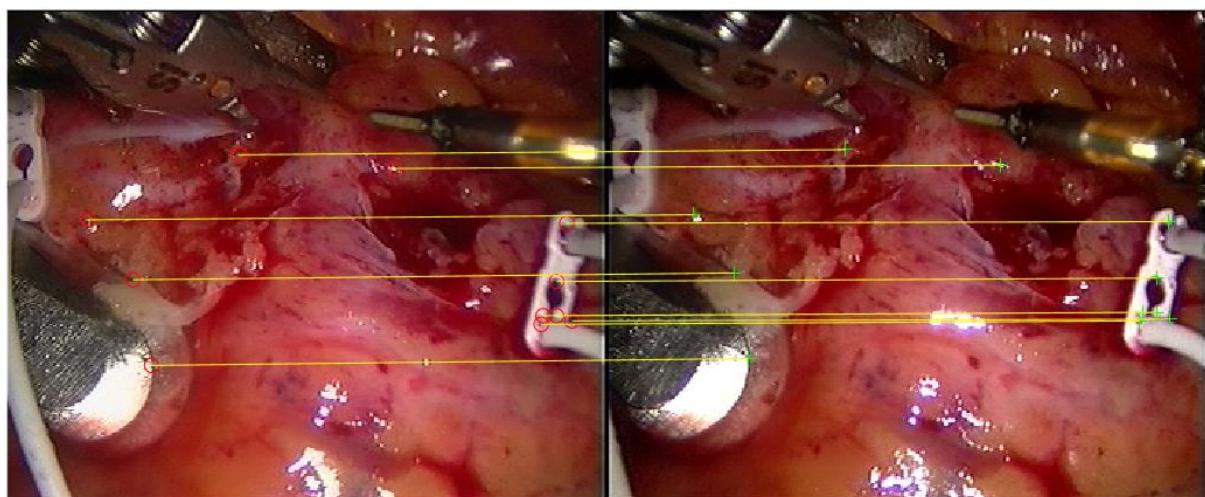
Orb



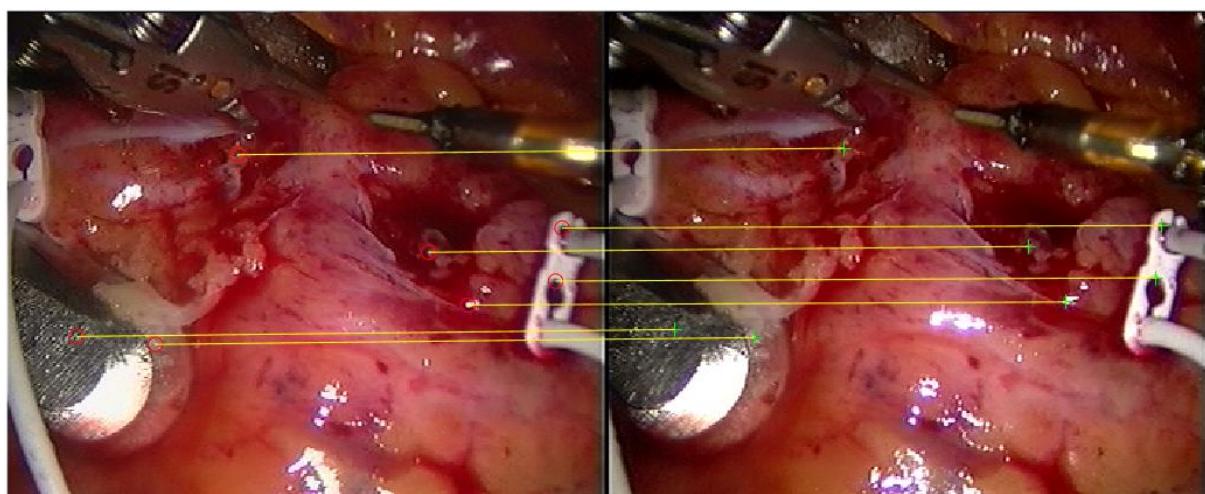
SURF



FAST



Harris



Auswertung

Alle Verfahren konnten hierbei Feature-Paare finden, deshalb wurden nur besonders gute und schlechte Ergebnisse gezeigt. Auffällig ist, dass bei den Corner Detektoren große Schwankungen zu erkennen sind. ORB erzielt sehr gute Ergebnisse, während FAST und Harris schlechte Ergebnisse liefern. Um eine Begründung hierfür zu finden, müsste weiter in die Implementierung der Detection und Description Methoden der jeweiligen Verfahren geblickt werden, was den Rahmen dieser Auswertung sprengt. Bei den Blob Verfahren SURF und KAZE konnten mittelmäßige bis gute Ergebnisse beobachtet werden.

Fazit

Über alle Testbilder hinweg haben sowohl SURF als auch ORB (das als direkte Alternative zu SIFT und SURF entwickelt wurde) die konstantesten Ergebnisse geliefert. Beide Verfahren sind außerdem, vor allem im Vergleich mit KAZE, extrem schnell. KAZE ist besonders für anspruchsvolle Szenarien geeignet, in denen Echtzeitverarbeitung keine Rolle spielt, oder keine ausreichenden Ergebnisse liefert. Die Anzahl der gefundenen Features ist mit KAZE sehr hoch, es empfiehlt sich jedoch beim Feature Matching nur einzigartige Features zuzulassen (Option: unique = true), da KAZE die Angewohnheit hat viele Features im Bild A mit einem Feature im Bild B zu verbinden.

Reflektion

Die von uns erstellte Applikation unterstützt den Nutzer beim Vergleichen verschiedener Feature Detection Verfahren für seine Anwendungszwecke. Die Entwicklung mit MATLAB war unkompliziert, was mit Hinblick auf die unterliegenden komplexen Themenbereiche erstaunlich ist.

Aktuell ist allerdings nur ein Bruchteil der in MATLAB verfügbaren Parameter (für Feature Detection, Description und Matching) in der Benutzeroberfläche sichtbar, obwohl die Ergebnisse durch Auswahl passender Parameter erheblich verbessert werden können. Auch der direkte Vergleich zwischen den Verfahren, also das direkte Gegenüberstellen der Ergebnisse ist derzeit nur über Umwege (Abspeichern, extern Öffnen) möglich. Ein letzter Aspekt der im Rahmen unserer Arbeit nicht weiter untersucht werden konnte, ist die Nutzung verschiedener Feature-Deskriptoren für gleiche Punkte. Wie sinnvoll ist die Verwendung eines SURF Deskriptors für Feature-Punkte die durch die BRISK Feature-Detektion gefunden wurden?

Ausblick

Bei einer größeren Erweiterung des Feature-Umfangs empfiehlt es sich jedoch die Umsetzung außerhalb von MATLAB durchzuführen, da dieses viele Limitierungen in der Gestaltung der Applikation mit sich bringt. MATLAB hat sich jedoch für das Prototyping bewährt.

Für eine Weiterentwicklung unserer App sind verschiedene Anpassungen möglich, um die Usability zu verbessern. Die Anzeige von Zusatzinformationen, zum Beispiel die Anzahl gefundener Features oder besseres Feedback nach ausgeführten Operationen wären sinnvolle Ergänzungen.

Weiterhin könnte das Zuschneiden von Bildern in der App durch Objekterkennung unterstützt werden. Dafür kommen verschieden komplexe Systeme in Frage, von einer einfachen kantengestützen Separation bei uniformen Hintergründen bis hin zum Einsatz von Deep Learning in Kombination mit anderen Bildverarbeitungsverfahren.

Um dem Nutzer das Gegenüberstellen der Ergebnisse verschiedener Verfahren zu vereinfachen müssten mehrere Prototypen entwickelt werden, um unterschiedliche Visualisierungen zu evaluieren. Zu viele Ergebnisbilder gleichzeitig darzustellen würde zum Beispiel zu einem Verlust von Detailinformationen führen, da diese sich die verfügbare Auflösung teilen müssten.

Abschließend könnte das Einführen weiterer Optionen in der Oberfläche die Anwendbarkeit der App enorm erhöhen.

Auf Basis solcher zusätzlichen Optionen könnten auch Parameter-Presets implementiert werden, vorgefertigte Sets an Parametern die für einen bestimmten Einsatz optimiert sind.

Solche Presets könnten ebenfalls von Nutzern eingereicht werden, wodurch die wissenschaftliche Gemeinschaft vom Fachwissen einzelner Experten profitieren könnte.

Zusatz: Anwendungsbeispiel mit Puzzleteilen



Neben der entwickelten MATLAB App stellen wir noch ein Anwendungsbeispiel zur Verfügung (*separate mlx*). Das Skript zeigt eine kantenbasierte Objekterkennung für Puzzleteile und versucht mit Hilfe von Feature Matching, für jedes erkannte Puzzleteil die richtige Stelle im fertigen Bild zu finden.

Die Datei ist vollständig kommentiert, daher folgt an dieser Stelle keine zusätzliche Dokumentation.