RBE 549 Computer Vision
Fall 2015
Final Project Report

"Object Detection of Escher Blocks"

Team Ocelot

Member	Signature	Contribution (%)
Dikshya Swain		25
Samyuktha Devadoss		25
Srishti Srivastava		25
Suriya Madhan Pachaimuthu		25

Grading:		
	Approach	/20
	Justification	/15
	Analysis	/20
	Testing & examples	/15
	Documentation	/15
	Presentation	/10
	Difficulty	/5
	Total	/100

List of Contents

List of Figures

Summary

Point Feature Matching techniques were implemented in detecting faces of an Escher block in various stages. The face was detected successfully by placing the block in a cluttered environment, even in the presence of another block. Partially occluded faces were also detected. Inbuilt functions were analyzed and fixed according to our requirements and were used in solving the Escher block puzzle. SURF algorithm was adopted to first determine the cube number, followed by rotating the cube by considering the left top image of each face as a reference and matching them with all the 24 templates and finally orienting the cube which helps us in successful solving of the puzzle.

The initial attempt of tracking the white points of each block, in order to recognize the block from a cluttered environment had various limitations because of the possibility of the threshold matching of the white color in the block with other white objects in the background. A detailed comparison has been made between the SURF and Harris technique.

Introduction

The objective of the project was to use computer vision techniques to detect an assigned object. The object given to our team was an Escher Block puzzle. It is a cube with images of M.C. Escher's artwork on each of its 6 faces as shown in Figure 1. The Escher block consists of 8 smaller cubes and each smaller cube face is quarter of the image on the bigger cube's face as shown in Figure 2.

Figure 1. Escher block

The goal of the puzzle is to solve the 6 faces simultaneously. Each of the 6 images is divided into 4 parts and forms 24 unique faces. However there are 8 cubes and 48 faces which implies each quarter of the image appears twice on each cube. It was observed that the combination of 3 adjacent faces is unique to each cube.

Based on the object's physical properties, different object detection techniques were considered. Since, the object has transparent edges, flat and smooth surface object detection using edge detection and surface detection techniques were neglected and the detailed designs of the images on each face of the cubes led to feature matching, extraction and recognition techniques. The advanced features and capabilities of MATLAB met our project requirements. Hence our project was carried out in MATLAB.

Figure 2. Six images on faces of Escher Block

MATLAB is a high-level proprietary programming language with extensive set of toolboxes and powerful functions and interactive environment for wide variety of applications. Image Acquisition Toolbox, Image Processing Toolbox and Computer Vision System in MATLAB can be used to acquire images and video, visualize data, and develop processing algorithms and analysis techniques for object detection and recognition.

Image Acquisition Toolbox simplifies the process of acquiring images and videos by providing a consistent interface across operating systems and hardware devices. MATLAB provides webcam support through hardware support packages which enables live video input. The image acquisition engine can acquire frames as fast as the camera and a PC can support for high speed imaging through programmatic interface in MATLAB, and a block for Simulink.

Computer Vision Toolbox provides various algorithms, functions, tools and methods for

- Object detection and tracking
- Training of object detection, object recognition, and image retrieval systems
- Camera calibration for single and stereo cameras
- Stereo vision
- 3-D point cloud processing
- Feature detection, extraction, and matching

Image Processing Toolbox has a wide range of algorithms, functions and apps for performing image analysis, image segmentation, image enhancement, noise reduction, geometric transformations, and image registration for an image. Visualization functions and apps can be utilized to examine a region of pixels, adjust color and contrast, explore images and videos, create contours or histograms, and manipulate regions of interest (ROIs).

The following section of the report describes the methodology and implementation of the algorithm for the object detection and stereo matching of Escher Blocks. The output obtained from implementing algorithm under various conditions such as cluttered environment, occlusions and analysis of various parameters of the algorithm are shown in the output section. In the application section, the algorithm developed for solving the cube puzzle is explained elaborately with detailed pictorial representations. The MATLAB codes for the following algorithms is included in the Appendix section.

Methodology

Object tracking:

A live video of the object was used as input to track the object. The object tracking was implemented by trying to detect the white color in the faces of the cube. The images in the cube were in grayscale hence it was easier to detect the white color and track the object. An rgb frame was acquired from the live video. The red, blue and green layers were acquired from the frame. Each of these layer images were then converted to binary image based on a threshold value 0.7. This threshold value was set in order to differentiate the white color in the object and the white color in the environment. The common region of the three extracted layer images were obtained and blob analysis was executed to show the bounded region of the object finally, in the live video, the object tracking was shown by highlighting the region of the object as shown in Figure 2.

Figure 3. Object Tracking

The object tracking method based on white color detection had a drawback. Its output cannot be used to recognize the object. As the cube's face images have similar grayscale values and intricate patterns and designs, more sophisticated method like feature detection, matching and extraction was used.

Feature detection, extraction, and matching:

A feature is an important and prominent part of an image, such as a corner, blob, edge, or line. These localized features are often called key point features or interest points. Point features can be used to find a wide set of corresponding locations in different images to perform object instance and category recognition. A key advantage of interest points is that they permit matching even in the presence of clutter (occlusion) and .large scale and orientation changes. Feature extraction enables you to derive a set of feature vectors, also called descriptors, from a set of detected features. Feature matching is the comparison of two sets of feature descriptors obtained from different images to provide point correspondences between images.

Computer Vision System Toolbox offers capabilities for feature detection and extraction that include BRISK, MSER, and SURF detection for blobs and regions and Harris

SURF:

In computer vision, Speeded up Robust Features (SURF) is a local feature detector and descriptor that can be used for object recognition. It is a detector and a descriptor for points of interest in images where the image is transformed into coordinates, using the multi-resolution pyramid technique. MATLAB provides inbuilt functions to implement SURF.

SURF Algorithm:

This algorithm detects a specific object based on finding point correspondences between the template and the scene image. It can detect objects despite a scale change or in-plane rotation. It is also robust to small amount of out-of-plane rotation and occlusion. The algorithm has following steps

1. Save templates
2. Get input images -scene images from video
3. Detect strongest feature points of both template and scene image
4. Extract feature descriptors from the images
5. Determine the putative point matches
6. Locate the object in the scene using putative matches

Advantages of SURF over HARRIS:

SURF, known as Speeded up Robust Features uses an integer approximation method to extract features in an image. SURF Features are a vector of values that describes a small patch around the interest point. It is a key point detector and descriptor. This algorithm returns SURF points which in turn can be used to match with the templates. It primarily works on obtaining SURF points from two images and matching them.

For our object, SURF worked well in a cluttered environment and environment that had a background scene. It is because our object has non-repeating texture patterns in each face of the block. These texture patterns tend to give unique features to match with the template. When the points are spread out, the algorithm matches accurately. For these environments the SURF algorithm had no false matches. We had opted HARRIS algorithm only when there is an image without any background. In this case, the algorithm tend to take the corner points in the image and match with the corresponding points in the template. The HARRIS algorithm primarily works on detecting interest points (corners) in the image but does not calculate the descriptor. For images with no background, the SURF algorithm tend to give false matches because the images on the face are composed of black and white, so it tend to take feature points and match randomly. It does not have a background to differentiate the object whereas the HARRIS algorithm perfectly matches the object because of the non-repeating texture patterns. The corners point of every image are different. The SURF algorithm can be made highly reliable by altering the inbuilt function 'estimateGeometricTransform' which will be explained in the below sections.

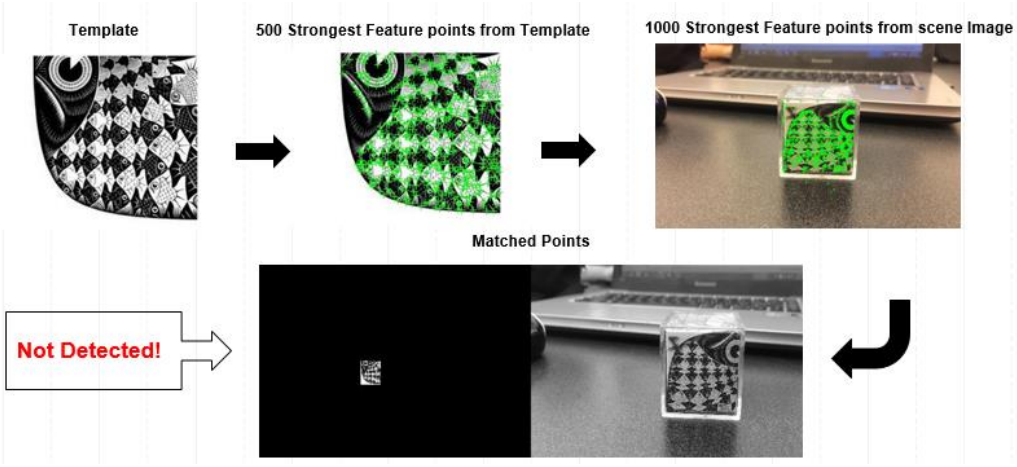

Figure 4. Rotation Invariance - HARRIS

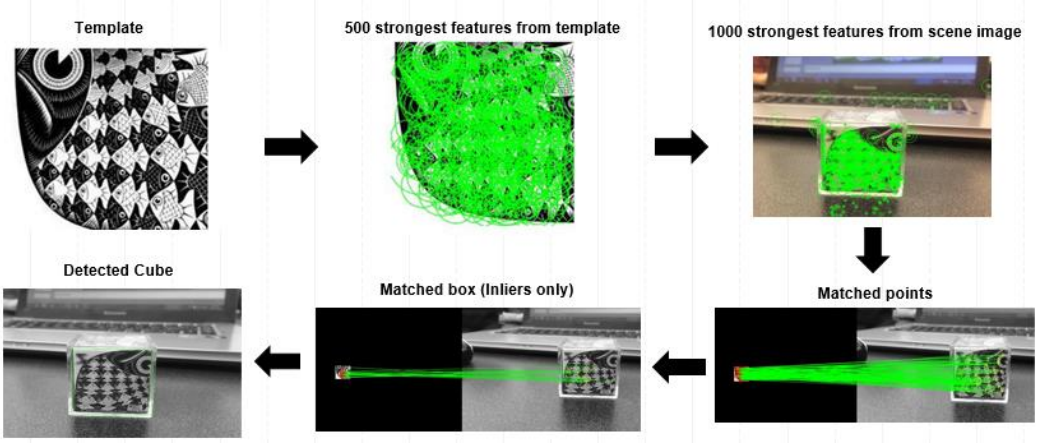

Figure 5. Rotation Invariance - SURF

Few tests have been performed to determine the efficiency of the algorithms. For a rotated image with background, the SURF algorithm performed well compared to HARRIS algorithm as the image had background even though the HARRIS algorithm is rotational invariant. Since the background image can have many corner points, the HARRIS algorithm tend to take in those points and give false matches. Figure 3 and Figure 4 depicts the matching of the scene image to the template by HARRIS and SURF algorithms respectively. Even for a 50% occluded image with background, the SURF algorithm performs better than the HARRIS algorithm. Figure 5 and Figure 6 depicts the matching of the 50% occluded image to the template by HARRIS and SURF algorithms respectively.

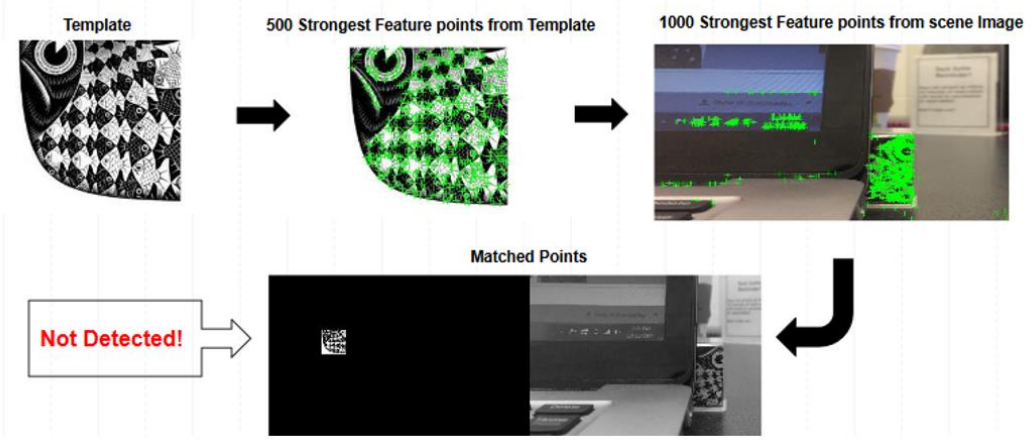

Figure 6. 50% occluded – HARRIS

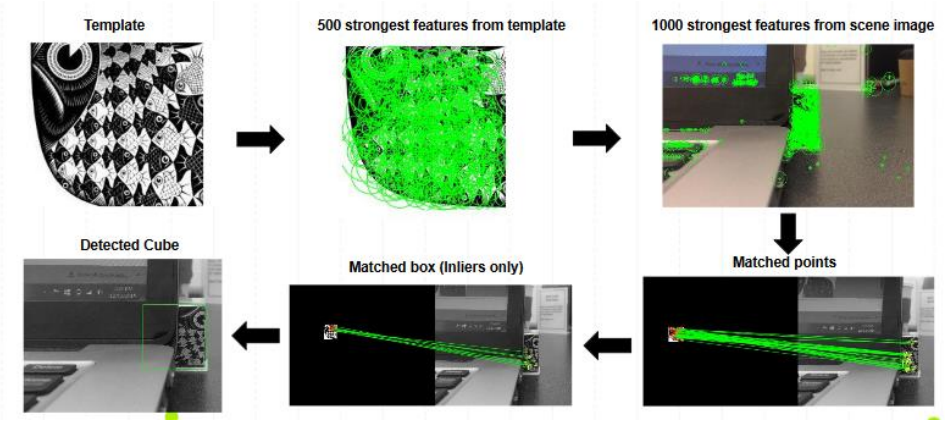

Figure 7. 50% occluded - SURF

<u>Analysis of MALTAB functions</u>:

estimateGeometricTransform

It is an inbuilt MATLAB function which accepts matched points as input parameters and returns the inliers, a 2D transform matrix, and status. Its main use is to remove the outliers from the matched points. This function uses the MSAC (M-estimate Sample Consensus) algorithm, which is a variation of RANSAC (Random Sample Consensus) algorithm, to remove the outliers. RANSAC is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers. It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability, with this probability increasing as more iterations are allowed. The main idea behind MSAC is to evaluate the quality of the consensus set (i.e. the data that fit a model and a certain set of parameters) calculating its likelihood. The syntax of the function is:

function [tform, inlierPoints1, inlierPoints2, status] ...

= estimateGeometricTransform(matchedPoints1, matchedPoints2, transformType, varargin)

where matchedPoints1 and matchedPoints2 are the mapped points from template to test image. The transformType can be ‘affine’, ‘similarity’, or ‘projective’. In our application, ‘affine’ was used. The function returns ‘status’ which was used in our application to compare the test image with multiple templates. The status can have the following three values:

0: No error

1: matchedPoints1 and matchedPoints2 do not contain enough points.

2: Not enough inliers have been found.

When the status output is not given, the function will throw an error if matchedPoints1 and matchedPoints2 do not contain enough points or if not enough inliers have been found.

The main parameters the function depends on are maxNumTrials, Confidence, and maxDistance.

- maxNumTrials - It is the maximum number of random trials (iterations) for finding inliers in a given data set. Robustness of the function increases with increase in this number, but at a cost of decreasing computational speed. The default value is 1000.
- Confidence - A numeric scalar, C, $0 < C < 100$, specifying the desired confidence (in percentage) for finding the maximum number of inliers. Increasing this value will improve the robustness of the output at the expense of additional computation. The default value is 99.
- maxDistance - A positive numeric scalar specifying the maximum distance in pixels that a point can differ from the projection location of its associated point. The default value is 1.5.

These default values correspond to optimal percentage of efficiency. But in the case of our object, we observed that even though most points were correctly mapped from the template to the test image, the estimateGeometricTransform function assumed most of them to be outliers (as shown in fig), hence giving "no match". The efficiency needed to be decreased so that we would get more inliers in our test image. So, we proceeded to tweak these three parameters and observed results with different values (this analysis is explained in detail in the Results section). The optimal values observed were:

MaxNumTrials = 500

MaxDistance = 2.5

Confidence = 75

which can be seen in Figure 8.

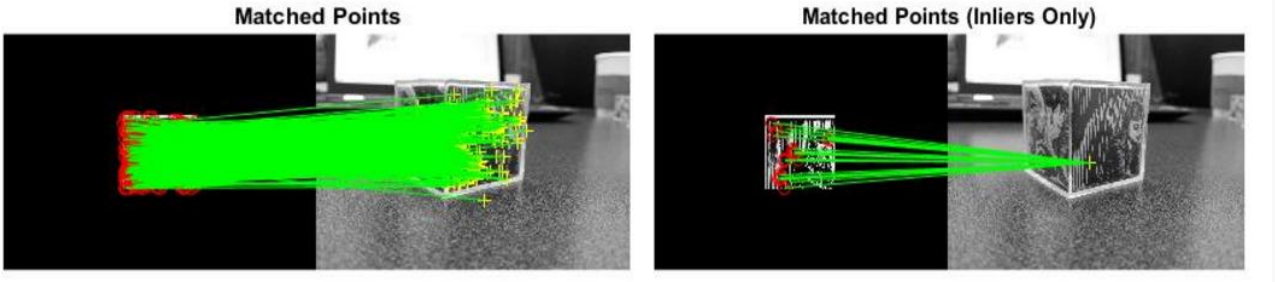

Figure 8. Output image

Stereo Matching:

In stereo matching, we take two rectified and undistorted stereo images and the output in a dense disparity map. The disparity map shown below is the corresponding pixel difference between the left and right stereo images. It also measure the apparent motion in pixels for every point and make an intensity image out of the measurements. In the disparity map, the brighter shades represent more shift and lesser distance from the point of view (camera). The darker shades represent lesser shift and therefore greater distance from the camera.

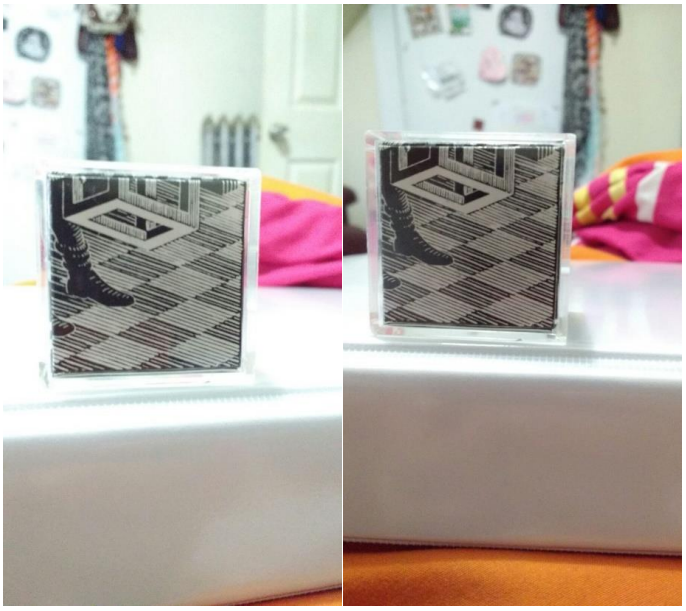

Figure 9. Left and Right Stereo Image

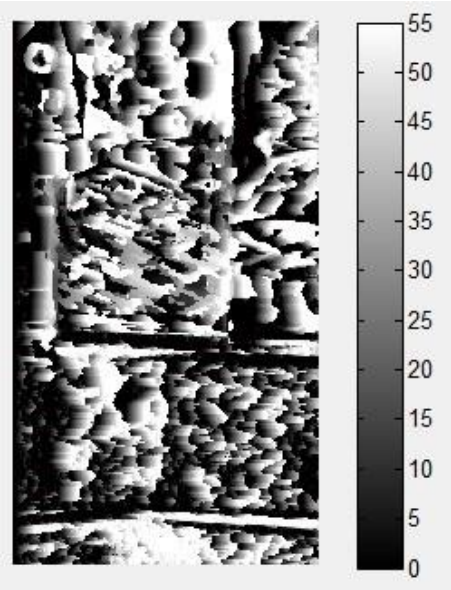

Figure 10. Disparity Map

Results

SURF was implemented to detect the Escher Cubes in under different conditions. The results are listed below.

1. Object Detection in cluttered environment

The object was converted to grayscale and SURF was used to detect the image. The detected image is shown by yellow box.

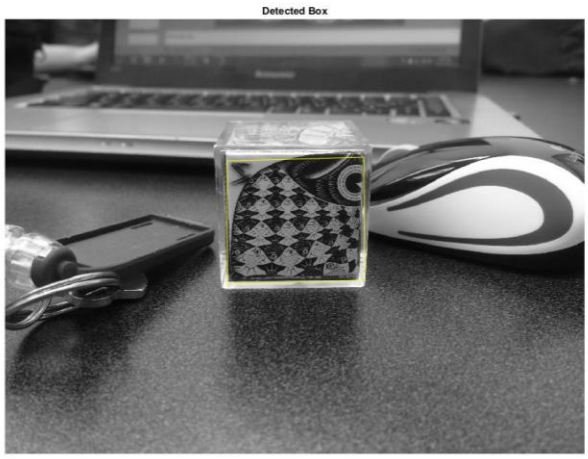

Figure 11. Objection detection in cluttered environment

2. **Multiple object detection**

Two cubes with different faces were placed side by side. One template was used and was correctly detected in the test image. This means that no false matches were given.

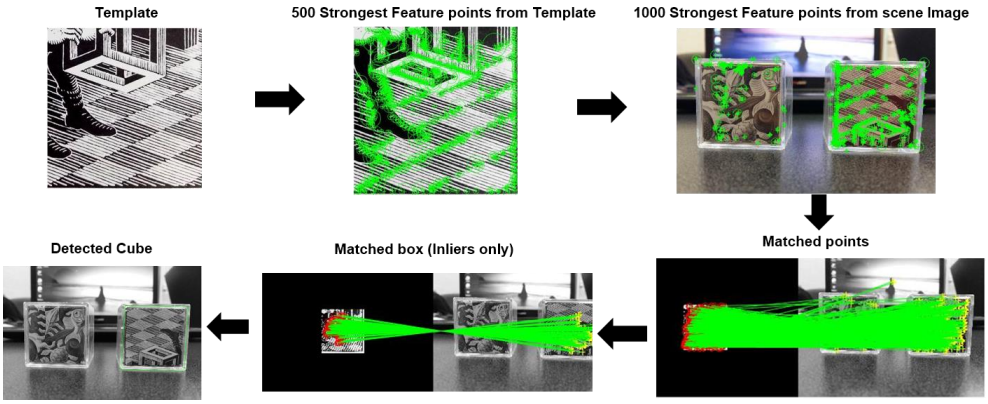

Figure 12. Multiple Object Detection

3. **Occlusion**

Images at 25%, 50% and with arbitrary rotation were used as test images. SURF was able to handle the partial occlusion.

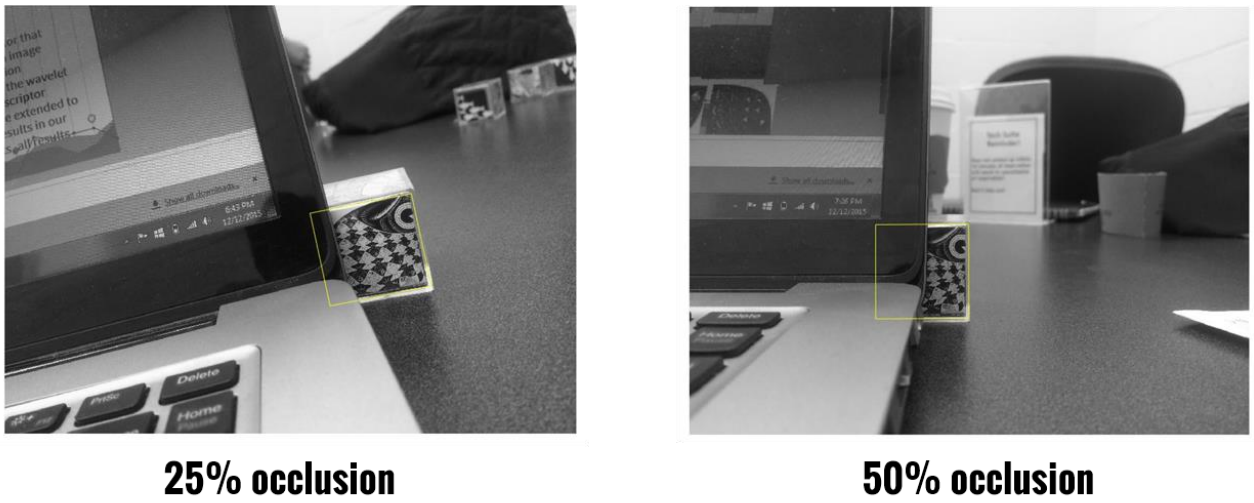

Figure 13. Occlusion

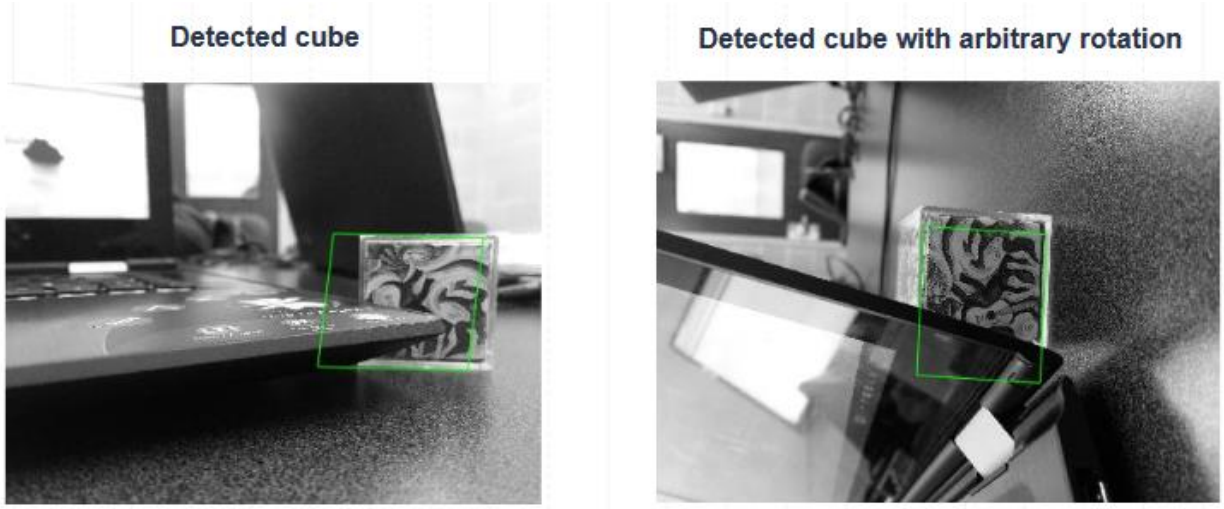

Figure 14. Arbitrary Orientation

4. **Analysis of inbuilt function**

4.1 Confidence

The confidence level is set at default value of 99%. At this level, almost all matched points were tagged as outliers which resulted in “no match”.

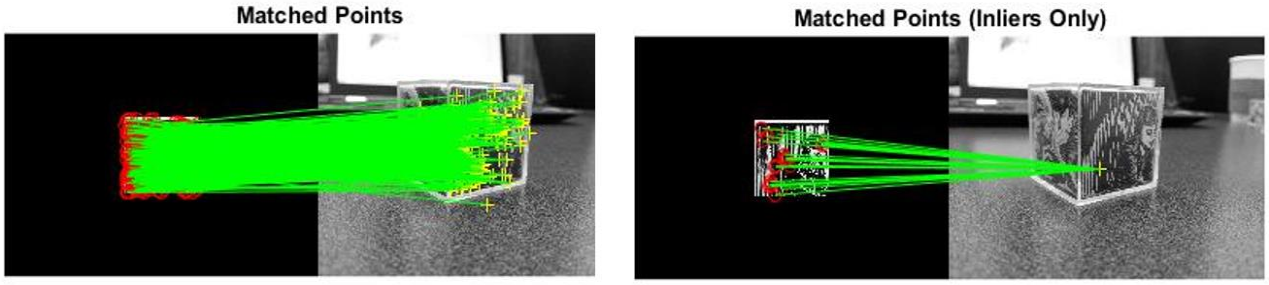

Figure 15. Confidence 75

Further decreasing the confidence to 50 increased the number of inliers even more and resulted in a detected object.

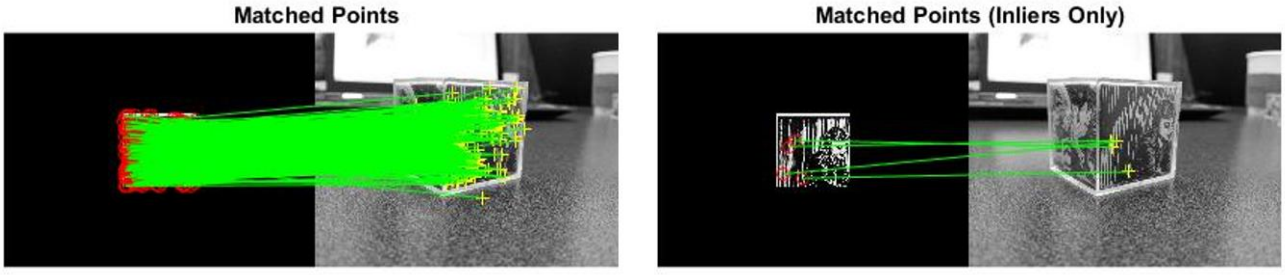

Figure 16. Confidence 50

.

Figure 17. Detected Face

4.2 MaxDistance

MaxDistance default value was 1.5 but increasing it increased the number of inliers. At the value of 2 pixels, we got:

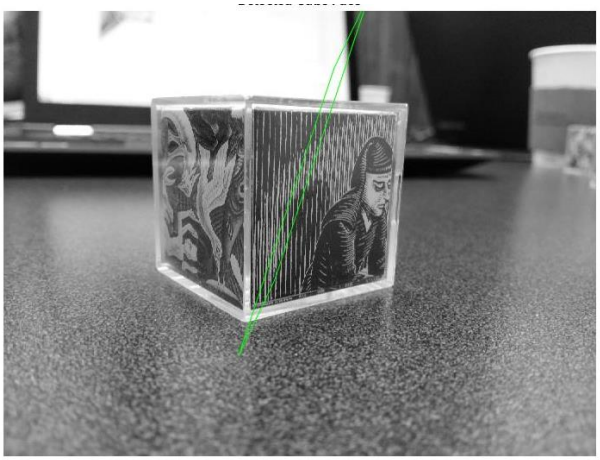

Figure 18. MaxDistance 1.5

Further increasing it to 2.5, we got better results. But further increasing it to 3 resulted in "no match" i.e. the efficiency of the code was reduced too much. So, the optimal value for our object seems to be 2.5 pixels.

5. **3D rotation with our values for above parameters**

The analysis of the inbuilt function was used to detect the cube in different 3D orientations because a more efficient code was needed to be able to handle the 3D rotations. For no roatation:

Figure 19. Detected Cube Face – No Rotation

For some rotation, the code was able to handle it:

Figure 20. With Rotation

But increasing the rotation further resulted in no match.

Application

Solving Escher's Block Using Various Object Recognition Algorithms

Escher's Block is solved in three parts namely Cube Number Identification, Cube Rotation and Cube Orientation. SURF was used to detect the cubes for this task. This method of object detection works best for objects that exhibit non-repeating texture patterns, which give rise to unique feature matches. This technique is not likely to work well for uniformly-colored objects, or for objects containing repeating patterns. Note that this algorithm is designed for detecting a specific object.

The eight cubes were all stored in eight different arrays containing the list of faces associated with them. Each of the 24 faces was previously labeled from A to X and these labels were used to store the cube-face-combinations.

```
cube(1).arr=['U','D','G','R','R','J'];
cube(2).arr=['V','L','T','Q','B','M'];
cube(3).arr=['X','F','V','M','O','A'];
cube(4).arr=['W','E','S','N','H','L'];
cube(5).arr=['X','J','B','T','I','O'];
cube(6).arr=['W','I','A','S','U','N'];
cube(7).arr=['P','C','C','H','F','P'];
cube(8).arr=['K','Q','D','G','K','E'];
```

PART 1: Cube Number Identification

To solve the puzzle, each cube number must be identified and it is done by SURF algorithm. The cube face is detected from the live video and the image is matched with 24 templates and label of the matched face is returned. This is done for three faces per cube. The unique feature of Escher's Block is that the combination of any three faces of the cube is unique to that cube. We use this property to identify the cube number of all cubes by taking in images from the user. Once we have all three labels, we run it through each cube's array and determine the cube number. Once we determine the cube number, we would essentially place it in respective position. The position of the cube's are shown to the user via GUI.

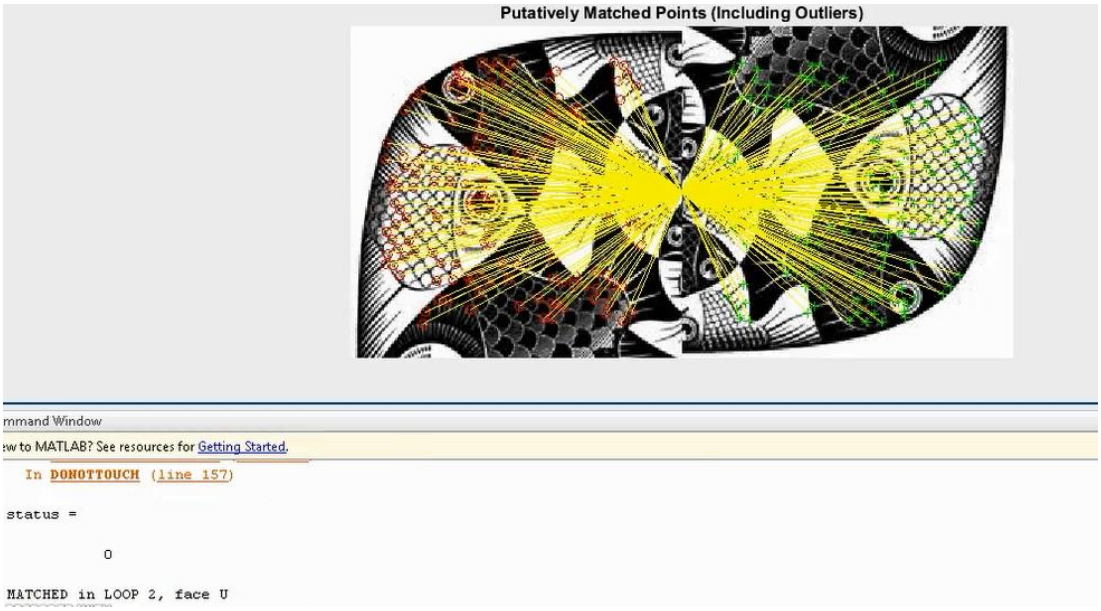

Figure 21. Putatively Matched Points

PART 2: Cube Rotation

Once cubes are in its respective position, we would need to fix the left top cube's image of any side. The left top cube is rotated to match with the left top templates of six images. The left top templates are stored in positions (1,5,9,13,17 & 21). We would appropriately run the left top cube's image to all the templates and fix the position. Once the left top image is fixed, we can match the right top, right bottom and left bottom cubes to their respective templates and fix their position. Then we move to their respective orientation.

Figure 22. Matching Points

PART 3: Cube Orientation

When we have all parts of the image in the same face, we would rotate the cube about its own axis to place it in correct orientation. We perform this operation automatically to determine the geometric transformation between a pair of images. When one image is distorted relative to another by rotation and scale, we use detectSURFFeatures and estimateGeometricTransform to find the rotation angle and scale factor. The transformation corresponding to the matching point pairs using the statistically robust M-estimator Sample Consensus (MSAC) algorithm, which is a variant of the RANSAC algorithm. It removes outliers while computing the transformation matrix. We can see varying results of the transformation computation because of the random sampling employed by the MSAC algorithm. we could have used a corner detector, detectFASTFeatures, to complement the SURF feature detector which finds blobs. Image content and image size also impact the number of detected features. After all these operations on the cube, the puzzle gets solved.

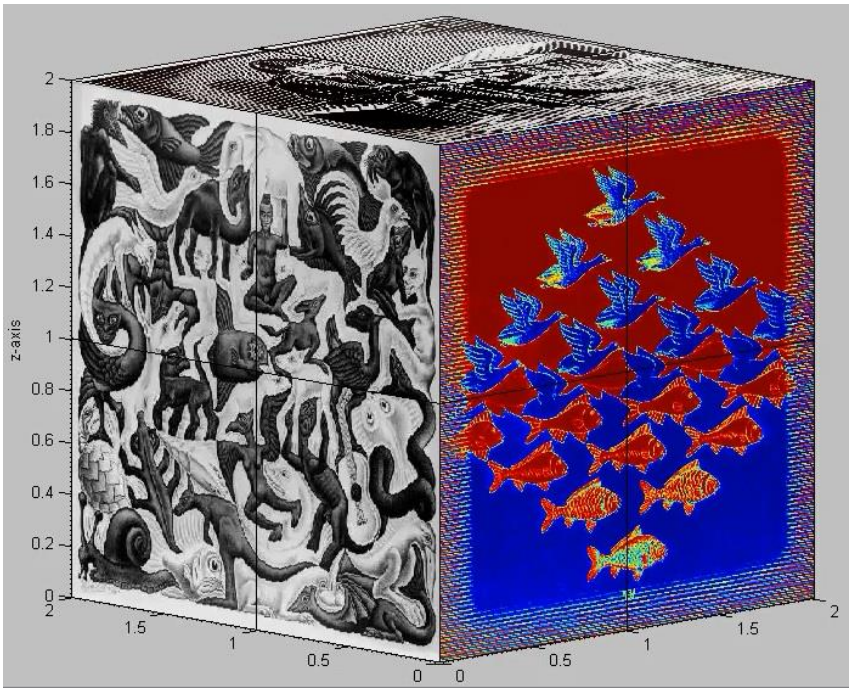

Figure 23. 3D Cube in MATLAB

Conclusion

SURF algorithm was mainly used in detecting the face and orientation of the Escher block. The test block was detected in various scenarios like from a cluttered environment, in the presence of another block in the vicinity, by partially occluding the object etc. Inbuilt MATLAB functions like estimateGeometricTransform were modified according to our requirement which helped us in getting more inliers in our test image, which resulted in better matching, hence used in solving the Escher block puzzle.

Stereo Matching was tried and implemented. The left and right-eye images were taken of the cube and a disparity map was generated.

Object tracking from a live feed was also successfully implemented. The main objective was to detect the white color in the faces of the cube. A definite threshold was maintained in order to differentiate the white color of the object from the other white objects in the environment. Though the results were accurate, this method has a limitation of the possibility of error in detecting the cube when another white object of the same threshold is present in the environment as well.

References

[1] http://www.mathworks.com/products/matlab/[2]
http://www.mathworks.com/products/computer-vision/features.html#key-features
[3] http://www.mathworks.com/products/computer-vision/features.html
[4] http://mesh.brown.edu/engn1610/szeliski/04-featuredetectionandmatch ing.pdf
[5] IFT and SURF Performance Evaluation against Various Image Deformations on Benchmark Dataset, *Khan, N.Y. ; Comput. Sci. Dept., Otago Univ., Dunedin, New Zealand ; McCane, B. ; Wyvill, G*.
[6] Tracking and Recognition of Objects using SURF Descriptor and Harris Corner Detection, *J.Jasmine Anitha and S.M.Deepa*
[7] Object recognition from local scale-invariant features, *Lowe, D.G. ; Dept. of Comput. Sci., British Columbia Univ., Vancouver, BC, Canada*
[8] Distinctive Image Features from Scale-Invariant Keypoints, *David G. Lowe*
[9] Overview of the RANSAC Algorithm, *Konstantinos G. Derpanis*
[10] An Evaluation of the Performance of RANSAC Algorithms for Stereo Camera Calibration, *A. J. Lacey, N. Pinitkarn and N. A. Thacker*

Appendix

Code :SURF.m

```
clc; clear; close all;

addpath('R:\CVProject\Faces');
addpath('R:\CVProject\Photos');
addpath('R:\CVProject\Templates');

for i=1:24
  face(i).label=char(64+i) %Labeling 24 faces
end

myFolder = 'R:\CVProject\Templates'; %Folder where images are stored
if ~isdir(myFolder)
  errorMessage = sprintf('Error: The following folder does not exist:\n%s', myFolder);
  uiwait(warndlg(errorMessage));
  return;
end

filePattern = fullfile(myFolder, '*.jpg'); %Find all files with extension .jpg
jpegFiles = dir(filePattern);
x=1;

for k = 1:length(jpegFiles)
  baseFileName = jpegFiles(k).name;
  fullFileName = fullfile(myFolder, baseFileName);
  fprintf(1, 'Now reading %s\n', fullFileName);
  Q = imread(fullFileName);
  temp(k).image=Q;
  face(x).image=Q(1:size(Q,1)/2,1:size(Q,2)/2,:); %Check numbering of faces
  face(x+1).image=Q(size(Q,1)/2+1:size(Q,1),1:size(Q,2)/2,:);
  face(x+2).image=Q(size(Q,1)/2+1:size(Q,1),size(Q,2)/2+1:size(Q,2),:);
  face(x+3).image=Q(1:size(Q,1)/2,size(Q,2)/2+1:size(Q,2),:);

  x=x+4;
end


cube(1).arr=['U','D','G','R','R','J'];
cube(2).arr=['V','L','T','Q','B','M'];
cube(3).arr=['X','F','V','M','O','A'];
cube(4).arr=['W','E','S','N','H','L'];
cube(5).arr=['X','J','B','T','I','O'];
```

```
cube(6).arr=['W','I','A','S','U','N'];
cube(7).arr=['P','C','C','H','F','P'];
cube(8).arr=['K','Q','D','G','K','E'];

set(0,'DefaultFigureWindowStyle','docked'); %Will dock all images. Put in startup.m to set as default
% set(0,'DefaultFigureWindowStyle','normal') %to switch back to undocked mode

for i=1:8
   figure(i);
   for x=1:6
      for j=1:24
         if (cube(i).arr(x)==face(j).label);
            subplot(2,3,x);
            imshow(face(j).image);
            drawnow;
         end
      end
   end
end

% tilefigs; %function to tile all figure windows. NOT for docked windows


%% PFM
% for i=1:24
  boxImage=face(17).image;
  figure;
  imshow(boxImage);
  if(ndims(boxImage)>2)
     boxImage=rgb2gray(boxImage);
  end
  title('Template');
  boxPoints = detectSURFFeatures(boxImage);
  figure;
  imshow(boxImage);
  title('500 Strongest Feature Points from Template');
  hold on;
  plot(selectStrongest(boxPoints, 1000));
  % plot(boxPoints, 'showPixelList', true, 'showEllipses', false);
  im=imread('rot3.jpg');
  figure;
  imshow(im);
  if(ndims(im)>2)
     im=rgb2gray(im);
  end
```

```
  scenePoints = detectSURFFeatures(im);
  title('1000 Strongest Feature Points from Scene Image');
  hold on;
  plot(selectStrongest(scenePoints, 500));
  % plot(scenePoints, 'showPixelList', true, 'showEllipses', false);

  [boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
  [sceneFeatures, scenePoints] = extractFeatures(im, scenePoints);

  boxPairs = matchFeatures(boxFeatures, sceneFeatures);

  matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
  matchedScenePoints = scenePoints(boxPairs(:, 2), :);
  figure;
  showMatchedFeatures1(boxImage, im, matchedBoxPoints, ...
     matchedScenePoints, 'montage');
  title('Matched Points');

  [tform, inlierBoxPoints, inlierScenePoints,status] = ...
     estimateGeometricTransform1(matchedBoxPoints,  matchedScenePoints,  'affine');  %Make
this function return the status

  display(status);
  % display(status);
  % display(status);
%    if status==0
     figure;
     showMatchedFeatures1(boxImage, im, inlierBoxPoints, ...
        inlierScenePoints, 'montage');
     title('Matched Points (Inliers Only)');

     boxPolygon = [1, 1;...                        % top-left
        size(boxImage, 2), 1;...                % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-right
        1, size(boxImage, 1);...                % bottom-left
        1, 1];                % top-left again to close the polygon

     newBoxPolygon = transformPointsForward(tform, boxPolygon);

     figure;
     imshow(im);
     hold on;
     line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'g');
     title('Detected Cube Face');
%       break;
%    end
```

```
% end

estimatGeometricTransform.m - our version of theinbuilt function

function [tform, inlierPoints1, inlierPoints2, status] ...
  = estimateGeometricTransform1(matchedPoints1, matchedPoints2, ...
  transformType, varargin)

statusCode = struct(...
  'NoError',          int32(0),...
  'NotEnoughPts',     int32(1),...
  'NotEnoughInliers',  int32(2));

% Parse and check inputs
[points1, points2, maxNumTrials, confidence, maxDistance, sampleSize, ...
  status, classToUse] = parseInputs(statusCode, matchedPoints1, ...
  matchedPoints2, transformType, varargin{:});

% return identity matrix in case of failure
failedMatrix = eye([3,3], classToUse);

% Compute the geometric transformation
if status == statusCode.NoError
  [isFound, tmatrix, inliers] = msac(points1, points2, maxNumTrials, ...
     confidence, maxDistance, sampleSize, classToUse);
  if ~isFound
     status = statusCode.NotEnoughInliers;
  end

  % Do an extra check to verify the tform matrix. Check if matrix is
  % singular or contains infs or nans.
  if isequal(det(tmatrix),0) || any(~isfinite(tmatrix(:)))
     status = statusCode.NotEnoughInliers;
     tmatrix = failedMatrix;
  end
else
  tmatrix = failedMatrix;
end

% Extract inlier points
if status == statusCode.NoError
  inlierPoints1 = matchedPoints1(inliers, :);
  inlierPoints2 = matchedPoints2(inliers, :);
else
  inlierPoints1 = matchedPoints1([]);
  inlierPoints2 = matchedPoints2([]);
```

```
  tmatrix = failedMatrix;
end

% Report runtime error if the status output is not requested
reportError = (nargout ~= 4);
if reportError
  checkRuntimeStatus(statusCode, status);
end

isSimilarityOrAffine = sampleSize < 4;

if isSimilarityOrAffine
  % Use the 3x2 affine2d syntax to have last column automatically
  % added to tform matrix. This prevents precision issues from
  % propagating downstream.
  tform = affine2d(tmatrix(:,1:2));
else % projective
  tform = projective2d(tmatrix);
end

%=========================================================================
=======
% Check runtime status and report error if there is one
%=========================================================================
=======
function checkRuntimeStatus(statusCode, status)
coder.internal.errorIf(status==statusCode.NotEnoughPts, ...
  'vision:estimateGeometricTransform:notEnoughPts');

coder.internal.errorIf(status==statusCode.NotEnoughInliers, ...
  'vision:estimateGeometricTransform:notEnoughInliers');

%=========================================================================
=======
% Parse and check inputs
%=========================================================================
=======
function [points1, points2, maxNumTrials, confidence, maxDistance, ...
  sampleSize, status, classToUse] = parseInputs(statusCode, ...
  matchedPoints1, matchedPoints2, transform_type, varargin)

isSimulationMode = isempty(coder.target);
if isSimulationMode
  % Instantiate an input parser
  parser = inputParser;
  parser.FunctionName = 'estimateGeometricTransform';
```

```
    % Specify the optional parameters
    parser.addParameter('MaxNumTrials', 1000);
    parser.addParameter('Confidence',   99);
    parser.addParameter('MaxDistance',  1.5);

    % Parse and check optional parameters
    parser.parse(varargin{:});
    r = parser.Results;

    maxNumTrials = 500;
    confidence   = 75;
    maxDistance  = 3;

else
    % Instantiate an input parser
    parms = struct( ...
        'MaxNumTrials',       uint32(0), ...
        'Confidence',         uint32(0), ...
        'MaxDistance',        uint32(0));

    popt = struct( ...
        'CaseSensitivity', false, ...
        'StructExpand',    true, ...
        'PartialMatching', false);

    % Specify the optional parameters
    optarg       = eml_parse_parameter_inputs(parms, popt,...
        varargin{:});
    maxNumTrials = eml_get_parameter_value(optarg.MaxNumTrials,...
        1000, varargin{:});
    confidence   = eml_get_parameter_value(optarg.Confidence,...
        99, varargin{:});
    maxDistance  = eml_get_parameter_value(optarg.MaxDistance,...
        1.5, varargin{:});
end

% Check required parameters
sampleSize = checkTransformType(transform_type);

[points1, points2] = vision.internal.inputValidation.checkAndConvertMatchedPoints(...
    matchedPoints1, matchedPoints2, ...
    mfilename, 'MATCHEDPOINTS1', 'MATCHEDPOINTS2');

status  = checkPointsSize(statusCode, sampleSize, points1, points2);
```

```
% Check optional parameters
checkMaxNumTrials(maxNumTrials);
checkConfidence(confidence);
checkMaxDistance(maxDistance);

classToUse = getClassToUse(points1, points2);

maxNumTrials = int32(maxNumTrials);
confidence   = cast(confidence,  classToUse);
maxDistance  = cast(maxDistance, classToUse);
sampleSize   = cast(sampleSize,  classToUse);


%=========================================================================
=======
function status = checkPointsSize(statusCode, sampleSize, points1, points2)

coder.internal.errorIf( size(points1,1) ~= size(points2,1), ...
  'vision:estimateGeometricTransform:numPtsMismatch');

coder.internal.errorIf( ~isequal(class(points1), class(points2)), ...
  'vision:estimateGeometricTransform:classPtsMismatch');

if size(points1,1) < sampleSize
  status = statusCode.NotEnoughPts;
else
  status = statusCode.NoError;
end

%=========================================================================
=======
function r = checkMaxNumTrials(value)
validateattributes(value, {'numeric'}, ...
  {'scalar', 'nonsparse', 'real', 'integer', 'positive', 'finite'},...
  'estimateGeometricTransform', 'MaxNumTrials');
r = 1;

%=========================================================================
=======
function r = checkConfidence(value)
validateattributes(value, {'numeric'}, ...
  {'scalar', 'nonsparse', 'real', 'positive', 'finite', '<', 100},...
  'estimateGeometricTransform', 'Confidence');
r = 1;
```

```
%=========================================================================
=======
function r = checkMaxDistance(value)
validateattributes(value, {'numeric'}, ...
  {'scalar', 'nonsparse', 'real', 'positive', 'finite'},...
  'estimateGeometricTransform', 'MaxDistance');
r = 1;

%=========================================================================
=======
function sampleSize = checkTransformType(value)
list = {'similarity', 'affine', 'projective'};
validatestring(value, list, 'estimateGeometricTransform', ...
  'TransformType');

switch(lower(value(1)))
  case 's'
     sampleSize = 2;
  case 'a'
     sampleSize = 3;
  otherwise
     sampleSize = 4;
end

%=========================================================================
=======
function c = getClassToUse(points1, points2)
if isa(points1, 'double') || isa(points2, 'double')
  c = 'double';
else
  c = 'single';
end

%=========================================================================
=======
function flag = isTestingMode
isSimulationMode = isempty(coder.target);
coder.extrinsic('vision.internal.testEstimateGeometricTransform');
if isSimulationMode
  flag = vision.internal.testEstimateGeometricTransform;
else
  flag = eml_const(vision.internal.testEstimateGeometricTransform);
end

%=========================================================================
=======
```

```
% Algorithms for computing the transformation matrix.
%=========================================================================
=======
function T = computeSimilarity(points1, points2, classToUse)
numPts = size(points1, 1);
constraints = zeros(2*numPts, 5, classToUse);
constraints(1:2:2*numPts, :) = [-points1(:, 2), points1(:, 1), ...
  zeros(numPts, 1), -ones(numPts,1), points2(:,2)];
constraints(2:2:2*numPts, :) = [points1, ones(numPts,1), ...
  zeros(numPts, 1), -points2(:,1)];
[~, ~, V] = svd(constraints, 0);
h = V(:, end);
T = coder.nullcopy(eye(3, classToUse));
T(:, 1:2) = [h(1:3), [-h(2); h(1); h(4)]] / h(5);
T(:, 3)   = [0; 0; 1];

%=========================================================================
=======
function T = computeAffine(points1, points2, classToUse)
numPts = size(points1, 1);
constraints = zeros(2*numPts, 7, classToUse);
constraints(1:2:2*numPts, :) = [zeros(numPts, 3), -points1, ...
  -ones(numPts,1), points2(:,2)];
constraints(2:2:2*numPts, :) = [points1, ones(numPts,1), ...
  zeros(numPts, 3), -points2(:,1)];
[~, ~, V] = svd(constraints, 0);
h = V(:, end);
T = coder.nullcopy(eye(3, classToUse));
T(:, 1:2) = reshape(h(1:6), [3,2]) / h(7);
T(:, 3)   = [0; 0; 1];

%=========================================================================
=======
function T = computeProjective(points1, points2, classToUse)
numPts = size(points1, 1);
p1x = points1(:, 1);
p1y = points1(:, 2);
p2x = points2(:, 1);
p2y = points2(:, 2);
constraints = zeros(2*numPts, 9, classToUse);
constraints(1:2:2*numPts, :) = [zeros(numPts,3), -points1, ...
  -ones(numPts,1), p1x.*p2y, p1y.*p2y, p2y];
constraints(2:2:2*numPts, :) = [points1, ones(numPts,1), ...
  zeros(numPts,3), -p1x.*p2x, -p1y.*p2x, -p2x];
[~, ~, V] = svd(constraints, 0);
h = V(:, end);
```

```
T = reshape(h, [3,3]) / h(9);

%==========================================================================
=======
function N = computeLoopNumber(sampleSize, confidence, pointNum, inlierNum)

pointNum = cast(pointNum, 'like', inlierNum);
inlierProbability = (inlierNum/pointNum)^sampleSize;

if inlierProbability < eps(class(inlierNum))
  N = intmax('int32');
else
  conf = cast(0.01, 'like', inlierNum) * confidence;
  one  = ones(1,    'like', inlierNum);
  num  = log10(one - conf);
  den  = log10(one - inlierProbability);
  N    = int32(ceil(num/den));
end

%==========================================================================
=======
function tform = computeTForm(sampleSize, points1, points2, indices, classToUse)

[samples1, normMatrix1] = ...
  vision.internal.normalizePoints(points1(indices, :)', 2, classToUse);
[samples2, normMatrix2] = ...
  vision.internal.normalizePoints(points2(indices, :)', 2, classToUse);

samples1 = samples1';
samples2 = samples2';

switch(sampleSize)
  case 2
     tform = computeSimilarity(samples1, samples2, classToUse);
  case 3
     tform = computeAffine(samples1, samples2, classToUse);
  otherwise % 4
     tform = computeProjective(samples1, samples2, classToUse);
     tform = tform / tform(end);
end
tform = normMatrix1' * (tform / normMatrix2');

%==========================================================================
=======
function dis = evaluateTForm(sampleSize, threshold, tform, points1, points2)
```

```
pt = points1 * tform(1:2, 1:2);
pt = bsxfun(@plus, pt, tform(3,1:2));

if sampleSize == 4
  denom = bsxfun(@plus, points1 * tform(1:2, 3), tform(3,3));

  tf = abs(denom) > eps(class(points1));

  pt(tf,:)  = bsxfun(@rdivide, pt(tf,:), denom(tf));

  % Mark these points invalid by setting it to a location far away from
  % point2
  pt(~tf,:) = bsxfun(@plus, points2(~tf,:), threshold);
end

delta = pt - points2;
dis   = hypot(delta(:,1),delta(:,2));
dis(dis > threshold) = threshold;

%=========================================================================
=======
function [isFound, tform, inliers] = msac(points1, points2, maxNumTrials, ...
      confidence, maxDistance, sampleSize, classToUse)

threshold = maxDistance;
numPts    = size(points1, 1);
idxTrial  = 1;
numTrials = int32(maxNumTrials);
maxDis    = cast(threshold * numPts, classToUse);
bestDis   = maxDis;
bestTForm = eye([3,3], classToUse);

% Create a random stream. It uses a fixed seed for the testing mode and a
% random seed for other mode.
if isTestingMode()
  rng('default');
end

points1 = cast(points1, classToUse);
points2 = cast(points2, classToUse);

while idxTrial <= numTrials

  indices = randperm(numPts, sampleSize);

  tform = computeTForm(sampleSize, points1, points2, indices, classToUse);
```

```
  dis = evaluateTForm(sampleSize, threshold, tform, points1, points2);

  accDis = sum(dis);

  if accDis < bestDis
     bestDis = accDis;
     bestTForm = tform;
     inlierNum = cast(sum(dis < threshold), classToUse);
     num = computeLoopNumber(sampleSize, confidence, numPts, inlierNum);
     numTrials = min(numTrials, num);
  end
  idxTrial = idxTrial + 1;
end

distances = evaluateTForm(sampleSize, threshold, bestTForm, points1, points2);
inliers = (distances < threshold);
isFound = (sum(inliers) >= sampleSize);

if isFound
  tform = computeTForm(sampleSize, points1, points2, inliers, classToUse);
  tform = tform / tform(3,3);
else
  tform = eye([3,3], classToUse);
end
```

TASK.m

```
clc; clear; close all;

%% Code from SURF.m will go here

%% TASK Detect cube, tell cube number, Check rotation
% Use Harris to check rotation. 6 faces x 4 possible rotations = 24 max matches
newlabel=[];

for i = 1:24
  boxImage=face(i).image;
  figure;
  imshow(boxImage);
  if (ndims(boxImage)>2)
     boxImage=rgb2gray(boxImage);
  end
  title('Template');
  boxPoints = detectHarrisFeatures(boxImage);
```

```
figure;
imshow(boxImage);
title('1000 Strongest Feature Points from Template');
hold on;
plot(selectStrongest(boxPoints, 1000));
% plot(boxPoints, 'showPixelList', true, 'showEllipses', false);
% im=imread('clutteredDesk.jpg');
im=webim;
figure;
imshow(im);
if (ndims(im)>2)
   im=rgb2gray(im);
end

scenePoints = detectHarrisFeatures(im);
title('1000 Strongest Feature Points from Scene Image');
hold on;
plot(selectStrongest(scenePoints, 1000));
% plot(scenePoints, 'showPixelList', true, 'showEllipses', false);

[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(im, scenePoints);

boxPairs = matchFeatures(boxFeatures, sceneFeatures);

matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, im, matchedBoxPoints, ...
   matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');




[tform, inlierBoxPoints, inlierScenePoints, status] = ...
   estimateGeometricTransform(matchedBoxPoints, matchedScenePoints, 'affine'); %Make this
function return the status


display(status);


if (status == 0)
   newlabel=[newlabel face(i).label];
   break;
```

```
  end
end


for g=1:8

  c=0;
  for h=1:6
     if newlabel(1)==cube(g).arr(h)
        c=c+1;
     end
     if newlabel(2)==cube(g).arr(h)
        c=c+1;
     end
     if newlabel(3)==cube(g).arr(h)
        c=c+1;
     end
     if c==3
        fprintf('\nCube %d Found\n', g);
        break;
     end
  end
end

webim=imread('21.jpg');
webim=imrotate(webim,180);
status=1;
for i = 1:4:24
  boxImage=face(i).image;
  figure;
  imshow(boxImage);
  if (ndims(boxImage)>2)
     boxImage=rgb2gray(boxImage);
  end
  title('Guitar template');
  boxPoints = detectHarrisFeatures(boxImage);
  figure;
  imshow(boxImage);
  title('500 Strongest Feature Points from Box Image');
  hold on;
  plot(selectStrongest(boxPoints, 500));
  % plot(boxPoints, 'showPixelList', true, 'showEllipses', false);
  % im=imread('clutteredDesk.jpg');
  im=webim;
  figure;
  imshow(im);
```

```
  im=rgb2gray(im);
  scenePoints = detectHarrisFeatures(im);
  title('1000 Strongest Feature Points from Scene Image');
  hold on;
  plot(selectStrongest(scenePoints, 1000));
  % plot(scenePoints, 'showPixelList', true, 'showEllipses', false);

  [boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
  [sceneFeatures, scenePoints] = extractFeatures(im, scenePoints);

  boxPairs = matchFeatures(boxFeatures, sceneFeatures);

  matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
  matchedScenePoints = scenePoints(boxPairs(:, 2), :);
  figure;
  showMatchedFeatures(boxImage, im, matchedBoxPoints, ...
     matchedScenePoints, 'montage');
  title('Putatively Matched Points (Including Outliers)');


  %while(status= IGNORE THIS


  [tform, inlierBoxPoints, inlierScenePoints, status] = ...
     estimateGeometricTransform(matchedBoxPoints, matchedScenePoints, 'affine'); %Make this
function return the status


  display(status);


  if (status == 0)
     for j = 1:4
        display(i);

        original = face(i).image;
        original=rgb2gray(original);
        imshow(original);
        text(size(original,2),size(original,1)+15, ...
           'Image courtesy of Massachusetts Institute of Technology', ...
           'FontSize',7,'HorizontalAlignment','right');

        scale = 0.7;
        J = imresize(original, scale); % Try varying the scale factor.

        %           theta = 30;
```

```
        distorted = webim; % Try varying the angle, theta.
        distorted=rgb2gray(distorted);
        figure, imshow(distorted)

        ptsOriginal  = detectSURFFeatures(original);
        ptsDistorted = detectSURFFeatures(distorted);

        [featuresOriginal,  validPtsOriginal]  = extractFeatures(original,  ptsOriginal);
        [featuresDistorted, validPtsDistorted] = extractFeatures(distorted, ptsDistorted);

        indexPairs = matchFeatures(featuresOriginal, featuresDistorted);

        matchedOriginal  = validPtsOriginal(indexPairs(:,1));
        matchedDistorted = validPtsDistorted(indexPairs(:,2));

        figure;
        showMatchedFeatures(original,distorted,matchedOriginal,matchedDistorted);
        title('Putatively matched points (including outliers)');

        [tform, inlierDistorted, inlierOriginal] = estimateGeometricTransform(...
           matchedDistorted, matchedOriginal, 'similarity');

        figure;
        showMatchedFeatures(original,distorted,inlierOriginal,inlierDistorted);
        title('Matching points (inliers only)');
        legend('ptsOriginal','ptsDistorted');

        Tinv  = tform.invert.T;

        ss = Tinv(2,1);
        sc = Tinv(1,1);
        scaleRecovered = sqrt(ss*ss + sc*sc)
        thetaRecovered = atan2(ss,sc)*180/pi

        break;
      end
    end
    if (status == 0)
      fprintf('MATCHED in LOOP 1\n');
      break;
    end
    fprintf('\n NO MATCH!! ROTATE THIS CUBE \n');
end
```