University of Burgundy

Assignment II

Object Recognition & Augmented Reality with Homographies

Kimberly Grace Sevillano Colina

1 Introduction

In this project, we will explore object recognition and robust model fitting using RANSAC for homography estimation. Our goal is to automatically locate a given reference object, such as a painting, in different images of a scene. This is useful for applications like augmented reality or robotics, where a robot with a camera needs to recognize and locate objects in its environment.

We will first detect keypoints, extract features, and match points between images. Then, we will use RANSAC to estimate a robust transformation between the reference view and target views, ensuring robustness against incorrect matches. We will focus on using tensors in PyTorch(optional) for this assignment, without relying on high-level OpenCV implementations. However, you may use OpenCV for debugging or result verification, and consult OpenCV tutorials on image matching and homography estimation.

2 Part I - Environment Setup & Useful Commands

2.1 Environment Setup

To set up the conda environment for working on the local machine, follow the steps below:

- Create a new conda environment: `conda create -n cv_recognition`
- Activate the environment: `conda activate cv_recognition`
- Install Jupyter Notebook: `conda install jupyter`
- Navigate to the workspace directory of the course and run Jupyter: `jupyter - notebook`

2.2 Input Images

The input images for this task are located in the `data/` folder within the `CV_Assignment_2_RecognitionAR` file.

Figure 1: These are all the images that are shown in the folder.

3 Part II - Extracting Features and Matching using OpenCV

3.1 Feature Extraction and Keypoint Detection

In this subsection, we focus on extracting features and detecting keypoints in the image. To accomplish this, we use the Scale-Invariant Feature Transform (SIFT) detector and descriptor, implemented in OpenCV. We create a custom SIFT detector object with user-defined parameters for contrast threshold and the number of features to be detected.

The function `extract_features_custom` is defined as follows:

```
def extract_features_custom(image, contrast_threshold=0.02, n_features=1000):
    # Create a SIFT detector object with custom parameters
    sift = cv2.SIFT_create(contrastThreshold=contrast_threshold, nfeatures=n_features)
    # Detect keypoints and compute descriptors
    keypoints, descriptors = sift.detectAndCompute(image, None)
    # Select the top N keypoints
    n_keypoints = 1000
    keypoints = keypoints[:n_keypoints]
    return keypoints, descriptors
```

Figure 2: Keypoints detected by SIFT for each image.

3.2 Matching Keypoints

In the matching keypoints subsection, we focus on finding corresponding keypoints between two images. The function *find_matches_bf_op* takes two images, their keypoints, descriptors, and a ratio threshold as input. It utilizes a brute-force approach to match keypoints, calculating the Euclidean distance between descriptor pairs.

$$d = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

where d is the distance between two descriptors a and b, and n is the length of the descriptors.

The function performs a ratio test by comparing the distances of the closest and second closest match for each descriptor. If the ratio of these distances is less than the provided ratio threshold (default 0.7), the match is

considered valid and added to the list of matches. Finally, the function returns the matches and a visualization of the matched keypoints between the two images.

$$\text{Ratio} = \frac{d_{closest}}{d_{second\_closest}}$$

where Ratio is the distance ratio, $d_{closest}$ is the distance to the closest match, and $d_{second\_closest}$ is the distance to the second closest match. If the ratio is less than the provided ratio threshold, the match is considered valid.

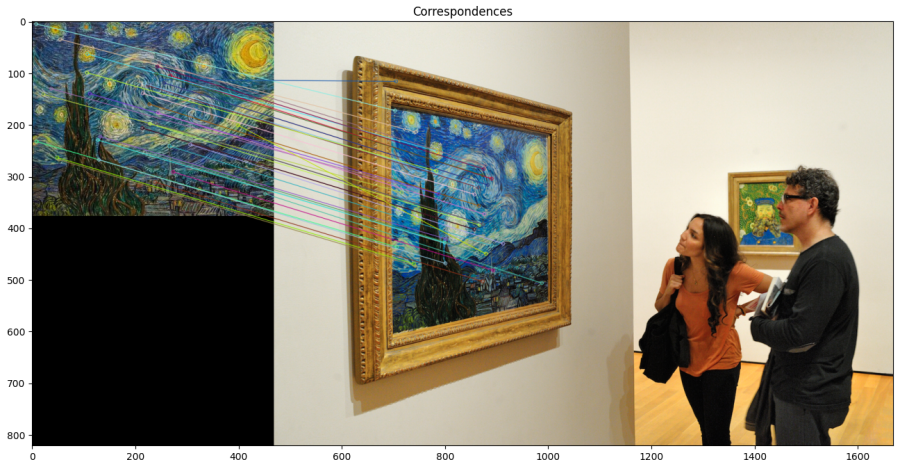


Figure 3: The figure shows the correspondences found between the pair of example images.

4 Part III Robust 2D Homography Model Fitting

4.1 Model Fitting Constraints for 2D Homography

In this subsection, we describe the model fitting constraints for 2D Homography transformation. The function get_corresponding_points retrieves the corresponding 2D points from the matched keypoints of two images.

To fit a 2D Homography transformation model, a minimum of four non-collinear corresponding 2D points are required. This is because a 2D Homography is a 3x3 matrix (eight degrees of freedom plus one for scaling), and each point correspondence provides two constraints. Thus, with four point correspondences, we have eight constraints, which is sufficient to estimate the Homography matrix.

4.2 Direct Linear Transformation (DLT) Algorithm

In this subsection, we describe the Direct Linear Transformation (DLT) algorithm used for estimating the Homography matrix given a set of 2D to 2D point correspondences. Given a scene image $\vec{X}$ (homogeneous representation) and a projective transformation H, we have the camera image $\vec{X}' = H\vec{X}$.

If we write $\vec{X}i = (x_i, y_i, w_i)^T$, $\vec{X}'_i = (x'_i, y'_i, w'_i)^T$ and $H = \begin{pmatrix} \vec{h}_1^T \\ \vec{h}_2^T \\ \vec{h}_3^T \end{pmatrix}$, where $\vec{h}_i^T$ is a row vector denoting the ith row of H, we have

$$\begin{pmatrix} \vec{0}^T & -w'_i\vec{X}_i^T & y'_i\vec{X}_i^T \\ w'_i\vec{X}_i^T & \vec{0}^T & -x'_i\vec{X}_i^T \end{pmatrix} \begin{pmatrix} \vec{h}_1 \\ \vec{h}_2 \\ \vec{h}_3 \end{pmatrix} = \vec{0}$$

This can be written as

$$A_i\vec{h} = \vec{0}$$

where A_i is a 2×9 matrix and $\vec{h}$ is 9×1.

Therefore, $n(n \geq 4)$ correspondences will provide a matrix A with dimension $2n \times 9$ and the solution of h can be written as:

$$\text{minimize } \left\|A\vec{h}\right\| \text{ s.t. } \|h\| = 1 \text{is equivalent to minimize } \vec{h}^T A^T A\vec{h} \text{ s.t. } \|h\| = 1.$$

The solution is the eigenvector of $A^T A$ associated with the smallest eigenvalue.

But we know that the DLT algorithm can lead to unstable results because of the numerical calculation, inaccurate point correspondences, and so on. This motivates a robust algorithm, such as RANSAC.

4.2.1 Normalized DLT

For the numerical calculation issues in the DLT algorithm, a normalization process should be applied. This step is very important for less well-conditioned problems such as DLT.

Given $n \geq 4$ point correspondences $\vec{X}i$ and $\vec{X}i'$, a similarity transformation T_1

$$T_1 = \begin{pmatrix} s & 0 & t_x \\ 0 & s & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

which consists of a translation and scaling, will take points $\vec{X}_i$ to a new set of points $\overrightarrow{\hat{X}}_i = T_1\vec{X}_i$ such that the centroid of the new points has the coordinate $(0,0)^T$, and their average distance from the origin is $\sqrt{2}$.

Suppose $\vec{X}_i = (x_i, y_i, 1)^T$, we have

$$\overrightarrow{\hat{X}}_i = T_1\vec{X}_i = \begin{pmatrix} sx_i + t_x \\ sy_i + t_y \\ 1 \end{pmatrix} = \begin{pmatrix} \hat{x}_i \\ \hat{y}_i \\ 1 \end{pmatrix}$$

The centroid of $\overrightarrow{\hat{X}}_i$ is

$$\begin{pmatrix} \overline{\hat{x}} \\ \overline{\hat{y}} \\ 1 \end{pmatrix} = \begin{pmatrix} s\overline{x} + t_x \\ s\overline{y} + t_y \\ 1 \end{pmatrix} = (0,0,1)^T$$

where the $\bar{x}$ and $\bar{y}$ denote the mean.
Therefore, $t_x = -s\bar{x}$ and $t_y = -s\bar{y}$.

The average distance between $\overrightarrow{\hat{X}}_i$ and the origin is

$$\frac{1}{n}\sum_i \sqrt{\hat{x}_i^2 + \hat{y}_i{}^2} = \frac{1}{n}\sum_i \sqrt{(sx_i - s\bar{x})^2 + (sy_i - s\bar{y})^2} = \frac{s}{n}\sum_i \sqrt{(x_i - \bar{x})^2 + (y_i - \bar{y})^2} = \sqrt{2}. \quad (1)$$

Therefore, s can be computed as:

$$s = \frac{\sqrt{2}}{\frac{1}{n}\sum_i \sqrt{(x_i - \bar{x})^2 + (y_i - \bar{y})^2}} \quad (2)$$

Similarly, a similarity transformation T_2 will take points $\vec{X}_i'$ to a new set of points $\overrightarrow{\hat{X}}_i' = T_2\vec{X}_i'$. Apply DLT to the correspondences $\overrightarrow{\hat{X}}i$ and $\overrightarrow{\hat{X}}_i$ and $\overrightarrow{\hat{X}}_i'$, we obtain a homography $\hat{H}$. Since

$$\vec{X}_i' = H\vec{X}_i = T_2^{-1}\overrightarrow{\hat{X}}_i' = T_2^{-1}\hat{H}\overrightarrow{\hat{X}}_i = T_2^{-1}\hat{H}T_1\vec{X}_i \quad (3)$$

we have the desired homography $H = T_2^{-1}\hat{H}T_1$.

In summary, the normalized DLT algorithm first applies a similarity transformation to the original point correspondences to make the computation more numerically stable. After that, the standard DLT algorithm is applied to the transformed points. Finally, the computed homography is transformed back to the original point space using the inverse of the similarity transformations.

Algorithm 1 2D Homography Matrix Estimation using DLT recipe

INPUT: $n \geq 4$ 2D point correspondences $\mathbf{x_i} \leftrightarrow \mathbf{x'_i}$
OUTPUT: H (2D homography matrix)

1: Compute the similarity transformation T for $\mathbf{x_i}$, transforming points to $\tilde{\mathbf{x}}_i$ with centroid $(0,0)^\top$ and average distance $\sqrt{2}$ from the origin
2: Compute the similarity transformation T′ for $\mathbf{x'_i}$, transforming points to $\tilde{\mathbf{x}}'_i$ with centroid $(0,0)^\top$ and average distance $\sqrt{2}$ from the origin
3: Apply DLT algorithm to correspondences $\tilde{\mathbf{x}}_i \leftrightarrow \tilde{\mathbf{x}}'_i$ to obtain homography $\tilde{\mathrm{H}}$
4: Compute H $\leftarrow$ T′$^{-1}\tilde{\mathrm{H}}$ T
5: **return** H

4.3 RANSAC Algorithm for Homography Estimation

The RANSAC (Random Sample Consensus) algorithm is a robust method for estimating the homography matrix H by iteratively selecting random subsets of point correspondences and fitting the model, while identifying and discarding outliers.

The RANSAC algorithm for homography estimation is as follows:

Algorithm 2 RANSAC Algorithm for Homography Estimation

INPUT: Data set S, containing inliers and outliers, model instantiation parameters s, distance threshold t, inliers threshold T, number of trials N
OUTPUT: A robustly fitted model using the consensus set S_i

1: **for** $i = 1$ to N **do**
2: Randomly select a sample of s data points from S and instantiate the model from this subset
3: Determine the consensus set S_i of data points within distance threshold t of the model
4: **if** size of $S_i \geq T$ **then**
5: Re-estimate the model using all points in S_i
6: Terminate
7: **end if**
8: **end for**
9: Select the largest consensus set S_i after N trials
10: Re-estimate the model using all points in S_i
11: **return** the robustly fitted model

In the context of homography estimation, the RANSAC algorithm can be connected with the DLT algorithm as follows:

Randomly select 4 point correspondences (minimal set) and compute the homography H_{curr} using the normalized DLT algorithm. For each putative correspondence, calculate the symmetric transfer error:

$$d_i = d(\vec{X}'_i, H_{\text{curr}}\vec{X}_i) + d(\vec{X}i, H^{-1}_{\text{curr}}\vec{X}'_i) \tag{4}$$

Count the number of inliers m which have a distance $d_i < t$. If the number of inliers is greater than a threshold or provides a better fit, update the best homography $H_{\text{best}} = H_{\text{curr}}$ and record all the inliers. Update the number of trials N using the inlier ratio and desired probability of success p:

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^4)} \tag{5}$$

where $\epsilon = 1 - \frac{m}{n}$ is the outlier ratio.

After running the RANSAC algorithm, re-estimate the homography matrix H using the DLT algorithm with all inliers. Transform the points in the second image using H^{-1} to obtain the reconstructed scene image and compare it to the original scene image.

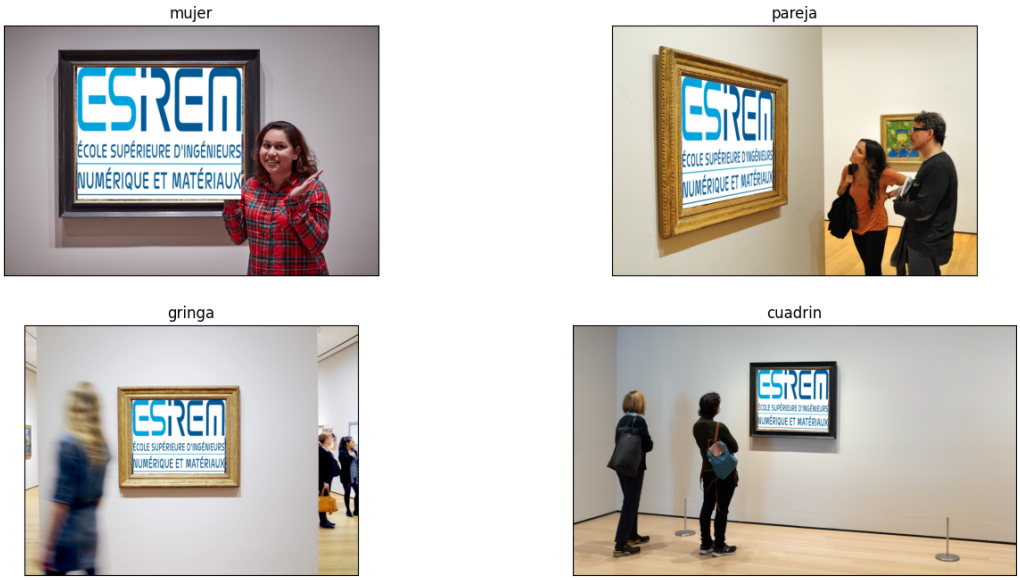


Figure 4: The figure shows the Esirem image substituting the paint location in each of the photos, after passing through the DLT and ransac algorithms.

4.4 Comparison with OpenCV Implementation

In this section, we compare the performance of our implementation of homography estimation with the OpenCV implementation, i.e., the cv.findHomography function with the RANSAC flag disabled, and using our own RANSAC routine in the estimation.

Figure 5: The mean difference between the homography matrix $H_{1_{cv}}$ and $H_{1_{own}}$ of the first image is 0.813.

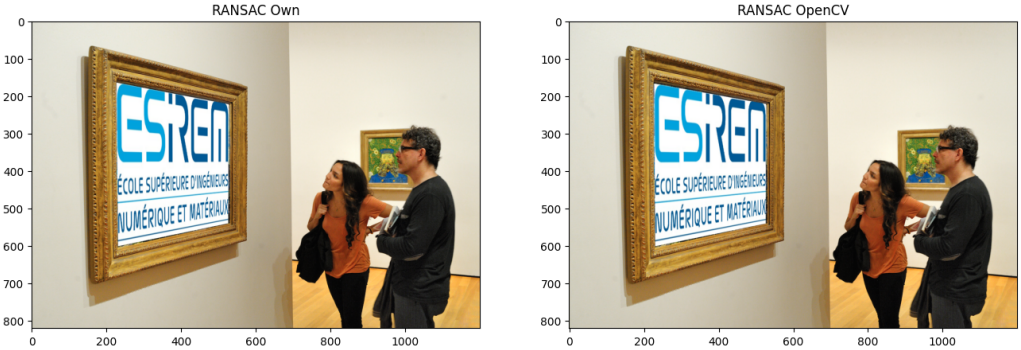


Figure 6: The mean difference between the homography matrix $H_{2_{cv}}$ and $H_{2_{own}}$ of the second image is 0.444.

Figure 7: The mean difference between the homography matrix $H_{3_{cv}}$ and $H_{3_{own}}$ of the third image is 0.0047.

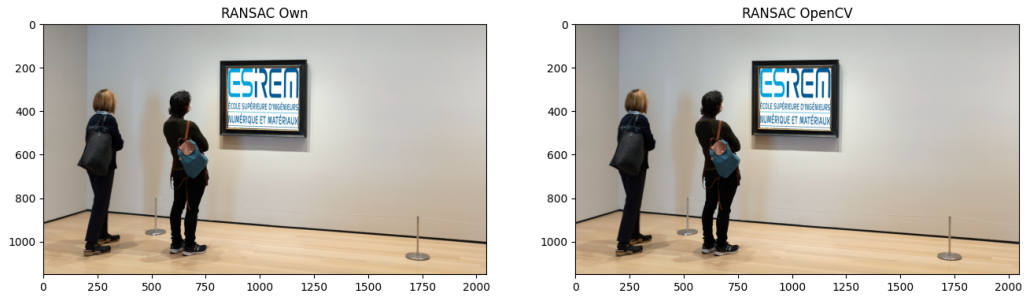


Figure 8: The mean difference between the homography matrix $H_{4_{cv}}$ and $H_{4_{own}}$ of the fourth image is 0.07896.

5 Extra Points: FAST Detector and ORB Descriptor

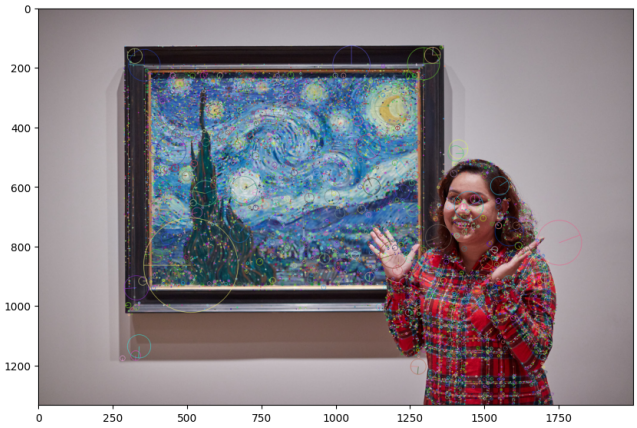
In this section, we explore the use of the FAST detector for keypoint detection and the ORB descriptor for feature description. The function $extract\_features\_FAST\_ORB$ demonstrates their application:

```
def extract_features_FAST_ORB(image):
    # Initialize FAST detector
    fast = cv2.FastFeatureDetector_create()
    # Detect keypoints with FAST
    keypoints = fast.detect(image, None)
    # Initialize ORB descriptor
    orb = cv2.ORB_create()
    # Compute descriptors with ORB
    keypoints, descriptors = orb.compute(image, keypoints)
    # Return keypoints and descriptors
    return keypoints, descriptors
```

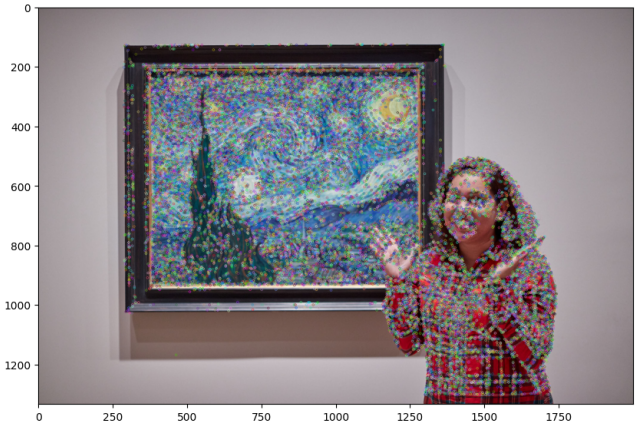
The FAST (Features from Accelerated Segment Test) detector is an efficient corner detection method that identifies keypoints in an image by analyzing the pixel intensity differences around a circular region. It is faster than other keypoint detectors like SIFT and SURF due to its simplicity and reduced computational requirements.

The ORB (Oriented FAST and Rotated BRIEF) descriptor is a binary descriptor that combines the strengths of the FAST detector and BRIEF (Binary Robust Independent Elementary Features) descriptor. ORB is computationally efficient and provides robustness to rotation and scale changes, making it a suitable choice for real-time applications.

By comparing the keypoints detected using the SIFT algorithm and those detected using the FAST detector and ORB descriptor, we can observe differences in the distribution, density, and number of keypoints in the images. The figure below illustrates the difference in keypoints between the SIFT and FAST-ORB methods:



(a)



(b)

Figure 9: Comparison of keypoints detected by SIFT (a) with 11796 and FAST-ORB (b) with 24545.

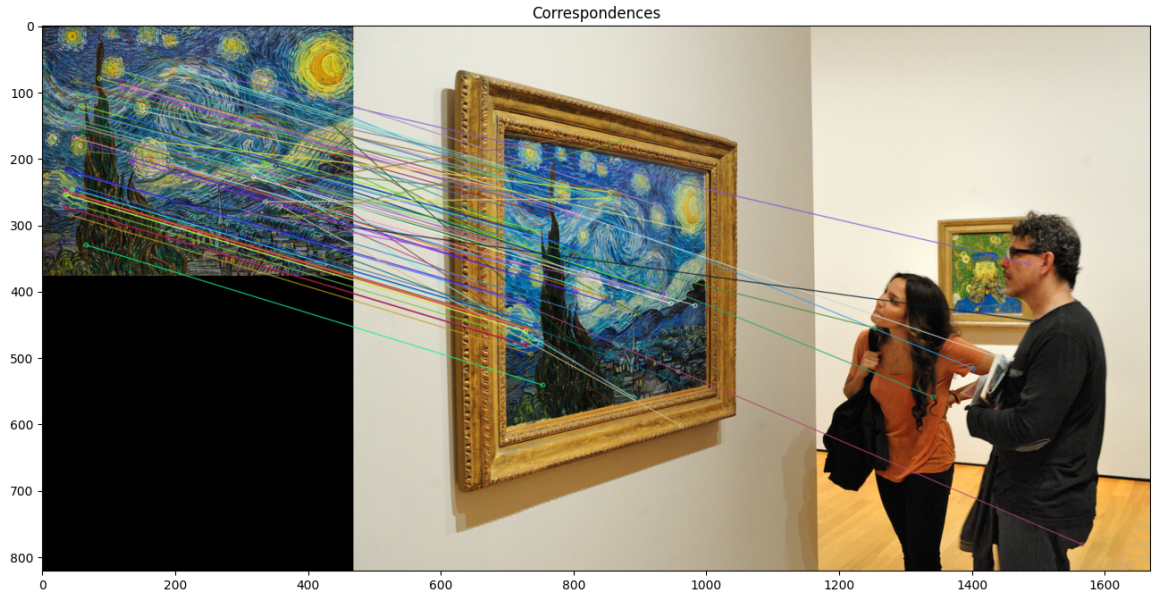


Figure 10: The figure shows the correspondences (FAST-ORB) found between the pair of example images.

5.1 Comparing Results

In the final section of this report, we have compared the performance of keypoint and descriptor extraction using SIFT and ORB-FAST. Although ORB-FAST proves to be a faster algorithm, its performance is not on par with

Figure 11: The figure shows the Esirem image substituting the paint location in each of the photos, using FAST detector and ORB descriptor when extracting features.

SIFT in terms of the number of inliers and the inlier ratio, which were found to be approximately 0.1.

Despite calculating a higher number of iterations (N) for optimal rendering with ORB-FAST, the lower inlier count and inlier ratio suggest that SIFT may still provide more reliable results for certain applications. Therefore, when deciding between SIFT and ORB-FAST, it is essential to consider the specific requirements of the application, including real-time processing demands and the robustness to transformations.