Top Image Systems

Digital Business Starts Here.

Automated Signature Verification

Implementation Guide

Version 5.2

contact@topimagesystems.com

www.topimagesystems.com

27 April, 2020

Contents

About Automated Signature Verification

Top Image Systems Automated Signature Verification (ASV) automatically verifies signatures by comparing them with existing, validated signature samples. The signature sample images can be in any graphic file format that can be loaded into a .NET bitmap. There is no minimum resolution requirement for the images.

To improve verification accuracy, ASV first pre-processes the images internally, removing borders, noise and superfluous white space, and normalizing them to a standard pixel width and height. It then applies six different verification methods, using digital signals such as enclosed areas, shadow areas, or intersections to measure differences and similarities between the signature sample and the signature image. Each of these measurements produce distance and weight values, and ultimately result in an overall final distance value. If this final distance value lies below the predefined threshold, the signature is considered to match the sample and is therefore a valid signature. If the final distance value lies above the predefined threshold, the signature is not considered a match and is therefore invalid. See Signature Verification Tester for more information on these measurements.

The ASV Signature Verification Tester is a graphic user interface that allows you to examine the verification results. You can use this tool for testing purposes, to examine problem signatures that did not pass verification, or simply to gain an understanding of how ASV works.

ASV can be implemented as a standalone system or as a functional add-on that can be easily integrated with eFLOW or with other existing systems via an API. The sample projects provided with the application demonstrate different implementation scenarios. See Sample projects for more information.

Installation

This section explains how to install ASV and obtain a license.

System requirements

The .NET Framework 4.0 or higher must be installed on the machine on which ASV is installed.

Installation package

The ASV installation package is delivered as a zip file, *TiS_ASV_[Version_Number].zip*.

The zip file contains the following folders. Only the **Bin** folder is required to run the application. The other folders are optional.

Folder	Contents
Bin	The ASV application files.
eFlow 5.2 Demo App *ChequeSignatureProcessing*	The eFLOW demo application *ChequeSignatureProcessing*. This demonstrates how ASV can be integrated into an eFLOW application that processes documents requiring signature verification.
Sample_Images	Sample signature images for use with the *ChequeSignatureProcessing* application or the Verification Tester tool.
Sig_DBs	Signatures database for use with the *ChequeSignatureProcessing* demo application.
VS_2012_API_Sample_ Projects	API sample projects. These contain source files for command line applications. See Sample projects for more information.

Install ASV

1. Copy the installation zip file to the machine on which you want to install ASV and extract it to a folder of your choice.
2. When prompted to enter a password, enter **tis**, then click **OK**.

Note: If you only intend to install ASV and do not require any of the sample files, you can extract the zip file elsewhere and just copy the *Bin* folder to the installation machine.

Obtain a license

After installing ASV, you must obtain a license. You need a separate license for each machine on which ASV is installed.

To obtain a license:

1. In the *Bin* folder, double-click on the file *ASV_HostIDGenerator.exe*.

   A license key is displayed in the **TiS ASV HostID Generator** window.

2. Click the **Copy to clipboard** button.
3. Paste the license key into an email and send the email to support@kofax.com.

4. Top Image Systems will send you a license file with a *.siglic* extension by return email.
5. Save the *.siglic* license file in the ASV *Bin* folder.

Signature Verification Tester

The ASV Signature Verification Tester allows you to compare two signature images and view the verification results. You can use this tool for testing purposes, to examine problem signatures that did not pass verification, or simply to gain an understanding of how ASV works.

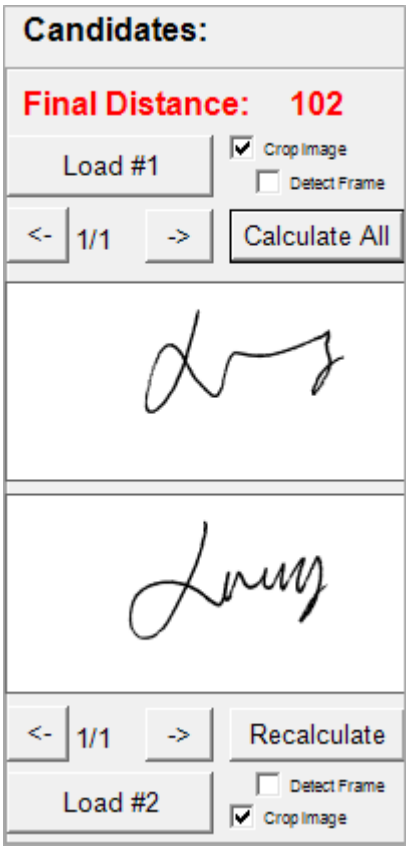
ASV applies six different verification methods, using digital signals such as enclosed areas, shadow areas, or intersections to measure differences and similarities between the signature sample and the signature image. Each of these measurements produce distance and weight values, and ultimately result in an overall final distance value. If this final distance value lies below the predefined threshold, the signature is considered to match the sample and is therefore a valid signature. If the final distance value lies above the predefined threshold, the signature is not considered a match and is therefore invalid. Most measurements use Levenshtein analysis to calculate the distance value.

Start the Verification Tester

To start the Verification Tester, in the ASV *Bin* folder, double-click the *Sig_Verification_Tester.exe* file.

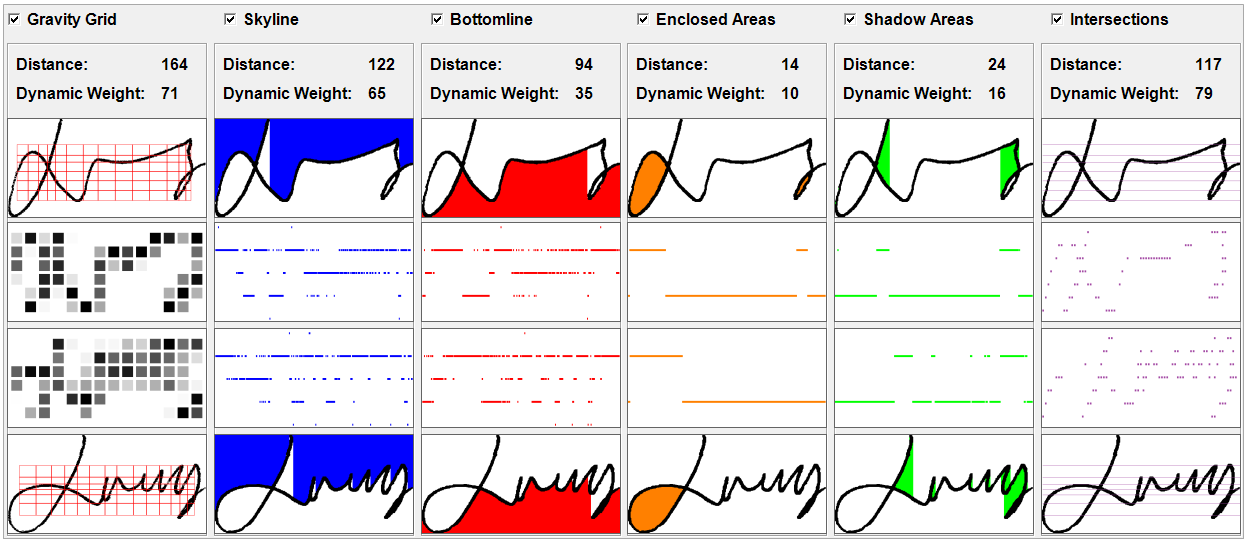
Compare images

1. Click the **Load #1** button and select the first signature image.
2. Click the **Load #2** button and select the second signature image.
3. If necessary, crop the image or remove frames. See Crop images and remove frames.
4. Click the **Calculate All** button.



The signatures are compared and the **Final Distance** value is calculated. If the **Final Distance** value is equal to or below 100, the signature is considered a match. If this value is above 100, the signature is not considered a match.

The results of each measurement are displayed.

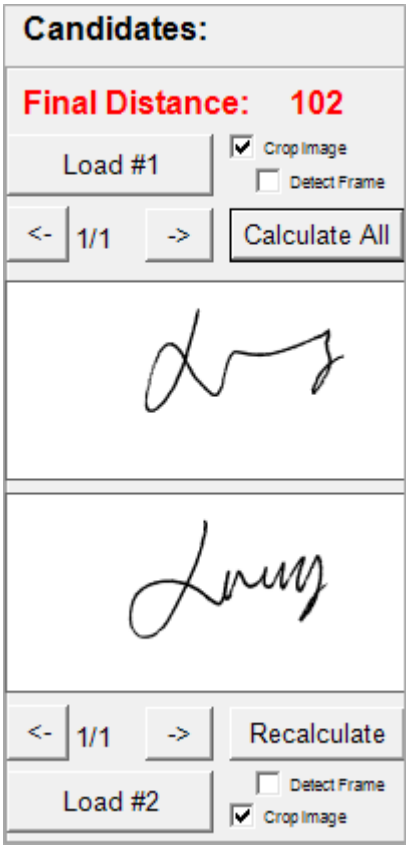


Crop images and remove frames

Superfluous white space and borders around the signature can distort verification results, so you should remove these from the image before calculating.

- To crop an image, thereby removing superfluous white space, check the **Crop image** check box.
- To remove borders, check the **Crop image** check box and then check the **Detect Frame** check box.

Note: These actions are only performed internally. The original signature images are not changed.



Gravity Grid

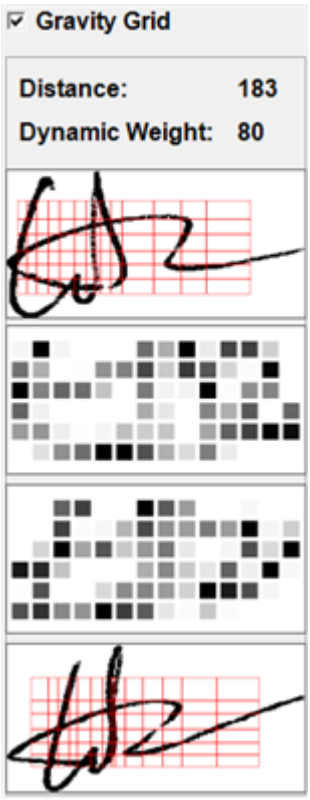
The **Gravity Grid** measurement calculates the pixel density within a dynamically calculated grid.

180,000 image pixels are spatially mapped for subsequent grid division. Grid division splits the image at the median density point both horizontally and vertically until a 14 by 6 grid has been formed.

Because each division is at the median of the space created by previous gridline, the grid spacing varies. The pixel density within each gravity grid block is then calculated as a percentage:

- 100% = every pixel is black
- 0% = every pixel white

This represents a normalized pixel density map of the overall signature. Each **Gravity Grid** block is compared to the corresponding value in the reference signature and the distance between the density percentages are calculated and averaged for the entire grid, yielding an overall distance value.



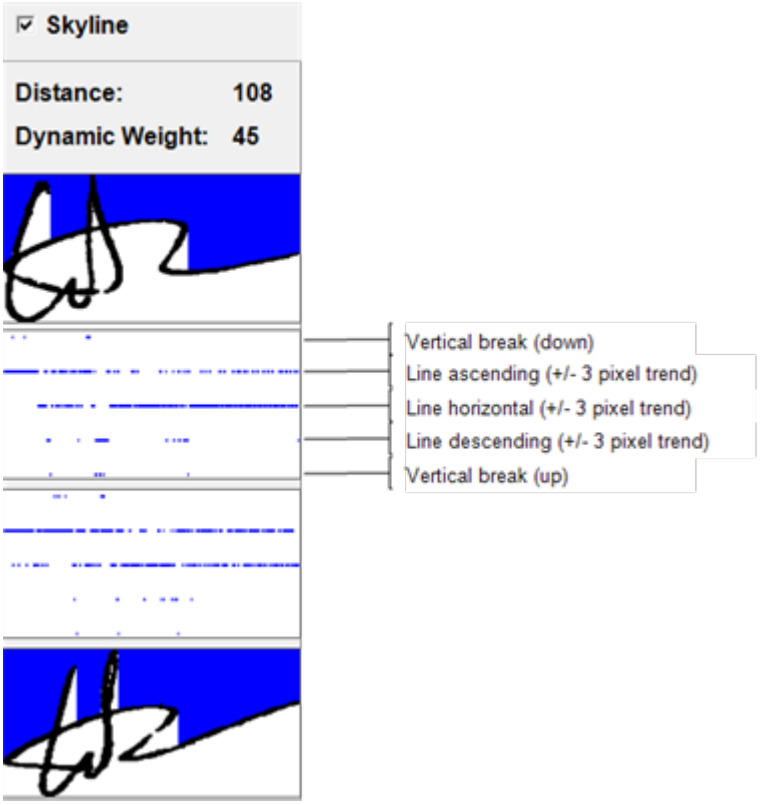
Skyline

The **Skyline** line is segmented into a 600 character string for a Levenshtein analysis.

Each character has 5 possible values corresponding to the following states:

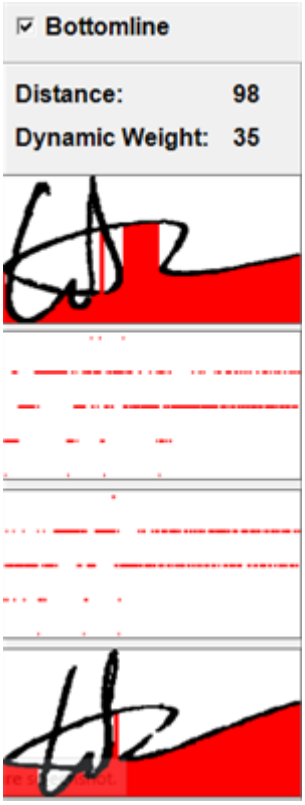
- Vertical break (down)
- Line ascending (+/- 3 pixel trend)
- Line horizontal (+/- 3 pixel trend)
- Line descending (+/- 3 pixel trend)
- Vertical break (up)

The **Skyline** measurement takes the normalized images and traces a line that is equal to the intersections of vertical lines drawn from top of the image boundary until an intersection with a signature pixel is reached.



Bottomline

Bottomline is similar to the **Skyline** measurement, only differing in that the vertical intersection lines are drawn from the bottom of image boundary instead of the top. See Skyline for more information.



Enclosed Areas

The **Enclosed Areas** measurement takes the normalized images and calculates every pixel that is fully encapsulated by the signature form. The segmentation is calculated into a 600 character binary string with the following states:

- Enclosed
- Not enclosed

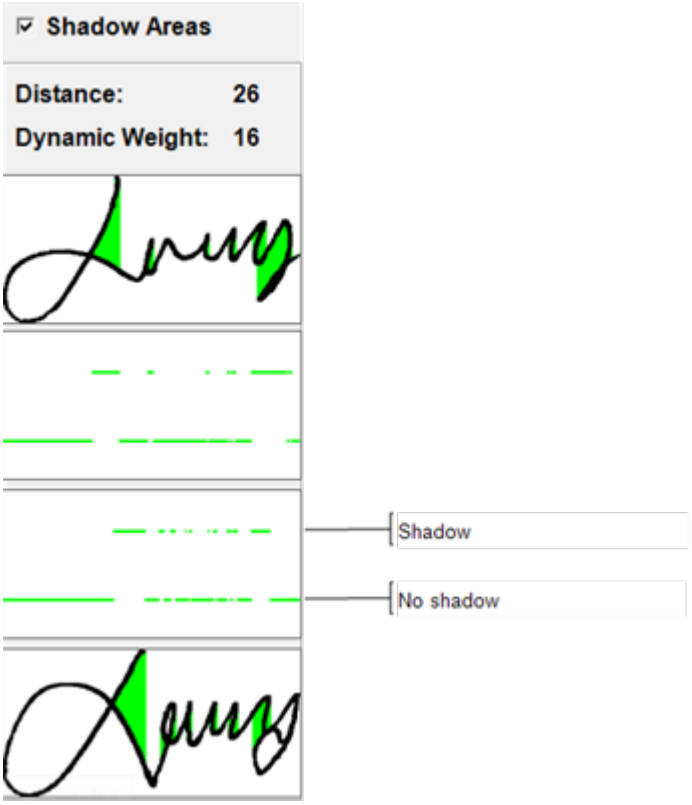
This resulting data is segmented for a Levenshtein analysis.

Shadow Areas

The **Shadow Areas** measurement takes the normalized images and calculates every pixel that cannot be reached by vertical lines drawn from top or the bottom of the image boundary. The segmentation is calculated into a 600 character binary string with the following states:

- Shadow
- No shadow

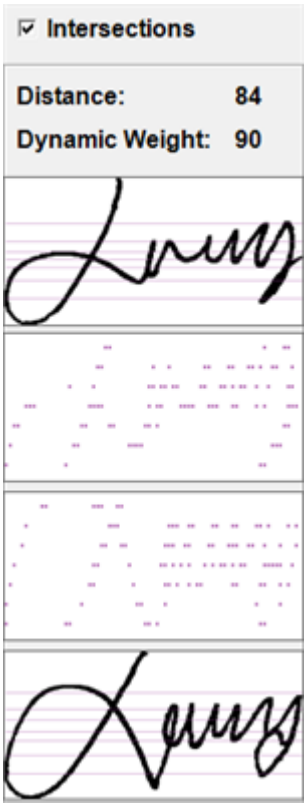
This resulting data is then segmented for a Levenshtein analysis.



Intersections

The **Intersections** measurement considers each pixel of the signature that intersects with the horizontal lines of the **Gravity Grid**.

The segmentation results in a 600 character string, with each character having 7 possible values. This yields a total of 4200 possible feature points for sequence comparison.



Sample projects

Sample Visual Studio API projects demonstrating various implementation scenarios are provided with the ASV installation package in the *VS_2012_API_Sample_Projects* folder. See ASV classes for detailed information on the classes and methods used in these projects.

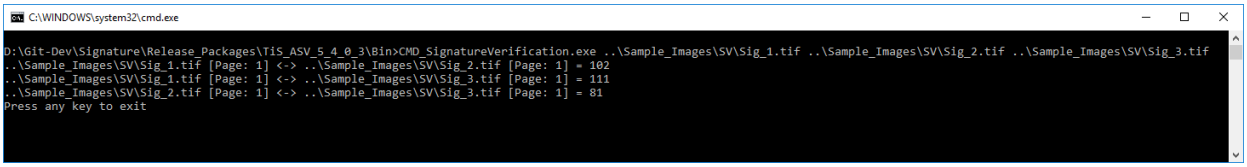
CMD_SignatureVerification

Location: *[Package folder]\VS_2012_API_Sample_Projects\CMD_SignatureVerification\CMD_SignatureVerification.csproj*.

CMD_SignatureVerification is a command line project (VS 2015 – C#.Net) that demonstrates how to use ASV to verify signatures that are stored as images in files (for example, **.tif* files).

In this example, the creation of a *Signature_Interface* is shown and then used to compare three signatures with each other. The signature files are passed as command line parameters to the application and the application then loads and compares the files via *DoSignatureVerification*. The output is displayed in a console and shows the verification result.

The following image shows the result that is produced by the test batch file *Test_Sig_Verification.bat*, which is available for testing in the *[Package folder]\Bin* folder.



CMD_ChequeSignatureProcessing

Location: *[Package folder]\VS_2012_API_Sample_Projects\CMD_ChequeSignatureProcessing\CMD_ChequeSignatureProcessing.csproj*.

CMD_ChequeSignatureProcessing is a command line project (VS 2015 – C#.Net) that demonstrates how to detect signatures in a given check image and compare them with pre-extracted signatures stored in a database.

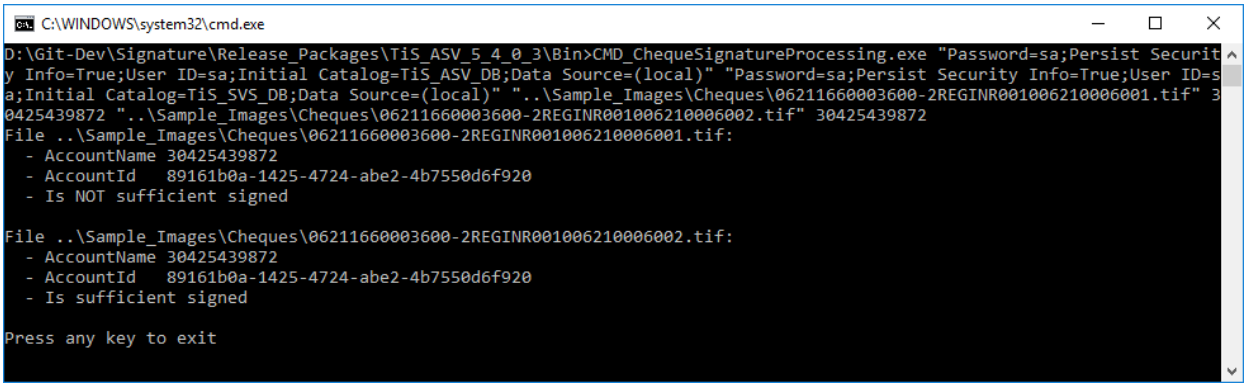
The signature files and the *AccountNo* in the *TiS_SVS_DB* database are passed as command line parameters and the image file of the check is validated against the account structure to establish whether sufficient signatures are present on the check. For example:

- Parameter 1 (mandatory): TIS_SVS_DB connection string
- Parameter 2 (mandatory): Image file of the check
- Parameter 3 (mandatory): Account number that is checked against parameter 2 for sufficient approval
- Parameter 4 (optional): Image file of the check
- Parameter 5 (optional): Account number that is checked against parameter 4 for sufficient approval

The features of the signatures in the database are extracted in an offline run and the results that are needed for later comparison are stored in the database. This saves a lot of resources during runtime and avoids having to perform feature extraction for the same specimen signatures again and again.

In this example, the creation of a *Signature_Interface* is shown and then *DoSignatureDetection* is used on a sample check. Found signatures are taken to extract features using *DoSignatureFeatureExtraction* and then compared against signatures in the database, which can be retrieved and checked per account using *TiS_SVS_DB* class methods such as *GetAccountIdByAccountName* and *IsAccountSufficientSigned*. The used samples require previous installation of the database samples *TiS_SVS_DB* and *TiS_ASV_DB*. The output is displayed in a console and shows the verification result.

The following image shows the result that is produced by the test batch file *Test_ChequeSignatureProcessing.bat*, which is available for testing in the *[Package folder]\Bin* folder.



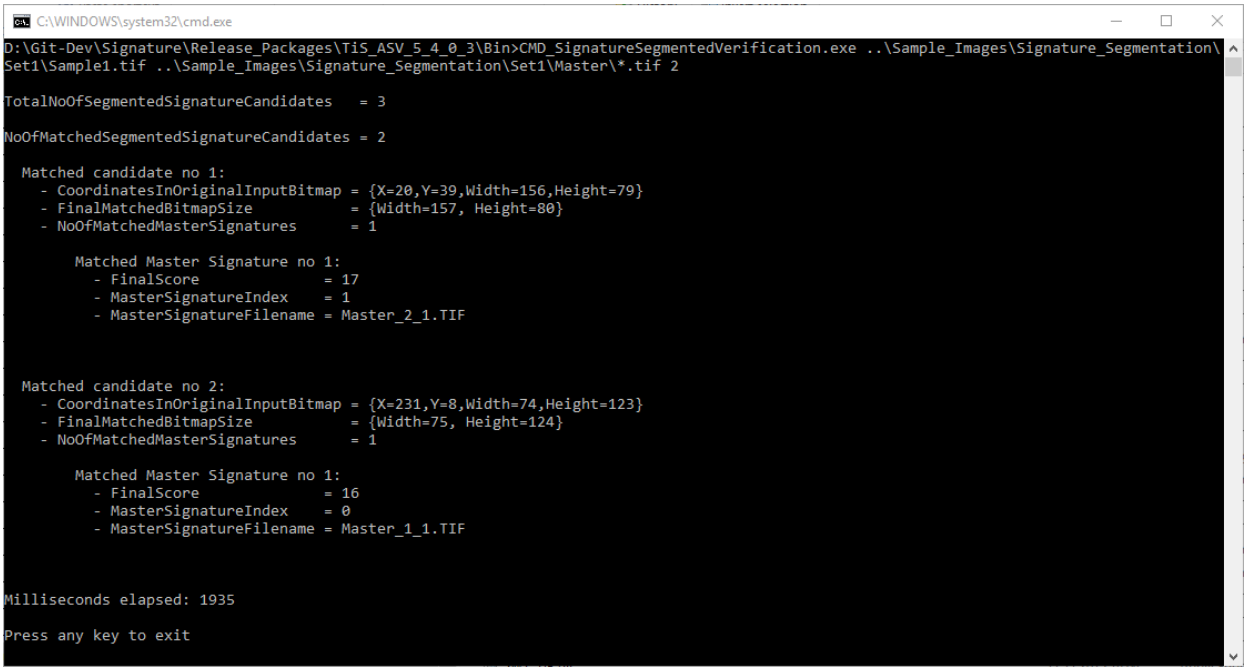
CMD_SignatureSegmentedVerification

Location: *[Package folder]\VS_2012_API_Sample_Projects\CMD_SignatureSegmentedVerification\CMD_SignatureSegmentedVerification.csproj*.

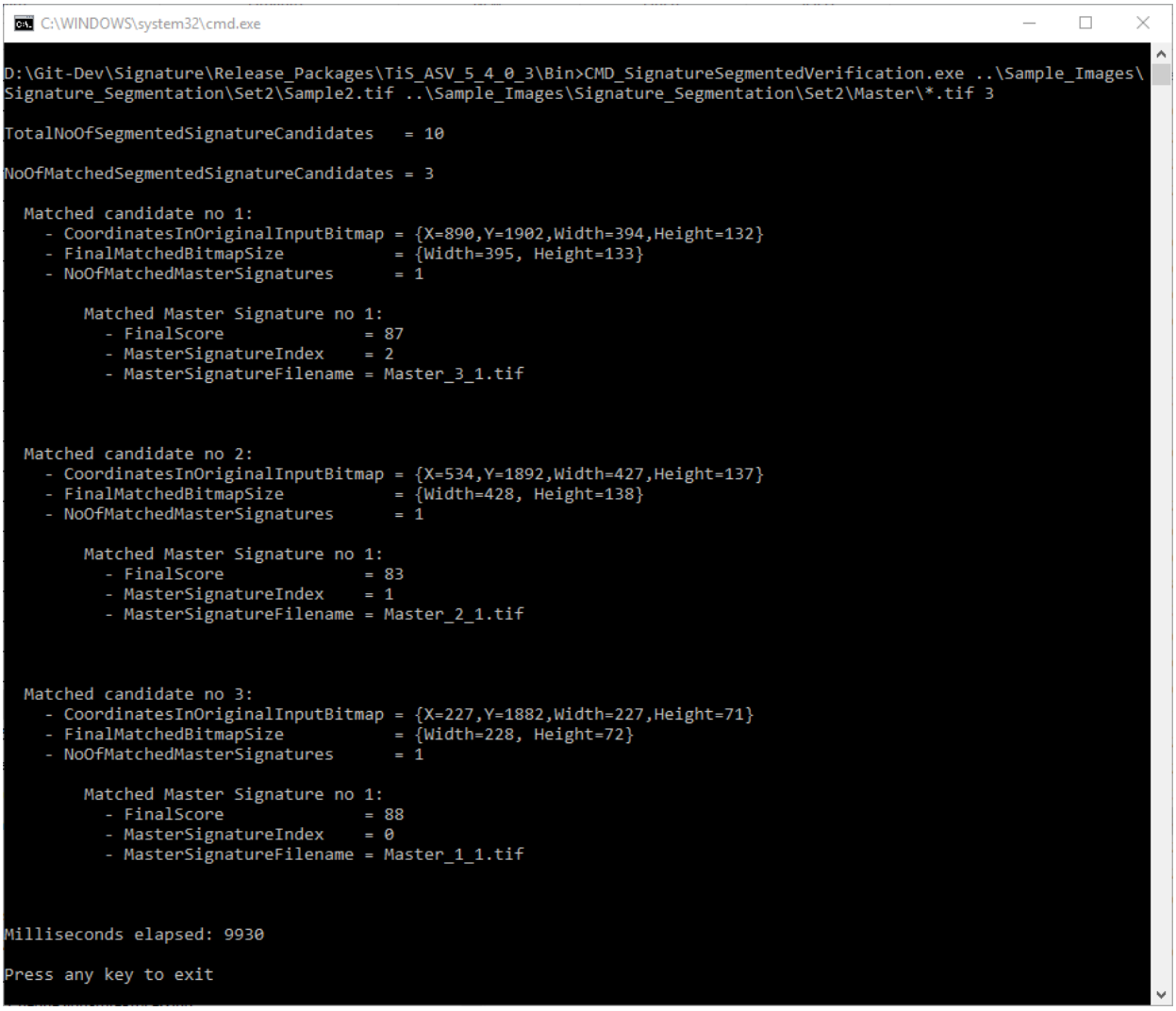
CMD_SignatureSegmentedVerification is a command line project (VS 2015 – C#.Net) that demonstrates how to detect and segment signatures in an unspecific document and compare them with several specimen signatures stored as TIFs in a folder.

In this example, the creation of a *Signature_Interface* is shown and then used to detect, segment and verify signatures. The sample document that is used for detection and segmentation is passed as a command line parameter to the application, as well as the folder containing the specimen signature files. Detection and verification is done using *DoSignatureSegmentationAndVerification*. The output is displayed in a console and shows the *SegmentationAndVerification* result.

The following image shows the result that is produced by the test batch file *Test_Sig_Segmentation_And_Verification Set1.bat*, which is available for testing in the *[Package folder]\Bin* folder.



The following image shows the result that is produced by the test batch file *Test_Sig_Segmentation_And_Verification Set2.bat*, which is available for testing in the *[Package folder]\Bin* folder.



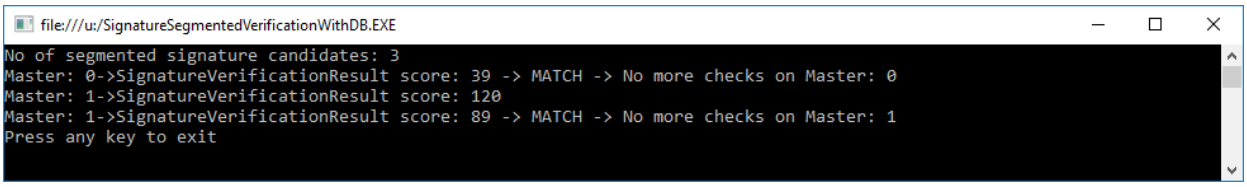
SignatureSegmentedVerificationWithDB

Location: *[Package folder]\VS_2012_API_Sample_Projects\SignatureSegmentedVerificationWithDB\SignatureSegmentedVerificationWithDB.csproj*.

SignatureSegmentedVerificationWithDB is a command line project (VS 2015 – C#.Net) that demonstrates how to detect signatures in a given check image and compare them with pre-extracted signatures in a database. The features of the signatures in the database are extracted in an offline run and the results that are needed for later comparison are stored in the database. This saves a lot of resources during runtime and avoids having to perform feature extraction for the same specimen signatures again and again.

In this example, the creation of a *Signature_Interface* is shown and then *DoSignatureDetection* is used on a sample check. Found signatures are taken to extract features using *DoSignatureFeatureExtraction* and then compared against signatures in the database, which can be retrieved using *TiS_ASV_DB* class methods. The used samples are defined as constants inside the project and require previous installation of the database sample *TiS_ASV_DB*. The output is displayed in a console and shows the verification result.

The following image shows the result produced by running the test project in Visual Studio.



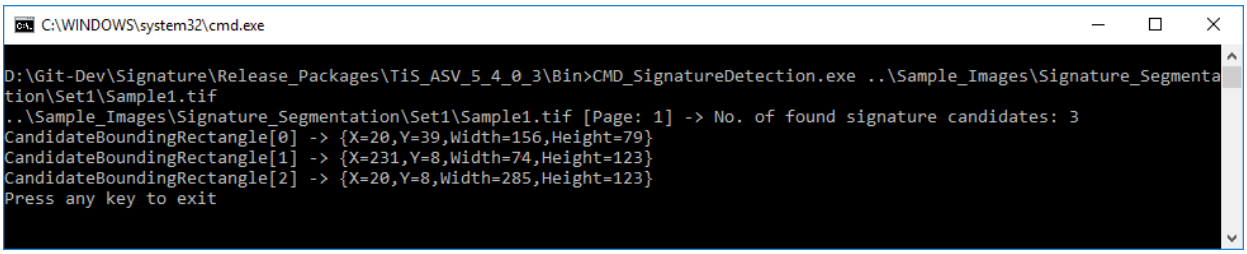
CMD_SignatureDetection

Location: *[Package folder]\VS_2012_API_Sample_Projects\CMD_SignatureDetection\CMD_SignatureDetection.csproj*.

CMD_SignatureDetection is a command line project (VS 2015 – C#.Net) that demonstrates how to detect signatures in unspecific documents that are stored as image files.

In this example, the creation of a *Signature_Interface* is shown and then used to detect signatures. The sample document that is used for detection is passed as a command line parameter to the application. Detection is done using *DoSignatureDetection*. The output is displayed in a console and shows the detection result as a list of signature candidates and the bounding rectangles.

The following image shows the result that is produced by the test batch file *Test_Sig_Detection.bat*, which is available for testing in the *[Package folder]\Bin* folder.



ASV classes

This section describes the ASV classes.

Signature_Interface:IDisposable

The *Signature_Interface* class contains methods for accessing all important ASV functions. All functionality described below can be called using this class.

Constructor

public Signature_Interface(string sApplicationStartupPath)

Parameters

Parameter	Description
sApplicationStartupPath	Path to the configuration files *Signature Verification_Configuration.txt* and *Signature Detection_Configuration.txt*.

Methods

DoSignatureVerification compares two images of signatures and delivers a distance value as the main result. The smaller this value is, the more alike the two signatures are. In the standard version of ASV, values <= 100 can be considered good enough to regard the signatures as a MATCH. Values > 100 are then a NOMATCH.

The *Crop* parameter for an image should always be set to *True* if the signatures in the image have not already been cropped to exactly fit the pixels of the signature. Cropping is then first performed in the function.

The *DetectFrame* parameter for an image should only be set to *True* if the signature in the image is within a frame, or if parts of the frame are still contained in the image. If these are horizontal or vertical, they are recognized and the frames or frame parts are removed, then cropping is performed again.

DoSignatureVerification - string

public Signature_Verification_Result DoSignatureVerification(string sFile1, int iPage1, bool bCrop1, bool bDetectFrame1, string sFile2, int iPage2, bool bCrop2, bool bDetectFrame2)

Parameters

Parameter	Description
sFile1	Full path to the first file with signature image #1
iPage1	Page number (0 based) in signature image #1
bCrop1	Cropping for sFile 1 activated/deactivated

Parameter	Description
bDetectFrame1	Frame detection for sFile 1 activated/deactivated
sFile2	Full path to the first file with signature image #2
iPage2	Page number (0 based) in signature image #2
bCrop2	Cropping for sFile 2 activated/deactivated
bDetectFrame2	Frame detection for sFile 2 activated/deactivated

Return value

Signature_Verification_Result (see class Signature_Verification_Result).

DoSignatureVerification - bitmap

public Signature_Verification_Result DoSignatureVerification(Bitmap bmp1, int iPage1, bool bCrop1, bool bDetectFrame1, Bitmap bmp2, int iPage2, bool bCrop2, bool bDetectFrame2)

Parameters

Parameter	Description
bmp1	.Net bitmap of the first signature
iPage1	Page number (0 based) in signature image #1
bCrop1	Cropping for bmp1 activated/deactivated
bDetectFrame1	Frame detection for bmp1 activated/deactivated
bmp2	.Net bitmap of the second signature
iPage2	Page number (0 based) in signature image #2
bCrop2	Cropping for bmp2 activated/deactivated
bDetectFrame2	Frame detection for bmp2 activated/deactivated

Return value

Signature_Verification_Result (see class Signature_Verification_Result).

DoSignatureDetection - string

public Signature_Detection_Result DoSignatureDetection(string sFile, int iPage)

DoSignatureDetection searches for possible signatures in the returned image. Small or insignificant structures are filtered out and the remaining structures, depending on their arrangement, are combined to find signature candidates. These are then passed back as a return value with information about their location in the original image.

Parameters

Parameter	Description
sFile	Full path to the file with the document page(s)
iPage1	Page number (0 based) in the document sFile

Return value

Signature_Detection_Result (see class Signature_Detection_Result).

DoSignatureDetection - bitmap

public Signature_Detection_Result DoSignatureDetection(Bitmap bmp1, int iPage)

Parameters

Parameter	Description
bmp1	.Net bitmap with document page(s)
iPage1	Page number (0 based) in signature image #1

Return value

Signature_Detection_Result (see class Signature_Detection_Result).

Signature_Verification_Result : IDisposable

Important public members

public int iTotalScore: Total distance calculated from the sum of the individual distances and weights of all used features.

In the standard version this is:

iTotalScore<=100 => MATCH

iTotalScore>100 => NOMATCH

Signature_Detection_Result:IDisposable

Important public members

public int iNoOfFoundSignatureCandidates: Number of found signature candidates.

Methods

Get_Signature_Candidate - bitmap

public Bitmap Get_Signature_Candidate_Bitmap(int iCandidateIndex)

Parameters

Parameter	Description
iCandidateIndex	Index (0 based) of the signature candidate

Return value

Bitmap of the examined candidate.

Get_Signature_Candidate - rectangle

public Rectangle Get_Signature_Candidate_Rectangle(int iCandidateIndex)

Parameters

Parameter	Description
iCandidParametateIndex	Index (0 based) of the signature candidate

Return value

Rectangle of the found candidate in the input image.