NIST API Requirements for Fingerprint SDK Biometric System Evaluation
ITL / Image Group

Test Protocol

The testing protocol is based on a protocol similar to the **Face Recognition Vendor Test** (FRVT) and **Fingerprint Vendor Technology Evaluation** (FpVTE) protocols. More information about these tests can be found off of their respective websites, *www.frvt.org* or *fpvte.nist.gov*.

A biometric *signature* refers to the biometric information for a given individual. This might be fingerprint, face or iris imagery. Let *G* represent some set of *gallery* signatures of known identity, and *P* represent a set of *probe* signatures of unknown identity. Further, let *g* represent one such image in the gallery *G*, and *p* some unknown probe image from *P.*

In evaluation, the identity of all signatures will be known. The distinction between target and query sets acknowledges that some biometric signatures are more indicative of enrolled, or background data (the gallery) and others as signatures to be verified or identified (the probe set).

NIST evaluates biometric systems according to the following operational model:

1. A template is created for each target element *t* in set *T.* The successful creation of a template is considered a *successful* enrollment, otherwise the is considered an enrollment *failure*.
2. Each image *q* in set *Q* is compared to each successfully generated template (i.e., all query elements are pair wise compared against all target elements). For each comparison, the score representing the similarity between the images is recorded. The score is assumed to be a floating-point number that represents the *similarity* between the images. Higher scores indicate similar biometric signatures, lower scores dissimilar ones. A pretermined *null* similarity score (typically zero) would be used given a system error, such as failure to enroll the target or query signature.

The collection of similarity scores across all comparisons yields a similarity matrix, *Q*x*T.* A typical test might consist of 12,000 (6,000 pairs) biometric signatures. A single test run will result in 36,000,000 comparisons producing 6,000 *genuine* or *match* scores and 35,994,000 *imposter*, or *non-match* scores. An image that cannot be enrolled will be included in the comparison statistics and will be treated as a zero match score.

Interfaces

To perform the tests, we require at minimum the following three interfaces (shown in C-style pseudocode prototypes) provided by the manufacturer. The interfaces should be in similar form to the following; they need not be an exact match.

(1) Get template size

```
unsigned int
get_template_size(const unsigned int height,
                  const unsigned int width);
```

Description

This function returns the maximum number of bytes for templates given the height and width of an input image.

Parameters

`height` **(input)**: The number of pixels indicating the height of the image.

`width` **(input)**: The number of pixels indicating the width of the image.

Return Value

This function returns the maximum number of bytes for a template.

(2) Enroll

```
int
enroll(const byte* const raw_image,
       const unsigned int height,
       const unsigned int width,
       byte* template);
```

Description

This function takes a raw image as input and outputs the corresponding template. The memory for the template is allocated before the call (i.e., **enroll** does not handle the memory allocation for the **template** parameter). The function returns either **success** (true) or **failure** (false). **Failure** indicates a failure to enroll the image. This will result in a *null* template that would be passed along in later comparisons.

Parameters

`raw_image` **input)**: The uncompressed, raw image to be used for template creation.

`height` **(input)**: The number of pixels indicating the height of the image.

`width` **(input)**: The number of pixels indicating the width of the image**.**

`template` **(output)**: The processed template of size returned by get_template_size.

Return Value

This function returns *zero* on **success** or a documented *non-zero* error code otherwise.

(3) **Compare**

```
int
compare(const byte* const probe_template,
        const unsigned int probe_height,
        const unsigned int probe_width,
        const byte* const gallery_template,
        const unsigned int gallery_height,
        const unsigned int gallery_width,
        float* score);
```

or, (difference highlighted in bold)

```
int
compare(const byte* const probe_image,
        const unsigned int probe_height,
        const unsigned int probe_width,
        const byte* const gallery_template,
        const unsigned int gallery_height,
        const unsigned int gallery_width,
        float* score);
```

Description

This function compares two templates and outputs the match score. The match score is a floating-point number. An alternative interface allows a vendor to substitute the template parameter (query_template) with the raw image (query_image). Naturally, it is expected that a vendor document whether their compare interface uses images or templates. It may be assumed that memory for the score value is allocated before the call.

Height and width parameters are provided, indicating the size for both the probe and gallery images. Vendors can make use of them if their algorithms are dependent on the size of the image. Often, the size of the target and query image will be the same.

Parameters

`probe_template` **(input)**: The processed query template returned from enroll. Alternatively, **probe_image** can be substituted for the **probe_template** parameter.

`probe` **(input)**: The number of pixels indicating the height of the query image.

`probe_width` **(input)**: The number of pixels indicating the width of the query image.

`target_template` **(input)**: The processed target template return from enroll.

`target_height` **(input)**: The number of pixels indicating the height of the target image.

`target_width` **(input)**: The number of pixels indicating the width of the target image.

`score` **(output)**: The score indicating the similarity between the two templates (or image and template).

Return Value

This function returns *zero* on **success** or a documented *non-zero* error code otherwise. On failure, the function is expected to return some default "null" similarity score. **Similarity scores should not be quantized**. This allows for more accurate system performance analysis.

(4) **Additional Functions or Parameters**

When initialization and configuration functions or additional parameters are needed, supplementary documentation must be provided describing their use. For example, how a complete match process may be invoked.

Performance & Packaging

NIST will be testing systems on either Linux or Microsoft Windows platforms commercial, off-the-shelf PCs. As of the fall of 2003, specifications of a "typical" machine are a (single processor), 2 GHz Intel Pentium IV with one GB RAM. In the event that other additional or alternate hardware is required, special arrangements will have to be made by and/or with NIST.

Because a typical test might consist of millions of comparisons, the combined enrolment and comparison time must not be excessive. An 'excessive' time is test-specific; a few seconds per comparison is acceptable for a few hundred comparisons, but would be unsatisfactory for millions of comparisons.

For example, for fingerprint vendors, we are recommending that the average enrollment time (on our 'typical' PC) should not exceed one second per enrollment, or 10 milliseconds per comparison. Based on these constraints, a 6,000 by 6,000 similarity matrix should take under 13.5 hours to compute. If a vendor's software has a time/performance tradeoff, then this tradeoff should be fully disclosed in the vendor's documentation. This will not only help NIST select a valid system parameterization given the constraints of any particular test, but illuminate to the target audience of internal and external reports, that such a tradeoff is present for a particular vendor's software.