





# Dedicated to machine vision software development

VISION fOr VISION specializes in the development of software for machine vision applications. We create standard and custom vision algorithms, tune them for best performance, adapt them to your platforms and assemble complete solutions that suit your needs.

If you couldn't find an off-the-shelf package to address your vision problem, this is where VISION fOr VISION can help you by providing consultancy, doing research & development and writing efficient code for you.

We enjoy twenty years of active development of image processing algorithms and machine vision products. Our mission is to help you expand your business in connection to computer vision by providing tailor-made software.

## Example projects

- Fast blob analysis for real-time particle detection
- Steel coil 3D inspection for planarity
- Registration of point clouds from range images
- Remote measurement of building vibrations
- Car plate detection and recognition
- Barcode and 2D code readers on mobile phones
- Location of rotated text on a display
- Stereoscopic map analysis for people detection
- Morphing of bitmap and vector images

# Why would you need us?

Since 2007, VISION fOr VISION helps vision integrators and OEMs achieve optimized results in a cost effective way.

Successful vision projects depend on the quality of the input images, itself relying on the system setup, lighting, optics, cameras and their proper adjustment. This takes good electromechanical integration skills.

On the other side they require a good mastering of the numerous image processing and machine learning techniques. Very often, the specificities of the problem need to be exploited, there is no size-fits-all solution. Every new case may require non-standard approaches.

It can take years to master all these technologies, and nothing replaces practice and experience. Trust a well-trained team able to judge at a glance the feasibility of a problem and to suggest an adequate processing pipeline, tailor-made from scratch or from existing resources.

## Software libraries

Simple problems can be addressed with configurable, GUI-based software. But practitioners are well aware that in more complex cases, nothing beats traditional programming in terms of development power. An alternative is provided by open source software, from which several image processing packages are available.

Anyway, these are not specifically focused on machine vision applications and are filled with black-box functions, to be taken "as-is". VISION fOr VISION develops and extends **mViz**, a large set of functions designed with machine vision applications in mind. Having full mastery of our code basis, we are best positioned to advice about usage, troubleshoot if needed, or augment the product when needed.

# mViz, comprehensive machine vision libraries

For processing, analysis, inspection, guidance and identification

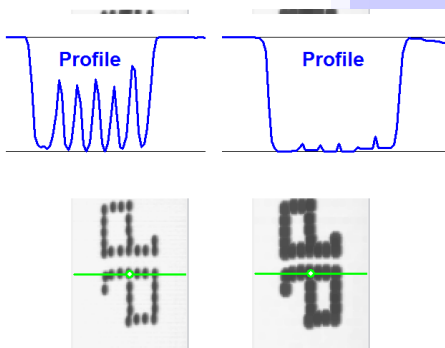
*Compatible with the C++, C++.NET, C# and Visual Basic environments*

*Linux gcc, 32 and 64 bits*

mViz is not just “another image processing package”. We want full customer satisfaction and we know that the learning curve for a first project can be costly. This is why free consultancy hours are included to help you establish a proof of concept very quickly and at a low risk.

mViz is licensed by separate modules or in bundles, run-times per machine, with an attractive pricing scheme. Feature enhancements, porting and special adaptations are possible on request.

## Image Processing



The processing functions are used to prepare the image by reducing unwanted nuisances such as noise or blur, or enhance some properties such as connexity of characters, contrast... They usually turn images into other images.

This function set comprises point-to-point operations such as pixel arithmetic, lookup table transforms or shading correction. It also handles geometric transforms like rotation and scaling that allow f.i. deskewing or size normalization.

**Point-to-point arithmetic & logic, grayscale & color transforms, shading correction, convolutions, grayscale**

## morphology, geometric transforms, polar unwarping and much more

```
// Sample code

// Rotate 30°, scale +20% around the center, with interpolation
Geometry::Rotate(XY(320, 240), XY(320, 240), 30, 1.2, Source,
Rotated, true);

// Two points shading correction with Black and White reference
Operator::Correct(Black, White, Source, Corrected, 0, 255);
```

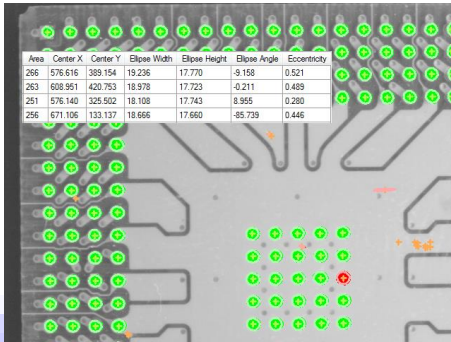
### Typical running times

(in milliseconds, 640x480 images, Core i7 @ 3.4 GHz)

Addition of two images	0.022
Pixel-wise maximum of two images	0.022
Pixel-wise greater-or-equal comparison	0.024
Automatic binarization	0.59
Two-point shading correction	0.71
Sobel magnitude	0.15
Prewitt magnitude	0.15
Binomial low-pass (3x3)	0.099
Uniform low-pass (25x25)	1.2
Sharpening filter (3x3)	0.12
Uniform high-pass (25x25)	1.3
User-defined convolution (3x3)	6.0
User-defined convolution (5x5)	11
Gaussian filter (15x15)	4.3
Bilateral filter (5x5)	14
Morphological dilation (3x3 square)	0.040
Morphological opening (7x7 diamond)	0.34
Morphological top-hat (5x5 octagon)	0.24
Median filter (7x7)	6.3
Downsampling (2:1)	0.83
Upsampling (1:2)	3.1
Rotation (nearest neighbor)	0.41
Rotation (bilinear)	1.2
Polar unwarping (bilinear)	2.6

# Image Analysis

The analysis functions extract condensed information from images or regions in order to characterize and classify 1D or 2D features.



They encompass histogram analysis, straight or curved profile extraction, and blob analysis. When numerical values are obtained, they can be used to compare objects to known references and discriminate between them.

## Profile processing, histogram statistics, blob analysis, region masking, contouring

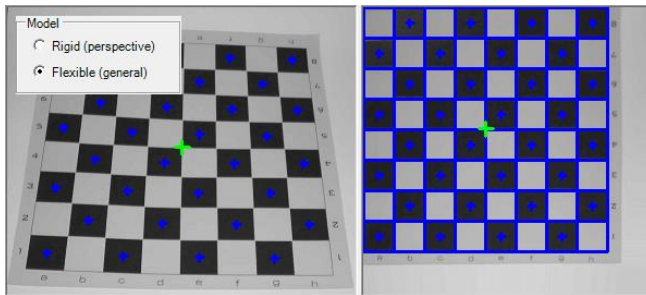
```
// Get the trimmed gray-level mean with 10% trimming
double Mean= Histogram::TrimmedMean(0.1, Source);

// Accumulate the histogram of the first (largest) blob
Blobs.PixelHistogram(0, Source, Histo);
```

## Typical running times

Cumulate 16 rows	0.0080
Trace a blob contour	0.020
Compute a contour area	0.0010
Fill a contour	0.089
Mean of an image	0.070
Entropy of an image	0.19
Compute the image histogram	0.19
Variance of the histogram	<0.001
Skewness of the histogram	<0.001
Best threshold from an histogram	0.0020
Segment an image (200 blobs)	0.16
Evaluate the gray-level averages (200 blobs)	<0.001
Evaluate the gray-level maxima (200 blobs)	<0.001
Evaluate the ellipse of inertia (200 blobs)	0.0070
Evaluate the Feret box (200 blobs)	0.0020
Sort blobs by area (200 blobs)	0.0040
Compute a blob histogram	0.13

# Image Calibration



An image is said to be calibrated when a mapping has been established between the pixel coordinates and some real-world coordinate system established on the

observed surface.

The image calibration set provides support to accurately compute the direct and inverse matching from a number of "anchor points", and to deal with perspective distortion as well as optical deformations.

## Scaling, isometry, Affinity, Perspective & Distortion Transforms

```
// Adjust the perspective from source to calibrated points
Model.Append(XY(RawX, RawY), XY(GridX, GridY));
Model.Fit(Perspective);

// Undistort the source image, without bilinear interpolation
Model.Register(Source, Calibrated, false);
```

## Typical running times

Affine calibration (nearest neighbor)	0.42
Affine calibration (bilinear)	1.2
Perspective calibration (bilinear)	4.6
Distortion calibration (bilinear)	4.8
Full calibration (bilinear)	5.9



# Edge Gauging

The edges of objects usually correspond to sharp transitions in the intensities in the image.



Gauging enables very accurate measurements of edges in order to determine positions, distances, sizes, angles...

This function set is specialized in profile analysis (peak detection) and fitting of straight / circular lines. It performs robust selection of the relevant edge points.

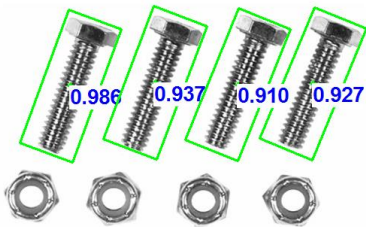
## Automatic edge detection, outlier rejection, point model fitting, robust sub-pixel measurement

```
// Place a straight line measurement gauge  
Gauge.Beg= XY(139, 88); Gauge.End= XY(343, 87);  
  
// Fit the line and get the angle  
Gauge.Detect(Source, BestStep);  
double Angle= Gauge.FittedAngle;
```

## Typical running times

Detect edges along a profile (100 pixels)	<0.001
Fit a straight edge (21 profiles)	0.0060
Fit a circular edge (63 profiles)	0.039

# Pattern Matching



Pattern matching, also called template matching, is an excellent approach to locate known objects in an image. It works by providing a sample image of the item to be located and then searching for similar shapes in the target image.

The pattern matching tool of mViz supports translation, rotation and scaling and tolerates linear changes in light intensity. It is based on the well-known normalized correlation

score.

## Model training, sub-pixel location, rotation & scaling

```
// Train the pattern from a region of interest
Source.Window(41, 170, 219, 194);
Template.Train(Source);

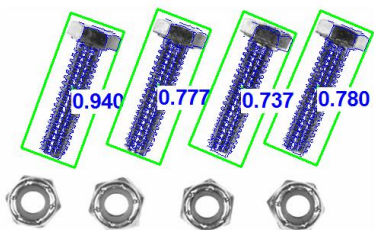
// NGC search in the whole image with a rotation tolerance
Template.MinAngle= -30; Template.MaxAngle= 30;
Template.Find(Source);
```

## Typical running times

Search (160x160 template)	0.17
Search $\pm 30^\circ$ rotation (160x160)	1.6
Search $\pm 30^\circ$ rotation, $\pm 10\%$ scaling (160x160)	5.7

# Geometric Matching

By contrast to standard pattern matching, the geometric finder uses edge information rather than area information. This provides better robustness to occlusion, clutter, blur, non-linear intensity changes.



The geometric and standard pattern matching tools share a common API and can easily be traded for one another.

## Model training, sub-pixel location, full rotation & scaling

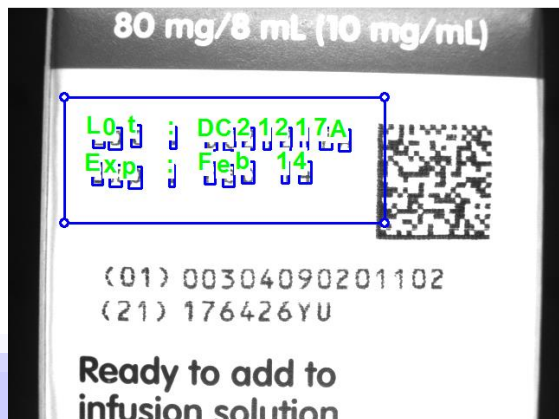
```
// Train the pattern from a region of interest
Source.Window(41, 170, 219, 194);
Template.Train(Source);

// Geometric search in the whole image with a rotation tolerance
Template.MinAngle= -30; Template.MaxAngle= 30;
Template.Find(Source, true);
```

## Typical running times

Search (160x160 template)	0.25
Search $\pm 30^\circ$ rotation (160x160)	0.91
Search $\pm 30^\circ$ rotation, $\pm 10\%$ scaling (160x160)	3.4

# Character Reading



Many industrial applications require part identification. One of the ways to distinguish object is by printing a human-readable serial number on the surface or on a stickled label.

The character reading set serves two purposes: either to be able to read the content of the marking

(OCR), or to verify that a given content has been properly printed (OCV). mViz uses user pre-trained fonts for the most accurate recognition. A priori information on the text layout can be used to further increase reliability.

## Font training, deskewing, auto-segmentation, printed character recognition & verification

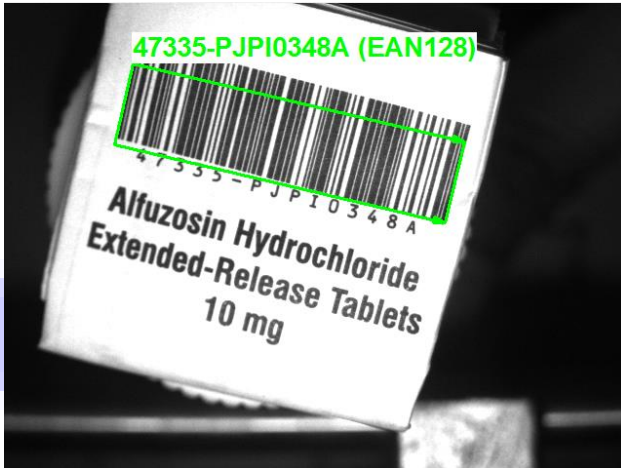
```
// Load the pre-recorded font from a file
OCR.Read("Fonts\\OCR-B.fnt");

// Perform the recognition from a region of interest
Source.Window(41, 170, 219, 194);
OCR.CharsRead(Source);
char* String= OCR.AsciString;
```

## Typical running times

Read characters (among 54 in font)	0.70
Read characters (adaptive threshold)	1.9

# Bar Code Reading & Verification



A well-known alternative to printed characters are the barcodes. They come in numerous flavors and allow storing a number of digits and/or alphabetic characters. Depending on the type, the payload ranges from a few digits to a few tens of

characters.

Verification is made according to ISO standards.

## **Automatic code location, recognition and decoding, scale and rotation invariant, EAN/UPC, GS1 Databar, Code 39, (color) Pharmacode... symbologies**

```
// Find the Pharmacode from the whole image
Barcode.DetectPharmacode= true;
bool Success= Barcode.Decode(Source);

// Read the decided string
char* String= Barcode.AsciiString;
```

## **Typical running times**

Read a EAN128 code	0.67
Read a EAN128 code (6 directions)	2.3

# Dot Code Reading & Verification



Another alternative to printed characters appeared more recently and is known as dot codes or 2D codes. They encode more information in the same space by allowing variations in two dimensions rather than one. To ensure data integrity, they also embed error

detection and correction mean.

Verification is made according to ISO standards.

## **Automatic code location, recognition and decoding, scale and rotation invariant, Data Matrix, Aztec Code, QR... symbologies**

```
// Find the Data Matrix code from the whole image
bool Success= DataMatrix.Decode(Source);

// Read the decided string
char* String= DataMatrix.AsciiString;
```

### **Typical running times**

Read a Data Matrix code	4.6
Read a QR code	2.5

# API Overview

```
// Image, 2D Pixel Array
class Image
{
    void Read; (x2)
    void Write; (x2)

    // Sizing
    void Set; (x2)

    // Window of Interest
    void Window; (x3)
    ReadOnly<int> X, Y, Width, Height;
    bool HasWindow;

    // Pixel Data
    bool PixelGray1;
    void PixelGray1;
    GrayPixel& PixelGray;
    Gray16Pixel& PixelGray16;
    Gray32Pixel& PixelGray32;
    Gradient& PixelGrad;
    RgbPixel& PixelRgb;
    RgbaPixel& PixelRgba;
    ReadOnly<byte> Buffer;
    ReadOnly<int> Pitch;

    // Graphics
    void Draw const; (x3)
    void DrawWindow const;
    enum DraggingHandles HitHandle;
    ReadOnly<enum DraggingHandles> Handle;
    void Drag;
    XY Center const;

    // Storage
    ReadOnly<enum PixelTypes> Type;
    ReadOnly<int> StoreWidth, StoreHeight;
};

// Image Sequence Reading
class Source
{
    // Access
    void Append; (x2)
    void OpenSequenceFile; (x2)
    void OpenCamera;
    void Close;
    ReadOnly<SourceMode> Mode;

    // Frame Reading
    ReadOnly<int> FrameCount;
    ReadOnly<double> FrameRate;
    int FrameIndex;
    bool Grab;
    double Exposure;
    double Gain;

    // Frame Transfer
    void CopyToClipboard;
    void PasteFromClipboard;

    // Data Transfer
    void CopyToClipboard;

    // I/O
    bool Input;
    void Output;
};

// Profile, 1D Data Container
class Profile
{
    // Sizing
    void Set; (x2)
    void Clear;
    void Append; (x3)

    // Sample
    void RowGet;
    void RowAdd;
    void RowVariance;
    void RowMaximize;
    void RowMinimize;
    void RowRange;
    void ColumnGet;
    void ColumnAdd;
    void ColumnVariance;
    void ColumnMaximize;
    void ColumnMinimize;
    void ColumnRange;
    void LineGet;
    void LineAdd;
    void LineVariance;
    void LineMaximize;
    void LineMinimize;
    void LineRange;
    void CurveGet;

    // Interval of Interest
    void Window; (x2)
    ReadOnly<int> I, Length;
    bool HasWindow;

    // Statistics
    int Quantile;
    int Median;
    double Mean;
    double TrimmedMean;
    void Range;
    double Variance;

    // Filter
    void GaussLowPass;
    void BoxLowPass;
    void GaussHighPass;
    void BoxHighPass;
    void Integrate;
    void Derive;

    // Analyze
    void EdgeDetect;

    // Define LUT
    void GainOffset;
    void Threshold;
    void BiThreshold;
    void Quantize;
    void Logarithm;
    void Gamma;
    void PseudoColor;

    // Sample Data
    Gray32Pixel& Sample; (x2)
    ReadOnly<int*> Buffer;
    ReadOnly<int> Pitch;

    // Graphics
    void Plot;
    char* ToAsciiString;

    // Storage
    ReadOnly<enum PixelTypes> Type;
```

```

ReadOnly<int> Scale;
ReadOnly<int> StoreLength;

// Graphics
void SetWindow;
int X, Y, Width, Height;
XY Beg, End;
int XCol, YRow, Breadth;
bool IsSum;
enum DraggingHandles Direction;
void Draw;
enum DraggingHandles HitHandle;
ReadOnly<enum DraggingHandles> Handle;
void Drag;
};

// Path, Arc/Line Curve Container
class ArcPath
{
    // Path Data
    void Clear;
    void Append; (x2)
    ArcSite& Vertex;

    // Graphics
    void Draw const;
    enum DraggingHandles HitHandle;
    int ClosestHandle;
    ReadOnly<enum DraggingHandles> Handle;
    ReadOnly<int> HitIndex;
    void Drag;

    // Storage
    ReadOnly<int> Length;
    bool Closed;
};

// Path, Curve Container
class Path
{
    // Path Data
    void Clear;
    void Append; (x2)
    void PolygonToRaster;
    int ClosestTo const;
    Site& Vertex;

    // Contouring
    void Follow;

    // Processing
    void Smooth const;

    // Analysis
    void Box const;
    double Perimeter const;
    int Area const;
    void Center const;
    void Ellipse const;
    bool FilledCenter const;
    bool FilledEllipse const;
    void ConvexHull;
    int Mass const;
    void Centroid const;
    void GrayHistogram const;
    void Vectorize; (x2)

    // Graphics
    void Draw const;
    enum DraggingHandles HitHandle;
    int ClosestHandle;
    ReadOnly<enum DraggingHandles> Handle;
    ReadOnly<int> HitIndex;
    bool PointInPath;

    bool Overlapping;
    void Drag;
    void Insert;
    void Delete;
    char* ToAsciiString;

    // Storage
    ReadOnly<int> Length;
    Site* Buffer;
    bool Closed;
};

// Poly, Polygonal Chain or Polygon
Container
class Poly
{
    // Poly Data
    void Clear;
    void Append; (x2)
    int ClosestTo const;
    XY& Vertex;

    // Graphics
    void Draw const;
    int ClosestHandle;
    ReadOnly<enum DraggingHandles> Handle;
    ReadOnly<int> HitIndex;
    bool PointInPolygon;
    bool Overlapping;
    void Drag;
    void Insert;
    void Delete;

    // Storage
    ReadOnly<int> Length;
    XY* Buffer;
    bool Closed;
};

// Pixel-to-pixel Arithmetic
class Operator
{
    // Copy
    void Copy; (x4)

    // Color
    void Invert;
    void Convert;
    void DeBayer;
    void ColorsLighten;
    void ColorsSaturate;
    void ColorsHueSet;

    // Image Arithmetic
    void Add; (x2)
    void Subtract; (x3)
    void Multiply; (x2)
    void Divide; (x3)
    void Maximize; (x4)

    void Blend;

    // Compare
    void EqualTest; (x2)
    void GreaterTest; (x3)
    void GreaterEqualTest; (x3)

    // Bitwise
    void And; (x2)
    void Or; (x2)
    void Xor; (x2)

    // Binarize
    int Binarize; (x6)
};

```



```

// Correct Shading
void AddCorrect; (x2)
void MultiplyCorrect; (x2)
void Correct; (x2)

// Scale Grays
void GainOffset;
void Equalize; (x2)
void Transform;
int GrayDepth;

// Profile Arithmetic
void Copy;
void Add; (x2)
void Subtract; (x3)
void Multiply; (x2)
void Divide; (x3)
void Maximize; (x4)
};

// Linear Filtering
class Kernel
{
// Gradient Filters
void Gradient;
void GradientSum;
void GradientX;
void GradientY;
void Sobel;
void SobelSum;
void SobelX;
void SobelY;
void Prewitt;
void PrewittSum;
void PrewittX;
void PrewittY;
void CorrelatedGradient;
void BoxDeviation;

// Low-pass Filters
void Smooth;
void Smooth8;
void Binomial;
void BoxLowPass;
void GaussLowPass;
void Bilateral;

// High-pass Filters
void Sharpen4;
void Sharpen8;
void Unsharp;
void BoxHighPass;

// Storage
ReadOnly<int> Width, Height;

// General Convolution
void Set;
double& Coefficient;
enum GrayClipping Clip;
double Offset;
void Normalize;
void Convolve;
};

// Morphological Filtering
class Morpho
{
// Structel
void Set;
void Box;
void Square;
void Octagon;

void Diamond;
void Horizontal;
void Vertical;
ReadOnly<int> Width, Height, Diagonal;

// Filtering
void Dilate;
void Erode;
void Open;
void Close;
void WhiteHat;
void BlackHat;
void Gradient;
void MidRange;
void PseudoMedian;
void Contrast;
void Median;
void Rank;
int BinaryThin;
int BinaryThick;

// Segmentation
void Isophotes;
int Watershed;
int Watershed;
void AverageRegions;
void Distance;
void KillConnect;
};

// Image Statistics
class Histogram:
{
// Frequency Data
int& Bin;
void ClearBins;

// Compute
void Compute; (x3)
void Cumulate;

// Statistics
int Count; (x2)
double Mean; (x4)
int Median; (x2)
int Mode; (x2)
double TrimmedMean; (x2)
void Range; (x4)
double Variance; (x4)
double MedianAbsoluteDeviation; (x2)
int Quantile; (x2)
double Skewness; (x2)
double Kurtosis; (x2)
double Entropy; (x2)
int BestThreshold; (x2)
int EntropyThreshold; (x2)
int IsodataThreshold; (x2)

// Filter
void GaussLowPass;
void BoxLowPass;
void GaussHighPass;
void BoxHighPass;
void Integrate;
void Derive;

// Image Quality
double Noise;
void Saturation;
void ShowSaturation;

// Compare Images
double SumAbsoluteDifferences;
double SumSquaredDifferences;
double NormalizedCorrelation;

```

```

void Normalize;
void ShowDifferences;

// Graphics
void Plot;
char* ToAsciiString;
};

// Geometric Transformations
class Geometry
{
    void Read; (x2)
    void Write; (x2)

    // Simple Transforms
    void Flip;
    void Mirror;
    void Turn;
    void DownSample;
    void UpSample;

    // Linear Transforms
    void Rotate; (x2)
    void Skew;
    void Affine;

    // Polar Transform
    void Unwind;

    // Tabulated Transform
    void Transform;

    // Simple Calibration
    void SimpleCalibration; (x2)
    ReadOnly<double> ScaleToPixel;
    ReadOnly<double> ScaleFromPixel;

    // Calibration
    void CalibrateTarget;
    XY Calibrate const;
    XY Decalibrate const;

    // Image Correction
    void Undistort;

    // Registration
    ReadOnly<RegistrationModels> Model;
    void ModelFit;
    XY Register const;
    void Register;
    XY ReverseRegister const;
    void ReverseRegister;

    // Anchor Points
    void Clear;
    void Append;
    ReadOnly<int> NumAnchors;
    XY& Src;
    XY& Dst;

    // Graphics
    void DrawAnchors;
    enum DraggingHandles HitHandle;
    ReadOnly<enum DraggingHandles> Handle;
    ReadOnly<int> AnchorIndex;
    void Drag; (x2)
};

// Pyramid, fine-to-coarse representation
class Pyramid
{
    // Building
    void Build;

    // Access by Level
    ReadOnly<int> NumLevels;
    const Image& Bitmap;
    Image& Bitmap;

    // Tile Statistics
    void TileMean;
    void TileMidRange;
    void TileDeviation;
    void TileRange;

    // Graphics
    void Draw; (x2)
};

class FeatureTable
{
    // Sizing
    void ClearFeatures;
    void ClearSamples;
    int AppendIntFeature;
    int AppendFloatFeature;
    int AppendSample;
    ReadOnly<int> NumFeatures;
    ReadOnly<int> NumSamples;

    // Sample Data
    enum PixelTypes& Type;
    int& IntFeature;
    float& FloatFeature;

    // Selection by Features
    bool IsSelected;
    void SampleSelect;
    void SamplesSelect; (x2)
    ReadOnly<int> NumSelectedSamples;

    // Sorting by Features
    void SamplesSort;
};

class Region
{
    ReadOnly<int> Width, Height;

    // Sizing
    void Clear;
    void Append;

    // Generate
    int Binarize; (x3)

    // Add a Region
    void BlobAppend;
    void PathAppend;
    void QuadAppend;

    // Region Arithmetic
    void Unite;
    void Intersect;
    void Subtract;

    // Region Reshaping
    void Convexify;

    // Filling
    void Fill const; (x3)
    void SeedFill;

    // Outlining
    void Follow;

    // Graphics
    ReadOnly<int> NumRuns;
};

```

```

void Run;
void Draw;
};

// Blob Analyser
class Blobs:
{
    // Segment Image
    ReadOnly<int> NumBlobs;
    int Segment; (x5)

    // Segment in Mask
    int Segment; (x5)

    // Binary Blob Features
    int Area const;
    int ConvexArea const;
    void Box const;
    void TiltedBox const;
    bool Border const;
    void Center const;
    void Ellipse const;
    void FeretBox const;
    void DiametralBox const;
    int Perimeter;
    enum RangeMode WindowBlob;

    // Weighted Blob Features
    double Average const;
    double Deviation const;
    int Median const;
    int Minimum const;
    int Maximum const;
    int Mass const;
    void Centroid const;
    void Ellipsoid const;

    // Non-scalar Features
    void PixelHistogram const;
    void Outline const;
    void ConvexHull const;

    // Evaluate Features
    void Evaluate;

    // Select by Feature
    void ByFeatureSelect; (x2)
    void BorderSelect;
    void BoxInWindowSelect;
    void BlobInWindowSelect;
    ReadOnly<int> NumSelectedBlobs;

    // Sort by Feature
    void ByFeatureSort;

    // Transform Blobs
    void Convexify;

    // Features Data
    enum Features AppendUserFeature;
    enum PixelTypes Type;
    const wchar_t* UnicodeName;
    bool IsComputed;
    int& IntFeature;
    float& FloatFeature;
    int StraightIndex;
    void ClearBlobs;

    // Features Distribution
    double FeatureAverage;
    double FeatureDeviation;
    double FeatureMinimum;
    double FeatureMaximum;
    double FeatureQuantile;
    double FeatureCovariance;

    // Graphics
    void DrawBlob;
    void DrawBox;
    void DrawTiltedBox;
    void DrawCenter;
    void DrawEllipse;
    void DrawHull;
    void DrawCentroid;
    void DrawEllipsoid;
    void DrawFeretBox;
    void DrawDiametralBox;
    void DrawOutline;
    int HitBlob;
    int HitBlobIndex;
    void ToImage;
    enum Features WhichFeature;
};

// Point Edge Detector
class EdgePoint
{
    // Location
    void Locate;
    XY From, To;
    ReadOnly<int> XMin, YMin, XMax, YMax;

    // Measurement
    int Breadth;
    int Smoothing;
    int Noise;
    int MinimumSlope;
    int MinimumStep;
    int MinimumSpan;
    enum Polarities Polarity;
    bool SplitPeaks;
    bool Detect; (x2)

    // Detection Results
    int NumPoints;
    Edge FittedEdge;
    XY FittedPoint;

    // Graphics
    void SetWindow; (x2)
    void Draw; (x2)
    enum DraggingHandles HitHandle;
    ReadOnly<enum DraggingHandles> Handle;
    int HandleOctant;
    void Drag;
};

// Linear Edge Detector
class EdgeLine:
{
    // Location
    void Locate;
    XY Beg, End;
    double Width;

    // Measurement
    double Density;
    double Clearance;
    bool Oblique;
    bool Detect; (x2)
    ReadOnly<int> NumSamples;
    int SampleIndex;
    XY FittedPoint;

    // Fit
    XY FittedBeg;
    XY FittedEnd;
    ReadOnly<double> FittedAngle;
    ReadOnly<int> NumDetectedPoints;
};

```

```

ReadOnly<double> Deviation;
ReadOnly<int> NumFittedPoints;
ReadOnly<double> RobustDeviation;
void Realign;

// Graphics
void Draw; (x2)
enum DraggingHandles HitHandle;
enum DraggingHandles Handle;
ReadOnly<enum DraggingHandles> Handle;
int HandleOctant;
void Drag;
};

// Curved Edge Detector
class EdgeArc:
{
// Location
void Locate; (x2)
XY Beg, Mid, End;
double Width;
XY Center;
ReadOnly<double> Radius;

// Measurement
double Density;
double Clearance;
bool Oblique;
bool Circle;
bool Detect; (x2)
ReadOnly<int> NumSamples;
int SampleIndex;
XY FittedPoint;

// Fit
XY FittedCenter;
ReadOnly<double> FittedRadius;
ReadOnly<int> NumDetectedPoints;
ReadOnly<double> Deviation;
ReadOnly<int> NumFittedPoints;
ReadOnly<double> RobustDeviation;
void Realign;

// Graphics
void Draw; (x2)
enum DraggingHandles HitHandle;
enum DraggingHandles Handle;
int HandleOctant;
void Drag;
};

// Rectangular Edge Detector
class EdgeRectangle:
{
// Location
void Locate;
XY Size;
ReadOnly<double> Angle;
double Width;

// Measurement
double Density;
double Clearance;
bool Oblique;
bool Detect; (x2)
ReadOnly<int> NumSamples;
int SampleIndex;
XY FittedPoint;

// Fit
XY FittedMiddle;
XY FittedSize;
ReadOnly<double> FittedAngle;
ReadOnly<double> Deviation;
ReadOnly<double> RobustDeviation;

int NumDetectedPoints;
int NumFittedPoints;
void Realign;

// Graphics
void Draw; (x2)
enum DraggingHandles HitHandle;
enum DraggingHandles Handle;
int HandleOctant;
void Drag; (x2)
};

// Segmented Edge Map Container
class EdgeMap
{
// Raster Map
void GradientVector;
int GradientMaxima;
int StrongEdges;
void StepEdges;

// Vector Map
int Follow;
void IsoCurves;
int Vectorize;
ReadOnly<int> NumPaths;

// Graphics
void Draw const;
int ClosestTo const;
void Clear;

// Vertex/Arc/Path Data
int NumVertexes;
Site & Vertex;
int NumArcs;
ArcSite & Arc;
};

// General Purpose Classifier
class Classifier
{
void Read; (x2)
void Write; (x2)

// Features Management
void AppendFeature;
ReadOnly<int> NumFeatures;
PixelTypes Type;
wchar_t* Name;
void ClearFeatures;

// Samples Management
void AppendSample;
ReadOnly<int> NumSamples;
void ClearSamples;

// Value Access
bool & Bool;
GrayPixel & Gray;
Gray16Pixel & Gray16;
RgbPixel & Rgb;
int & Int;
float & Float;
double & Dble;

// Training
void TrainClusters;
void TrainNeighbors;
double ClusterCenter;
double ClusterCovariance;

// Matching
Metrics Mode;
int SlowNearestNeighbor; (x3)
};

```

```

int NearestNeighbor; (x3)
int NearestCenter;
int NearestCluster;

// Performance Assessment
double InterDistance;
int CrossValidateClusters;
int CrossValidateNeighbors;

// Text Input/Output
char* ToAsciiString;
wchar_t* ToUnicodeString;
void FromAsciiString;
void FromUnicodeString;

// Graphics
void PlotY; (x2)
void PlotXY; (x2)

// Clipboard
void CopyToClipboard;
void PasteFromClipboard;
};

// Template Location
class Locator
{
void Read; (x2)
void Write; (x2)

// Training
void Train; (x3)
ReadOnly<int> MaxAreaDepths;
ReadOnly<int> MaxEdgesDepths;
const Image& TemplateImage;
const Image& MaskImage;
void TrainStringTemplate;

// Searching
int MaxLocations;
double MinScore;
double MinAngle, MaxAngle; (x2)
bool Edges;
MatchingScores ScoreType;
int Symmetry;
int MaxDepths, StopDepth;
void Find;
ReadOnly<int> NumLocations;

// Found Locations
Pose& Location;
XY Register;

// Normalization
void Register;

// Graphics
void Draw;
void DrawModel;
enum DraggingHandles HitHandle;
ReadOnly<enum DraggingHandles> Handle;
ReadOnly<int> LocationIndex;
};

// Shape Recognition
class ShapeMatcher
{
void Read; (x2)
void Write; (x2)

// Training
void Train; (x2)
void TrainBlob;
ReadOnly<int> NumTrained;

void Clear;

// Recognition
double MaxSimilarity;
double MinAngle, MaxAngle; (x2)
int Recognize;
void RecognizeBlobs;
ReadOnly<Pose> Location;
ReadOnly<int> Class;

// Registration
void Register;

// Graphics
void DrawTrained;
void DrawMatched;
int HitTrained;
ReadOnly<int> HitIndex;
void DeleteHit;
};

// Optical Character CharReader
class CharReader:
{
void Read; (x2)
void Write; (x2)

// Character Segmentation
void CharsSegment;
int Quadrant;
bool LocalThreshold;
int Threshold;
bool WhiteOnBlack;
int CharWidth, CharHeight;
bool GuessSize;
bool KeepRatio;
bool RepairBroken;
int Dotted;
int Masking;
bool AlignChars;
bool EvenlySpaceChars;
int MaxTouching;
int SplitRows;
ReadWrite<CharReader, bool>
VariableWidth;
double GapFraction;
double DotFraction;
ReadOnly<int> NumChars;
wchar_t Separator;
void PresetCharBoxes;
void PresetCharBox;

// Training
FontCharImage& FontChar;
int FontWidth, FontHeight;
void FontCharInsert; (x2)
void FontStringInsert; (x2)
void FontCharDelete;
ReadOnly<int> NumFontChars;
void LayoutTrain;
void LayoutReset;
ReadOnly<int> NumLayoutChars;
ReadOnly<int> NumLayoutStrings;

// Reading
void CharsDecode;
void CharsRead;
double MinimumScore;
bool UseSafeScores;
bool DiscardBadScores;
void Filter; (x2)
int JitterX, JitterY;
ReadOnly<char*> AsciiFilter;
ReadOnly<wchar_t*> UnicodeFilter;
bool InsertSpaces;
};

```

```

bool InsertNewlines;
double MinAngle, MaxAngle;
ReadOnly<char*> AsciiString;
ReadOnly<wchar_t*> UnicodeString;
ReadOnly<double> AverageScore;
double CharScore;
int MatchingFontChar;
int CharTrueWidth;
LayoutModes LayoutMode;

// Graphics
void Draw;
bool GetBoxes;
void DrawLayout;
bool GetLayoutBoxes;
void DrawFontChars;
enum DraggingHandles HitChar;
int CharIndex;
enum DraggingHandles HitFontChar;
ReadOnly<enum DraggingHandles> Handle;
int FontCharIndex;
void CharThumbnail;
bool New;
};

// Barcode Reading
class Code1DReader
{
    // Decoding
    bool Decode;
    bool DetectEAN128;
    bool DetectEAN13;
    bool DetectEAN8;
    bool DetectUPCA;
    bool DetectUPCE;
    bool Detect2of5;
    bool Detect39;
    bool Detect39Extended;
    bool Detect93;
    bool DetectCodabar;
    bool DetectDatabar;
    bool DetectDatabarLimited;
    bool DetectPharmacode;
    int NumDirections;
    int Density;
    int Noise;
    int Step;
    bool Reverse;
    bool CheckQuietZone;
    int MinimumBars;

    // Decoding Results
    ReadOnly<int> NumDecoded;
    void Choose;
    void Choose;
    ReadOnly<char*> AsciiString;
    ReadOnly<wchar_t*> UnicodeString;
    ReadOnly<enum Symbologies> Symbology;
    ReadOnly<bool> ValidCRC;
    ReadOnly<bool> ReadFNC1;
    ReadOnly<bool> ReaderProgramming;
    ReadOnly<bool> Append;

    // Quality Indicators
    bool CheckQuality;
    ReadOnly<char> AsciiGrade;
    ReadOnly<wchar_t> UnicodeGrade;
    ReadOnly<char*> AsciiProfileGrades;
    ReadOnly<wchar_t*> UnicodeProfileGrades;

    // Color Training
    void ColorTrain;
    int RgbTolerance;
    ReadOnly<bool> ValidColors;
    ReadOnly<int> NumColorBars;

    // Graphics
    void Draw;
    ReadOnly<int> X, Y, Width, Height;
    ReadOnly<double> Angle;
    void ColorReset;
    RgbPixel BarColor;
};

// Dot Code Reading
class Code2DReader
{
    // Properties
    enum CodePolarities EnabledPolarities;
    enum CodeViews EnabledViews;
    Sizes SizeFrom;
    Sizes SizeTo;
    ReadOnly<Sizes> Size;
    bool DetectDataMatrix;
    bool DetectQR;
    bool SymbologyIdentifier;
    bool AdaptiveThreshold;
    int Threshold;
    int StartLevel;
    bool AccurateEdges;

    // Decoding
    bool Decode;
    ReadOnly<int> NumDecoded;
    void Choose;

    ReadOnly<char*> AsciiString;
    ReadOnly<wchar_t*> UnicodeString;
    ReadOnly<Symbologies> Symbology;
    ReadOnly<int> StringLength;
    ReadOnly<enum CodePolarities> Polarity;
    ReadOnly<enum CodeViews> View;
    ReadOnly<float> CellSize;
    ReadOnly<bool> ReadFNC1;
    ReadOnly<bool> ReaderProgramming;
    ReadOnly<int> SymbolPosition;
    ReadOnly<int> SymbolTotal;
    ReadOnly<int> SymbolFileID1;
    ReadOnly<int> SymbolFileID2;

    // Quality Indicators
    bool CheckQuality;
    ReadOnly<char*> AsciiGrades;
    ReadOnly<wchar_t*> UnicodeGrades;
    ReadOnly<float> SymbolContrast;
    ReadOnly<float> ContrastUniformity;
    ReadOnly<float> AxialNonUniformity;
    ReadOnly<float> GridNonUniformity;
    ReadOnly<float> UnusedError;
    ReadOnly<float> PrintGrowthHorizontal;
    ReadOnly<float> PrintGrowthVertical;

    // Graphics
    void Draw;
    ReadOnly<int> X, Y, Width, Height;
    ReadOnly<double> Angle;
    XY Corner;
    float PrintGrowthNominal;
    float PrintGrowthMinimum;
    float PrintGrowthMaximum;
};

// Graphics
class Graph
{
    // Display
    void*& DrawTo;
    double Zoom;
    void Offset;
};

```

```

void Offset;
double Resolution;
int TextSize;
AlphaModes AlphaRendering;
void* DrawToDefault;
void DrawToImage;
void ReleaseImage;

// Graphics Attributes
enum PredefinedColors DotColor,
LineColor, FillColor, TextColor;
void TextOrigin;
ReadOnly<char*> TextFont;
void SetTextFont;
bool AutoColor;
bool SmoothZoom;

// Graphics Primitives
void Dot; (x3)
void Cross; (x3)
void Line; (x3)
void Box; (x2)
void Circle; (x2)
void Arc; (x2)
void Ellipse; (x2)
void Text; (x6)
void Handle; (x3)
void Distance; (x6)
void Angle; (x2)
const RgbPixel Color;
void EraseWindow;
void ZoomToWindow;
void WaitClose;
};

// Global Parameters
class Status
{
// Error Processing
enum ErrorCodes& Error;
char* AsciiError;

// Miscellaneous
int JpegQuality;
};

wchar_t* UnicodeError;
char* AsciiFunction;
wchar_t* UnicodeFunction;

void IgnoreError;
void ThrowError;
void ShowError;
void PrintError;
void AssertError;
void ErrorBehavior;
Callback ErrorCallback;
void* ErrorContext;

bool Raise;

// Angle Unit
double AngleUnit;
const double FromDegrees;
const double ToDegrees;
const double FromRadians;
const double ToRadians;

// Timing
void StartWatch;
int ReadWatch;
void Wait;

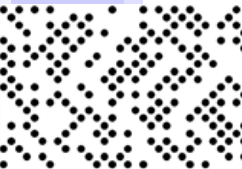
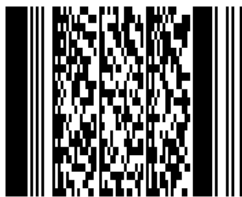
// Keypress
int Key;

// Versioning
const wchar_t* UnicodeVersion;
const char* AsciiVersion;

// Multithreading
void AttachThread;
void ReleaseThread;

// Data Storage
bool Shallow;
bool License;

```



[www.visionforvision.eu](http://www.visionforvision.eu)