CV Workbench

A platform for computer vision experiments

# Overview

The CV-Workbench is a framework to perform reproducible image processing experiments using scripts that allow comparison of image processing results over multiple images. Users define scripts that are experiments with multiple image processing operator steps that process input data and store outputs. Scripts facilitate reproducing experiments with different data. The advantages of the system are:

* The system is simple. It provides an easy way to set up the inputs, parameters, and outputs to image processing operations. Scripts can be run against different data and be easily changed to allow experimenting with operations.
* It is easily extensible using C++. There is a regular code format for programming image processing operations, and they can be added without necessarily changing the base system.
* The system is open source and readily available for educational settings or personal use.

An application may be setting up labs for an image processing course. Scripts can be provided to demonstrate different techniques, then students can modify the scripts and turn them in for assignments. This is not a full image processing system, so it may be more appropriate for robotics applications where it can be extended with specific image processing operators that are needed but not all image processing methods are required.

An experiment management module guides runs the image processing scripts on Linux using the command line. The module handles reading and writing images and data and applying operators with the specified parameters. Experiment results are logged, and the scripts can be re-run in a reproducible way. A large set of test scripts are available as samples.

# Method

Scripts are written as [JSON](https://www.json.org/json-en.html) text files whose format clearly defines the operators and parameters to use. Operators use and produce JPEG images or images in an internal format. JPEG images have unsigned values and image operators may produce signed outputs. These require an integer storage type that holds a larger range of pixel values, though they cannot be viewed as directly as JPEG images. Internal format images can be converted to JPEG images to visualize them. The system produces other types of data such as histograms and Hough accumulators. These can be visualized using tools such as Microsoft Excel or gnuplot. The system is designed to run operations, observe the output data, change the parameters as desired, and end up with scripts that can be run against large image sets. The system can have operations added as needed by experiments.

The following sections describe:

* The script format, how to specify operators used, and where the data is stored on the filesystem.
* The image processing operations available, with their inputs, the parameters, the outputs, and function. The Internet and textbooks can provide more details on the image processing functions.
* The storage formats for each type of data, and log output produced by the operations.

# Operators

Operators can be separated into image processing function domains. Sample operator domains follow.

| Domain | Description |
| --- | --- |
| Binary operators | Transformations of binary images |
| Correlation | Matching features or patterns of image values to images. |
| Feature detection | Image- and region-level operators to detect and characterize arbitrary shaped features in images. |
| Filters | Pixel-level operators that transform pixel values or make computations on pixel neighborhoods that are stored in pixels. |
| Geometric operations | Projecting images based on different viewing geometries. |
| Histogram operators | Creating, transforming, or extracting. parameters of histograms. |
| Transforms | Various image-wide changes. |

Within each domain, there are operator classes for specific groups of operators. Sample operator domain classes follow.

| Domain | Operator classes |
| --- | --- |
| Filters | Edge detectors  Convolutions  Thresholding |
| Correlation | Feature matching  Pattern matching |
| Feature detection | Region growing and classification  Hough line and generalized shape detection |
| Binary operators | Morphological operations |
| Histogram operators | Histogramming image pixel values  Histogramming Hough accumulator counts  Histgramming feature values  Histogram equalization  Threshold selection |
| Image operators | Image transforms  Image coding  Image compression |
| Geometric operators | Resampling  Warping  Pyramids |

Specific operators supported in the current release are described below.

# Experiments

Each script runs an experiment that consists of a series of experiments steps. Each step consists of:

* One or more input images or other types of data
* An image processing operator
* Parameters controlling the operator function
* One or more output image or other types of data

These scripts may be run against a set of sample images to test the effectiveness of the processing. Data produced by experiments are saved. The processing results for operators are logged for later reviews.

# Data Type Formats

Each data type is stored in its own format. The data types supported in the current release are the following. Much of the data is stored in signed integer format specific to this program, because the data values involved lie outside 0 to 255. Some operators will produce an unsigned 8-bit value on output that can be visualized as a JPEG image. Other operators can produce tab-delimited text output viewable in Microsoft Excel, for example. These are specific to the operator involved.

| Data type | Description |
| --- | --- |
| Image | Two-dimensional matrices of grayscale or color values. |
| Histogram | Frequency distributions for sets of values, such as pixel values in images. |
| Hough accumulator data | Hough transform (rho, theta) counts for detection of lines in edge images. |
| Hough peak lines | Lines in polar coordinates selected from Hough accumulators. |

## Image data type

Images are a matrix of pixel values consisting of one or more components. Grayscale images have one component value per pixel; color images have three RGB component values. Each component pixel value is stored as one of three binary formats. OpenCV “depth” codes are used to describe the binary formats.

| Value depth | Value range | Description |
| --- | --- | --- |
| CV\_8U | 0..255 | unsigned byte values |
| CV\_32S | INT\_MIN..INT\_MAX | signed 32-bit integer values |
| CV\_32F | FLT\_MIN..FLT\_MAX | signed 32-bit floating values |

CV\_8U images are in JPEG format. The JPEG image file format is defined by the [JPEG standard](https://jpeg.org/). Images generated by operators generally use CV\_8U, CV\_32S, or CV\_32F as input and generate CV\_32S or CV\_32F on output, except as noted in the operator descriptions. CV\_32S and CV\_32F images may be used interchangeably without having to specify the depth in operators, though CV\_32F images are output if any input image is CV\_32F. The "transform-intensity-map" operator can map CV\_32S and CV\_32F images to CV\_8U JPEG for display.

The binary image data files are defined by the workbench and have the following format. Color RGB pixels are stored as three consecutive pixel values, one for each component. The three consecutive pixel values represent the red, green, and blue components.

| Field | Type | Description |
| --- | --- | --- |
| nrows | 32-bit integer | The number of image rows |
| ncols | 32-bit integer | The number of image columns |
| components | 32-bit integer | The number of image components. Grayscale images have 1 component; color images have 3 components. |
| depth | 32-bit integer | Pixel value data type.   * CV\_8U has value 0 * CV\_32S has value 1 * CV\_32F has value 2 |
| image data | nrows \* ncols \* components binary pixel values | Each value is CV\_8U, CV\_32S, or CV\_32F as specified by the depth. Color pixels are consecutive 32-bit integer red, green, and blue values. |

## Histogram data type format

Histograms count the number of different values for any data type containing a range of values, such as images or Hough accumulators. The histogram bins correspond to data values ranging from a lower to an upper value. The number of histogram bins or nbins, the lower value or lower\_value, and the upper value or upper\_value determine into which bin a value is counted. Values below the lower value are counted in the first bin, and values above the upper value are counted in the last bin. The bin corresponding to a value is given by this:

if value < lower\_value

bin = 0

else if value > upper\_value

bin = nbins – 1

else

bin = round((nbins – 1) \* (value – lower\_value) / (upper\_value – lower\_value))

The histogram data type format indicates how histogram data is stored. There are both binary and text stored histogram formats. Binary histogram files are used only within this software. Text histogram files may be plotted with gnuplot or imported into Microsoft Excel for review.

### Binary histogram file format

Binary histogram files contain the statistics for a value distribution for the input values, such as an image or Hough accumulator, and statistics for the histogram, such as the minimum, maximum, and total counts for all bins. The binary histogram data files are defined by the workbench, and have the following format:

| Field | Type | Description |
| --- | --- | --- |
| nbins | 32-bit integer | The number of histogram bins |
| lower\_value | 32-bit float | The lower pixel value accumulated in the first histogram bin |
| upper\_value | 32-bit float | The upper pixel value accumulated in the last histogram bin |
| histogram data | nbins \* 32-bit integers | The histogram bin data |

### Text histogram file format

Histogram text files are stored as input to display with gnuplot. Two files are created. A data file *output-filename*.txt is generated with each histogram bin stored as a record with the format:

*bin-value bin-count*

where the *bin-value* is the value that is accumulated in a given bin. It has this formula, where bin no. ranges from 0 to nbins-1:

*bin-value* = lower\_value + bin no. \* (upper\_value - lower\_value) / (nbins – 1)

A gnuplot script file *output-filename.*gp is generated to display the histogram as:

set style line 1 \

linecolor rgb '#0060ad' \

linetype 1 linewidth 2 \

pointtype 7 pointsize 0

plot '././image\_grayscale\_edge\_histogram.txt' with linespoints linestyle 1

pause -1 "Hit any key to continue"

The script is run with this command, assuming [gnuplot](https://sourceforge.net/projects/gnuplot/files/gnuplot/) is installed:

$ gnuplot *output-filename.*gp

The text data file can also be imported into Microsoft Excel to review.

## Hough accumulator data type format

The Hough accumulator data type contains counts from the accumulated (rho, theta) polar lines calculated for each image point by the Hough algorithm. Lines are computed for angles theta between 0 and 180 degrees, where theta is incremented by a fixed number of degrees, theta\_inc, between samples. The number of angles, nthetas, samples is:

nthetas = 180 / theta\_inc

For an image size (nrows, ncols), the polar line distance rho is computed from each (row, col) point with:

rho = (col–ncols/2+0.5) \* cos(theta) + (nrows/2–row-0.5) \* sin(theta)

The number of rho values, nrhos, for images is:

nrhos = sqrt(rows2 + columns2)

Each rho is accessed in the accumulator by bin index rho\_index that corresponds to a rho with the formulas:

rho\_index = rho + nrhos / 2

rho = rho\_index – nrhos / 2

The Hough accumulator is indexed by (rho\_index, theta\_index), where theta\_index is the index of the sampled angle used. theta\_index corresponds to the angle:

theta degrees = theta\_index \* theta\_inc

theta\_index = theta degrees / theta\_inc

Polar lines (rho, theta) are generally input and stored as (rho, theta), except as noted.

### Hough accumulator data binary file format

Binary Hough accumulator files contain the Hough accumulator counts. The binary Hough accumulator data files are defined by the workbench, and include the following fields:

| Field | Type | Description |
| --- | --- | --- |
| theta\_inc | 32-bit integer | The increment between sampled angles in degrees |
| nrows | 32-bit integer | The number of nrows in the image used to create the accumulator |
| ncols | 32-bit integer | The number of columns in the image used to create the accumulator |
| hough accumulator data | nrhos \* nthetas 32-bit integers | The Hough accumulator (rho\_index, theta\_index) count data |

### Hough accumulator data text file format

Hough accumulator text data is stored as tab-delimited text files with the following format. These files can be imported into Microsoft Excel for review.

|  |  |
| --- | --- |
| first line: | rho values from -nrhos/2 to nrhos/2 |
| following lines: | sample theta in degrees, followed by each accumulator count for the corresponding (rho, theta\_index) |

## Hough peak line data type format

Histogram peaks are polar (rho, theta) lines selected from the Hough accumulator for a given image. Each peak includes the accumulator count for that line.

### Hough peak line data binary file format

The binary Hough peak line data is defined by the workbench and has the following format. Polar line data is stored as (rho\_index, theta\_index). The corresponding rho and theta can be derived from:

rho = rho\_index – max\_rho / 2

theta degrees = theta\_index \* theta\_inc

| Field | Type | Description |
| --- | --- | --- |
| nlines | 32-bit integer | The number of polar lines (rho\_index, theta\_index, count) |
| theta\_inc | 32-bit integer | The increment between sampled angles in degrees |
| nrhos | 32-bit integer | The number of rho values sampled |
| Hough peak line data | 3 \* nlines 32-bit integers | The polar line (rho\_index, theta\_index, count) data |

### Hough peak line data text file format

Hough peak line text data is stored as tab-delimited text files with each line as a record in the following format. These files can be imported into Microsoft Excel for review.

*rho-index theta\_index count*

# Scripts

Scripts are written in JSON format to define the operators in an experiment. Experiments have this JSON format:

{

"experiment": {

"steps": [

{

*step-definition*

},

{

*step-definition*

},

a*dditional-steps*

]

}

}

Each *step-definition* has the format:

{

"id": *step-index*,

*operator-applied*,

“input-data”: [

*input-data-descriptors*

],

“output-data”: [

*output-data-descriptors*

],

“parameters”: [

*parameters-applied*

]

}

Each *operator-applied* has the format:

"operator": "*operator-name*",

Multiple data descriptors may be specified for a step, for example if multiple input data sources are combined to produce the output, or if it is desired to produce both the binary and text formats for histogram data. The input-data-descriptors and output-data-descriptors are specified with:

{

"id": *descriptor-index*,

"type": "*data-type*,

"format": "*data-format*",

"repository": "*data-storage-location*",

*additional-data-descriptor-parameters*

}

Each *parameters-supplied* has this format, where all parameters including numbers have text values:

"*parameter-name*": "*value*"

## Data Descriptors

The data descriptor determines what type of data is handled by the data descriptor, in what format is that data stored, and in which type of repository is the data found. The *data-type* and *data-format* have the following values.

| Data type | *data-type* | *data-format* |
| --- | --- | --- |
| Image | “image” | “binary” for CV\_32S or CV\_32S data “jpeg” for CV\_8U JPEG images |
| Histogram | “histogram” | “binary” for binary histogram data  “text” for output text histogram data to display using gnuplot |
| Hough accumulator data | “hough” | “binary” for binary Hough accumulator data  “text” for output tab-delimited Hough accumulator data |
| Hough peak lines | “polar-line” | “binary” for binary Hough peak line data  “text” for output tab-delimited Hough peak line data |
| Text data | “data” | “text” for multiline text |

Input and output data may be stored in different types of repositories. Not all data types and data formats are supported for any given repository. Repositories to be supported include the following. Only filesystem repositories are supported in the current release.

| Data repository | Description |
| --- | --- |
| “filesystem” | Files on the filesystem |
| “berkeley\_db” | In Berkeley DB records |
| “internet” | Streamed Internet data (input only) |

Each repository type has specific additional parameters. Only data stored on the filesystem is available in the current release.

## Filesystem data repository

The filesystem data descriptor is specified with:

"id": *descriptor-index*,

"type": "*data-type*",

"format": "*data-format*",

"repository": "filesystem",

"directory": "*directory-location*",

"filename": "*filename-base*",

"ext": "*filename-extension*"

The filesystem data repository supports all data types and data formats listed above. The file path associated with the data descriptor is “*directory*/*filename-base*.*filename-extension*". The extension may default to specific values for certain operations, such as that the default extension for image JPEG data is “jpg”.

## Script run results logging

Scripts run results are added back into the original experiment run JSON text file and saved into a log file. Results are logged into different levels of the script. The format in which results are added to the top-level of the experiment script follows. Results data added is shown in **boldface**. The log results added for the experiment are:

{

"experiment": {

**“run” : {**

**“script-path”: “*script-path*”,**

**“run-time”: “*timestamp*”,**

**“username”: “*username*”,**

**“version”: “*program-version*”**

**},**

"steps": [

{

*step-definition*

},

{

*step-definition*

},

a*dditional-steps*

]

}

}

Results added to each *step-definition* have the format:

{

"id": *step-index*,

*operator-applied*,

“input-data”: [

*input-data-descriptors*

],

“output-data”: [

*output-data-descriptors*

],

“parameters”: [

*parameters-applied*

],

**“results”: {**

***operator-specific-run-results***

**}**

}

The *operator-specific-run-results* are described in the operator sections below.

In the event errors occur running operators, the *step-definition* will have the format:

{

"id": *step-index*,

*operator-applied*,

“input-data”: [

*input-data-descriptors*

],

“output-data”: [

*output-data-descriptors*

],

“parameters”: [

*parameters-applied*

],

**“errors”: [**

***operator-run-errors***

**]**

}

The *operator-run-errors* have the format:

“module-id”: “error-message”

where *module* is the routine in which the error occurred, *id* is a unique id for the error within the module, and *error-message* describes what error occurred. The *module-id* can be used to look up an explanation of the error in a separate error message document.

Where relevant, the end of experiment log output includes the output image pixel statistics:

| Field | Description |
| --- | --- |
| nrows | The number of image rows |
| ncols | The number of image columns |
| components | The number of image components. Grayscale images have 1 component, color images have 3 components. |
| depth | Pixel value data type.   * CV\_8U * CV\_32S * CV\_32F |
| pixel count | image nrows \* ncols \* components |
| pixel mean | pixel value mean |
| pixel standard deviation | pixel value standard deviation |
| min pixel value | minimum pixel value |
| max pixel value | maximum pixel value |

## Running the script

Scripts currently can be run only in Linux. The Linux program to runs scripts is the following, and can be found at this link:

[cv-workbench](https://github.com/kushnertodd/cv-workbench/blob/master/bin/cv-workbench)

A copy of this document describing that version of the program can be found at:

[cv-workbench.docx](https://github.com/kushnertodd/cv-workbench/blob/master/doc/cv-workbench.docx)

The program can be run under an Intel-based Linux distribution such as [Ubuntu](https://ubuntu.com/). A version of Linux under which the program can be run on Windows 10 that is fairly easy to install uses [Windows Subsystem for Linux WSL2](https://docs.microsoft.com/en-us/windows/wsl/install) with Ubuntu or Pengwin.

The program is run under Linux with this command line:

$ ./cv-workbench *script-file*

The *script-file* is as described above. Serious errors that prevent running the script, such as a missing or invalid script file, are written to the terminal.

## Sample Operator Scripts

Following are sample scripts for a Sobel operator and an image intensity map transform.

### Sobel Operator script experiment step

A sample script experiment step specifying the Sobel operator is:

{

"id": 1,

"operator": "filter-edge-sobel",

"input-data": [

{

"id": 1,

"type": "image",

"repository": "filesystem",

"format": "jpeg",

"directory": ".",

"filename": "square45-90\_gray",

"ext": "jpg"

}

],

"output-data": [

{

"id": 1,

"type": "image",

"repository": "filesystem",

"format": "binary",

"directory": ".",

"filename": "square45-90\_sobel\_gray",

"ext": "bin"

}

],

"parameters": {

"orientation": "0"

}

}

The significant elements of the script are:

| Script segment | Description |
| --- | --- |
| {  "id": 1,  "operator": "filter-edge-sobel", | This is the first experiment step, id 1, for the Sobel edge operator |
| "input-data": [  {  "id": 1,  "type": "image",  "repository": "filesystem",  "format": "jpeg",  "directory": ".",  "filename": "square45-90\_gray",  "ext": "jpg"  }  ], | The input data for the operator is a CV\_8U JPEG image file named “square45-90\_gray.jpg” in the current directory |
| "output-data": [  {  "id": 1,  "type": "image",  "repository": "filesystem",  "format": "binary",  "directory": ".",  "filename": "square45-90\_sobel\_gray",  "ext": "bin"  }  ], | The output data for the operator is a CV\_32S binary image file named “square45-90\_sobel\_gray.bin” in the current directory. The Sobel operator defines that the image is CV\_32S. |
| "parameters": {  "orientation": "0"  }  } | The script parameter defines the Sobel orientation as 0 degrees. The mask used is:  [-1, 0, 1]  [-2, 0, 2]  [-1, 0, 1] |

### Image Intensity Map Operator script experiment step

A sample script experiment step specifying mapping image intensities for an input binary CV\_32S or CV\_32F image to a CV\_8U JPEG output image follows. The operator detects that the input image is in either a CV\_32S or CV\_32F format and handles either one.

{

"id": 1,

"operator": "transform-intensity-map",

"input-data": [

{

"id": 1,

"type": "image",

"repository": "filesystem",

"format": "binary",

"directory": ".",

"filename": "square45-90\_sobel\_gray",

"ext": "bin"

}

],

"output-data": [

{

"id": 1,

"type": "image",

"repository": "filesystem",

"format": "jpeg",

"directory": ".",

"filename": "square45-90\_sobel\_gray",

"ext": "jpg"

}

],

"parameters": {

"lower-in": "-1020",

"upper-in": "1020",

"lower-out": "0",

"upper-out": "255",

"depth": "CV\_8U"

}

The significant elements of the script are:

| Script segment | Description |
| --- | --- |
| {  "id": 1,  "operator": "transform-intensity-map", | This is the first experiment step, id 1, for the image intensity map operator |
| "input-data": [  {  "id": 1,  "type": "image",  "repository": "filesystem",  "format": "binary",  "directory": ".",  "filename": "square45-90\_sobel\_gray",  "ext": "bin"  }  ], | The input data for the operator is a CV\_32S or CV\_32F binary image file named “square45-90\_gray.bin” in the current directory |
| "output-data": [  {  "id": 1,  "type": "image",  "repository": "filesystem",  "format": "jpeg",  "directory": ".",  "filename": "square45-90\_sobel\_gray",  "ext": "jpg"  }  ], | The output data for the operator is a CV\_8U binary JPEG file named “square45-90\_sobel\_gray.jpg” in the current directory. The CV\_8U depth is implied by specifying a JPEG output format. |
| "parameters": {  "lower-in": "-1020",  "upper-in": "1020",  "lower-out": "0",  "upper-out": "255"  }, | The script parameter define lower\_in, upper\_in, lower\_out, and upper\_out intensity mapping parameters. |

# Operators

Operators are predefined for various domains and classes as indicated below. The current version supports only a limited set of operators, but more are added for each release. The following operators are currently defined. Those marked with ‘\*’ are not yet implemented in the current version.

| Domain | Class | Instance |  | Description |
| --- | --- | --- | --- | --- |
| Filter | Edge | Color |  | Color image edge operator |
| Filter | Edge | Gaussian | \* | Difference of Gaussian edge operator |
| Filter | Edge | Kirsch |  | Kirsch edge operator |
| Filter | Edge | Laplacian | \* | Laplacian of Gaussian detector |
| Filter | Edge | Prewitt |  | Prewitt edge operator |
| Filter | Edge | Roberts |  | Roberts edge operator |
| Filter | Edge | Sobel |  | Sobel edge operator |
| Filter | Image | Morphology |  | Map binary images with dilate, erode, open, close, gradient, top hat, or black at morphology transforms. |
| Filter | Smooth | Average | \* | Average smoothing operator |
| Filter | Smooth | Gaussian | \* | Gaussian smoothing operator |
| Filter | Smooth | Median | \* | Average smoothing operator |
| Histogram | Hough | Create |  | Histogram Hough accumulator counts |
| Histogram | Image | Create |  | Histogram pixel value counts for an image |
| Histogram | Peak | Detect | \* | Find histogram peaks exceeding a threshold |
| Hough | Draw | Line |  | Draw an image line segment for a polar (rho, theta) line |
| Hough | Draw | Lines | \* | Draw image line segments for polar (rho, theta) lines detected by the hough-peak-detect operator |
| Hough | Image | Create |  | Accumulate Hough (rho, theta) counts from an image |
| Hough | Peak | Detect |  | Find Hough peak lines exceeding a threshold |
| Transform | Image | Copy | \* | Copy image data and components from input to output images |
| Transform | Image | Create |  | Create test image from text file or from scratch optionally with superimposed features |
| Transform | Intensity | Map |  | Map CV\_32S or CV\_32F image intensities from an input intensity range to an output intensity range. Output images may be JPEG CV\_8U images or binary CV\_32S or CV\_32F images. |

Typical use cases are:

| Data type | Use | Steps |
| --- | --- | --- |
| Image | Generate edges | * edge filter image * get image histogram and statistics * analyze histogram and statistics to get edge thresholds * select edges * generate JPEG images for viewing |
| Hough | Detect lines | * generate edge image * create Hough accumulator * get Hough accumulator histogram and statistics * analyze histogram and statistics to get Hough line thresholds * select peaks * draw lines in original JPEG image for viewing |

## filter-edge-color Operator

The color edge operator produces an image pixels are replaced with a measure of the color distance between a pixel and its neighbors where the distance is measured with:

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “binary” or “jpeg” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “image” | “binary” | output CV\_32S, or CV\_32F edge image |

## filter-edge-kirsch Operator

The Kirsch edge operator convolves the image with kernels oriented to each of 8 directions as follows, where “[…], […], […]” are the nrows of the kernel:

| orientation | kernel |
| --- | --- |
| N | [-3, -3, 5], [-3, 0, 5], [-3, -3, 5] |
| NW | [-3, 5, 5], [-3, 0, 5], [-3, -3, -3] |
| W | [5, 5, 5], [-3, 0, -3], [-3, -3, -3] |
| SW | [5, 5, -3], [5, 0, -3], [-3, -3, -3] |
| S | [5, -3, -3], [5, 0, -3], [5, -3, -3] |
| SE | [-3, -3, -3], [5, 0, -3], [5, 5, -3] |
| E | [-3, -3, -3], [-3, 0, -3], [5, 5, 5] |
| NE | [-3, -3, -3], [-3, 0, 5], [-3, 5, 5] |

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “binary” or “jpeg” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “image” | “binary” | output CV\_32S, or CV\_32F edge image |

The parameter to the Kirsch operator is:

| parameter | description |
| --- | --- |
| orientation | “N”, “NW”, “W”, “SW”, “S”, “SE”, “E”, or “NE” direction as above |

## filter-edge-laplacian Operator

Planned but not yet implemented in the current release.

## filter-edge-prewitt Operator

The Prewitt edge operator convolves the image with kernels with orientations corresponding to 0 or 90 degrees as follows, where “[…], […], […]” are the nrows of the kernel:

| orientation | kernel |
| --- | --- |
| 0 | [-1, 0, 1], [-1, 0, 1], [-1, 0, 1] |
| 90 | [1, 1, 1], [0, 0, 0], [-1, -1, -1] |

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “jpeg” or “binary” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “image” | “binary” | output CV\_32S, or CV\_32F edge image |

The parameter to the Prewitt operator is:

| parameter | description |
| --- | --- |
| orientation | “0” or “90” degrees as above |

## filter-edge-roberts Operator

The Roberts edge operator convolves the image with kernels with orientations corresponding to 0 or 90 degrees as follows, where “[…], […]” are the nrows of the kernel:

| orientation | kernel |
| --- | --- |
| 0 | [0, 1], [-1, 0] |
| 90 | [1, 0], [0, -1] |

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “jpeg” or “binary” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “image” | “binary” | output CV\_32S, or CV\_32F edge image |

The parameter to the Roberts operator is:

| parameter | description |
| --- | --- |
| orientation | “0” or “90” degree as above |

## filter-edge-sobel Operator

The Sobel edge operator convolves the image with kernels with orientations corresponding to 0 or 90 degrees as follows, where “[…], […], […]” are the nrows of the kernel:

| orientation | kernel |
| --- | --- |
| 0 | [-1, 0, 1], [-2, 0, 2], [-1, 0, 1] |
| 90 | [1, 2, 1], [0, 0, 0], [-1, -2, -1] |

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “jpeg” or “binary” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “image” | “binary” | output CV\_32S, or CV\_32F edge image |

The parameter to the Sobel operator is:

| parameter | description |
| --- | --- |
| orientation | “0” or “90” degrees as above |

## filter-image-morphology Operator

The image morphology operator applies one of six different mathematical morphology operations on an image. Each operation applies a binary mask called a *structuring element* at each pixel position and decides as to what output value to produce. Erosion selects the minimum value under the mask and dilation selects the maximum value. This will give the appropriate result for both binary and grayscale images. The operator produce output images of the same depth as the input images.

| Operation | Compound | Description |
| --- | --- | --- |
| Erosion |  | Output the minimum value under the structuring element |
| Dilation |  | Output the maximum value under the structuring element |
| Opening | x | Erosion followed by dilation |
| Closing | x | Dilation followed by eroding |
| Gradient | x | Subtract the eroded image from the dilated image |
| Top Hat | x | Subtract the opened image from the original image |
| Black Hat | x | Subtract the original image from the closed image |

The operator takes the following data descriptors. The output data format is ignored.

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “jpeg” or “binary” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “image” | Same as input format. | output CV\_8U, CV\_32S, or CV\_32F intensity image |

The parameters to the Morphology operator are:

| parameter | description |
| --- | --- |
| structuring-element | Shape of the structuring element: “rectangle”, “cross”, “ellipse” |
| height | Rectangle, ellipse, or cross pixel height |
| width | Rectangle, ellipse, or cross pixel width |
| thickness | Cross pixel thickness |
| operation | “erode”, “dilate”, “open”, “close”, “gradient”, “top-hat”, or “black-hat” |

## filter-smooth-average Operator

The filter smooth average operator convolves the image with a kernel that averages the pixels under the kernel. For a kernel that is size nrows x ncols, the output pixel is the sum of the pixels in a rectangle nrows x ncols around the input pixel, divided by 1/(nrows \* ncols).

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “jpeg” or “binary” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “image” | “binary” | output CV\_32F edge image |

The parameters to the smooth average operator are:

| parameter | description |
| --- | --- |
| nrows | kernel height |
| ncols | kernel width |

## filter-smooth-gaussian Operator

The filter smooth gaussian operator convolves the image with a gaussian kernel that does a weighted average of the pixels under the kernel. The 1-dimensional gaussian using variance can be computed by convolving with this this formula:

The gaussian has a separable kernel, so the 2-dimensional gaussian can be computed asymmetrically by convolving with the above 1-dimensional formula in the x direction and with the following formula using variance in the y direction:

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “jpeg” or “binary” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “image” | “binary” | output CV\_32S, or CV\_32F edge image |

The parameters to the smooth average operator are:

| parameter | description |
| --- | --- |
| nrows | kernel height |
| ncols | kernel width |
| sigma-x | column variance |
| sigma-y | row variance |

## filter-smooth-median Operator

Planned but not yet implemented in the current release.

## histogram-hough-create Operator

The create Hough accumulator histogram operator creates a histogram for a Hough accumulator and stores it in binary or text format.

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “hough” | “binary” | input Hough accumulator |
| output | 1 | “histogram” | “binary” or “text” | output Hough accumulator histogram |

For text output format, the output data descriptor extension is ignored. The output text file has the extension “.txt” and the output gnuplot script has the extension “.gp” as described in section 5.2.2.

The parameters to the create hough histogram operator are:

| parameter | description |
| --- | --- |
| nbins | The number of histogram bins |
| lower-value | The minimum value accumulated in the first histogram bin |
| upper-value | The maximum value accumulated in the last histogram bin |

## histogram-image-create Operator

The create image histogram operator creates a histogram for an image and stores it in binary or text format. The mapping of input to output intensity values are determined as follows.

if input value < lower\_value

bin = 0;

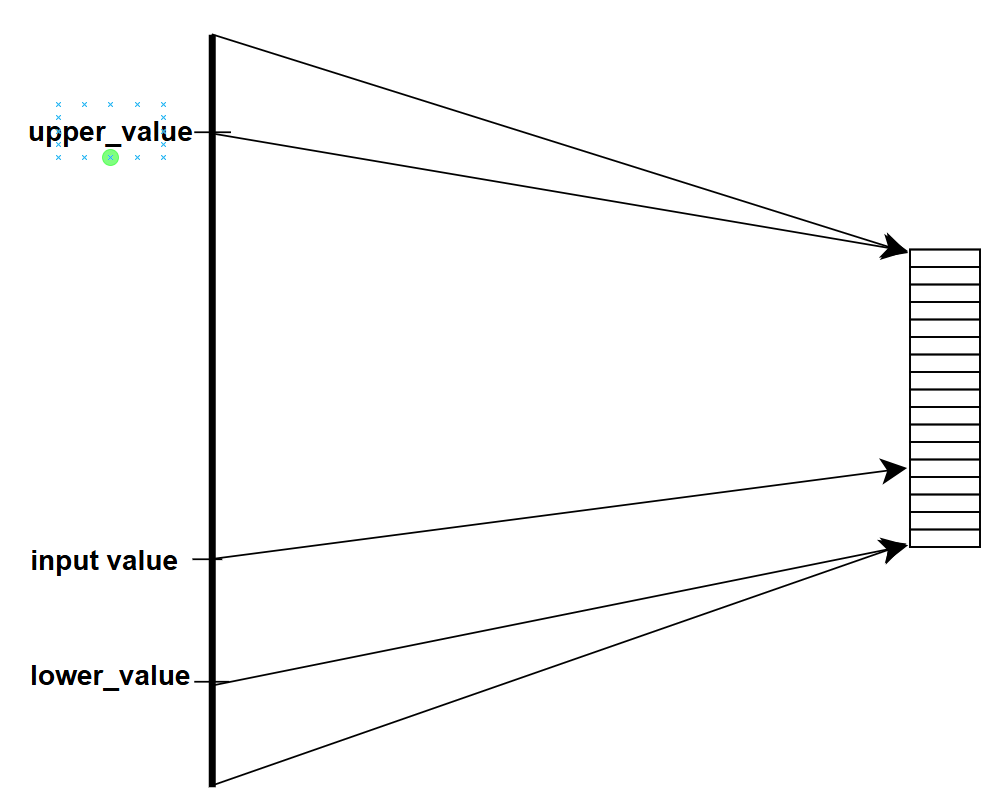
else if input value > upper\_value

bin = nbins - 1;

else

bin = (nbins - 1) \* (input value - lower\_value) / (upper\_value - lower\_value)

This is a diagram of the mapping.



The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “jpeg” or “binary” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “histogram” | “binary” or “text” | output image histogram |

For text output format, the output data descriptor extension is ignored. The output text file has the extension “.txt” and the output gnuplot script has the extension “.gp” as described in section 7.2.2.

The parameters to the create image histogram operator are. The lower\_value and upper\_value parameters may be omitted and default to the image minimum and maximum values.

| parameter | description |
| --- | --- |
| nbins | The number of histogram bins |
| lower-value | The minimum value accumulated in the first histogram bin |
| upper-value | The maximum value accumulated in the last histogram bin |

## histogram-peak-detect Operator

The Hough detect peak operator extracts the top (rho, theta) polar lines from a Hough accumulator amd stores it in binary or text format as described in section 5.4.

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “hough” | “binary” | input Hough accumulator data |
| output | 1 | “peak-lines” | “binary” or “text” | output Hough accumulator peak data |

The parameters to the create image Hough operator is:

| parameter | description |
| --- | --- |
| npeaks | the number of (rho, theta) polar lines to find |

## hough-peak-detect Operator

The Hough peak detect operator finds polar lines (rho, theta) in the Hough accumulator data where the counts exceed a threshold.

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “hough” | “binary” | input Hough accumulator data |
| output | 1 | “peak” | “binary” or “text” | output Hough accumulator polar line peaks. |

The parameters to the draw Hough peak detect operator are as follows. Only one of either the threshold or max\_peaks parameter should be supplied.

| parameter | description |
| --- | --- |
| threshold | Hough accumulator count threshold for selecting peaks |
| max-peaks | The maximum number of peaks to return |

## hough-draw-line Operator

The Hough draw line operator draws a line segment corresponding to a polar line (rho, theta) in an image and stores the resulting image. JPEG input images can be stored as either JPEG or binary output images, but binary input images can only be stored as binary output images. Grayscale input images produce grayscale output images, and color input images produce color output images. Lines are drawn in the color output image out\_component component if specified. The output color image component defaults to 1 if omitted and is ignored for grayscale images.

The operator takes the following data descriptors. The output data format is ignored.

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “jpeg” or “binary” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “image” | same as input | output CV\_8U, CV\_32S, or CV\_32F image with drawn line. |

The parameters to the draw Hough draw line operator are:

| parameter | description |
| --- | --- |
| rho | Polar line rho |
| theta | Polar line angle |
| pixel-value | Pixel value for drawn line segment |
| out-component | Color output image component in which to draw line |

This is a sample output.



## hough-draw-lines Operator

Planned but not yet implemented in the current release.

## hough-image-create Operator

The create image Hough operator creates a Hough accumulator from an image and stores it in binary or text format as described in section 7.3.

The operator takes the following data descriptors.

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “jpeg” or “binary” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “hough” | “binary” or “text” | output Hough accumulator data |

The parameters to create image Hough operator is:

| parameter | description |
| --- | --- |
| rho-inc | increment for the Hough accumulator rho. Defaults to 1. |
| theta-inc | increment for the Hough accumulator angle. The number of angles are 180 / theta\_inc. Defaults to 3. |
| pixel-threshold | pixel value threshold above which image points are candidates for accumulating. Defaults to 0. |
| accumulate | the value to accumulate for each (rho, theta) pair associated with the pixel   * “unit” to accumulate the value 1 for pixel hough points (default) * “value” to accumulate the pixel value for pixel hough points |
| min-theta | start theta. Defaults to 0. |
| max-theta | ending theta. Defaults to 180. |

## hough-peak-detect Operator

The Hough peak detect operator finds polar lines (rho, theta) in the Hough accumulator data where the counts exceed a threshold.

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “hough” | “binary” | input Hough accumulator data |
| output | 1 | “peak” | “binary” or “text” | output Hough accumulator polar line peaks. |

The parameters to the Hough peak detect operator are as follows. Only one of either the threshold or max\_peaks parameter should be supplied.

| parameter | description |
| --- | --- |
| threshold | Hough accumulator count threshold for selecting peaks |
| max-peaks | The maximum number of peaks to return |

## transform-image-combine Operator

The transform image combine operator does a linear recombination of two images. The formula for the output values are:

output(row, col) = scale1 \* image1(row, col) + scale2 \* image2(row, col) + offset

The operator takes the following data descriptors. Input images must be the same (nrows, ncols) size.

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “jpeg”, “binary”, “text” | input JPEG CV\_8U, binary CV\_32S/F, or text CV\_32S intensity value file. |
| input | 2 | “image” | “jpeg”, “binary”, “text” | input JPEG CV\_8U, binary CV\_32S/F, or text CV\_32S intensity value file. |
| output | 1 | “image” | “binary” | output CV\_32F combined image. |

These are the parameters to the draw transform image create operator.

| parameter | description |
| --- | --- |
| scale-1 | Scale factor for first image value, defaults to 1.0 |
| scale-2 | Scale factor for second image value, defaults to 1.0 |
| offset | Offset for output image value, defaults to 0.0 |

## transform-image-copy Operator

The transform image copy operator copies the specified image area and components from input to output images. The min-col, min-row, max-col, and max-row parameters specify the output image size. If all are omitted, the entire output image is a copy of the input image.

The copy components capability is not yet implemented, only grayscale images are supported. The function of the operator will depend on whether input and output images are grayscale or color.

| Input Image | Output Image | Description |
| --- | --- | --- |
| Grayscale | Grayscale | Copy the input image to the output image. |
| Grayscale | Color | * Copy the input image to the output image out\_component component. * Copy the input image to the first output image component if the out\_component parameter is omitted. |
| Color | Grayscale | * Copy the input image in\_component component to the output image in\_component component. * Copy the first input image component to the output image if the in\_component parameter is omitted. |
| Color | Color | Copy from the input image to the output image as below. |

There are several variations depending whether the in\_component or out\_component parameters are specified.

| in\_component | out\_component | description |
| --- | --- | --- |
| specified | specified | Copy the input image in\_component to the output image out\_component component |
| specified | omitted | Copy the input image in\_component component to the in\_component component of the output image |
| omitted | specified | Copy the input image out\_component component to the output image out\_component component |
| omitted | omitted | Copy all input image components to the output image |

The parameters to the draw transform image copy operator are:

| parameter | description |
| --- | --- |
| in-component | The component of the input image to copy (not yet implemented) |
| out-component | The component of the output image to copy from the input image (not yet implemented) |
| min-col | Starting upper left-hand input image column, defaults to 0 |
| min-row | Starting upper left-hand input image column, defaults to 0 |
| max-col | Starting upper left-hand input image column, defaults to input image columns - 1 |
| min- row | Starting upper left-hand input image column, defaults to 0, defaults to input image rows - 1 |

## transform-image-create Operator

The transform image create operator creates an image from text file or from scratch, optionally with superimposed features. This is useful to create test images. The operator takes the following data descriptors. Various parameters optionally draw points, lines, or rectangles in the image.

* An image is created from scratch and the ‘nrows’ and ‘ncols’ parameters are required. The output image is CV\_8U data format.
* A draw command text input file is required.
* The output format is JPEG with pixel values limited 0 to 255.

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “data” | “text”, or omitted | input draw commands file. |
| output | 1 | “image” | “jpeg” or “binary” | output CV\_8U created image. |

These are the parameters to the draw transform image create operator. The features will be added with the foreground value.

| parameter | Description |
| --- | --- |
| nrows | The number of image rows for a new image. Required if a text input image is omitted, else ignored. |
| ncols | The number of image columns for a new image. Required if a text input image is omitted, else ignored. |
| background | The base pixel value of the output image 0..255, defaults to 0 |
| foreground | The pixel value of added features the output image 0..255, defaults to 255 |

The draw commands file contains a list of lines as follows.

|  |  |
| --- | --- |
| P *row*,*col* | A point to add to the image as |
| L *row1,col1,row2,col2* | A line segment with endpoints ‘(row1,col1):(row2,col2**)’** to add to the image |
| R *row1,col1,row2,col2* | A rectangle with corners ‘(row,col):(row,col)’ to add to the image, where the first ‘row,col’ is the upper left-hand corner and the second ‘row,col’ is the lower right-hand corner |
| F *row1,col1,row2,col2* | Similar to the ‘rectangle’ parameter, but creates a filled rectangle. |

## transform-intensity-map Operator

The transform intensity map operator maps the input image pixel value range to the specified output image pixel value range. Input JPEG and binary images can map to CV\_8U JPEG images or CV\_32S and CV\_32F binary images The mapping of input to output intensity values are determined as follows.

if input value < lower\_in

output value = lower\_out

else if input value >= upper\_in

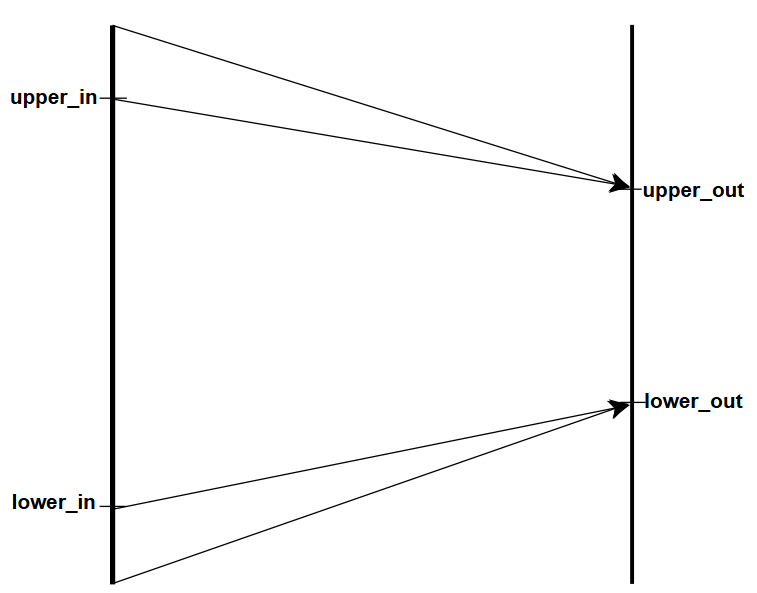
output value = upper\_out

else

output value =

lower\_out + (input value - lower\_in) \* (upper\_out - upper\_in) / (lower\_out - lower\_in)

This is a diagram of the mapping.



The operator can threshold images by specifying lower\_in = upper\_in = threshold value. Lower\_out then is the output pixel value for input pixel values < threshold, and upper\_out is the output pixel value for input pixel values >= threshold. Values for JPEG output images will be CV\_8U and truncated to 0..255.

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “jpeg” or “binary” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “image” | “jpeg” or “binary” | output CV\_8U, CV\_32S, or CV\_32F mapped intensity image. |

The parameters to the draw transform intensity map operator are as follows.

| parameter | description |
| --- | --- |
| depth | The output image depth: CV\_8U, CV\_32S, or CV\_32F. Defaults to the input image depth, ignored for JPEG output images. |
| lower-in | The lower mapped pixel value of the input image |
| upper-in | The lower mapped pixel value of the input image |
| lower-out | The lower mapped pixel value of the output image |
| upper-out | The upper mapped pixel value of the output image |
| standard-deviations | The number of standard deviations around the mean to use as the range |

* Parameters lower-out and upper-out are required.
* If the standard-deviations option is specified, lower-in and upper-in are the mean +/- the number of standard deviations.
* If the standard-deviations option is not specified
  + if either lower-in or upper-in parameters are not specified, they default to the lower or upper image values.
  + if either lower-in or upper-in parameters are not specified, they default to the 0 or 255.

## transform-intensity-convert Operator

The transform intensity convert operator uses the following functions to convert the input image pixel value to the output image.

* absolute value
* logarithm
* square
* square toot

The absolute value function has no effect on input JPEG images that have unsigned pixel values, and the logarithm and square root functions on signed binary images causes errors for negative pixel values. The logarithm function returns 0.0 for pixel values less than 1.0.

The operator takes the following data descriptors:

| Descriptor type | Id | type | Format | Description |
| --- | --- | --- | --- | --- |
| input | 1 | “image” | “jpeg” or “binary” | input CV\_8U, CV\_32S, or CV\_32F intensity image |
| output | 1 | “image” | “jpeg” or “binary” | output CV\_8U, CV\_32S, or CV\_32F mapped intensity image. |

The parameters to the draw transform intensity map operator are as follows.

| parameter | description |
| --- | --- |
| function | “abs” for absolute value “log” for logarithm  “sqr” for square  “sqrt” for square root |

* The “function” parameter is required.
* Only one function parameter may be specified