**Application Programming Interface for the Optic Flow Analysis using MPEG encoded Videos on UNIX OSs**

Florian Raudies, Swapnaa Jayaraman, Rick Gilmore

**Abstract**

Over several years we analyzed the optic flow in videos taken with head-mounted cameras, sometimes synced with eye-tracking data from a mobile eye-tracker. We used this analysis tool to empirically measure the distribution of visual motion speeds and patterns available to children and infants. This application programming interface (API) has the purpose of making our processing pipeline more widely usable for other researchers and research laboratories and the open-source community by providing high-level descriptions routines in the form of an API. We open-sourced this API together with a reference implementation on github <https://github.com/opticflow/analysis> .

**Processing Pipeline**

For the analysis of optic flow, we developed a processing pipeline that starts with the MPEG (H.264) encoded videos and ends with the visualization of distributions of visual motion speeds and motion patterns. In brief, the pipeline consists of the following steps: The (i) step is the extraction of image frames from videos for further processing – to ensure that we got all the frames before starting the processing. To decode MPEG files, we interfaced MATLAB scripts to ffmpeg system calls. The (ii) step is the calculation of optic flow from consecutive image pairs. This step is the most compute intensive. To accelerate this step, we ported methods from MATLAB into C that are called from MATLAB through the mex-interface. The (iii) step is the computation of motion speeds, optionally within a circular aperture. The (iv) step is the construction of motion templates that can depend on image position within the visual field. Subsequently, the optic flow is projected onto these motion templates to define “responses” for these motion patterns. A circular aperture can be applied optionally for this projection as well.

**Installation and Configuration**

To deploy our software on an UNIX system, we assume that you have ffmpeg installed on your system, and have a GCC compiler added to your MATLAB to compile the ‘C native’ methods for the estimation of optic flow. To retrieve information about your compiler embedded into MATLAB, execute the command ‘mex.getCompilerConfigurations’ in your MATLAB command window. To get the information about your ffmpeg install/path, execute ‘which ffmpeg’ in your bash shell.

To install a GCC compiler in your MATLAB run ‘mex –setup’ in MATLAB. This should setup a default mexopts.sh in your user folder ~/.matlab/R2013a/. Save this default file and overwrite it with the ‘mexopts.sh’ provided file. This will unset the option –ansi and set the optimization level to –O3. To compile the source code from within MATLAB run

‘mex –v estimateFlow.c flowlib.c imagelib.c’.

To test your install, we provided a comprehensive test suite. Run ‘Code/toolbox/flow/TestAll.m’ from the folder ‘Code/toolbox/flow’, which should pass. Notice that these tests run for several 10s of seconds and can be slower depending on your machine. If your install is working the output should be ‘ALL TESTS PASSED.’ Alternatively, the test bench may stop in between with a failure message.

**Installing ffmpeg for Unix Systems**

For the install we used an Ubuntu 14.04 LTS. Download the source code from <https://ffmpeg.org/download.html> in our case the release 2.8.4. Then use the following commands to unzip, untar, configure, compile, and install the ffmpeg on your system. The sequence of commands is

bzip2 –d ffmpeg-2.8.4.tar.bzip2

tar –xvf ffmpeg-2.8.4.tar

cd ffmpeg-2.8.4

./configure

make

make install

To test the install run the commands which ffmpeg that should return /usr/local/bin/ffmpeg and which ffprobe that should return /usr/local/bin/ffprobe.

To see the version run the command ffmpeg –version which should return 2.8.4 in our case.

**Performance Evaluation**

The method that estimates optic flow takes most of the computation time and, thus, is the bottleneck of our pipeline. For this reason, we ported the MATLAB code to native C. We compare the runtime of the MATLAB and C implementations of the estimation method for optic flow on two different systems. The first system runs MATLAB on Windows 7 and uses the Microsoft Visual C++ 2008 compiler to compile the native C code. This system (I) has an Intel i5 M520 clocked at 2.4 GHz and 4 GBy of RAM. The second system (II) runs Ubuntu 14.04.3 LTS and uses the GCC 4.8.4 to compile the native C code. This system has 2 Intel Xeon CPUs X5675 @ 3.07GHz (with 12 cores each and hyper threading) and 188 GBy of RAM.

System I – Windows running on a laptop

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input size | 80 x 60 | 160 x 120 | 320 x 240 | 640 x 480 | 1280 x 960 |
| Levels \* | 14 | 21 | 27 | 34 | 40 |
| C (sec) | 0.12 | 0.57 | 2.20 | 11.58 | 52.00 |
| MATLAB (sec) | 1.62 | 4.30 | 18.24 | 98.39 | 445.11 |
| Speedup of C | 12.96 | 7.48 | 8.29 | 8.50 | 8.56 |

\* Number of levels in the image pyramid.

System II – Ubuntu running on a server

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input size | 80 x 60 | 160 x 120 | 320 x 240 | 640 x 480 | 1280 x 960 |
| Levels | 14 | 21 | 27 | 34 | 40 |
| C (sec) | 0.12 | 0.45 | 2.67 | 8.87 | 41.06 |
| MATLAB (sec) | 3.01 | 6.42 | 86.64 | 230.13 | 807.77 |
| Speedup of C | 26.18 | 14.39 | 32.40 | 25.94 | 19.67 |

Parameters used for the simulation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| nMaxLevel | sFactor | nWarp | nIter | lambda |
| 30 | 0.9 | 1 | 50 | 50 |

The performance numbers are from a single trial and are “rough” estimates. Notice that the MATLAB implementation uses multiple threads for some functions, such as filtering. Our C implementation is purely single threaded, thus far.

**Application Programming Interface (API)**

**Overview of Code/toolbox**

|  |  |
| --- | --- |
| **Utilities (utilities/)** | |
| DirNames = dirFilter(  dirName,  dirPrefix) | Finds all directories in a directory with the given prefix. If the prefix is ‘’, then all directories are returned. |
| FileNames = fileFilter(  dirPath,  fileSuffix) | Finds all files in a directory for a given suffix. |
| FileNames = fileFilterForDirectories(  dirPath,  fileSuffix) | Find all files in the directory and its subdirectories for a given a suffix. |
| nFile = nFileFilter(  dirPath,  fileSuffix) | Counts all files in a directory for a given suffix. |
| [iMin iMax] = indexMinMaxFileFilter(  dirPath,  fileNamePattern) | Gets the minimum and maximum appearing index for a specific filter pattern. |
| IndexForSegment = indexFileFilter(  dirPath,  fileNamePattern) | Returns all indices of first and last index for segments of consecutive indices that match the file name pattern. |
| saveFlow(Dx, Dy, outFile) | Saves the flow. |
| [Dx, Dy] = loadFlow(inFile) | Loads the flow. |
| saveHistogram(H, B, outFile) | Saves the histogram with bins. |
| [H, B] = loadHistogram(inFile) | Loads the histogram and bins. |
| savePatternMatch(Match, PatternType,  PatternPosition, outFile) | Saves pattern matching results. |
| [Match, PatternType, PatternPosition] =  loadPatternMatch(inFile) | Loads pattern matching results. |
| saveEntropyForAllFrames(Entropy,  IndexForFrame,Bin,outFile) | Saves the entropy for all frames. |
| [Entropy,IndexForFrame, Bin] =  loadEntropyForAllFrames(inFile) | Loads the entropy for all frames. |
| videoInfo = getVideoInfo(inFile) | Pulls information about a video file. |
| [status,msg] = extractFramesFromVideo(  inFile, outFile, opt) | Extracts image frames from a video file. |
| [H, B] = histogram(Data, Bin) | Computes the histogram of the data. |
| [H, Bx, By] = jointHistogram(X, Y,  BinX, BinY) | Computes the joint histogram for the data variables X and Y. |
| e = entropy(X, Data) | Computes the entropy for the data. |
| H = jointEntropy(X, Y, BinX, BinY) | Computes the joint entropy. |
| info = loadAnnotation(inFile,  formatString) | Loads the annotation file that belongs to a video. |
|  |  |
| **Computation of optic flow from image frames (flow/)** | |
| [Dx, Dy] = computeFlow2(  Img1, Img2, opt) | Computes the flow for a pair of images. |
| info = computeFlowForImages(inFile,  iFirstFrame, iLastFrame,  outFile, opt) | Computes the flows for all specified image pairs. |
| info = computeFlowForAllImages(  inFile, outFile, opt) | Computes the flows for all subsequent image pairs in a folder. |
|  |  |
| **Analysis of flow speeds (flow/speed/)** | |
| [S, H, B] = computeSpeedHistogram(  Dx, Dy, opt) | Computes the speed histogram. |
| info = computeSpeedHistogramForFlows(  inFile, IndexFirstFrame,  IndexLastFrame, outFile, opt) | Computes the speed histograms for all specified flows. |
| info= computeSpeedHistogramForAllFlows(  inFile, outFile, opt) | Computes the speed histogram for all flows within a folder. |
|  |  |
| **Analysis of flow pattern (flow/pattern/)** | |
| [DxP, DyP, opt] = flowPattern(  nX, nY, opt) | Create the flow patterns of given types. |
| [M, opt] = computePatternMatch(Dx, Dy,  DxP, DyP, opt) | Match the flow with flow patterns. |
| info = computePatternMatchForFlows(  inFile, IndexFirstFrame,  IndexLastFrame, outFile, opt) | Match specified flows with patterns. |
| info = computePatternMatchForAllFlows(  inFile, outFile, opt) | Match flows within one folder with patterns. |
|  |  |
| **Image processing methods (image/)** | |
| ImgRGB = histogramEqualize(ImgRGB,  H) | Histogram equalization for an image in RGB color format. |
| info = equalizeImageForAllImages(  inFile, outFile, opt) | Histogram-equalizes images within a folder. |
|  |  |
| **Analysis of entropy for images (image/entropy/)** | |
| e = computeEntropyForImage(Img, opt) | Compute the entropy of an image. |
| info = computeEntropyForImages(  inFile, IndexFirstFrame,  IndexLastFrame, outFile,  opt) | Compute the entropy for all specified images. |
| info = computeEntropyForAllImages(  inFile, outFile, opt) | Compute the entropy for all images within a folder. |
|  |  |
| **Analysis of entropy for flow (flow/entropy/)** | |
| E = computeEntropyForFlow(Dx, Dy, opt) | Compute the entropy of a flow field. |
| info = computeEntropyForFlows(  inFile, IndexFirstFrame,  IndexLastFrame, outFile, opt) | Compute the entropy for all specified flows. |
| info = computeEntropyForAllFlows(  inFile, outFile, opt) | Compute the entropy for all flows within one folder. |

**Utility functions**

FileNames = fileFilter(dirPath, fileSuffix)

Produces a file listing of all files ending with the fileSuffix in the directory with dirPath. This method does not allow any recursive traversal of sub-directories.

* dirPath: Path for input directory.
* fileSuffix: All files of the listing must have the file ending ‘fileSuffix’.

RETURNs

* FileNames: Listing of filtered file names with their full paths.

**Example:** The script is in Code/toolbox/utilities/ExampleFileFilter.m . The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script

dirPath = '../Data/SleepingBag/';

Videos = fileFilter(dirPath, 'MP4');

nVideos = length(Videos);

fprintf('The folder %s contains the video files.\n', dirPath);

for iVideos = 1:nVideos, fprintf('\t%s.\n', Videos{iVideos}); end

nFile = nFileFilter(dirPath, fileSuffix)

Counts the files with the suffix fileSuffix in the directory dirPath.

* dirPath: Path for input directory.
* fileSuffix: All files of the listing must have the file ending ‘fileSuffix’.

RETURNs

* nFile: Number of files with the suffix ‘fileSuffix’ in the directory dirPath.

**Example:** The script is in Code/toolbox/utilities/ExampleNFileFilter.m . The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

dirPath = '../Data/SleepingBag/';

fileSuffix = 'MP4';

nVideos = nFileFilter(dirPath, fileSuffix);

fprintf(['The folder %s contains %d video file(s) with ',...

'file extension %s.\n', dirPath, nVideos, fileSuffix);

[iMin iMax] = indexMinMaxFileFilter(dirPath, fileNamePattern)

Finds the minimum and maximum index of files following the naming pattern 'fileNamePattern' in the directory 'dirPath'. This method does NOT check that the indices appear consecutively.

* dirPath: Path to the directory that contains the files.
* fileNamePattern: Pattern of the file name, e.g. 'Img%05d.png'.

RETURNs

* iMin: Minimum index that matches the pattern. If none of the files matches iMin = intmax.
* iMax: Maximum index that matches the pattern. If none of the files matechs iMax = intmin.

**Example:** The script is in Code/toolbox/utilities/ExampleIndexMinMaxFileFilter.m . The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

dirPath = '../Data/SleepingBag/Image/';

fileNamePattern = 'Img%05d.png';

[iMin, iMax] = indexMinMaxFileFilter(dirPath, ...

fileNamePattern);

fprintf(['Minimum %d and maximum %d file index for images in'... 'folder %s.\n', iMin, iMax, dirPath);

IndexForSegment = indexFileFilter(dirPath, fileNamePattern)

Finds all the segments of consecutive indices in the directory 'dirPath' that match the 'fileNamePattern'.

* dirPath: Path to the directory that contains the files.
* fileNamePattern: Pattern of the file name, e.g. 'Img%05d.png'.

RETURNs

* IndexForSegment: Start and end index for each segment. Has the dimensions: nSegment x 2. If there are no matches returns an empty matrix.

**Example:** The script is in Code/toolbox/utilities/ExampleIndexFileFilter.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

dirPath = '../Data/SleepingBag/Image/';

fileNamePattern = 'Img%05d.png';

IndexForSegment = indexFileFilter(dirPath, fileNamePattern)

saveFlow(Dx, Dy, outFile)

Saves the optic flow variables ‘Dx’ and ‘Dy’.

* Dx: Horizontal flow component in units of pixels/frame.
* Dy: Vertical flow component in units of pixels/frame.
* outFile: The output filename

RETURNs: None

[Dx, Dy] = loadFlow(inFile)

Loads the optic flow variables ‘Dx’ and ‘Dy’ from the file ‘inFile’.

* inFile: Input file name of the optic flow field.

RETURNs

* Dx: Horizontal flow component in units of pixels/frame.
* Dy: Vertical flow component in units of pixels/frame.

saveHistogram(H, B, fileName)

Uses the Matlab save method to save 'H' and 'B'.

* H: Histogram data.
* B: Bins for the histogram.

RETURNs: None

[H, B] = loadHistogram(inFile)

Loads the histogram data and corresponding bins from a Matlab file.

* inFile: Input file name.

RETURNs

* H: Histogram data.
* B: Bins for the histogram data.

savePatternMatch(Match, PatternType, PatternPosition, outFile)

Saves the pattern matching information into a Matlab file.

* Match: The matching values for each flow pattern.
* PatternType: Cell array with pattern types as strings.
* PatternPosition: Positions for patterns.
* outFile: Output file pattern e.g. 'Pat%05d.mat'.

RETURNs

None

[Match, PatternType, PatternPosition] = loadPatternMatch( inFile)

Loads the pattern matching information from a Matlab file.

* inFile: Input file pattern, e.g. 'Pat%05d.mat'.

RETURNs

* Match: Matching result.
* PatternType: Flow pattern type is a cell array with strings.
* PatternPostion: Pattern positions.

saveEntropyForAllFrames(Entropy,IndexForFrame,Bin,outFile)

Saves the entropy from all frames in a Matlab file.

* Entropy: Matrix with the entropy per frame information. Each row in this matrix corresponds to one frame index.
* IndexForFrame: Index of the frame that corresponds to each row entry in the entropy matrix.
* Bin: Bins used for the computation of the entropy.
* outFile: Output file with path for the entropy, e.g. 'Entropy.mat'.

Returns: None

[Entropy, IndexForFrame, Bin] = loadEntropyForAllFrames( inFile)

Loads the entropy from a Matlab file.

* inFile: Input file name with path, e.g. 'Entropy.mat'.

RETURNs

* Entropy: The entropy matrix, where each row corresponds to one frame with the index from 'IndexForFrame'.
* IndexForFrame: Index of the frame that corresponds to the row in the entropy matrix.
* Bin: The bins used to compute the entropy. Notice, if there were multiple variables (as in the case of optic flow) each row corresponds to the bins for a variable.

videoInfo = getVideoInfo(inFile)

This method returns the meta-information of a video file in a structure. This is useful, e.g. for the estimation of space requirements to store all image frames or the size of single image frames.

* inFile: Input filename of the video file.

REUTRNs

* info: Object with information about the video such as duration, frame size, codec, audio, fps, … These are attributes of a class hierarchy.

**Example:** The script is in Code/toolbox/utilities/ExampleGetVideoInfo.m . The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script. This example will only run in an environment where ffprobe are part of the system path.

fileName = '../Data/SleepingBag/SleepingBag.MP4';

videoInfo = getVideoInfo(fileName);

fileName = videoInfo.getFileName(); % filename

duration = videoInfo.getDuration(); % sec

fps = videoInfo.getFramesPerSecond(); % fps

R = videoInfo.getResolution(); % width x height

fprintf('The video %s has the:\n', fileName);

fprintf(' - resolution [%d x %d]\n', R(1), R(2));

fprintf(' - duration %2.5f sec\n', duration);

fprintf(' - fps %2.3f frames/second\n', fps);

[status,msg] = extractFramesFromVideo(inFile, outFile, opt)

Extracts frames from a video file at a specific frame rate.

* inFile: Input filename of the video file.
* outFile: Pattern of an output filename used within ‘sprintf’ to generate filenames for image frames, e.g. ‘Img%05d.png’. The file ending is used to identify the image format.
* opt: Structure with field(s):
* fps: Frames per second that should be extracted in frames/second.

RETURNS

* status: Error code from ffmpeg. If status = 0, then there was no error.
* msg: Any output of ffmpeg that was printed to std::out is returned as string, which can be parsed for validation.

**Example:** The script is in Code/toolbox/utilities/ExampleExtractFramesFromVideo.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script. This script will only run if ffprobe and ffmpeg are part of the system path.

maxStorage = 2; % GBy

videoFile = '../Data/SleepingBag/SleepingBag.MP4';

outDir = '../Data/SleepingBag/Image/';

mkdir(outDir); % Ensure that the output directory exists.

imgFile = [outDir, 'Img%05d.png'];

videoInfo = getVideoInfo(videoFile);

fps = videoInfo.getFramesPerSecond();

duration = videoInfo.getDuration();

R = videoInfo.getResolution();

nFrame = floor(duration\*fps);

isStorage = ceil(4 \* prod(R) \* nFrame /2^10 /2^10 /2^10); % GBy

fprintf('The required storage is %d GBy.\n',isStorage);

assert(isStorage<=maxStorage, ...

'Storage is %d GBy but the user set the maximum %dGBy.\n',...

isStorage, maxStorage);

opt.fps = fps;

[status, msg] = extractFramesFromVideo(videoFile, imgFile, opt);

if status,

fprintf('Extracted %d frames from %s successfully.\n', ...

nFrame,videoFile);

end

[H, B] = histogram(Data, Bin)

This function computes the histogram for data with the binning:

H(1) H(2) H(nBin+1)

Data<=Bin(1) Bin(1)<Data<=Bin(2) ... Bin(nBin)<Data

B = 0.5\*(Bin(2:end)+Bin(1:end-1))

For most applications the data H(2:end-1) is displayed,

* Data: Data to calculate the histogram for.
* Bin: Edges of bins.

RETURNs

* H: Histogram data with nBin+1 entries.
* B: Centers of internal bins with nBin-1 entries.

**Example:** The script is in Code/toolbox/utilities/ExampleHistogram.m.

nData = 10^5;

muForData = 2;

sigmaForData = 4;

nBin = 17;

Data = sigmaForData\*randn(nData,1) - muForData;

Bin = linspace(-3\*sigmaForData,3\*sigmaForData,nBin);

[H, B] = histogram(Data, Bin);

figure; bar(B,H(2:end-1));

jointHistogram(X, Y, BinX, BinY)

Computes the joint histogram for the first/second data source:

H(i,j) = |{BinY(i)<=y<BinY(i+1) /\ BinX(i)<=x<BinX(i+1)

: x e X, y e Y}|

This counts the occurrences of data within the interval of the two bins. In total H has |X x Y| data entries or nX x nY data entries.

* X: First data source.
* Y: Second data source.
* BinX: Bins for first data source.
* BinY: Bins for second data source.

RETURNs

* H: The joint histogram data with dimensions: (nBinY+1) x (nBinX+1)
* Bx: Bins for first data source.
* By: Bins for second data source.

**Example:** The script is in Code/toolbox/utilities/ExampleJointHistogram.m.

sigmaX = 5;

sigmaY = 7;

muX = -2;

muY = 3;

nX = 1\*10^4;

nY = 2\*10^4;

nBinX = 13;

nBinY = 15;

X = sigmaX \* randn(nX,1) - muX;

Y = sigmaY \* randn(nY,1) - muY;

BinX = linspace(-2\*sigmaX,2\*sigmaX,nBinX);

BinY = linspace(-2\*sigmaY,2\*sigmaY,nBinY);

[H, Bx, By] = jointHistogram(X, Y, BinX, BinY);

figure;

imagesc(H(2:end-1,2:end-1),'XData',Bx,'YData',By); colormap gray;

e = entropy(Data, Bin)

Computes the entropy -sum(P x log2(P)) for P(i) being the relative occurrence of data in the interval given by Bin(i).

* Data: Input data.
* Bin: This is the binning of the data that is used when computing the entropy.

RETURNs

* e: The entropy value in bits.

**Example:** The script is in Code/toolbox/utilities/ExampleEntropy.m.

sigma = 5;

nData = 10^7;

Data = sigma\*randn(nData,1);

Bin = linspace(-2\*sigma, 2\*sigma, 4\*sigma);

eIs = entropy(Data, Bin);

eShouldBe = 0.5\*log2(2\*pi\*exp(1)\*sigma^2);

fprintf(' Entropy is %e and should be %e.\n', eIs, eShouldBe);

e = jointEntropy(X, Y, BinX, BinY)

Computes the joint entropy for the first/second random variable as:

H(i,j) = |{BinY(i)<=y<BinY(i+1) /\ BinX(i)<=x<BinX(i+1)

: x e X, y e Y}|

P(i,j) = H(i,j)/sum\_{i,j}{ P(i,j }

e = sum\_{i,j}{ P(i,j) log2(P(i,j)) }

* X: First data source.
* Y: Second data source.
* BinX: Bins for first data source.
* BinY: Bins for second data source.

RETURNs

* e: The joint entropy value in bits.

**Example:** The script is in Code/toolbox/utilities/ExampleJointEntropy.m.

sigmaX = 3;

sigmaY = 7;

nDataX = 10^5;

nDataY = 10^5;

DataX = sigmaX\*randn(nDataX,1);

DataY = sigmaY\*randn(nDataY,1);

BinX = linspace(-2\*sigmaX, 2\*sigmaX, 4\*sigmaX);

BinY = linspace(-2\*sigmaY, 2\*sigmaY, 4\*sigmaY);

eIs = jointEntropy(DataX, DataY, BinX, BinY);

eShouldBe = 1+log2(2\*pi) + 0.5\*log2(sigmaX^2 \* sigmaY^2);

fprintf(' Entropy is %e and should be %e.\n', eIs, eShouldBe);

info = loadAnnotation(inFile, formatString)

This loads annotations from CSV files.

* inFile: Input file name.
* formatString: Formatting of the data per row.

RETURNs

* info: Structure with GENERIC fields according to the header names in the CSV file. Each field has all row values for that name as data.

**Example:** The script is in Code/toolbox/utilities/ExampleLoadAnnotation.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

formatString = '%d%d%d%s%d';

inFile = '../Data/AnnotationSamples/036MR\_11.csv';

info = loadAnnotation(inFile, formatString);

Names = fieldnames(info);

nNames = length(Names);

fprintf('The file %s has the following fields:\n', inFile);

for iName = 1:nNames,

name = Names{iName};

data = info.(name);

fprintf(' %s with type %s and %d values.\n', ...

name, formatString(2\*iName), length(data));

end

**Computation of optic flow from image frames**

[Dx, Dy] = computeFlow2(Img1, Img2, opt)

Compute the optic flow for an image pair.

* Img1: 1st input image.
* Img2: 2nd input image.
* opt: Structure with fields:
* nLevel: This is the maximum number of levels used in the image pyramid. The actual number of levels used can be smaller and depends on the input image. The number of levels is at most 1+ceil(log(16.0/min(nY, nX))/log(sFactor)), where nY and nX denote the height and width of the image, respectively.
* ScaleOrSize: Rescale the image by this factor or resize the image to the size [row column].
* nWarp: Number of warping steps.
* nIter: Number of iterations between the primal/dual problem.
* lambda: A parameter that controls the smoothness of the solution.

RETURNs

* Dx: Horizontal flow component in units of pixels/frame.
* Dy: Vertical flow component in units of pixels/frame.

**Example:** The script is in Code/toolbox/flow/ExampleComputeFlow2.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

imgFile = '../Data/SleepingBag/Image/Img%05d.png';

Img1 = double(rgb2gray(imread(sprintf(imgFile, 1))))/255;

Img2 = double(rgb2gray(imread(sprintf(imgFile, 2))))/255;

[Dx, Dy] = computeFlow2(Img1, Img2, struct());

Img = flowToImage(Dx,Dy);

figure;

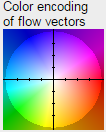
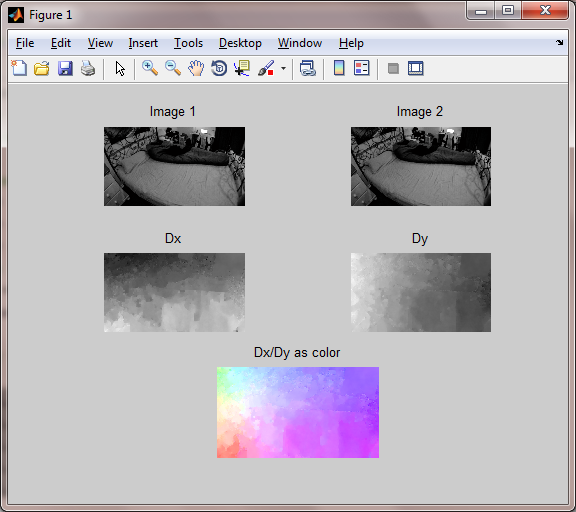
subplot(2,3,1); imshow(Img1); title('Image 1');

subplot(2,3,2); imshow(Img2); title('Image 2');

subplot(2,3,3); imshow(Dx,[]); title('Dx');

subplot(2,3,4); imshow(Dy,[]); title('Dy');

subplot(3,2,5:6); imshow(Img); title('Dx/Dy as color');



Output of the example. The example does not produce the color code shown to the right.

info = computeFlowForImages(inFile, IndexFirstFrame, IndexLastFrame, outFile, opt)

Computes the optic flow for a sequence of images.

* inFile: Pattern for the input file name, e.g. ‘Img%05d.png’, used in sprintf to generate the filename for each image frame. The method estimates ‘iLastFrame-iFirstFrame’ flow fields, because two frames are required to estimate a single optic flow field.
* iFirstFrame: The index of the first frames for each segment.
* iLastFrame: The index of the last frames for each segment.
* outFile: Pattern for the output file name, e.g. ‘Flw%05d.mat’, but instead of ‘mat’ it could be another file format (e.g. ‘pcm’ or ‘flw’) too. Output files have the indices iFirstFrame…iLastFrame-1. The file ending defines the file format used to save the optic flow.
* opt: Structure with the fields:
* ScaleOrSize: Rescale the image by this factor or resize the image to the size [row column].
* verbose: Print status updates about the progress of the computation.
* And all other arguments from the method ‘estimateFlow2’.

RETURNs

* info: Structure with the information about the calculated optic flow. Fields are:
* nFrame: Number of computed optic flow fields.
* nSegment: Number of segments. A segment is defined through consecutive indices.
* nY: Height of the frames in pixels.
* nX: Width of the frames in pixels.
* excTimeInM: Time of execution in minutes.
* excTimeInH: Time of execution in hours.
* excTimeInD: Time of execution in days.

**Example:** The script is in Code/toolbox/flow/ExampleComputeFlowForImages.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

imgDir = '../Data/SleepingBag/Image/';

flwDir = '../Data/SleepingBag/Flow/';

imgFile = [imgDir, 'Img%05d.png'];

flwFile = [flwDir, 'Flw%05d.mat'];

nFrame = nFileFilter(imgDir, 'png');

mkdir(flwDir);

opt.ScaleOrSize = 0.25;

opt.verbose = 1;

info = computeFlowForImages(imgFile,1,nFrame,flwFile,opt);

fprintf(['Computation of %d frames of size %d x %d pixels',... 'took:\n', info.nFrame, info.nX, info.nY);

fprintf(' %2.2f minutes.\n',info.excTimeInM);

fprintf(' %2.2f hours.\n',info.excTimeInH);

fprintf(' %2.2f days.\n',info.excTimeInD);

info = computeFlowForAllImages(inFile, outFile, opt)

Estimates the optic flow for all images within the folder of inFile. The other

* inFile: Pattern for the input file name see computeFlowForImages for more details.
* outFile: Pattern for the output file name see computeFlowForImages for more details.
* opt: Structure with the fields, are the same as for the method computeFlowForImages.

RETURNs

* info: Structure with the information about the calculated optic flow. Fields are:
* nFrame: Number of computed optic flow fields.
* nSegment: Number of segments. A segment is defined through consecutive indices.
* nY: Height of the frames in pixels.
* nX: Width of the frames in pixels.
* excTimeInM: Computation time in minutes.
* excTimeInH: Computation time in hours.
* excTimeInD: Computation time in days.

**Example:** The script is in Code/toolbox/flow/ExampleComputeFlowForAllImages.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

imgFile = '../Data/SleepingBag/Image/Img%05d.png';

flwDir = '../Data/SleepingBag/Flow/';

flwFile = [flwDir, 'Flw%05d.mat'];

mkdir(flwDir);

opt.ScaleOrSize = 0.25;

opt.verbose = 1;

info = computeFlowForAllImages(imgFile, flwFile, opt);

fprintf(['The computation of %d frames at size %d x %d',... ['pixels took:\n', info.nFrame, info.nX, info.nY);

fprintf(' %2.2f minutes.\n', info.excTimeInM);

fprintf(' %2.2f hours.\n', info.excTimeInH);

fprintf(' %2.2f days.\n', info.excTimeInD);

**Analysis of flow speeds**

[S, H] = computeSpeedHistogram(Dx, Dy, opt)

Computes the histogram of the flow speeds for the optic flow.

* Dx: Horizontal flow component in units of pixel/frame.
* Dy: Vertical flow component in units of pixel/frame.
* opt: Structure with fields:
* aperture: ‘None’ (which is the default) or ‘Circular’. For the circular aperture a circle is placed around the center of the image plane and only the measurements within that circle are returned. The remaining measurements of speeds within the image plane are set to NaN.
* nBin: Number of bins for the histogram. The default value is 11.
* Range: Range [minValue, maxValue] for minimum and maximum speed in pixels/frames. If no range is given then the minValue = minimum of all speeds and maxValue = maximum of all speeds.
* fps: Frames per second.
* hFoV: Horizonal field of view in RAD.

RETURNs

* S: Image speed in degrees/frame. This matrix has the same size as the matrices ‘Dx’ / ‘Dy’.
* H: Histogram for the image speeds within the aperture. The counts are within the interval Range and the centers for the bins are returned in B.
* B: Centers of the bins in pixel/frame.

**Example:** The script is in Code/toolbox/flow/speed/ExampleComputeSpeedHistogram.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

[Dx, Dy] = loadFlow(sprintf(...

'../Data/SleepingBag/Flow/Flw%05d.mat',1));

opt.nBin = 17;

opt.aperture = 'Circular';

opt.hFoV = 170/180\*pi; % degrees per pixel.

opt.fps = 59.94; % frames per second.

[S, H, B] = computeSpeedHistogram(Dx, Dy, opt);

figure('Position',[50 50 800 500]);

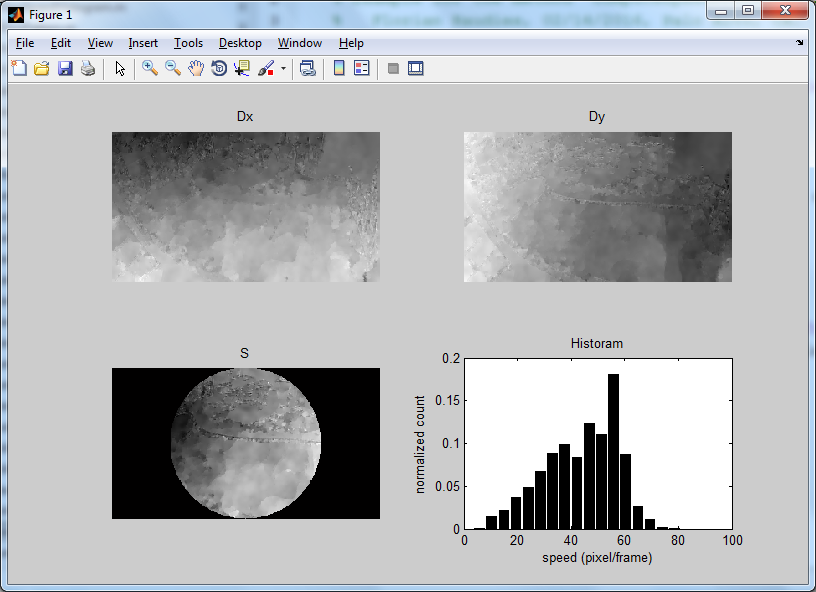
subplot(2,2,1); imshow(Dx,[]); title('Dx');

subplot(2,2,2); imshow(Dy,[]); title('Dy');

subplot(2,2,3); imshow(S,[]); title('S');

subplot(2,2,4); bar(B,H); xlabel('speed (pixel/frame)');

ylabel('normalized count'); title('Historam');



Example with circular aperture.

info = computeHistogramForFlows(inFile, IndexFirstFrame, IndexLastFrame, outFile, opt)

Computes the flow speed and histogram thereof for optic flows specified through the inFile and indices.

* inFile: Pattern for the input file name, e.g. ‘Flw%05d.mat’, used in sprintf to generate the filename for each flow field. The method computes iLastFrame-iFirstFrame+1 speeds.
* IndexFirstFrame: The index of the first frame in the segments.
* IndexLastFrame: The index of the last frame in the segments.
* outFile: Pattern for the output file name, e.g. ‘Hst%05d.mat’. Output files have the indices iFirstFrame…iLastFrame.
* opt: Structure with the fields:
* aperture: ‘None’ (which is the default) or ‘Circular’.
* verbose: [0|1] If 1 prints information about the progress. 1 is the default.
* fps: Frames per second.
* hFoV: Horizontal field of view in RAD.

RETURNs

* info: Structure with the information about the calculated histograms. Fields are:
* nFrame: Number of computed flow speeds.
* nSegment: Number of segments.
* nY: Height of the frames in pixels.
* nX: Width of the frames in pixels.
* excTimeInM: Computation time in minutes.
* excTimeInH: Computation time in hours.
* excTimeInD: Computation time in days.

**Example:** The script is in Code/toolbox/flow/speed/ExampleComputeSpeedHistogramForFlows.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

hstDir = '../Data/SleepingBag/Histogram/';

hstFile = [hstDir, 'Histogram.mat'];

flwDir = '../Data/SleepingBag/Flow/';

flwFile = [flwDir, 'Flw%05d.mat'];

mkdir(hstDir);

nFrame = nFileFilter(flwDir, 'mat');

opt.aperture = 'Circular';

opt.fps = 59.94;

opt.hFoV = 170/180\*pi;

info = computeSpeedHistogramForFlows(flwFile, 1, ...

nFrame, hstFile, opt);

fprintf(['Computed histogram of flow speeds for %d frames',... 'each %d x %d pixels.\n', info.nFrame, info.nY, info.nX);

fprintf(['Computation took %2.2f minutes, %2.2f hours, ', ...

'or %2.2f days.\n', info.excTimeInM, info.excTimeInH, ...

info.excTimeInD);

info = computeHistogramForAllFlows(inFile, outFile, opt)

Computes the histogram of flow speeds for optic flows of one directory.

* inFile: Pattern for the input file name, e.g. ‘Flw%05d.mat’.
* outFile: Pattern for the output file name, e.g. ‘Hst%05d.mat’. Output files have the indices iFirstFrame…iLastFrame.
* opt: Structure with the fields:
* aperture: ‘None’ (which is the default) or ‘Circular’.
* verbose: [0 | 1] If 1 prints additional information about the progress of the computation. Default 0.
* fps: Frames per second.
* hFoV: Horizontal field of view in RAD.

RETURNs

* info: Structure with the information about the calculated flow speed. Fields are:
* nFrame: Number of computed flow speeds.
* nSegment: Number of segments.
* nY: Height of the frames in pixels.
* nX: Width of the frames in pixels.
* excTimeInM: Computation time in minutes.
* excTimeInH: Computation time in hours.
* excTimeInD: Computation time in days.

**Example:** The script is in Code/toolbox/flow/speed/ExampleComputeSpeedHistogramForAllFlows.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

hstDir = '../Data/SleepingBag/Histogram/';

hstFile = [hstDir, 'Histogram.mat'];

flwFile = '../Data/SleepingBag/Flow/Flw%05d.mat';

mkdir(hstDir);

opt.aperture = 'Circular';

opt.verbose = 1;

opt.fps = 59.94;

opt.hFoV = 170/180\*pi;

info = computeSpeedHistogramForAllFlows(flwFile, hstFile, opt);

fprintf(['Computed histogram of flow speeds for %d frames',... 'each %d x %d pixels.\n', info.nFrame, info.nY, info.nX);

fprintf(['Computation took %2.2f minutes, %2.2f hours, ', ...

'or %2.2f days.\n', info.excTimeInM, info.excTimeInH, ...

info.excTimeInD);

**Analysis of flow pattern**

[DxP, DyP, opt] = flowPattern(nX, nY, opt)

Defines the space of flow patterns assuming that the y-axis points downward and the x-axis points to the right. ATTENTION: This method brings the pattern type into the canonical order as given below.

* nX: Number of pixels in the horizontal dimension.
* nY: Number of pixels in the vertical dimension.
* opt: Structure with fields:
* PatternType = {‘exp’, ‘con’, ‘cw’, ‘ccw’, ‘left’, ‘right’, ‘up’, ‘down’} These are abbreviations for the following:
  + ‘exp’: An expansion flow pattern.
  + ‘con’: An contraction flow pattern.
  + ‘cw’: A clockwise rotational flow pattern.
  + ‘ccw’: A counterclockwise rotational flow pattern.
  + ‘left’: A leftward pointing, laminar flow pattern.
  + ‘right’: A rightward pointing, laminar flow pattern.
  + ‘up’: An upward pointing, laminar flow pattern.
  + ‘down’: A downward pointing, laminar flow pattern.

Laminar flow patterns are characterized by having only one direction for the motion vectors in the image plane. Notice that only the first four patterns depend on the position within the image plane.

* PatternPosition = [0 0] Position of the flow pattern in vertical and horizontal direction with respect to the center of the image screen in percentage of the entire size of the image screen. Thus, pattern positions inside the image screen are in the range -0.5 and 0.5.

RETURNs

* DxP: Horizontal flow component of the flow pattern with the three dimensions: nY x nX x nP, where ‘nP’ is the number of patterns. In the example the number of motion patterns is eight, nP = 8.
* DyP: Vertical flow component of the flow pattern with the dimensions: nY x nX x nP.
* opt: Same structure as input with default values set.

**Example:** The script is in Code/toolbox/flow/pattern/ExampleFlowPattern.m.

% Constructs expansion, contraction, clockwise, counter

% clockwise, and laminar flow patterns.

nX = 320; % pixels

nY = 240; % pixels

[DxP, DyP, opt] = flowPattern(nX, nY, struct()); % Use defaults

nP = size(DxP,3);

figure('Name','8 Flow Patterns','Position',[50 50 800 400]);

for iP = 1:nP,

Dx = squeeze(DxP(:,:,iP));

Dy = squeeze(DyP(:,:,iP));

Img = flowToImage(Dx, Dy);

patternType = opt.PatternType{iP};

subplot(2,4,iP); imshow(Img);

title(sprintf('%s',patternType));

end

% Constructs expansion, contraction, clockwise, and counter

% clockwise patterns for nine positions.

nX = 32; % pixels

nY = 24; % pixels

opt.PatternType = {'exp','con','cw','ccw'};

opt.PatternPosition = [ -0.5 -0.5; 0 -0.5; 0.5 -0.5; ...

-0.5 0; 0 0; 0.5 0; ...

-0.5 0.5; 0 0.5; 0.5 0.5];

[DxP, DyP] = flowPattern(nX, nY, opt);

nPattern = length(opt.PatternType);

nPosition = length(opt.PatternPosition);

nP = size(DxP,3); % There are 4 x 9 = 36 patterns.

figure('Name','36 Flow Patterns','Position',[50 50 1200 600]);

for iP = 1:nP,

Dx = squeeze(DxP(:,:,iP));

Dy = squeeze(DyP(:,:,iP));

Img = flowToImage(Dx, Dy);

[iPosition, iPattern] = ind2sub([nPosition nPattern], iP);

patternType = opt.PatternType{iPattern};

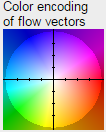
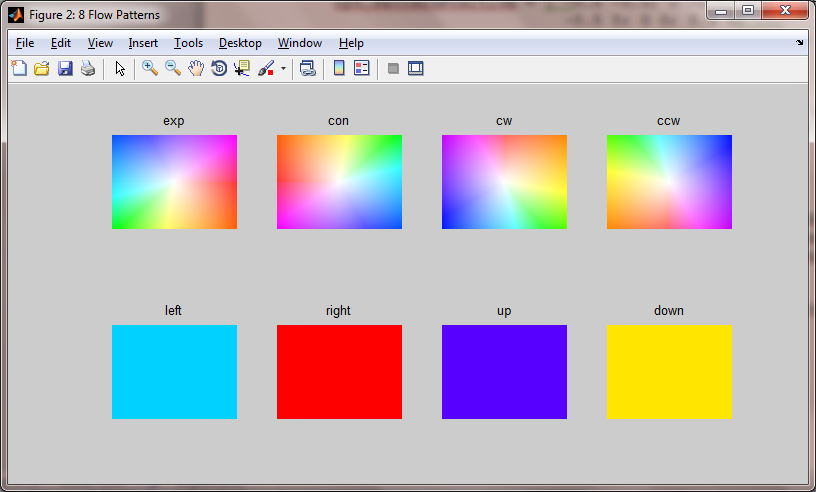
PatternPosition = opt.PatternPosition(iPosition,:);

subplot(4,9,iP); imshow(Img);

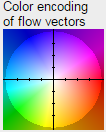
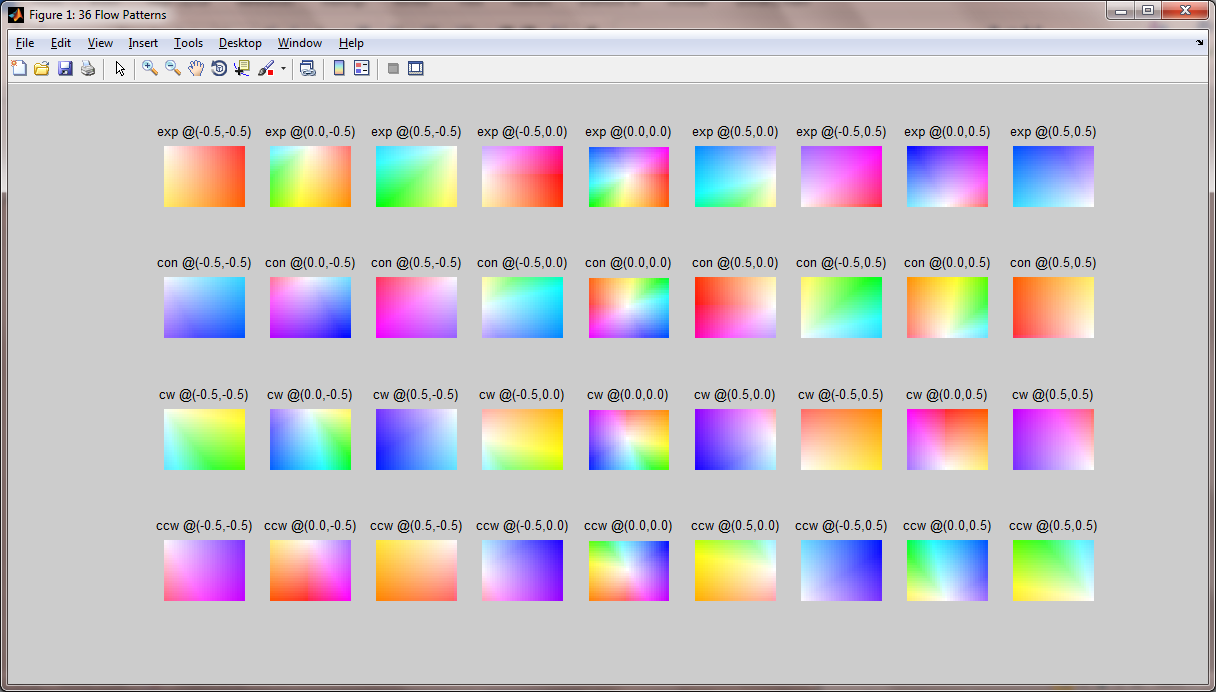
title(sprintf('%s @(%1.1f,%1.1f)',...

patternType,PatternPosition(1),PatternPosition(2)));

end



This is the 1st figure generated by the example. It shows the constructed flow patterns using the color code to the right. The color code on the right hand side is not generated by the example.



This is the 2nd figure generated by the example. It shows the constructed flow patterns that are spatially variable. The color code on the right hand side is not generated by the example.

[M, opt] = computePatternMatch(Dx, Dy, DxP, DyP, opt)

Matches the optic flow ‘Dx’ and ‘Dy’ onto the motion pattern ‘DxP’ and ‘DyP’.

* Dx: Horizontal component of the optic flow has dimensions: nY x nX.
* Dy: Vertical component of the optic flow has dimensions: nY x nX.
* DxP: Horizontal component of the motion pattern has dimensions: nY x nX x nP.
* DyP: Vertical component of the motion pattern has dimensions: nY x nX x nP.
* opt: Structure with fields:
* matchType = {‘normInnerProduct’ | ‘opponent’} This defines the matching type as inner product or using an opponent mechanism. The default is ‘normInnerProduct’.

RETURNs

* Matching values have the dimensions: nP x 1.
* opt: Structure with fields of all inputs and defaults that may have been set by the method.

**Example:** The script is in Code/toolbox/flow/pattern/ExampleComputePatternMatch.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

flwFile = sprintf(...

'../Data/SleepingBag/Flow/Flw%05d.mat', 1);

[Dx, Dy] = loadFlow(flwFile);

[nY, nX] = size(Dx);

[DxP, DyP, opt] = flowPattern(nX, nY, struct()); % Defaults.

opt.matchType = 'normReLuInnerProduct';

[M1, opt] = computePatternMatch(Dx, Dy, DxP, DyP, opt);

opt.matchType = 'normInnerProduct';

[M2, opt] = computePatternMatch(Dx, Dy, DxP, DyP, opt);

opt.PatternType = {'cw','exp','con','ccw','left','down','right','up'};

[DxP, DyP, opt] = flowPattern(nX, nY, opt);

opt.matchType = 'opponent';

[M3, opt] = computePatternMatch(Dx, Dy, DxP, DyP, opt);

Img = flowToImage(Dx, Dy);

figure('Position',[50 50 1100 700]);

subplot(2,3,1:3); imshow(Img); title('Flow');

subplot(2,3,4); bar(M1); set(gca,'XTickLabel',opt.PatternType);

ylabel('Match Score'); xlabel('Pattern Type');

title('Normalized, rectified, inner product');

subplot(2,3,5); bar(M2); set(gca,'XTickLabel',opt.PatternType);

ylabel('Match Score'); xlabel('Pattern Type');

title('Normalized inner product');

subplot(2,3,6); bar(M3); set(gca,'XTickLabel',opt.PatternType);

ylabel('Match Score'); xlabel('Pattern Type');

title('Opponent');

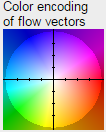
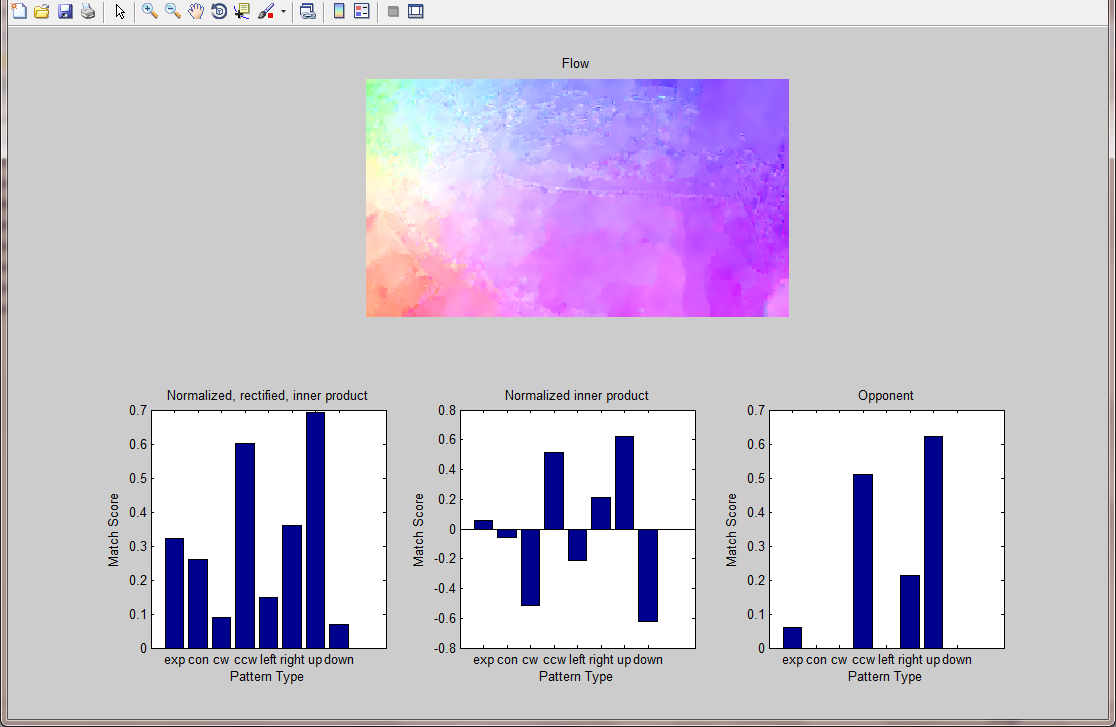


Figure of the example shows the input flow and the flow pattern matching results for the three matching mechanisms. Note that the opponent mechanism gives the clearest answer. The color code for the encoding of the flow is shown on the right and this color code is not part of the example.

info = patternMatchForFlows(inFile, IndexFirstFrame, IndexLastFrame, outFile, opt)

Matches the optic flows from inFile into pattern space constructed with opts and saves the matching results in the outFile.

* inFile: Input file pattern, e.g. ‘Flw%05d.flw’.
* IndexFirstFrame: First optic flow field for segments.
* IndexLastFrame: Last optic flow field for segments.
* outFile: Output file pattern, e.g. ‘Pat%05d.mat’.
* opt: Structure with fields:
* Same as for motionPattern and computePatternMatch combined.

RETURNs

* info: Structure with the information about the calculated flow patterns. Fields are:
* nFrame: Number of computed flow patterns.
* nP: Number of patterns.
* PatternType: Type of flow patterns.
* PatternPosition: Position of the flow patterns in the image plane.
* nY: Height of the frames in pixels.
* nX: Width of the frames in pixels.
* excTimeInM: Computation time in minutes.
* excTimeInH: Computation time in hours.
* excTimeInD: Computation time in days.

**Example:** The script is in Code/toolbox/flow/pattern/ExampleComputePatternMatchForFlows.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

flwDir = '../Data/SleepingBag/Flow/';

flwFile = [flwDir, 'Flw%05d.mat'];

patDir = '../Data/SleepingBag/Pattern/';

patFile = [patDir, 'Pattern.mat'];

mkdir(patDir);

nFrame = nFileFilter(flwDir, 'mat');

info = computePatternMatchForFlows(flwFile, 1, nFrame,... patFile, struct());

fprintf(['Computed flow pattern matches for %d frames each ',... '%d x %d pixels.\n', info.nFrame, info.nX, info.nY);

fprintf(['Computation took %2.2f minutes, %2.2f hours, ',...

'or %2.2f days.\n', info.excTimeInM, info.excTimeInH,

info.excTimeInD);

info = computePatternMatchForAllFlows(inFile, outFile, opt)

Matches all optic flows from inFile into pattern space constructed with opts and saves the matching results in the outFile.

* inFile: Input file pattern, e.g. ‘Flw%05d.flw’.
* outFile: Output file pattern, e.g. ‘Pat%05d.mat’.
* opt: Structure with fields:
* Same as for motionPattern and computePatternMatch combined.
* verbose: [0 | 1]. If 1 prints status information for the computation.

RETURNs

* info: Structure with the information about the calculated flow patterns. Fields are:
* nFrame: Number of computed flow patterns.
* nPattern: Number of patterns.
* PatternType: Type of flow patterns.
* PatternPosition: Position of the flow patterns in the image plane.
* nY: Height of the frames in pixels.
* nX: Width of the frames in pixels.
* excTimeInM: Computation time in minutes.
* excTimeInH: Computation time in hours.
* excTimeInD: Computation time in days.

**Example:** The script is in Code/toolbox/flow/pattern/ExampleComputePatternMatchForAllFlows.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

flwFile = '../Data/SleepingBag/Flow/Flw%05d.mat';

patDir = '../Data/SleepingBag/Pattern/';

patFile = [patDir, 'Pattern.mat'];

mkdir(patDir);

info = computePatternMatchForAllFlows(flwFile,

patFile, struct());

fprintf(['Computed flow pattern matches for %d frames each ',... '%d x %d pixels.\n', info.nFrame, info.nX, info.nY);

fprintf(['Computation took %2.2f minutes, %2.2f hours, ',...

'or %2.2f days.\n', info.excTimeInM, info.excTimeInH,

info.excTimeInD);

We worked with other velocity spaces (not Cartesian but polar) and other image spaces (not Cartesian but using a monopole and dipole mapping). In this API we excluded these alternative velocity space and image space to avoid further complexities.

**Image processing methods (image/)**

ImgRGB = histogramEqualize(ImgRGB, H)

Histogram equalization for an image in RGB color format.

* ImgRGB: Color image with RGB color space.
* H: Histogram that is used for equalization.

RETURNs

* ImgRGB: Equalized color image.

**Example:** The script is in Code/toolbox/image/ExampleHistogramEqualize.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

imgFile = '../Data/HistogramEqualization/036NII\_00054.png';

ImgRGB = imread(imgFile);

ImgYCBCR = rgb2ycbcr(ImgRGB);

ImgY = squeeze(ImgYCBCR(:,:,1));

[H, B] = histogram(ImgY(:),linspace(0,255,255));

CDF = cumsum(H);

minCDF = min(CDF);

ImgYEq = uint8(1 + floor((CDF(ImgY)-minCDF)/(numel(ImgY)-minCDF)\*255));

[HEq, BEq] = histogram(ImgYEq(:),linspace(0,255,255));

CDFEq = cumsum(HEq);

ImgEq = histogramEqualize(ImgRGB, H);

figure('Position',[50 50 900 400]);

subplot(2,3,1);

imshow(ImgRGB); title('Input Image');

subplot(2,3,2);

bar(B,H(2:end-1)); title('Histogram');

xlabel('Gray Value');

ylabel('Count');

subplot(2,3,3);

bar(B,CDF(2:end-1)); title('CDF');

xlabel('Gray Value');

ylabel('Cumulative Count');

subplot(2,3,4);

imshow(ImgEq); title('Equalized Image');

subplot(2,3,5);

bar(B,HEq(2:end-1)); title('Histogram Equalized');

xlabel('Gray Value');

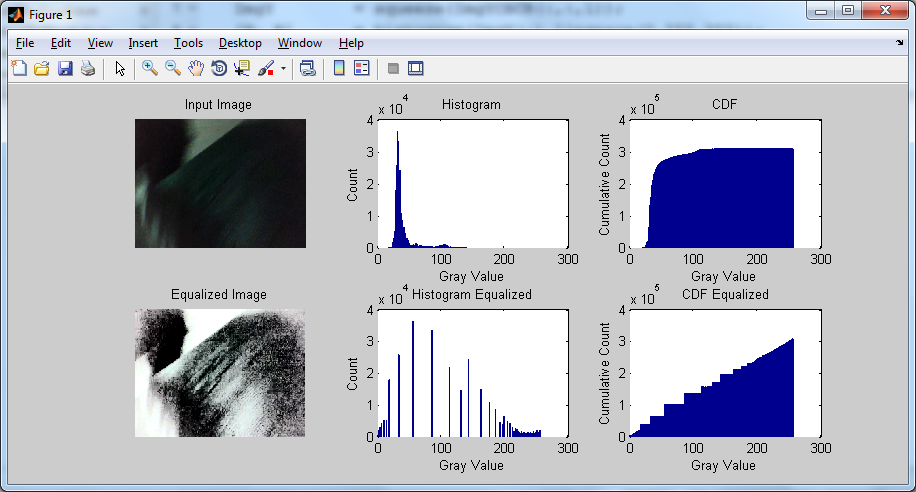
ylabel('Count');

subplot(2,3,6);

bar(B,CDFEq(2:end-1)); title('CDF Equalized');

xlabel('Gray Value');

ylabel('Cumulative Count');



The example script produces an output like the one above. The 1st row shows the input image and the histogram and cumulative distribution function (CDF) for the Y channel of the YCBCR color-encoded image. The 2nd row shows the histogram equalized image with its corresponding histogram and CDF.

info = equalizeImageForAllImages(inFile, outFile, opt)

Histogram-equalizes images within the file pattern ‘inFile’. All files from the input are used to build the histogram that is used for equalization. Thus, the transform for equalization is constant for all images.

* inFile: Input file pattern, e.g. 'Img%05d.png'.
* outFile: Output file pattern, e.g. 'Eq/Img%05d.png'.
* opt: Structure with fields:
* verbose: [0 | 1]. If 1 prints status information for the computation.

RETURNs

* info: Structure with the information about the calculated images. Fields are:
* nFrame: Number of equalized image frames.
* nSegment: Number of segments.
* excTimeInM: Computation time in minutes.
* excTimeInH: Computation time in hours.
* excTimeInD: Computation time in days.

**Analysis of entropy for images**

[e, opt] = computeEntropyForImage(Img, opt)

Computes the entropy in bits (base two) for the image using the specified range and number of bins.

* Img: Image data, typically has dimensions: nY x nX.
* opt: Structure with fields:
* ScaleOrSize: Rescale the image by this factor or resize the image to the size [row column].
* nBin: Number of bins to compute the entropy. The default is 11 bins.
* Range = [minValue, maxValue] Minimum and maximum value used when computing the mutual information. If no Range is given the default values minValue = minimum of all values and maxValue = maximum of all values is assumed.

RETURNs

* e: Entropy in bits.
* opt: Structure that contains the parameter information for the computation of the entropy.

**Example:** The script is in Code/toolbox/image/entropy/ExampleComputeEntropyForImage.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

imgFile = sprintf('../Data/SleepingBag/Image/Img%05d.png',1);

Img = rgb2gray(imread(imgFile));

opt.nBin = 256;

nCase = 4;

figure;

for iCase = 1:nCase,

subplot(2,2,iCase);

opt.ScaleOrSize = 1/4^(iCase-1);

[e, opt] = computeEntropyForImage(Img, opt);

imshow(imresize(Img,opt.ScaleOrSize));

title(sprintf(['%s\nfor size %d x %d pixels\n',...

'and the entropy of %2.2f bits.'],...

imgFile, opt.nY, opt.nX, e));

end

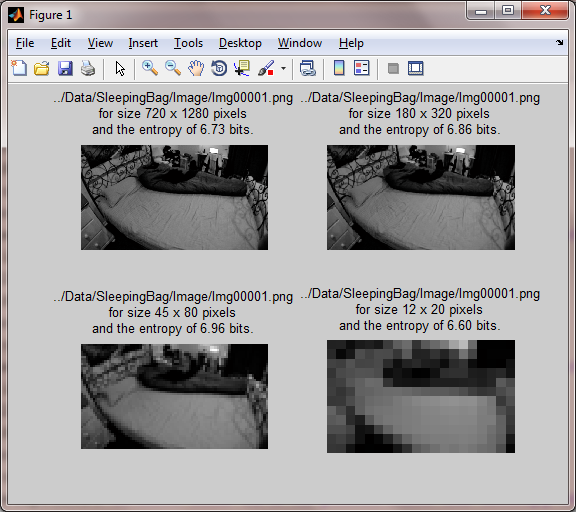


Figure of the example. This image has the entropy of 6.73 bits.

info = computeEntropyForImages(inFile, IndexFirstFrame, IndexLastFrame, outFile, opt)

Computes the entropy in bits (base two) for the images in the folder ‘inFile’ using the specified range and number of bins. The result is stored in one output file.

* inFile: Input file pattern for images, e.g. ‘Img%05d.png’.
* IndexFirstFrame: First frame of the images for segments.
* IndexLastFrame: Last frame of the images for segments.
* outFile: Output file for the entropy of all frames, e.g. ‘Entropy.mat’.
* opt: Structure with fields:
* The fields from computeEntropyForImage.
* verbose: [0 | 1] If 1 prints status information about the progress.

RETURNs

* info: Structure with the information about the calculated flow patterns. Fields are:
* nFrame: Number of computed flow patterns.
* nSegment: Number of segments.
* Bin: Bins used to compute the entropy.
* nY: Height of the frames in pixels.
* nX: Width of the frames in pixels.
* excTimeInM: Computation time in minutes.
* excTimeInH: Computation time in hours.
* excTimeInD: Computation time in days.

**Example:** The script is in Code/toolbox/image/entropy/ExampleComputeEntropyForImages.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

imgDir = '../Data/SleepingBag/Image/';

imgFile = [imgDir, 'Img%05d.png'];

nFrame = nFileFilter(imgDir,'png');

entropyDir = [imgDir, 'Entropy/'];

entropyFile = [entropyDir, 'Entropy.mat'];

mkdir(entropyDir);

opt.ScaleOrSize = 0.25;

info = computeEntropyForImages(imgFile, 1, nFrame, ...

entropyFile, opt);

fprintf(['Computed entropy for %d images each %d x %d ', ...

'pixels.\n', info.nFrame, info.nX, info.nY);

fprintf(['Computation took %2.2f minutes, %2.2f hours, ', ...

'or %2.2f days.\n', info.excTimeInM, info.excTimeInH,

info.excTimeInD);

info = computeEntropyForAllImages(inFile, outFile, opt)

Computes the entropy in bits (base two) for the images in the folder ‘inFile’. The result is stored in one output file with the name ‘outFile’.

* inFile: Input file pattern for images, e.g. ‘Img%05d.png’.
* outFile: Output file for the entropy of all frames, e.g. ‘Entropy.mat’.
* opt: Structure with fields:
* The fields from computeEntropyForImages.
* verbose: [0 | 1] If 1 prints status information about the progress.

RETURNs

* info: Structure with the information about the calculated entropy. Fields are:
* nFrame: Number of computed flow patterns.
* nSegment: Number of segments.
* Bin: Bins used to compute the entropy.
* nY: Height of the frames in pixels.
* nX: Width of the frames in pixels.
* excTimeInM: Computation time in minutes.
* excTimeInH: Computation time in hours.
* excTimeInD: Computation time in days.

**Example:** The script is in Code/toolbox/image/entropy/ExampleComputeEntropyForAllImages.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

imgDir = '../Data/SleepingBag/Image/';

imgFile = [imgDir, 'Img%05d.png'];

entropyDir = [imgDir, 'Entropy/'];

entropyFile = [entropyDir, 'Entropy.mat'];

mkdir(entropyDir);

opt.ScaleOrSize = 0.25;

info = computeEntropyForAllImages(imgFile, entropyFile, opt);

fprintf(['Computed entropy for %d images each %d x %d ', ...

'pixels.\n', info.nFrame, info.nX, info.nY);

fprintf(['Computation took %2.2f minutes, %2.2f hours, ', ...

'or %2.2f days.\n', info.excTimeInM, info.excTimeInH,

info.excTimeInD);

**Analysis of entropy for flow**

[E, opt] = computeEntropyForFlow(Dx, Dy, opt)

Computes the entropy in bits of each flow component separately and the joint entropy for both flow components.

* Dx: Horizontal flow component in pixel/frame.
* Dy: Vertical flow component in pixel/frame.
* opt: Structure with fields:
* nBin: Number of bins to compute the entropy.
* Range = [minValue, maxValue] Minimum and maximum value used when computing the mutual information.

RETURNs

* E: Entropy in bits for ‘Dx’, ‘Dy’, and the joint entropy of ‘Dx’ and ‘Dy’. The entropy E has the dimensions: 3 x 1.
* Structure ‘opt’ with all options, especially the default values for parameters.

**Example:** The script is in Code/toolbox/flow/entropy/ExampleComputeEntropyForFlow.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

[Dx, Dy] = loadFlow(sprintf(

'../Data/SleepingBag/Flow/Flw%05d.mat',1));

opt.nBin = 11;

[E, opt] = computeEntropyForFlow(Dx, Dy, opt);

figure;

subplot(2,2,1);

imshow(Dx,[]);

title(sprintf('Dx entropy = %2.2f bits', E(1)));

subplot(2,2,2);

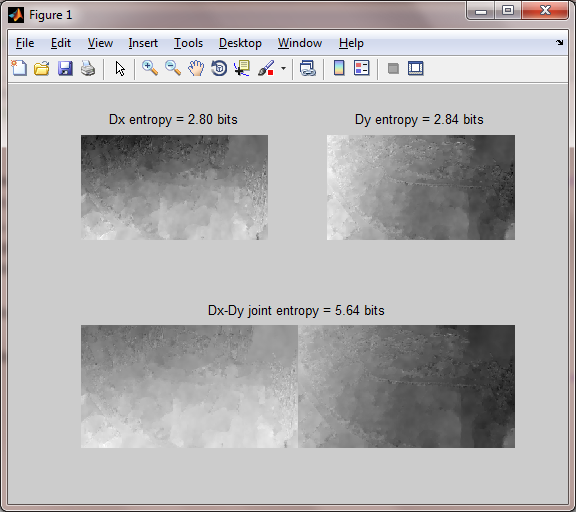
imshow(Dy,[]);

title(sprintf('Dy entropy = %2.2f bits', E(2)));

subplot(2,2,3:4);

imshow([Dx Dy],[]);

title(sprintf('Dx-Dy joint entropy = %2.2f bits', E(3)));



Output figure of the example that computes entropy for flow.

info = computeEntropyForFlows(inFile, IndexFirstFrame, IndexLastFrame, outFile, opt)

Computes the entropy in bits (base two) for the index specified flows in the folder ‘inFile’. The result is stored in one output file with the name ‘outFile’.

* inFile: Input file pattern for flows, e.g. ‘Flw%05d.mat’.
* IndexFirstFrame: First frame of the flow fields for segments.
* IndexLastFrame: Last frame of the flow fields for segments.
* outFile: Output file for the entropy of all frames, e.g. ‘Entropy.mat’.
* opt: Structure with fields:
* All those from ‘computeEntropyForFlow’.
* verbose = [0 | 1] If 1 prints status information about the progress of the computation.

RETURNs

* info: Structure with the information about the calculated entropy. Fields are:
* nFrame: Number of computed flow patterns.
* nSegment: Number of segments.
* BinY: Binning used for ‘Dy’.
* BinX: Binning used for ‘Dx’.
* nY: Height of the frames in pixels.
* nX: Width of the frames in pixels.
* excTimeInM: Computation time in minutes.
* excTimeInH: Computation time in hours.
* excTimeInD: Computation time in days.

**Example:** The script is in Code/toolbox/flow/entropy/ExampleComputeEntropyForFlows.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

flwDir = '../Data/SleepingBag/Flow/';

flwFile = [flwDir, 'Flw%05d.mat'];

nFrame = nFileFilter(flwDir, 'mat');

entropyDir = [flwDir, 'Entropy/'];

entropyFile = [entropyDir, 'Entropy.mat'];

mkdir(entropyDir);

info = computeEntropyForFlows(flwFile, 1, nFrame, ...

entropyFile, struct());

fprintf(['Computed histogram of flow speeds for %d frames ',...

'each %d x %d pixels.\n'], info.nFrame, info.nX, info.nY);

fprintf('Computation took %2.2f minutes, %2.2f hours, ',...

'or %2.2f days.\n', info.excTimeInM, info.excTimeInH,

info.excTimeInD);

info = computeEntropyForAllFlows(inFile, outFile, opt)

Computes the entropy in bits (base two) for all flows in the folder ‘inFile’. The result is stored in one output file with the name ‘outFile’.

* inFile: Input file pattern for flows, e.g. ‘Flw%05d.mat’.
* outFile: Output file for the entropy of all frames, e.g. ‘Entropy.mat’.
* opt: Structure with fields:
* All those from ‘computeEntropyForFlow’.
* verbose = [0 | 1] If 1 prints status information about the progress of the computation.

RETURNs

* info: Structure with the information about the calculated entropy. Fields are:
* nFrame: Number of computed flow patterns.
* nSegment: Number of segments.
* BinY: Binning used for ‘Dy’.
* BinX: Binning used for ‘Dx’.
* nY: Height of the frames in pixels.
* nX: Width of the frames in pixels.
* excTimeInM: Computation time in minutes.
* excTimeInH: Computation time in hours.
* excTimeInD: Computation time in days.

**Example:** The script is in Code/toolbox/flow/entropy/ExampleComputeEntropyForAllFlows.m. The example ASSUMES you are in the folder ‘Code’ when executing the Matlab script.

flwDir = '../Data/SleepingBag/Flow/';

flwFile = [flwDir, 'Flw%05d.mat'];

entropyDir = [flwDir, 'Entropy/'];

entropyFile = [entropyDir, 'Entropy.mat'];

mkdir(entropyDir);

info = computeEntropyForAllFlows(flwFile, entropyFile, struct());

fprintf(['Computed entropy for flow for %d frames ',...

'each %d x %d pixels.\n'], info.nFrame, info.nX, info.nY);

fprintf('Computation took %2.2f minutes, %2.2f hours, ',...

'or %2.2f days.\n', info.excTimeInM, info.excTimeInH,

info.excTimeInD);