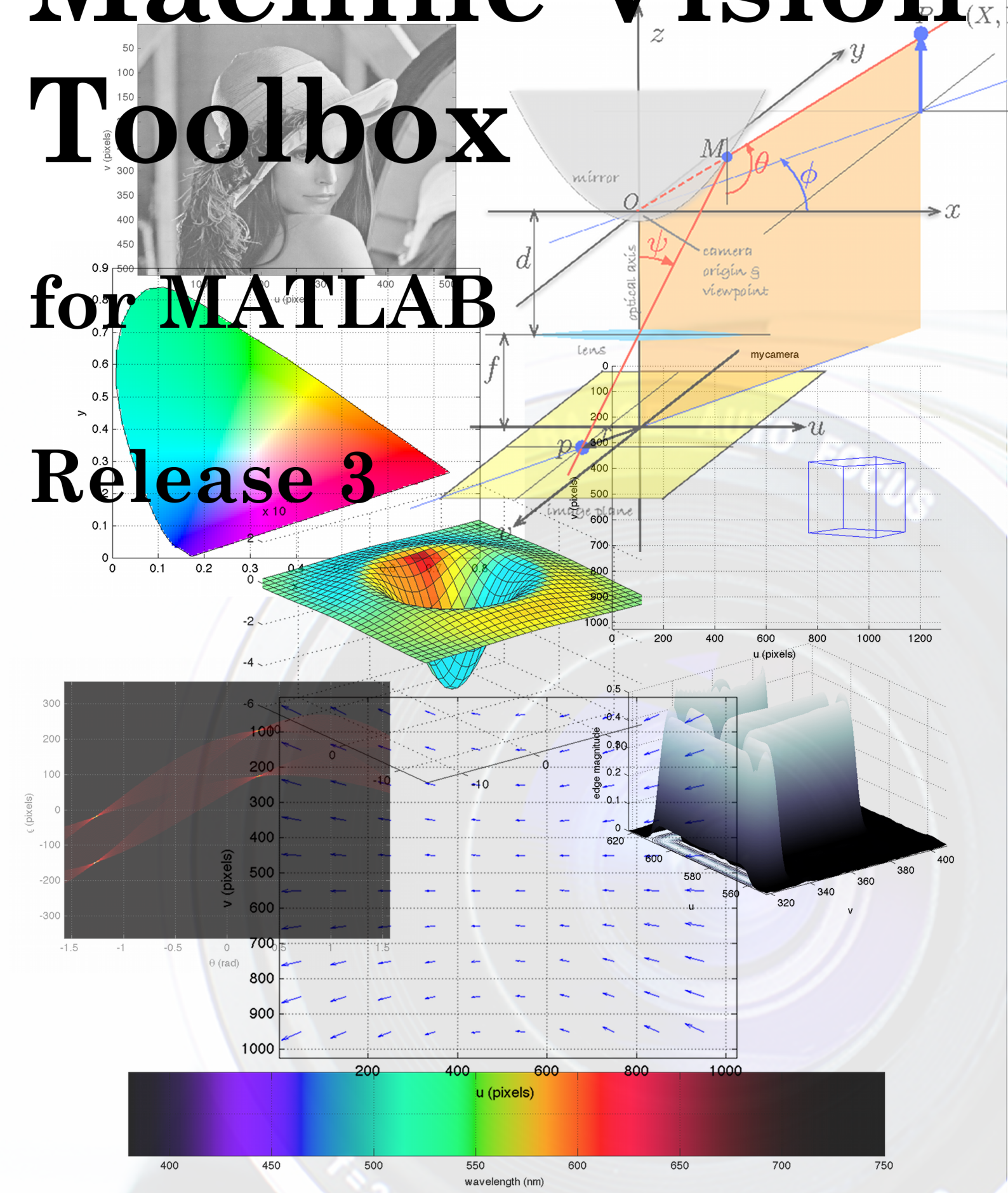
Machine Vision Toolbox for MATLAB

Release 3



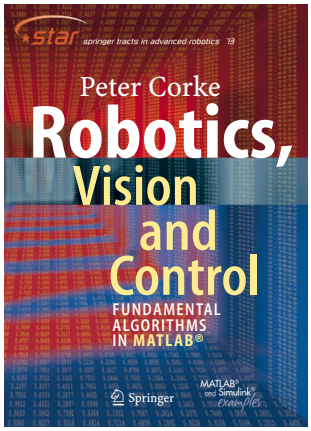
Peter Corke

Release	3.4
Release date	January 2015

Licence	LGPL
Toolbox home page	http://www.petercorke.com/robot
Discussion group	http://groups.google.com.au/group/robotics-tool-box

peter.i.corke@gmail.com
http://www.petercorke.com

Preface



This, the third release of the Toolbox, represents a decade of development. The last release was in 2005 and this version captures a large number of changes over that period but with extensive work over the last two years to support my new book "*Robotics, Vision & Control*" shown to the left.

The Machine Vision Toolbox (MVTB) provides many functions that are useful in machine vision and vision-based control. It is a somewhat eclectic collection reflecting my personal interest in areas of photometry, photogrammetry, colorimetry. It includes over 100 functions spanning operations such as image file reading and writing, acquisition, display, filtering, blob, point and line feature extraction, mathematical morphology, homographies, visual Jacobians, camera calibration and color space conversion. The Toolbox, combined with MATLAB® and a modern workstation computer, is a useful and convenient environment for investigation of machine vision algorithms. For modest image sizes the processing rate can be sufficiently "real-time" to allow for closed-loop control. Focus of attention methods such as dynamic windowing (not provided) can be used to increase the processing rate. With input from a firewire or web camera (support provided) and output to a robot (not provided) it would be possible to implement a visual servo system entirely in MATLAB®.

An image is usually treated as a rectangular array of scalar values representing intensity or perhaps range. The matrix is the natural datatype for MATLAB® and thus makes the manipulation of images easily expressible in terms of arithmetic statements in MATLAB® language. Many image operations such as thresholding, filtering and statistics can be achieved with existing MATLAB® functions. The Toolbox extends this core functionality with M-files that implement functions and classes, and mex-files for some compute intensive operations. It is possible to use mex-files to interface with image acquisition hardware ranging from simple framegrabbers to robots. Examples for firewire cameras under Linux are provided.

The routines are written in a straightforward manner which allows for easy understanding. MATLAB® vectorization has been used as much as possible to improve efficiency, however some algorithms are not amenable to vectorization. If you have the

MATLAB® compiler available then this can be used to compile bottleneck functions. Some particularly compute intensive functions are provided as mex-files and may need to be compiled for the particular platform. This toolbox considers images generally as arrays of double precision numbers. This is extravagant on storage, though this is much less significant today than it was in the past.

This toolbox is not a clone of the Mathwork's own Image Processing Toolbox (IPT) although there are many functions in common. This toolbox predated IPT by many years, is open-source, contains many functions that are useful for image feature extraction and control. It was developed under Unix and Linux systems and some functions rely on tools and utilities that exist only in that environment.

The manual is now auto-generated from the comments in the MATLAB® code itself which reduces the effort in maintaining code and a separate manual as I used to — the downside is that there are no worked examples and figures in the manual. However the book "*Robotics, Vision & Control*" provides a detailed discussion (over 600 pages, nearly 400 figures and 1000 code examples) of how to use the Toolbox functions to solve many types of problems in robotics and machine vision.

Contents

Functions by category

Color

Camera models

Image sources

Devices

Test patterns

Monadic operators

Type changing

Diadic operators

Spatial operators

Linear convolution

Kernels

Non-linear

Morphological

Similarity

Features

Region features

Line features

Point features

Other features

Multiview

Geometric

Stereo

Image sequence

Shape changing

Utility

Image utility

Image generation

Moments

Plotting

Homogeneous coordinates

3D

Integral image

Edge representation

General

Chapter 1

Introduction

1.1 What's changed

1.1.1 New features and changes to MVTB 3.4

This release represents continued evolution and refinement of the Toolbox, rather than a significant number of new features. There's been a shameful lack of releases, the last was October 2012. Points of note in this dot release include:

- The `mex` folder contains prebuilt MEX files for many platforms including 32- and 64-bit Windows, MacOS and Linux.
- Plücker coordinate class `Plucker` for describing lines in 3D.
- `idisp` has a more polished display and works with the major changes to graphics in MATLAB® 14b.
- The gamma correction function `igamma` has been renamed `igamm` since `igamma` is now a method of floating point numbers which cannot be overridden.
- `iconv` now performs correlation, not convolution. This was neccessary to ensure consistency with example in the book, but means that the function is very badly/confusingly named.
- The distance transform function `dxform` has been renamed `dtransform`.
- All Simulink models have been updated to work with the latest version of `roblocks.slx`.

For those with access to the Image Processing Toolbox (IPT) there are some alternative versions of a few MVTB functions that use IPT rather than provided MEX files. These are provided in the folder `vision/IPT` and can be copied into the `vision` folder to access that functionality.

1.2 How to obtain the Toolbox

The Machine Vision Toolbox is freely available from the Toolbox home page at

`http://www.petercorke.com`

The web page requests some information from you regarding such as your country, type of organization and application. This is just a means for me to gauge interest and to remind myself that this is a worthwhile activity.

The files are available in zip format (.zip). Download them all to the same directory and then unzip them. They all unpack to the correct parts of a hiearchy of directories (folders) headed by `rvctools`.

You may require one or more files, please read the descriptions carefully before downloading.

- `vision-3.X.zip` This file is essential, it is the core Toolbox and contains all the functions, classes, mex-files and Simulink models required for most of the RVC book.
- `images.zip` These are the images that are used for many examples in the RVC book. These images are all found automatically by the `iread()` function.
- `contrib.zip` A small number of Toolbox functions depend on third party code which is included in this file. Please note and respect the licence conditions associated with these packages. Those functions are: `igraphseg`, `imser`, and `CentralCamera.estpose`.
- `contrib2.zip` Additional third party code for the functions: `isift`, and `isurf`. Note that the code here is slightly modified version of the open-source packages.
- `images2.zip` This is a large file (150MB) containing the mosaic, campus, bridge-l and campus sequences which support the examples in Sections 14.6, 14.7 and 14.8 respectively.

If you already have the Robotics Toolbox installed then download the zip file(s) to the directory above the existing `rvctools` directory and then unzip them. The files from these zip archives will properly interleave with the Robotics Toolbox files.

Ensure that the folder `rvctools` is on your MATLAB® search path. You can do this by issuing the `addpath` command at the MATLAB® prompt. Then issue the command `startup_rvc` and it will add a number of paths to your MATLAB® search path. You need to setup the path every time you start MATLAB® but you can automate this by setting up environment variables, editing your `startup.m` script by pressing the "Update Toolbox Path Cache" button under MATLAB® General preferences.

1.2.1 Documentation

This document `vision.pdf` is a manual that describes all functions in the Toolbox. It is auto-generated from the comments in the MATLAB® code and is fully hyperlinked:

to external web sites, the table of content to functions, and the "See also" functions to each other.

The same documentation is available online in alphabetical order at http://www.petercorke.com/MVTB/r3/html/index_alpha.html or by category at http://www.petercorke.com/MVTB/r3/html/index.html.

Documentation is also available via the MATLAB® help browser, "Machine Vision Toolbox" appears under the Contents.

1.3 MATLAB version issues

The Toolbox has been tested under R2014b.

1.4 Use in teaching

This is definitely encouraged! You are free to put the PDF manual (`vision.pdf` or the web-based documentation `html/*.html` on a server for class use. If you plan to distribute paper copies of the PDF manual then every copy must include the first two pages (cover and licence).

1.5 Use in research

If the Toolbox helps you in your endeavours then I'd appreciate you citing the Toolbox when you publish. The details are

```
@book{Corke11a,
    Author = {Peter I. Corke},
    Date-Added = {2011-01-12 08:19:32 +1000},
    Date-Modified = {2012-07-29 20:07:27 +1000},
    Note = {ISBN 978-3-642-20143-1},
    Publisher = {Springer},
    Title = {Robotics, Vision \& Control: Fundamental Algorithms in {MATLAB}},
    Year = {2011}}
```

or

> P.I. Corke, Robotics, Vision & Control: Fundamental Algorithms in MATLAB. Springer, 2011. ISBN 978-3-642-20143-1.

which is also given in electronic form in the CITATION file.

1.6 Support

There is no support! This software is made freely available in the hope that you find it useful in solving whatever problems you have to hand. I am happy to correspond with

people who have found genuine bugs or deficiencies but my response time can be long and I can't guarantee that I respond to your email.

I can guarantee that I will not respond to any requests for help with assignments or homework, no matter how urgent or important they might be to you. That's what your teachers, tutors, lecturers and professors are paid to do.

You might instead like to communicate with other users via the Google Group called "Robotics and Machine Vision Toolbox"

http://groups.google.com.au/group/robotics-tool-box

which is a forum for discussion. You need to signup in order to post, and the signup process is moderated by me so allow a few days for this to happen. I need you to write a few words about why you want to join the list so I can distinguish you from a spammer or a web-bot.

1.6.1 Related software

Matlab Central http://www.mathworks.com/matlabcentral is a great resource for user contributed MATLAB code, and there are hundreds of modules available. VLFeat http://www.vlfeat.org is a great collection of advanced computer vision algorithms for MATLAB.

1.7 Acknowledgements

This release includes functions for computing image plane homographies and the fundamental matrix, contributed by Nuno Alexandre Cid Martins of I.S.R., Coimbra. RANSAC code by Peter Kovesi; pose estimation by Francesco Moreno-Noguer, Vincent Lepetit, Pascal Fua at the CVLab-EPFL; color space conversions by Pascal Getreuer; numerical routines for geometric vision by various members of the Visual Geometry Group at Oxford (from the web site of the Hartley and Zisserman book; the k-means and MSER algorithms by Andrea Vedaldi and Brian Fulkerson;the graph-based image segmentation software by Pedro Felzenszwalb; and the SURF feature detector by Dirk-Jan Kroon at U. Twente. The Camera Calibration Toolbox by Jean-Yves Bouguet is used unmodified.Functions such as SURF, MSER, graph-based segmentation and pose estimation are based on great code Some of the MEX file use some really neat macros that were part of the package VISTA Copyright 1993, 1994 University of British Columbia. See the file CONTRIB for details.

Chapter 2

Functions and classes

about

Compact display of variable type

about(**x**) displays a compact line that describes the class and dimensions of **x**.

about **x** as above but this is the command rather than functional form

Examples

```
>> a=1;
>> about a
a [double] : 1x1 (8 bytes)

>> a = rand(5,7);
>> about a
a [double] : 5x7 (280 bytes)
```

See also

whos

---

anaglyph

Convert stereo images to an anaglyph image

a = **anaglyph**(**left**, **right**) is an **anaglyph** image where the two images of a stereo pair are combined into a single image by coding them in two different colors. By default

the left image is red, and the right image is cyan.

anaglyph(**left**, **right**) as above but display the **anaglyph**.

a = **anaglyph**(**left**, **right**, **color**) as above but the string **color** describes the color coding as a string with 2 letters, the first for left, the second for right, and each is one of:

'r'	red
'g'	green
'b'	green
'c'	cyan
'm'	magenta

a = **anaglyph**(**left**, **right**, **color**, **disp**) as above but allows for disparity correction. If **disp** is positive the disparity is increased, if negative it is reduced. These adjustments are achieved by trimming the images. Use this option to make the images more natural/comfortable to view, useful if the images were captured with a stereo baseline significantly different the human eye separation (typically 65mm).

Example

Load the left and right images

```
L = iread('rocks2-l.png', 'reduce', 2);
R = iread('rocks2-r.png', 'reduce', 2);
```

then display the **anaglyph** for viewing with red-cyan glasses

```
anaglyph(L, R);
```

References

- Robotics, Vision & Control, Section 14.3, P. Corke, Springer 2011.

See also

stdisp

---

angdiff

Difference of two angles

d = **angdiff**(**th1**, **th2**) returns the difference between angles **th1** and **th2** on the circle. The result is in the interval [-pi pi). If **th1** is a column vector, and **th2** a scalar then return a column vector where **th2** is modulo subtracted from the corresponding elements of **th1**.

d = **angdiff**(**th**) returns the equivalent angle to **th** in the interval [-pi pi).

Animate

Create an animation

Helper class for creating animations. Saves snapshots of a figture as a folder of individual PNG format frames numbered 0000.png, 0001.png and so on.

Example

```
anim = Animate('movie');

for i=1:100

plot(...);
anim.add();

end
```

To convert the image files to a movie you could use a tool like ffmpeg

```
% ffmpeg -r 10 -i movie/*.png out.mp4
```

Animate.Animate

Create an animation class

a = **ANIMATE**(**name**, **options**) initializes an animation, and creates a folder called **name** to hold the individual frames.

Options

'resolution', R Set the resolution of the saved image to R pixels per inch.

Animate.add

Adds current plot to the animation

A.**ADD**() adds the current figure in PNG format to the animation folder with a unique sequential filename.

A.**ADD**(**fig**) as above but captures the figure **fig**.

See also

print

AxisWebCamera

Image from Axis webcam

A concrete subclass of ImageSource that acquires images from a web camera built by Axis Communications (www.axis.com).

Methods

grab	Aquire and return the next image
size	Size of image
close	Close the image source
char	Convert the object parameters to human readable string

See also

ImageSource, video

AxisWebCamera.AxisWebCamera

Axis web camera constructor

a = **AxisWebCamera**(**url**, **options**) is an **AxisWebCamera** object that acquires images from an Axis Communications (www.axis.com) web camera.

Options

'uint8'	Return image with uint8 pixels (default)
'float'	Return image with float pixels
'double'	Return image with double precision pixels
'grey'	Return greyscale image
'gamma', G	Apply gamma correction with gamma=G
'scale', S	Subsample the image by S in both directions.
'resolution', S	Obtain an image of size S=[W H].

Notes:

- The specified 'resolution' must match one that the camera is capable of, otherwise the result is not predictable.

AxisWebCamera.char

Convert to string

A.**char**() is a string representing the state of the camera object in human readable form.

See also

AxisWebCamera.display

AxisWebCamera.close

Close the image source

A.**close**() closes the connection to the web camera.

AxisWebCamera.grab

Acquire image from the camera

im = A.**grab**() is an image acquired from the web camera.

Notes

- Some web cameras have a fixed picture taking interval, and this function will return the most recently captured image held in the camera.

BagOfWords

Bag of words class

The BagOfWords class holds sets of features for a number of images and supports image retrieval by comparing new images with those in the 'bag'.

Methods

isword	Return all features assigned to word
occurrences	Return number of occurrences of word
remove_stop	Remove stop words
wordvector	Return word frequency vector
wordfreq	Return words and their frequencies
similarity	Compare two word bags
contains	List the images that contain a word
exemplars	Display examples of word support regions
display	Display the parameters of the bag of words
char	Convert the parameters of the bag of words to a string

Properties

K	The number of clusters specified
nstop	The number of stop words specified
nimages	The number of images in the bag

Reference

J.Sivic and A.Zisserman, "Video Google: a text retrieval approach to object matching in videos", in Proc. Ninth IEEE Int. Conf. on Computer Vision, pp.1470-1477, Oct. 2003.

See also

PointFeature

BagOfWords.BagOfWords

Create a BagOfWords object

b = **BagOfWords**(**f**, **k**) is a new bag of words created from the feature vector **f** and with **k** words. **f** can also be a cell array, as produced by ISURF() for an image sequence.

The features are sorted into **k** clusters and each cluster is termed a visual word.

b = **BagOfWords**(**f**, **b2**) is a new bag of words created from the feature vector **f** but clustered to the words (and stop words) from the existing bag **b2**.

Notes

- Uses the MEX function vl_kmeans to perform clustering (vlfeat.org).

See also

PointFeature, isurf

BagOfWords.char

Convert to string

s = B.**char**() is a compact string representation of a bag of words.

BagOfWords.contains

Find images containing word

k = B.**contains**(**w**) is a vector of the indices of images in the sequence that contain one or more instances of the word **w**.

BagOfWords.display

Display value

B.**display**() displays the parameters of the bag in a compact human readable form.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a BagOfWords object and the command has no trailing semicolon.

See also

BagOfWords.char

BagOfWords.exemplars

display exemplars of words

B.**exemplars**(**w**, **images**, **options**) displays examples of the support regions of the words specified by the vector **w**. The examples are displayed as a table of thumbnail images. The original sequence of images from which the features were extracted must be provided as **images**.

Options

'ncolumns', N	Number of columns to display (default 10)
'maxperimage', M	Maximum number of **exemplars** to display from any one image (default 2)
'width', **w**	Width of each thumbnail [pixels] (default 50)

BagOfWords.isword

Features from words

f = B.**isword**(**w**) is a vector of feature objects that are assigned to any of the word **w**. If **w** is a vector of words the result is a vector of features assigned to all the words in **w**.

BagOfWords.occurrence

Word occurrence

n = B.**occurrence**(**w**) is the number of occurrences of the word **w** across all features in the bag.

BagOfWords.remove_stop

Remove stop words

B.**remove_stop**(**n**) removes the **n** most frequent words (the stop words) from the bag. All remaining words are renumbered so that the word labels are consecutive.

BagOfWords.wordfreq

Word frequency statistics

[**w**,**n**] = B.**wordfreq**() is a vector of word labels **w** and the corresponding elements of **n** are the number of occurrences of that word.

BagOfWords.wordvector

Word frequency vector

wf = B.**wordvector**(**J**) is the word frequency vector for the **J**'th image in the bag. The vector is $K \times 1$ and the angle between any two WFVs is an indication of image similarity.

Notes

- The word vector is expensive to compute so a lazy evaluation is performed on the first call to this function

---

blackbody

Compute blackbody emission spectrum

E = **blackbody**(**lambda**, **T**) is the **blackbody** radiation power density [W/m^3] at the wavelength **lambda** [m] and temperature **T** [K].

If **lambda** is a column vector ($N \times 1$), then **E** is a column vector ($N \times 1$) of **blackbody** radiation power density at the corresponding elements of **lambda**.

Example

```
l = [380:10:700]'*1e-9; % visible spectrum
e = blackbody(l, 6500); % emission of sun
plot(l, e)
```

References

- Robotics, Vision & Control, Section 10.1, P. Corke, Springer 2011.

---

boundmatch

Match boundary profiles

x = **boundmatch**(**R1**, **r2**) is the correlation of the two boundary profiles **R1** and **r2**. Each is an $N \times 1$ vector of distances from the centroid of an object to points on its perimeter at equal angular increments spanning 2pi radians. **x** is also $N \times 1$ and is a correlation whose peak indicates the relative orientation of one profile with respect to the other.

[**x**,**s**] = **boundmatch**(**R1**, **r2**) as above but also returns the relative scale **s** which is the size of object 2 with respect to object 1.

Notes

- Can be considered as matching two functions defined over **s**(1).

See also

RegionFeature.boundary, xcorr

---

bresenham

Generate a line

p = **bresenham**(**x1**, **y1**, **x2**, **y2**) is a list of integer coordinates ($2 \times N$) for points lying on the line segement (**x1**,**y1**) to (**x2**,**y2**).

p = **bresenham**(**p1**, **p2**) as above but **p1**=[**x1**,**y1**] and **p2**=[**x2**,**y2**].

Notes

- Endpoints must be integer values.

See also

icanvas

---

camcald

Camera calibration from data points

C = **camcald**(**d**) is the camera matrix (3×4) determined by least squares from corresponding world and image-plane points. **d** is a table of points with rows of the form [X Y Z U V] where (X,Y,Z) is the coordinate of a world point and [U,V] is the corresponding image plane coordinate.

[**C**,**E**] = **camcald**(**d**) as above but **E** is the maximum residual error after back substitution [pixels].

Notes:

- This method assumes no lense distortion affecting the image plane coordinates.

See also

CentralCamera

Camera

Camera superclass

An abstract superclass for Toolbox camera classes.

Methods

plot	plot projection of world point to image plane
hold	control figure hold for image plane window
ishold	test figure hold for image plane
clf	clear image plane
figure	figure holding the image plane
mesh	draw shape represented as a mesh
point	draw homogeneous points on image plane
homline	draw homogeneous lines on image plane
lineseg	draw line segment defined by points
plot_camera	draw camera in world view
rpy	set camera attitude
move	clone Camera after motion
centre	get world coordinate of camera centre
delete	object destructor
char	convert camera parameters to string
display	display camera parameters

Properties (read/write)

npix	image dimensions (2×1)
pp	principal point (2×1)
rho	pixel dimensions (2×1) in metres
T	camera pose as homogeneous transformation

Properties (read only)

nu	number of pixels in u-direction
nv	number of pixels in v-direction
u0	principal point u-coordinate
v0	principal point v-coordinate

Notes

- Camera is a reference object.
- Camera objects can be used in vectors and arrays
- This is an abstract class and must be subclassed and a project() method defined.
- The object can create a window to display the Camera image plane, this window is protected and can only be accessed by the plot methods of this object.

Camera.Camera

Create camera object

Constructor for abstact **Camera** class, used by all subclasses.

C = **Camera**(**options**) creates a default (abstract) camera with null parameters.

Options

'name', N	Name of camera
'image', IM	Load image IM to image plane
'resolution', N	Image plane resolution: $N \times N$ or N=[W H]
'sensor', S	Image sensor size in metres (2×1) [metres]
'centre', P	Principal point (2×1)
'pixel', S	Pixel size: $S \times S$ or S=[W H]
'noise', SIGMA	Standard deviation of additive Gaussian noise added to returned image projections
'pose', T	Pose of the camera as a homogeneous transformation
'color', **C**	Color of image plane background (default [1 1 0.8])

Notes

- Normally the class plots points and lines into a set of axes that represent the image plane. The 'image' option paints the specified image onto the image plane and allows points and lines to be overlaid.

See also

CentralCamera, fisheyecamera, CatadioptricCamera, SphericalCamera

---

Camera.centre

Get camera position

p = C.**centre**() is the 3-dimensional position of the camera **centre** (3×1).

Camera.char

Convert to string

s = C.**char**() is a compact string representation of the camera parameters.

Camera.clf

Clear the image plane

C.**clf**() removes all graphics from the camera's image plane.

Camera.delete

Camera object destructor

C.**delete**() destroys all figures associated with the **Camera** object and removes the object.

Camera.display

Display value

C.**display**() displays a compact human-readable representation of the camera parameters.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a Camera object and the command has no trailing semicolon.

See also

Camera.char

Camera.figure

Return figure handle

H = C.**figure**() is the handle of the **figure** that contains the camera's image plane graphics.

Camera.hold

Control hold on image plane graphics

C.**hold**() sets "**hold** on" for the camera's image plane.

C.**hold**(**H**) **hold** mode is set on if **H** is true (or > 0), and off if **H** is false (or 0).

Camera.homline

Plot homogeneous lines on image plane

C.**homline**(**L**) plots lines on the camera image plane which are defined by columns of **L** $(3 \times N)$ considered as lines in homogeneous form: a.u + b.v + c = 0.

Camera.ishold

Return image plane hold status

H = C.**ishold**() returns true (1) if the camera's image plane is in hold mode, otherwise false (0).

Camera.lineseg

handle for this camera image plane

Camera.mesh

Plot mesh object on image plane

C.**mesh**(**x**, **y**, **z**, **options**) projects a 3D shape defined by the matrices **x**, **y**, **z** to the image plane and plots them. The matrices **x**, **y**, **z** are of the same size and the corresponding elements of the matrices define 3D points.

Options

'Tobj', T	Transform all points by the homogeneous transformation T before projecting them to the camera image plane.
'Tcam', T	Set the camera pose to the homogeneous transformation T before projecting points to the camera image plane. Temporarily overrides the current camera pose C.T.

Additional arguments are passed to plot as line style parameters.

See also

mesh, cylinder, sphere, mkcube, Camera.plot, Camera.hold, Camera.clf

---

Camera.move

Instantiate displaced camera

C2 = C.**move**(**T**) is a new camera object that is a clone of C but its pose is displaced by the homogeneous transformation **T** with respect to the current pose of C.

Camera.plot

Plot points on image plane

C.**plot**(**p**, **options**) projects world points **p** ($3 \times N$) to the image plane and plots them. If **p** is $2 \times N$ the points are assumed to be image plane coordinates and are plotted directly.

uv = C.**plot**(**p**) as above but returns the image plane coordinates **uv** ($2 \times N$).

- If **p** has 3 dimensions ($3 \times N \times S$) then it is considered a sequence of point sets and is displayed as an animation.

C.**plot**(**L**, **options**) projects the world lines represented by the array of Plucker objects ($1 \times N$) to the image plane and plots them.

li = C.**plot**(**L**, **options**) as above but returns an array ($3 \times N$) of image plane lines in homogeneous form.

Options

'Tobj', T	Transform all points by the homogeneous transformation T before projecting them to the camera image plane.
'Tcam', T	Set the camera pose to the homogeneous transformation T before projecting points to the camera image plane. Overrides the current camera pose C.T.
'fps', N	Number of frames per second for point sequence display
'sequence'	Annotate the points with their index
'textcolor', C	Text color for annotation (default black)
'textsize', S	Text size for annotation (default 12)
'drawnow'	Execute MATLAB drawnow function

Additional options are considered MATLAB linestyle parameters and are passed directly to **plot**.

See also

Camera.mesh, Camera.hold, Camera.clf, Plucker

Camera.plot_camera

Display camera icon in world view

C.**plot_camera**(**options**) draw a camera as a simple 3D model in the current figure.

Options

'Tcam', T	Camera displayed in pose T (homogeneous transformation 4×4)
'scale', S	Overall scale factor (default 0.2 x maximum axis dimension)
'color', C	Camera body color (default blue)
'frustrum'	Draw the camera as a frustrum (pyramid mesh)
'solid'	Draw a non-frustrum camera as a solid (default)
'mesh'	Draw a non-frustrum camera as a mesh
'label'	Show the camera's name next to the camera

Notes

- The graphic handles are stored within the Camera object.
- A line between the red faces is parallel to the x-axis, between the green faces is parallel to the y-axis.

Camera.point

Plot homogeneous points on image plane

C.**point**(**p**) plots points on the camera image plane which are defined by columns of **p** ($3 \times N$) considered as points in homogeneous form.

Camera.rpy

Set camera attitude

C.**rpy**(**R**, **p**, **y**) sets the camera attitude to the specified roll-pitch-yaw angles.

C.**rpy**(**rpy**) as above but **rpy**=[**R**,**p**,**y**].

CatadioptricCamera

Catadioptric camera class

A concrete class for a catadioptric camera, subclass of Camera.

Methods

project	project world points to image plane
plot	plot/return world point on image plane
hold	control hold for image plane
ishold	test figure hold for image plane
clf	clear image plane
figure	figure holding the image plane
mesh	draw shape represented as a mesh
point	draw homogeneous points on image plane
line	draw homogeneous lines on image plane
plot_camera	draw camera
rpy	set camera attitude
move	copy of Camera after motion
centre	get world coordinate of camera centre
delete	object destructor
char	convert camera parameters to string
display	display camera parameters

Properties (read/write)

npix	image dimensions in pixels (2×1)
pp	intrinsic: principal point (2×1)
rho	intrinsic: pixel dimensions (2×1) [metres]
f	intrinsic: focal length [metres]
p	intrinsic: tangential distortion parameters
T	extrinsic: camera pose as homogeneous transformation

Properties (read only)

nu	number of pixels in u-direction
nv	number of pixels in v-direction
u0	principal point u-coordinate
v0	principal point v-coordinate

Notes

- Camera is a reference object.
- Camera objects can be used in vectors and arrays

See also

CentralCamera, Camera

CatadioptricCamera.CatadioptricCamera

Create central projection camera object

C = **CatadioptricCamera**() creates a central projection camera with canonic parameters: f=1 and name='canonic'.

C = **CatadioptricCamera**(**options**) as above but with specified parameters.

Options

'name', N	Name of camera
'focal', F	Focal length (metres)
'default'	Default camera parameters: 1024×1024, f=8mm, 10um pixels, camera at origin, optical axis is z-axis, u- and v-axes parallel to x- and y-axes respectively.
'projection', M	Catadioptric model: 'equiangular' (default), 'sine', 'equisolid', 'stereographic'
'k', K	Parameter for the projection model
'maxangle', A	The maximum viewing angle above the horizontal plane.
'resolution', N	Image plane resolution: $N \times N$ or N=[W H].
'sensor', S	Image sensor size in metres (2×1)
'centre', P	Principal point (2×1)
'pixel', S	Pixel size: $S \times S$ or S=[W H].
'noise', SIGMA	Standard deviation of additive Gaussian noise added to returned image projections
'pose', T	Pose of the camera as a homogeneous transformation

Notes

- The elevation angle range is from -pi/2 (below the mirror) to maxangle above the horizontal plane.

See also

Camera, fisheyecamera, CatadioptricCamera, SphericalCamera

CatadioptricCamera.project

Project world points to image plane

uv = C.**project**(**p**, **options**) are the image plane coordinates for the world points **p**. The columns of **p** ($3 \times N$) are the world points and the columns of **uv** ($2 \times N$) are the corresponding image plane points.

Options

'Tobj', T	Transform all points by the homogeneous transformation T before projecting them to the camera image plane.
'Tcam', T	Set the camera pose to the homogeneous transformation T before projecting points to the camera image plane. Temporarily overrides the current camera pose C.T.

See also

Camera.plot

---

ccdresponse

CCD spectral response

R = **ccdresponse**(**lambda**) is the spectral response of a typical silicon imaging sensor at the wavelength **lambda** [m]. The response is normalized in the range 0 to 1. If **lambda** is a vector then **R** is a vector of the same length whose elements are the response at the corresponding element of **lambda**.

Notes

- Deprecated, use loadspectrum(lambda, 'ccd') instead.

References

- An ancient Fairchild data book for a silicon sensor.
- Robotics, Vision & Control, Section 10.2, P. Corke, Springer 2011.

See also

rluminos

---

ccodefunctionstring

Converts a symbolic expression into a C-code function

[**funstr**, **hdrstr**] = **ccodefunctionstring**(**symexpr**, **arglist**) returns a string representing a C-code implementation of a symbolic expression **symexpr**. The C-code implementation has a signature of the form:

```
void funname(double[][n_o] out, const double in1,
const double* in2, const double[][n_i] in3);
```


depending on the number of inputs to the function as well as the dimensionality of the inputs (n_i) and the output (n_o). The whole C-code implementation is returned in **funstr**, while **hdrstr** contains just the signature ending with a semi-colon (for the use in header files).

Options

'funname', name	Specify the name of the generated C-function. If this optional argument is omitted, the variable name of the first input argument is used, if possible.
'output', outVar	Defines the identifier of the output variable in the C-function.
'vars', varCells	The inputs to the C-code function must be defined as a cell array. The elements of this cell array contain the symbolic variables required to compute the output. The elements may be scalars, vectors or matrices symbolic variables. The C-function prototype will be composed accoringly as exemplified above.
'flag', sig	Specifies if function signature only is generated, default (false).

Example

```
% Create symbolic variables
syms q1 q2 q3

Q = [q1 q2 q3];
% Create symbolic expression
myrot = rotz(q3)*roty(q2)*rotx(q1)

% Generate C-function string
[funstr, hdrstr] = ccodefunctionstring(myrot,'output','foo', ...
'vars',{Q},'funname','rotate_xyz')
```

Notes

- The function wraps around the built-in Matlab function 'ccode'. It does not check for proper C syntax. You must take care of proper dimensionality of inputs and outputs with respect to your symbolic expression on your own. Otherwise the generated C-function may not compile as desired.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

ccode, matlabfunction

ccxyz

XYZ chromaticity coordinates

xyz = **ccxyz**(**lambda**) is the xyz-chromaticity coordinates (3×1) for illumination at wavelength **lambda**. If **lambda** is a vector ($N \times 1$) then each row of **xyz** ($N \times 3$) is the **xyz**-chromaticity of the corresponding element of **lambda**.

xyz = **ccxyz**(**lambda**, **E**) is the xyz-chromaticity coordinates ($N \times 3$) for an illumination spectrum **E** ($N \times 1$) defined at corresponding wavelengths **lambda** ($N \times 1$).

References

- Robotics, Vision & Control, Section 10.2, P. Corke, Springer 2011.

See also

cmfxyz

CentralCamera

Perspective camera class

A concrete class for a central-projection perspective camera, a subclass of Camera.

The camera coordinate system is:

```
0------------> u X
|
|
|   + (principal point)
|
|   Z-axis is into the page.
v Y
```

This camera model assumes central projection, that is, the focal point is at z=0 and the image plane is at z=f. The image is not inverted.

Methods

project	project world points and lines
K	camera intrinsic matrix
C	camera matrix
H	camera motion to homography
invH	decompose homography
F	camera motion to fundamental matrix
E	camera motion to essential matrix
invE	decompose essential matrix
fov	field of view
ray	Ray3D corresponding to point
centre	projective centre
plot	plot projection of world point on image plane
hold	control hold for image plane
ishold	test figure hold for image plane
clf	clear image plane
figure	figure holding the image plane
mesh	draw shape represented as a mesh
point	draw homogeneous points on image plane
line	draw homogeneous lines on image plane
plot_camera	draw camera in world view
plot_line_tr	draw line in theta/rho format
plot_epiline	draw epipolar line
flowfield	compute optical flow
visjac_p	image Jacobian for point features
visjac_p_polar	image Jacobian for point features in polar coordinates
visjac_l	image Jacobian for line features
visjac_e	image Jacobian for ellipse features
rpy	set camera attitude
move	clone Camera after motion
centre	get world coordinate of camera centre
estpose	estimate pose
delete	object destructor
char	convert camera parameters to string
display	display camera parameters

Properties (read/write)

npix	image dimensions in pixels (2×1)
pp	intrinsic: principal point (2×1)
rho	intrinsic: pixel dimensions (2×1) in metres
f	intrinsic: focal length
k	intrinsic: radial distortion vector
p	intrinsic: tangential distortion parameters
distortion	intrinsic: camera distortion [k1 k2 k3 p1 p2]
T	extrinsic: camera pose as homogeneous transformation

Properties (read only)

nu	number of pixels in u-direction
nv	number of pixels in v-direction
u0	principal point u-coordinate
v0	principal point v-coordinate

Notes

- Camera is a reference object.
- Camera objects can be used in vectors and arrays

See also

Camera

CentralCamera.CentralCamera

Create central projection camera object

C = **CentralCamera**() creates a central projection camera with canonic parameters: f=1 and name='canonic'.

C = **CentralCamera**(**options**) as above but with specified parameters.

Options

'name', N	Name of camera
'focal', F	Focal length [metres]
'distortion', D	Distortion vector [k1 k2 k3 p1 p2]
'distortion-bouguet', D	Distortion vector [k1 k2 p1 p2 k3]
'default'	Default camera parameters: 1024×1024, f=8mm, 10um pixels, camera at origin, optical axis is z-axis, u- and v-axes parallel to x- and y-axes respectively.
'image', IM	Display an image rather than points
'resolution', N	Image plane resolution: $N \times N$ or N=[W H]
'sensor', S	Image sensor size in metres (2×1)
'centre', P	Principal point (2×1)
'pixel', S	Pixel size: $S \times S$ or S=[W H]
'noise', SIGMA	Standard deviation of additive Gaussian noise added to returned image projections
'pose', T	Pose of the camera as a homogeneous transformation
'color', **C**	Color of image plane background (default [1 1 0.8])

See also

Camera, fisheyecamera, CatadioptricCamera, SphericalCamera

CentralCamera.C

Camera matrix

C = **C**.**C**() is the 3×4 camera matrix, also known as the camera calibration or projection matrix.

CentralCamera.centre

Projective centre

p = C.**centre**() returns the 3D world coordinate of the projective **centre** of the camera.

Reference

Hartley & Zisserman, "Multiview Geometry",

See also

Ray3D

CentralCamera.E

Essential matrix

E = C.**E**(**T**) is the essential matrix relating two camera views. The first view is from the current camera pose C.**T** and the second is a relative motion represented by the homogeneous transformation **T**.

E = C.**E**(**C2**) is the essential matrix relating two camera views described by camera objects C (first view) and **C2** (second view).

E = C.**E**(**f**) is the essential matrix based on the fundamental matrix **f** (3×3) and the intrinsic parameters of camera C.

Reference

Y.Ma, J.Kosecka, S.Soatto, S.Sastry, "An invitation to 3D", Springer, 2003. p.177

See also

CentralCamera.F, CentralCamera.invE

CentralCamera.estpose

Estimate pose from object model and camera view

T = C.**estpose**(**xyz**, **uv**) is an estimate of the pose of the object defined by coordinates **xyz** ($3 \times N$) in its own coordinate frame. **uv** ($2 \times N$) are the corresponding image plane coordinates.

Reference

"EPnP: An accurate O(n) solution to the PnP problem", V. Lepetit, F. Moreno-Noguer, and P. Fua, Int. Journal on Computer Vision, vol. 81, pp. 155-166, Feb. 2009.

CentralCamera.F

Fundamental matrix

F = C.**F**(**T**) is the fundamental matrix relating two camera views. The first view is from the current camera pose C.**T** and the second is a relative motion represented by the homogeneous transformation **T**.

F = C.**F**(**C2**) is the fundamental matrix relating two camera views described by camera objects C (first view) and **C2** (second view).

Reference

Y.Ma, J.Kosecka, S.Soatto, S.Sastry, "An invitation to 3D", Springer, 2003. p.177

See also

CentralCamera.E

CentralCamera.flowfield

Optical flow

C.**flowfield**(**v**) displays the optical flow pattern for a sparse grid of points when the camera has a spatial velocity **v** (6×1).

See also

quiver

CentralCamera.fov

Camera field-of-view angles.

a = C.**fov**() are the field of view angles (2×1) in radians for the camera x and y (horizontal and vertical) directions.

CentralCamera.H

Homography matrix

H = C.**H**(**T**, **n**, **d**) is a 3×3 homography matrix for the camera observing the plane with normal **n** and at distance **d**, from two viewpoints. The first view is from the current camera pose C.**T** and the second is after a relative motion represented by the homogeneous transformation **T**.

See also

CentralCamera.H

CentralCamera.invE

Decompose essential matrix

s = C.**invE**(**E**) decomposes the essential matrix **E** (3×3) into the camera motion. In practice there are multiple solutions and **s** $(4 \times 4 \times N)$ is a set of homogeneous transformations representing possible camera motion.

s = C.**invE**(**E**, **p**) as above but only solutions in which the world point **p** is visible are returned.

Reference

Hartley & Zisserman, "Multiview Geometry", Chap 9, p. 259

Y.Ma, J.Kosecka, **s**.Soatto, **s**.Sastry, "An invitation to 3D", Springer, 2003. p116, p120-122

Notes

- The transformation is from view 1 to view 2.

See also

CentralCamera.E

CentralCamera.invH

Decompose homography matrix

s = C.**invH**(**H**) decomposes the homography **H** (3×3) into the camera motion and the normal to the plane.

In practice there are multiple solutions and **s** is a vector of structures with elements:

- T, camera motion as a homogeneous transform matrix (4×4), translation not to scale
- n, normal vector to the plane (3×3)

Notes

- There are up to 4 solutions
- Only those solutions that obey the positive depth constraint are returned
- The required camera intrinsics are taken from the camera object
- The transformation is from view 1 to view 2.

Reference

Y.Ma, J.Kosecka, **s**.Soatto, **s**.Sastry, "An invitation to 3D", Springer, 2003. section 5.3

See also

CentralCamera.H

CentralCamera.K

Intrinsic parameter matrix

K = C.**K**() is the 3×3 intrinsic parameter matrix.

CentralCamera.plot_epiline

Plot epipolar line

C.**plot_epiline**(**f**, **p**) plots the epipolar lines due to the fundamental matrix **f** and the image points **p**.

C.**plot_epiline**(**f**, **p**, **ls**) as above but draw lines using the line style arguments **ls**.

H = C.**plot_epiline**(**f**, **p**) as above but return a vector of graphic handles, one per line.

CentralCamera.plot_line_tr

Plot line in theta-rho format

CentralCamera.**plot_line_tr**(**L**) plots lines on the camera's image plane that are described by columns of **L** with rows theta and rho respectively.

See also

Hough

CentralCamera.project

Project world points to image plane

uv = C.**project**(**p**, **options**) are the image plane coordinates $(2 \times N)$ corresponding to the world points **p** $(3 \times N)$.

- If Tcam ($4 \times 4 \times S$) is a transform sequence then **uv** ($2 \times N \times S$) represents the sequence of projected points as the camera moves in the world.
- If Tobj ($4 \times 4 \times S$) is a transform sequence then **uv** ($2 \times N \times S$) represents the sequence of projected points as the object moves in the world.

[**uv**,**vis**] = C.**project**(**p**, **options**) as above but **vis** ($S \times N$) is a logical matrix with elements true (1) if the point is visible, that is, it lies within the bounds of the image plane and is in front of the camera.

L = C.**project**(**L**, **options**) are the image plane homogeneous lines ($3 \times N$) corresponding to the world lines represented by a vector of Plucker coordinates ($1 \times N$).

Options

'Tobj', T	Transform all points by the homogeneous transformation T before projecting them to the camera image plane.
'Tcam', T	Set the camera pose to the homogeneous transformation T before projecting points to the camera image plane. Temporarily overrides the current camera pose C.T.

Notes

- Currently a camera or object pose sequence is not supported for the case of line projection.
- (u,v) values are set to NaN if the corresponding point is behind the camera.

See also

Camera.plot, Plucker

CentralCamera.ray

3D ray for image point

R = C.**ray**(**p**) returns a vector of Ray3D objects, one for each point defined by the columns of **p**.

Reference

Hartley & Zisserman, "Multiview Geometry", p 162

See also

Ray3D

CentralCamera.visjac_e

Visual motion Jacobian for point feature

J = C.**visjac_e**(**E**, **pl**) is the image Jacobian (5×6) for the ellipse **E** (5×1) described by u^2 + E1v^2 - 2E2uv + 2E3u + 2E4v + E5 = 0. The ellipse lies in the world plane **pl** = (a,b,c,d) such that aX + bY + cZ + d = 0.

The Jacobian gives the rates of change of the ellipse parameters in terms of camera spatial velocity.

Reference

B. Espiau, F. Chaumette, and P. Rives, "A New Approach to Visual Servoing in Robotics", IEEE Transactions on Robotics and Automation, vol. 8, pp. 313-326, June 1992.

See also

CentralCamera.visjac_p, CentralCamera.visjac_p_polar, CentralCamera.visjac_l

---

CentralCamera.visjac_l

Visual motion Jacobian for line feature

J = C.**visjac_l**(**L**, **pl**) is the image Jacobian ($2N \times 6$) for the image plane lines **L** ($2 \times N$). Each column of **L** is a line in theta-rho format, and the rows are theta and rho respectively.

The lines all lie in the plane **pl** = (a,b,c,d) such that aX + bY + cZ + d = 0.

The Jacobian gives the rates of change of the line parameters in terms of camera spatial velocity.

Reference

B. Espiau, F. Chaumette, and P. Rives, "A New Approach to Visual Servoing in Robotics", IEEE Transactions on Robotics and Automation, vol. 8, pp. 313-326, June 1992.

See also

CentralCamera.visjac_p, CentralCamera.visjac_p_polar, CentralCamera.visjac_e

---

CentralCamera.visjac_p

Visual motion Jacobian for point feature

J = C.**visjac_p**(**uv**, **z**) is the image Jacobian $(2N \times 6)$ for the image plane points **uv** $(2 \times N)$. The depth of the points from the camera is given by **z** which is a scalar for all points, or a vector $(N \times 1)$ of depth for each point.

The Jacobian gives the image-plane point velocity in terms of camera spatial velocity.

Reference

"A tutorial on Visual Servo Control", Hutchinson, Hager & Corke, IEEE Trans. R&A, Vol 12(5), Oct, 1996, pp 651-670.

See also

CentralCamera.visjac_p_polar, CentralCamera.visjac_l, CentralCamera.visjac_e

---

CentralCamera.visjac_p_polar

Visual motion Jacobian for point feature

J = C.**visjac_p_polar**(**rt**, **z**) is the image Jacobian $(2N \times 6)$ for the image plane points **rt** $(2 \times N)$ described in polar form, radius and theta. The depth of the points from the camera is given by **z** which is a scalar for all point, or a vector $(N \times 1)$ of depths for each point.

The Jacobian gives the image-plane polar point coordinate velocity in terms of camera spatial velocity.

Reference

"Combining Cartesian and polar coordinates in IBVS", P. I. Corke, F. Spindler, and F. Chaumette, in Proc. Int. Conf on Intelligent Robots and Systems (IROS), (St. Louis), pp. 5962-5967, Oct. 2009.

See also

CentralCamera.visjac_p, CentralCamera.visjac_l, CentralCamera.visjac_e

---

cie_primaries

Define CIE primary colors

p = **cie_primaries**() is a 3-vector with the wavelengths [m] of the CIE 1976 red, green and blue primaries respectively.

---

circle

Compute points on a circle

circle(**C**, **R**, **opt**) plots a **circle** centred at **C** (1×2) with radius **R** on the current axes.

x = **circle**(**C**, **R**, **opt**) is a matrix $(2 \times N)$ whose columns define the coordinates [x,y] of points around the circumferance of a **circle** centred at **C** (1×2) and of radius **R**.

C is normally 2×1 but if 3×1 then the **circle** is embedded in 3D, and **x** is $N \times 3$, but the **circle** is always in the xy-plane with a z-coordinate of **C**(3).

Options

'n', N	Specify the number of points (default 50)

---

closest

Find closest points in N-dimensional space.

k = **closest**(**a**, **b**) is the correspondence for N-dimensional point sets **a** $(N \times NA)$ and **b** $(N \times NB)$. **k** (1 x NA) is such that the element J = **k**(I), that is, that the I'th column of **a** is **closest** to the Jth column of **b**.

[**k**,**d1**] = **closest**(**a**, **b**) as above and **d1**(I)=—**a**(I)-**b**(J)— is the distance of the **closest** point.

[**k**,**d1**,**d2**] = **closest**(**a**, **b**) as above but also returns the distance to the second **closest** point.

Notes

- Is a MEX file.

See also

distance

---

cmfrgb

RGB color matching function

The color matching function is the RGB tristimulus required to match a particular spectral excitation.

rgb = **cmfrgb**(**lambda**) is the CIE color matching function ($N \times 3$) for illumination at wavelength **lambda** ($N \times 1$) [m]. If **lambda** is a vector then each row of **rgb** is the color matching function of the corresponding element of **lambda**.

rgb = **cmfrgb**(**lambda**, **E**) is the CIE color matching (1×3) function for an illumination spectrum **E** ($N \times 1$) defined at corresponding wavelengths **lambda** ($N \times 1$).

Notes

- Data from http://cvrl.ioo.ucl.ac.uk
- From Table I(5.5.3) of Wyszecki & Stiles (1982). (Table 1(5.5.3) of Wyszecki & Stiles (1982) gives the Stiles & Burch functions in 250 cm-1 steps, while Table I(5.5.3) of Wyszecki & Stiles (1982) gives them in interpolated 1 nm steps.)
- The Stiles & Burch 2-deg CMFs are based on measurements made on 10 observers. The data are referred to as pilot data, but probably represent the best estimate of the 2 deg CMFs, since, unlike the CIE 2 deg functions (which were reconstructed from chromaticity data), they were measured directly.
- These CMFs differ slightly from those of Stiles & Burch (1955). As noted in footnote a on p. 335 of Table 1(5.5.3) of Wyszecki & Stiles (1982), the CMFs have been ”corrected in accordance with instructions given by Stiles & Burch (1959)” and renormalized to primaries at 15500 (645.16), 19000 (526.32), and 22500 (444.44) cm-1

References

- Robotics, Vision & Control, Section 10.2, P. Corke, Springer 2011.

See also

cmfxyz, ccxyz

---

cmfxyz

matching function

The color matching function is the XYZ tristimulus required to match a particular wavelength excitation.

xyz = **cmfxyz**(**lambda**) is the CIE **xyz** color matching function ($N \times 3$) for illumination at wavelength **lambda** ($N \times 1$) [m]. If **lambda** is a vector then each row of **xyz** is the color matching function of the corresponding element of **lambda**.

xyz = **cmfxyz**(**lambda**, **E**) is the CIE **xyz** color matching (1×3) function for an illumination spectrum **E** ($N \times 1$) defined at corresponding wavelengths **lambda** ($N \times 1$).

Note

- CIE 1931 2-deg **xyz** CMFs from cvrl.ioo.ucl.ac.uk

References

- Robotics, Vision & Control, Section 14.3, P. Corke, Springer 2011.

See also

cmfrgb, ccxyz

---

col2im

Convert pixel vector to image

out = **col2im**(**pix**, **imsize**) is an image ($H \times W \times P$) comprising the pixel values in **pix** ($N \times P$) with one row per pixel where N=$H \times W$. **imsize** is a 2-vector (N,M).

out = **col2im**(**pix**, **im**) as above but the dimensions of **out** are the same as **im**.

Notes

- The number of rows in **pix** must match the product of the elements of **imsize**.

See also

im2col

---

colnorm

Column-wise norm of a matrix

cn = **colnorm**(**a**) is vector ($1 \times M$) of the normals of each column of the matrix **a** ($N \times M$).

---

colordistance

Colorspace distance

d = **colordistance**(**im**, **rg**) is the Euclidean distance on the rg-chromaticity plane from coordinate **rg**=[r,g] to every pixel in the color image **im**. **d** is an image with the same dimensions as **im** and the value of each pixel is the color space distance of the corresponding pixel in **im**.

Notes

- The output image could be thresholded to determine color similarity.

- Note that Euclidean distance in the rg-chromaticity space does not correspond well with human perception of color differences. Perceptually uniform spaces such as Lab remedy this problem.

See also

colorspace

---

colorize

Colorize a greyscale image

out = **colorize**(**im**, **mask**, **color**) is a color image where each pixel in **out** is set to the corresponding element of the greyscale image **im** or a specified **color** according to whether the corresponding value of **mask** is true or false respectively. The color is specified as a 3-vector (R,G,B).

out = **colorize**(**im**, **func**, **color**) as above but a the mask is the return value of the function handle **func** applied to the image **im**, and returns a per-pixel logical result, eg. @isnan.

Examples

Display image with values < 100 in blue

```
out = colorize(im, im<100, [0 0 1])
```

Display image with NaN values shown in red

```
out = colorize(im, @isnan, [1 0 0])
```

Notes

- With no output arguments the image is displayed.

See also

imono, icolor, ipixswitch

---

colorkmeans

Color image segmentation by clustering

L = **colorkmeans**(**im**, **k**, **options**) is a segmentation of the color image **im** into **k** classes. The label image **L** has the same row and column dimension as **im** and each pixel has a value in the range 0 to **k**-1 which indicates which cluster the corresponding pixel belongs to. A k-means clustering of the chromaticity of all input pixels is performed.

[**L**,**C**] = **colorkmeans**(**im**, **k**) as above but also returns the cluster centres **C** (**k** $\times$ 2) where the I'th row is the rg-chromaticity of the I'th cluster and corresponds to the label I. A k-means clustering of the chromaticity of all input pixels is performed.

[**L**,**C**,**R**] = **colorkmeans**(**im**, **k**) as above but also returns the residual **R**, the root mean square error of all pixel chromaticities with respect to their cluster centre.

L = **colorkmeans**(**im**, **C**) is a segmentation of the color image **im** into **k** classes which are defined by the cluster centres **C** (**k** $\times$ 2) in chromaticity space. Pixels are assigned to the closest (Euclidean) centre. Since cluster centres are provided the k-means segmentation step is not required.

Options

Various options are possible to choose the initial cluster centres for k-means:

'random'	randomly choose **k** points from
'spread'	randomly choose **k** values within the rectangle spanned by the input chromaticities.
'pick'	interactively pick cluster centres

Notes

- The k-means clustering algorithm used in the first three forms is computationally expensive and time consuming.
- Clustering is performed in xy-chromaticity space.
- The residual is an indication of quality of fit, low is good.

See also

rgb2xyz, kmeans

colorname

Map between color names and RGB values

rgb = **colorname**(**name**) is the **rgb**-tristimulus value (1×3) corresponding to the color specified by the string **name**. If **rgb** is a cell-array ($1 \times N$) of names then **rgb** is a matrix ($N \times 3$) with each row being the corresponding tristimulus.

XYZ = **colorname**(**name**, 'xyz') as above but the XYZ-tristimulus value corresponding to the color specified by the string **name**.

XY = **colorname**(**name**, 'xy') as above but the xy-chromaticity coordinates corresponding to the color specified by the string **name**.

name = **colorname**(**rgb**) is a string giving the name of the color that is closest (Euclidean) to the given **rgb**-tristimulus value (1×3). If **rgb** is a matrix ($N \times 3$) then return a cell-array ($1 \times N$) of color names.

name = **colorname**(XYZ, 'xyz') as above but the color is the closest (Euclidean) to the given XYZ-tristimulus value.

name = **colorname**(XYZ, 'xy') as above but the color is the closest (Euclidean) to the given xy-chromaticity value with assumed Y=1.

Notes

- Color name may contain a wildcard, eg. "?burnt"
- Based on the standard X11 color database rgb.txt.
- Tristimulus values are in the range 0 to 1

---

colorseg

Color image segmentation using k-means

THIS FUNCTION IS DEPRECATED, USE COLORKMEANS INSTEAD

Notes

- deprecated. Use COLORKMEANS instead.

See also

colorkmeans

---

colorspace

Color space conversion of image

out = **colorspace**(**s**, **im**) converts the image **im** to a different color space according to the string **s** which specifies the source and destination color spaces, **s** = 'dest<-src', or alternatively, **s** = 'src->dest'. Input and output images have 3 planes.

[**o1**,**o2**,**o3**] = **colorspace**(**s**, **im**) as above but specifies separate output channels or planes.

colorspace(**s**, **i1**,**i2**,**i3**) as above but specifies separate input channels.

Supported color spaces are:

'RGB'	R'G'B' Red Green Blue (ITU-R BT.709 gamma-corrected)
'YPbPr'	Luma (ITU-R BT.601) + Chroma
'YCbCr'/'YCC'	Luma + Chroma ("digitized" version of Y'PbPr)
'YUV'	NTSC PAL Y'UV Luma + Chroma
'YIQ'	NTSC Y'IQ Luma + Chroma
'YDbDr'	SECAM Y'DbDr Luma + Chroma
'JPEGYCbCr'	JPEG-Y'CbCr Luma + Chroma
'HSV'/'HSB'	Hue Saturation Value/Brightness
'HSL'/'HLS'/'HSI'	Hue Saturation Luminance/Intensity
'XYZ'	CIE XYZ
'Lab'	CIE L*a*b* (CIELAB)
'Luv'	CIE L*u*v* (CIELUV)
'Lch'	CIE L*ch (CIELCH)

Notes

- RGB input is assumed to be gamma encoded
- RGB output is gamma encoded
- All conversions assume 2 degree observer and D65 illuminant.
- Color space names are case insensitive.
- When R'G'B' is the source or destination, it can be omitted. For example 'yuv<-' is short for 'yuv<-rgb'.
- MATLAB uses two standard data formats for R'G'B': double data with intensities in the range 0 to 1, and uint8 data with integer-valued intensities from 0

to 255. As MATLAB's native datatype, double data is the natural choice, and the R'G'B' format used by **colorspace**. However, for memory and computational performance, some functions also operate with uint8 R'G'B'. Given uint8 R'G'B' color data, **colorspace** will first cast it to double R'G'B' before processing.

- If **im** is an $M \times 3$ array, like a colormap, **out** will also have size $M \times 3$.

Author

Pascal Getreuer 2005-2006

---

diff2

First-order difference

d = **diff2**(**v**) is the first-order difference ($1 \times N$) of the series data in vector **v** ($1 \times N$) and the first element is zero.

d = **diff2**(**a**) is the first-order difference ($M \times N$) of the series data in each row of the matrix **a** ($M \times N$) and the first element in each row is zero.

Notes

- Unlike the builtin function DIFF, the result of **diff2** has the same number of columns as the input.

See also

diff

---

distance

Euclidean distances between sets of points

d = **distance**(**a**,**b**) is the Euclidean distances between L-dimensional points described by the matrices **a** ($L \times M$) and **b** ($L \times N$) respectively. The **distance** **d** is $M \times N$ and element **d**(I,J) is the **distance** between points **a**(I) and **d**(J).

Example

```
A = rand(400,100); B = rand(400,200);
d = distance(A,B);
```

Notes

- This fully vectorized (VERY FAST!)
- It computes the Euclidean **distance** between two vectors by:

```
||A-B|| = sqrt ( ||A||^2 + ||B||^2 - 2*A.B )
```

Author

Roland Bunschoten, University of Amsterdam, Intelligent Autonomous Systems (IAS) group, Kruislaan 403 1098 SJ Amsterdam, tel.(+31)20-5257524, bunschot@wins.uva.nl Last Rev: Oct 29 16:35:48 MET DST 1999, Tested: PC Matlab v5.2 and Solaris Matlab v5.3, Thanx: Nikos Vlassis.

See also

closest

---

distributeblocks

Distribute blocks in Simulink block library

distributeblocks(**model**) equidistantly distributes blocks in a Simulink block library named **model**.

Notes

- The MATLAB functions to create Simulink blocks from symbolic expresssions actually place all blocks on top of each other. This function scans a simulink model and rearranges the blocks on an equidistantly spaced grid.
- The Simulink model must already be opened before running this function!

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

symexpr2slblock, doesblockexist

---

dockfigs

Control figure docking in the GUI

dockfigs causes all new figures to be docked into the GUI

dockfigs(1) as above.

dockfigs(0) causes all new figures to be undocked from the GUI

---

doesblockexist

Check existence of block in Simulink model

res = **doesblockexist**(**mdlname**, **blockaddress**) is a logical result that indicates whether or not the block **blockaddress** exists within the Simulink model **mdlname**.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

symexpr2slblock, distributeblocks

---

dtransform

Distance transform

dt = **dtransform**(**im**, **options**) is the distance transform of the binary image **im**. The value of each output pixel is the distance (pixels) to the closest set pixel.

Options

'Euclidean'	use Euclidean distance (default)
'cityblock'	use cityblock (Manhattan) distance
'show', T	display the evolving distance transform, with a delay of T seconds between frames

See also

imorph, distancexform, dxform

---

e2h

Euclidean to homogeneous

H = **e2h**(**E**) is the homogeneous version (K+1 $\times$ N) of the Euclidean points **E** ($K \times N$) where each column represents one point in R^K.

See also

h2e

EarthView

Image from Google maps

A concrete subclass of ImageSource that acquires images from Google maps.

Methods

grab	Grab a frame from Google maps
size	Size of image
close	Close the image source
char	Convert the object parameters to human readable string

Examples

Create an EarthView camera

```
ev = EarthView();
```

Zoom into QUT campus in Brisbane

```
ev.grab(-27.475722,153.0285, 17);
```

Show aerial view of Brisbane in satellite and map view

```
ev.grab('brisbane', 14)
ev.grab('brisbane', 14, 'map')
```

Notes

- Google limit the number of map queries limit to 1000 unique (different) image requests per viewer per day. A 403 error is returned if the daily quota is exceeded.
- Maximum size is 640×640 for free access, business users can get more.
- There are lots of conditions on what you can do with the images, particularly with respect to publication. See the Google web site for details.

Author

Peter Corke, with some lines of code from from get_google_map by Val Schmidt.

See also

ImageSource

EarthView.EarthView

Create EarthView object

ev = **EarthView**(**options**)

Options

'satellite'	Retrieve satellite image
'map'	Retrieve map image
'hybrid'	Retrieve satellite image with map overlay
'scale'	Google map scale (default 18)
'width', W	Set image width to W (default 640)
'height', H	Set image height to H (default 640)
'key', S	The Google maps key string

see also options for ImageSource.

Notes

- A key is required before you can use the Google Static Maps API. The key is a long string that can be passed to the constructor or saved as an environment variable GOOGLE_KEY. You need a Google account before you can register for a key.

Notes

- Scale is 1 for the whole world, 20 is about as high a resolution as you can get.

See also

ImageSource, EarthView.grab

EarthView.char

Convert to string

EV.**char**() is a string representing the state of the **EarthView** object in human readable form.

See also

EarthView.display

EarthView.grab

Grab an aerial image

im = EV.**grab**(**lat**, **long**, **options**) is an image of the Earth centred at the geographic coordinate (lat, long).

im = **EarthView.grab**(**lat**, **long**, **zoom**, **options**) as above with the specified zoom. **zoom** is an integer between 1 (zoom right out) to a maximum of 18-20 depending on where in the world you are looking.

[**im**,**E**,**n**] = **EarthView.grab**(**lat**, **long**, **options**) as above but also returns the estimated easting **E** and northing **n**. **E** and **n** are both matrices, the same size as **im**, whose corresponding elements are the easting and northing are the coordinates of the pixel.

[**im**,**E**,**n**] = **EarthView.grab**(**name**, **options**) as above but uses a geocoding web site to resolve the name to a location.

Options

'satellite'	Retrieve satellite image
'map'	Retrieve map image
'hybrid'	Retrieve satellite image with map overlay
'roadmap'	Retrieve a binary image that shows only roads, no labels. Roads are white, everything else is black.
'noplacenames'	Don't show placenames.
'noroadnames'	Don't show roadnames.

Examples

Zoom into QUT campus in Brisbane

```
ev.grab(-27.475722,153.0285, 17);
```

Show aerial view of Brisbane in satellite and map view

```
ev.grab('brisbane', 14)
ev.grab('brisbane', 14, 'map')
```

Notes

- If northing/easting outputs are requested the function deg2utm is required (from MATLAB Central)

- The easting/northing is somewhat approximate, see get_google_map on MATLAB Central.
- If no output argument is given the image is displayed using idisp.

---

edgelist

Return list of edge pixels for region

eg = **edgelist**(**im**, **seed**) is a list of edge pixels ($N \times 2$) of a region in the image **im** starting at edge coordinate **seed**=[X,Y]. The **edgelist** has one row per edge point coordinate (x,y).

eg = **edgelist**(**im**, **seed**, **direction**) as above, but the direction of edge following is specified. **direction** == 0 (default) means clockwise, non zero is counter-clockwise. Note that direction is with respect to y-axis upward, in matrix coordinate frame, not image frame.

[**eg**,**d**] = **edgelist**(**im**, **seed**, **direction**) as above but also returns a vector of edge segment directions which have values 1 to 8 representing W SW S SE E NW N NW respectively.

Notes

- Coordinates are given assuming the matrix is an image, so the indices are always in the form (x,y) or (column,row).
- **im** is a binary image where 0 is assumed to be background, non-zero is an object.
- **seed** must be a point on the edge of the region.
- The seed point is always the first element of the returned **edgelist**.
- 8-direction chain coding can give incorrect results when used with blobs founds using 4-way connectivty.

Reference

- METHODS TO ESTIMATE AREAS AND PERIMETERS OF BLOB-LIKE OBJECTS: A COMPARISON Luren Yang, Fritz Albregtsen, Tor Lgnnestad and Per Grgttum IAPR Workshop on Machine Vision Applications Dec. 13-15, 1994, Kawasaki

See also

ilabel

---

epidist

Distance of point from epipolar line

d = **epidist**(**f**, **p1**, **p2**) is the distance of the points **p2** ($2 \times M$) from the epipolar lines due to points **p1** ($2 \times N$) where **f** (3×3) is a fundamental matrix relating the views containing image points **p1** and **p2**.

d ($N \times M$) is the distance matrix where element **d**(i,j) is the distance from the point **p2**(j) to the epipolar line due to point **p1**(i).

Author

Based on fmatrix code by, Nuno Alexandre Cid Martins, Coimbra, Oct 27, 1998, I.S.R.

See also

epiline, fmatrix

---

epiline

Draw epipolar lines

epiline(**f**, **p**) draws epipolar lines in current figure based on points **p** ($2 \times N$) and the fundamental matrix **f** (3×3). Points are specified by the columns of **p**.

epiline(**f**, **p**, **ls**) as above but draw lines using the line style arguments **ls**.

H = **epiline**(**f**, **p**, **ls**) as above but return a vector of graphic handles, one per line drawn.

See also

fmatrix, epidist

FeatureMatch

Feature correspondence object

This class represents the correspondence between two PointFeature objects. A vector of FeatureMatch objects can represent the correspondence between sets of points.

Methods

plot	Plot corresponding points
show	Show summary statistics of corresponding points
ransac	Determine inliers and outliers
inlier	Return inlier matches
outlier	Return outlier matches
subset	Return a subset of matches
display	Display value of match
char	Convert value of match to string

Properties

p1	Point coordinates in view 1 (2×1)
p2	Point coordinates in view 2 (2×1)
p	Point coordinates in view 1 and 2 (4×1)
distance	Match strength between the points

Properties of a vector of FeatureMatch objects are returned as a vector. If F is a vector $(N \times 1)$ of FeatureMatch objects then F.p1 is a $2 \times N$ matrix with each column the corresponding view 1 point coordinate.

Note

- FeatureMatch is a reference object.
- FeatureMatch objects can be used in vectors and arrays
- Operates with all objects derived from PointFeature, such as ScalePointFeature, SurfPointFeature and SiftPointFeature.

See also

PointFeature, SurfPointFeature, SiftPointFeature

FeatureMatch.FeatureMatch

Create a new FeatureMatch object

m = **FeatureMatch**(**f1**, **f2**, **s**) is a new **FeatureMatch** object describing a correspondence between point features **f1** and **f2** with a strength of **s**.

m = **FeatureMatch**(**f1**, **f2**) as above but the strength is set to NaN.

Notes

- Only the coordinates of the PointFeature are kept.

See also

PointFeature, SurfPointFeature, SiftPointFeature

FeatureMatch.char

Convert to string

s = M.**char**() is a compact string representation of the match object. If M is a vector then the string has multiple lines, one per element.

FeatureMatch.display

Display value

M.**display**() displays a compact human-readable representation of the feature pair. If M is a vector then the elements are printed one per line.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a FeatureMatch object and the command has no trailing semicolon.

See also

FeatureMatch.char

FeatureMatch.inlier

Inlier features

m2 = M.**inlier**() is a subset of the **FeatureMatch** vector M that are considered to be inliers.

Notes

- Inliers are not determined until after RANSAC is run.

See also

FeatureMatch.outlier, FeatureMatch.ransac

FeatureMatch.outlier

Outlier features

m2 = M.**outlier**() is a subset of the **FeatureMatch** vector M that are considered to be outliers.

Notes

- Outliers are not determined until after RANSAC is run.

See also

FeatureMatch.inlier, FeatureMatch.ransac

FeatureMatch.p

Feature point coordinate pairs

p = M.**p**() is a $4 \times N$ matrix containing the feature point coordinates. Each column contains the coordinates of a pair of corresponding points [u1,v1,u2,v2].

See also

FeatureMatch.p1, FeatureMatch.p2

FeatureMatch.p1

Feature point coordinates from view 1

p = M.**p1**() is a $2 \times N$ matrix containing the feature points coordinates from view 1. These are the (u,v) properties of the feature F1 passed to the constructor.

See also

FeatureMatch.FeatureMatch, FeatureMatch.p2, FeatureMatch.p

FeatureMatch.p2

Feature point coordinates from view 2

p = M.**p2**() is a $2 \times N$ matrix containing the feature points coordinates from view 1. These are the (u,v) properties of the feature F2 passed to the constructor.

See also

FeatureMatch.FeatureMatch, FeatureMatch.p1, FeatureMatch.p

FeatureMatch.plot

Show corresponding points

M.**plot**() overlays the correspondences in the **FeatureMatch** vector M on the current figure. The figure must comprise views 1 and 2 side by side, for example by:

```
idisp({im1,im2})
m.plot()
```

M.**plot**(**ls**) as above but the optional line style arguments **ls** are passed to **plot**.

Notes

- Using IDISP as above adds UserData to the figure, and an error is created if this UserData is not found.

See also

idisp

FeatureMatch.ransac

Apply RANSAC

M.**ransac**(**func**, **options**) applies the RANSAC algorithm to fit the point correspondences to the model described by the function **func**. The **options** are passed to the RANSAC() function. Elements of the **FeatureMatch** vector have their status updated in place to indicate whether they are inliers or outliers.

Example

```
f1 = isurf(im1);
f2 = isurf(im2);
m = f1.match(f2);
m.ransac( @fmatrix, 1e-4);
```

See also

fmatrix, homography, ransac

FeatureMatch.show

Display summary statistics of the FeatureMatch vector

M.**show**() is a compact summary of the **FeatureMatch** vector M that gives the number of matches, inliers and outliers (and their percentages).

FeatureMatch.subset

Subset of matches

m2 = M.**subset**(**n**) is a **FeatureMatch** vector with no more than **n** elements sampled uniformly from M.

---

filt1d

1-dimensional rank filter

y = **filt1d**(**x**, **options**) is the minimum, maximum or median value $(1 \times N)$ of the vector **x** $(1 \times N)$ compute over an odd length sliding window.

Options

'max'	Compute maximum value over the window (default)
'min'	Compute minimum value over the window
'median'	Compute minimum value over the window
'width', W	Width of the window (default 5)

Notes

- If the window width is even, it is incremented by one.
- The first and last elements of **x** are replicated so the output vector is the same length as the input vector.

FishEyeCamera

Fish eye camera class

A concrete class a fisheye lense projection camera.

The camera coordinate system is:


```
0------------> u, X
|
|
|   + (principal point)
|
|   Z-axis is into the page.
v, Y
```

This camera model assumes central projection, that is, the focal point is at z=0 and the image plane is at z=f. The image is not inverted.

Methods

project	project world points to image plane
plot	plot/return world point on image plane
hold	control hold for image plane
ishold	test figure hold for image plane
clf	clear image plane
figure	figure holding the image plane
mesh	draw shape represented as a mesh
point	draw homogeneous points on image plane
line	draw homogeneous lines on image plane
plot_camera	draw camera
rpy	set camera attitude
move	copy of Camera after motion
centre	get world coordinate of camera centre
delete	object destructor
char	convert camera parameters to string
display	display camera parameters

Properties (read/write)

npix	image dimensions in pixels (2×1)
pp	intrinsic: principal point (2×1)
f	intrinsic: focal length [metres]
rho	intrinsic: pixel dimensions (2×1) [metres]
T	extrinsic: camera pose as homogeneous transformation

Properties (read only)

nu	number of pixels in u-direction
nv	number of pixels in v-direction

Notes

- Camera is a reference object.
- Camera objects can be used in vectors and arrays

See also

Camera

FishEyeCamera.FishEyeCamera

Create fisheyecamera object

C = **FishEyeCamera**() creates a fisheye camera with canonic parameters: f=1 and name='canonic'.

C = **FishEyeCamera**(**options**) as above but with specified parameters.

Options

'name', N	Name of camera
'default'	Default camera parameters: 1024×1024, f=8mm, 10um pixels, camera at origin, optical axis is z-axis, u- and v-axes are parallel to x- and y- axes respectively.
'projection', M	Fisheye model: 'equiangular' (default), 'sine', 'equisolid', 'stereographic'
'k', K	Parameter for the projection model
'resolution', N	Image plane resolution: $N \times N$ or N=[W H].
'sensor', S	Image sensor size [metres] (2×1)
'centre', P	Principal point (2×1)
'pixel', S	Pixel size: $S \times S$ or S=[W H].
'noise', SIGMA	Standard deviation of additive Gaussian noise added to returned image projections
'pose', T	Pose of the camera as a homogeneous transformation

Notes

- If K is not specified it is computed such that the circular imaging region maximally fills the square image plane.

See also

Camera, CentralCamera, CatadioptricCamera, SphericalCamera

FishEyeCamera.project

Project world points to image plane

uv = C.**project**(**p**, **options**) are the image plane coordinates for the world points **p**. The columns of **p** ($3 \times N$) are the world points and the columns of **uv** ($2 \times N$) are the corresponding image plane points.

Options

'Tobj', T	Transform all points by the homogeneous transformation T before projecting them to the camera image plane.
'Tcam', T	Set the camera pose to the homogeneous transformation T before projecting points to the camera image plane. Temporarily overrides the current camera pose C.T.

See also

FishEyeCamera.plot

---

fmatrix

Estimate fundamental matrix

f = **fmatrix**(**p1**, **p2**, **options**) is the fundamental matrix (3×3) that relates two sets of corresponding points **p1** ($2 \times N$) and **p2** ($2 \times N$) from two different camera views.

Notes

- The points must be corresponding, no outlier rejection is performed.
- Contains a RANSAC driver, which means it can be passed to ransac().
- **f** is a rank 2 matrix, that is, it is singular.

Reference

Hartley and Zisserman, 'Multiple View Geometry in Computer Vision', page 270.

Author

Based on fundamental matrix code by Peter Kovesi, School of Computer Science & Software Engineering, The University of Western Australia, http://www.csse.uwa.edu.au/,

See also

ransac, homography, epiline, epidist

---

gauss2d

Gaussian kernel

out = **gauss2d**(**im**, **sigma**, **C**) is a unit volume Gaussian kernel rendered into matrix **out** ($W \times H$) the same size as **im** ($W \times H$). The Gaussian has a standard deviation of **sigma**. The Gaussian is centered at **C**=[U,V].

---

gaussfunc

kernel

k = gauss1(, c, sigma)

Returns a unit volume Gaussian smoothing kernel. The Gaussian has a standard deviation of sigma, and the convolution kernel has a half size of w, that is, k is (2W+1) x (2W+1).

---

h2e

Homogeneous to Euclidean

E = **h2e**(**H**) is the Euclidean version (K-1 $\times$ N) of the homogeneous points **H** ($K \times N$) where each column represents one point in P^K.

See also

e2h

---

hist2d

MEX file to compute 2-D histogram.

[**h**,**vx**,**vy**] = **hist2d**(**x**,**y**)

or

[**h**,**vx**,**vy**] = **hist2d**(**x**,**y**,[x0 dx nx],[y0 dy ny])

Inputs:

```
x,y     data points.  {x(i),y(i)} is a single data point.
x0      lowest x bin's lower edge
dx      x bin width
nx      number of x bins
y0      lowest y bin's lower edge
dy      y bin width
ny      number of y bins
[x0,dx,nx] and [y0,dy,ny] default = [0,1,256]
```

Outputs:

```
h       histogram matrix.  h(i,j) = number of data points

satisfying vx(j) <= x < vx(j+1) and vy(i) <= y < vy(i+1).

vx      bin lower x-ordinates (one for each column of h)
vy      bin lower y-ordinates (one for each row of h)
```

Notes

- Data vectors **x** and **y** must be double

Author

Michael Maurer, 7 October 1994. Copyright 1994 by Michael Maurer.

---

hitormiss

Hit or miss transform

H = **hitormiss**(**im**, **se**) is the hit-or-miss transform of the binary image **im** with the structuring element **se**. Unlike standard morphological operations S has three possible values: 0, 1 and don't care (represented by NaN).

References

- Robotics, Vision & Control, Section 12.5.3, P. Corke, Springer 2011.

See also

imorph, ithin, itriplepoint, iendpoint

---

homline

Homogeneous line from two points

L = **homline**(**x1**, **y1**, **x2**, **y2**) is a vector (3×1) which describes a line in homogeneous form that contains the two Euclidean points (**x1**,**y1**) and (**x2**,**y2**).

Homogeneous points X (3×1) on the line must satisfy **L**'*X = 0.

See also

plot_homline

---

homography

Estimate homography

H = **homography**(**p1**, **p2**) is the **homography** (3×3) that relates two sets of corresponding points **p1** $(2 \times N)$ and **p2** $(2 \times N)$ from two different camera views of a planar object.

Notes

- The points must be corresponding, no outlier rejection is performed.
- The points must be projections of points lying on a world plane
- Contains a RANSAC driver, which means it can be passed to ransac().

Author

Based on **homography** code by Peter Kovesi, School of Computer Science & Software Engineering, The University of Western Australia, http://www.csse.uwa.edu.au/,

See also

ransac, invhomog, fmatrix

---

homtrans

Apply a homogeneous transformation

p2 = **homtrans**(**T**, **p**) applies homogeneous transformation **T** to the points stored columnwise in **p**.

- If **T** is in SE(2) (3×3) and
  - **p** is $2 \times N$ (2D points) they are considered Euclidean (R^2)
  - **p** is $3 \times N$ (2D points) they are considered projective ($\mathbf{p}^2$)
- If **T** is in SE(3) (4×4) and
  - **p** is $3 \times N$ (3D points) they are considered Euclidean (R^3)
  - **p** is $4 \times N$ (3D points) they are considered projective ($\mathbf{p}^3$)

tp = **homtrans**(**T**, **T1**) applies homogeneous transformation **T** to the homogeneous transformation **T1**, that is **tp**=**T***T1. If **T1** is a 3-dimensional transformation then **T** is applied to each plane as defined by the first two dimensions, ie. if **T** = $N \times N$ and **T**=$N \times N \times$ **p** then the result is $N \times N \times$ **p**.

See also

e2h, h2e

---

homwarp

Warp image by an homography

out = homwarp(**H**, **im**, **options**) is a warp of the image **im** obtained by applying the homography **H** to the coordinates of every input pixel.

[**out**,**offs**] = homwarp(**H**, **im**, **options**) as above but **offs** is the offset of the warped tile **out** with respect to the origin of **im**.

Options

'full'	output image contains all the warped pixels, but its position with respect to the input image is given by the second return value **offs**.
'extrapval', V	set unmapped pixels to this value (default NaN)
'roi', R	output image contains the specified ROI in the input image
'scale', S	scale the output by this factor
'dimension', D	ensure output image is $D \times D$
'size', S	size of output image S=[W,**H**]
'coords', {U,V}	coordinate matrices for im, each same size as im.

Notes

- The edges of the resulting output image will in general not be be vertical and horizontal lines.

See also

homography, itrim, interp2

Hough

Hough transform class

The Hough transform is a technique for finding lines in an image using a voting scheme. For every edge pixel in the input image a set of cells in the Hough accumulator (voting array) are incremented.

In this version of the Hough transform lines are described by:

```
d = y cos(theta) + x sin(theta)
```


where theta is the angle the line makes to horizontal axis, and d is the perpendicular distance between (0,0) and the line. A horizontal line has theta = 0, a vertical line has theta = pi/2 or -pi/2.

The voting array is 2-dimensional, with columns corresponding to theta and rows corresponding to offset (d). Theta spans the range -pi/2 to pi/2 in Ntheta steps. Offset is in the range -rho_max to rho_max where rho_max=max(W,H).

Methods

plot	Overlay detected lines
show	Display the Hough accumulator
lines	Return line features
char	Convert Hough parameters to string
display	Display Hough parameters

Properties

Nrho	Number of bins in rho direction
Ntheta	Number of bins in theta direction
A	The Hough accumulator (Nrho x Ntheta)
rho	rho values for the centre of each bin vertically
theta	Theta values for the centre of each bin horizontally
edgeThresh	Threshold on relative edge pixel strength
houghThresh	Threshold on relative peak strength
suppress	Radius of accumulator cells cleared around peak
interpWidth	Width of region used for peak interpolation

Notes

- Hough is a reference object.

See also

LineFeature

Hough.Hough

Create Hough transform object

ht = Hough(**E**, **options**) is the Hough transform of the edge image **E**.

For every pixel in the edge image **E** ($H \times W$) greater than a threshold the corresponding elements of the accumulator are incremented. By default the vote is incremented by

the edge strength but votes can be made equal with the option 'equal'. The threshold is determined from the maximum edge strength value x **ht**.edgeThresh.

Options

'equal'	All edge pixels have equal weight, otherwise the edge pixel value is the vote strength
'points'	Pass set of points rather than an edge image, in this case **E** $(2 \times N)$ is a set of N points, or **E** $(3 \times N)$ is a set of N points with corresponding vote strengths as the third row
'interpwidth', W	Interpolation width (default 3)
'houghthresh', T	Set **ht**.houghThresh (default 0.5)
'edgethresh', T	Set **ht**.edgeThresh (default 0.1);
'suppress', W	Set **ht**.suppress (default 0)
'nbins', N	Set number of bins, if N is scalar set Nrho=Ntheta=N, else N = [Ntheta, Nrho]. Default 400×401.

Hough.char

Convert to string

s = HT.**char**() is a compact string representation of the **Hough** transform parameters.

Hough.display

Display value

HT.**display**() displays a compact human-readable string representation of the **Hough** transform parameters.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a Hough object and the command has no trailing semicolon.

See also

Hough.char

Hough.lines

Find lines

L = HT.**lines**() is a vector of LineFeature objects that represent the dominant **lines** in the **Hough** accumulator.

L = HT.**lines**(**n**) as above but returns no more than **n** LineFeature objects.

Lines are the coordinates of peaks in the **Hough** accumulator. The highest peak is found, refined to subpixel precision, then all elements in an HT.suppress radius around are zeroed so as to eliminate multiple close minima. The process is repeated for all peaks.

The peak detection loop breaks early if the remaining peak has a strength less than HT.houghThresh times the maximum vote value.

See also

Hough.plot, LineFeature

Hough.plot

Plot line features

HT.**plot**() overlays all detected lines on the current figure.

HT.**plot**(**n**) overlays a maximum of **n** strongest lines on the current figure.

HT.**plot**(**n**, **ls**) as above but the optional line style arguments **ls** are passed to **plot**.

H = HT.**plot**() as above but returns a vector of graphics handles for each line.

See also

Hough.lines

Hough.show

Display the Hough accumulator as image

s = HT.**show**() displays the **Hough** vote accumulator as an image using the hot colormap, where 'heat' is proportional to the number of votes.

See also

colormap, hot

---

humoments

Hu moments

phi = **humoments**(**im**) is the vector (7×1) of Hu moment invariants for the binary image **im**.

Notes

- **im** is assumed to be a binary image of a single connected region

Reference

M-K. Hu, Visual pattern recognition by moment invariants. IRE Trans. on Information Theory, IT-8:pp. 179-187, 1962.

See also

npq

---

ianimate

Display an image sequence

ianimate(**im**, **options**) displays a greyscale image sequence **im** ($H \times W \times N$) or a color image sequence **im** (HxWx3xN) where N is the number of frames in the sequence.

ianimate(**im**, **features**, **options**) as above but with point features overlaid. **features** ($N \times 1$) is a cell array whose elements are vectors of feature objects for the corresponding frames of **im**. The feature is plotted using the feature object's plot method and additional options are passed through to that method.

Examples

Animate image sequence:

```
ianimate(seq);
```

Animate image sequence with overlaid corner features:

```
c = icorner(im, 'nfeat', 200);  % computer corners
ianimate(seq, c, 'gs');  % features shown as green squares
```

Options

'fps', F	set the frame rate (default 5 frames/sec)
'loop'	endlessly loop over the sequence
'movie', M	save the animation as a series of PNG frames in the folder M
'npoints', N	plot no more than N features per frame (default 100)
'only', I	display only the I'th frame from the sequence
'title', T	displays the specified title on each frame, T is a cell array $(1 \times N)$ of strings.

Notes

- If titles are not specified the title is "frame N"
- If the 'movie' is used the frames can be converted to a movie using a utility like ffmpeg, for instance:

```
ffmpeg -i *.png -r 5 movie.mp4
```

or to set the bit rate explicitly

```
ffmpeg -i *.png -b:v 64k movie.mp4
```

See also

PointFeature, iharris, isurf, idisp

---

ibbox

Find bounding box

box = **ibbox**(**p**) is the minimal bounding box that contains the points described by the columns of **p** $(2 \times N)$.

box = **ibbox**(**im**) as above but the box minimally contains the non-zero pixels in the image **im**.

Notes

- The bounding box is a 2×2 matrix [XMIN XMAX; YMIN YMAX].

---

iblobs

features

f = **iblobs**(**im**, **options**) is a vector of RegionFeature objects that describe each connected region in the image **im**.

Options

'aspect', A	set pixel aspect ratio, default 1.0
'connect', C	set connectivity, 4 (default) or 8
'greyscale'	compute greyscale moments 0 (default) or 1
'boundary'	compute boundary (default off)
'area', [A1,A2]	accept only blobs with area in the interval A1 to A2
'shape', [S1,S2]	accept only blobs with shape in the interval S1 to S2
'touch', T	accept only blobs that touch (1) or do not touch (0) the edge (default accept all)
'class', C	accept only blobs of pixel value C (default all)

The RegionFeature object has many properties including:

uc	centroid, horizontal coordinate
vc	centroid, vertical coordinate
p	centroid (uc, vc)
umin	bounding box, minimum horizontal coordinate
umax	bounding box, maximum horizontal coordinate
vmin	bounding box, minimum vertical coordinate
vmax	bounding box, maximum vertical coordinate
area	the number of pixels
class	the value of the pixels forming this region
label	the label assigned to this region
children	a list of indices of features that are children of this feature
edgepoint	coordinate of a point on the perimeter
edge	a list of edge points $2 \times N$ matrix
perimeter	edge length (pixels)
touch	true if region touches edge of the image
a	major axis length of equivalent ellipse
b	minor axis length of equivalent ellipse
theta	angle of major ellipse axis to horizontal axis
shape	aspect ratio b/a (always <= 1.0)
circularity	1 for a circle, less for other shapes
moments	a structure containing moments of order 0 to 2

References

- Robotics, Vision & Control, Section 13.1, P. Corke, Springer 2011.
- METHODS TO ESTIMATE AREAS AND PERIMETERS OF BLOB-LIKE OBJECTS: A COMPARISON Luren Yang, Fritz Albregtsen, Tor Lgnnestad and Per Grgttum IAPR Workshop on Machine Vision Applications Dec. 13-15, 1994, Kawasaki
- Area and perimeter measurement of blobs in discrete binary pictures. Z.Kulpa. Comput. Graph. Image Process., 6:434-451, 1977.

Notes

- The RegionFeature objects are ordered by the raster order of the top most point (smallest v coordinate) in each blob.
- Circularity is computed using the raw perimeter length scaled down by Kulpa's correction factor.

See also

RegionFeature, ilabel, idisplabel, imoments

---

icanny

edge detection

E = **icanny**(**im**, **options**) is an edge image obtained using the Canny edge detector algorithm. Hysteresis filtering is applied to the gradient image: edge pixels > th1 are connected to adjacent pixels > th0, those below th0 are set to zero.

Options

'sd', S	set the standard deviation for smoothing (default 1)
'th0', T	set the lower hysteresis threshold (default 0.1 x strongest edge)
'th1', T	set the upper hysteresis threshold (default 0.5 x strongest edge)

Reference

- "A Computational Approach To Edge Detection", J. Canny, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679698, 1986.

Notes

- Produces a zero image with single pixel wide edges having non-zero values.
- Larger values correspond to stronger edges.
- If th1 is zero then no hysteresis filtering is performed.
- A color image is automatically converted to greyscale first.

Author

Oded Comay, Tel Aviv University, 1996-7.

See also

isobel, kdgauss

---

iclose

closing

out = **iclose**(**im**, **se**, **options**) is the image **im** after morphological closing with the structuring element **se**. This is a morphological dilation followed by an erosion.

out = **iclose**(**im**, **se**, **n**, **options**) as above but the structuring element **se** is applied **n** times, that is **n** erosions followed by **n** dilations.

Notes

- For binary image a closing operation can be used to eliminate small black holes in white regions.
- Cheaper to apply a smaller structuring element multiple times than one large one, the effective structuring element is the Minkowski sum of the structuring element with itself **n** times.
- Windowing options of IMORPH can be passed. By default output image is same size as input image.

See also

iopen, idilate, ierode, imorph

---

icolor

Colorize a greyscale image

C = **icolor**(**im**) is a color image **C** ($H \times W \times 3$)where each color plane is equal to **im** ($H \times W$).

C = **icolor**(**im**, **color**) as above but each output pixel is **color** (3×1) times the corresponding element of **im**.

Examples

Create a color image that looks the same as the greyscale image

```
c = icolor(im);
```

each set pixel in im is set to [1 1 1] in the output.

Create an rose tinted version of the greyscale image

```
c = icolor(im, colorname('pink'));
```

each set pixel in im is set to [0 1 1] in the output.

Notes

- Can convert a monochrome sequence ($H \times W \times N$) to a color image sequence (HxWx3xN).

See also

imono, colorize, ipixswitch

---

iconcat

Concatenate images

C = **iconcat**(**im**,**options**) concatenates images from the cell array **im**.

iconcat(**im**,**options**) as above but displays the concatenated images using IDISP.

[**C**,**u**] = **iconcat**(**im**,**options**) as above but also returns the vector **u** whose elements are the coordinates of the left (or top in vertical mode) edge of the corresponding image within the concatenated image.

Options

'dir', D	direction of concatenation: 'horizontal' (default) or 'vertical'.
'bgval', B	value of padding pixels (default NaN)

Examples

Horizontally concatenate three images

```
c = iconcat({im1, im2, im3}, 'h');
```

Find the first column of each of the three images

```
[c,u] = iconcat({im1, im2, im3}, 'h');
```

where u is a 3-vector such that im3 starts in the u(3)'rd column of c.

Notes

- The images do not have to be of the same size, and smaller images are surrounded by background pixels which can be specified.
- Works for color or greyscale images.
- Direction can be abbreviated to first character, 'h' or 'v'.
- In vertical mode all images are right justified.
- In horizontal mode all images are top justified.

See also

idisp

iconv

Image cross-correlation

C = **iconv**(**im1**, **im2**, **options**) is the cross-correlation of images **im1** and **im2**. The smaller image is taken as the kernel and correlated with the larger image.

Options

'same'	output image is same size as largest input image (default)
'full'	output image is larger than the input image
'valid'	output image is smaller than the input image, and contains only valid pixels

Notes

- If the larger image is color (has multiple planes) the kernel is applied to each plane, resulting in an output image with the same number of planes.
- The kernel must be greyscale.
- This function is a convenience wrapper for the MATLAB function CONV2.
- Works for double, uint8 or uint16 images. Image and kernel must be of the same type and the result is of the same type.
- This function is badly named, it peforms cross-correlation not convolution.

See also

conv2

---

icorner

Corner detector

f = **icorner**(**im**, **options**) is a vector of PointFeature objects describing corner features detected in the image **im**. This is a non-scale space detector and by default the Harris method is used but Shi-Tomasi and Noble are also supported.

If **im** is an image sequence a cell array of PointFeature vectors for the correspnding frames of **im**.

The PointFeature object has many properties including:

u	horizontal coordinate
v	vertical coordinate
strength	corner strength
descriptor	corner descriptor (vector)

See PointFeature for full details

Options

'detector', D	choose the detector where D is one of 'harris' (default), 'noble' or 'klt'
'sigma', S	kernel width for smoothing (default 2)
'deriv', D	kernel for gradient (default kdgauss(2))
'cmin', CM	minimum corner strength
'cminthresh', CT	minimum corner strength as a fraction of maximum corner strength
'edgegap', E	don't return features closer than E pixels to the edge of image (default 2)
'suppress', R	don't return a feature closer than R pixels to an earlier feature (default 0)
'nfeat', N	return the N strongest corners (default Inf)
'k', K	set the value of k for the Harris detector
'patch', P	use a $P \times P$ patch of surrounding pixel values as the feature vector. The vector has zero mean and unit norm.
'color'	specify that **im** is a color image not a sequence

Example

Compute the 100 strongest Harris features for the image

```
c = icorner(im, 'nfeat', 100);
```

and overlay them on the image

```
idisp(im);
c.plot();
```

Notes

- Corners are processed in order from strongest to weakest.
- The function stops when:
  - the corner strength drops below cmin, or
  - the corner strength drops below cMinThresh x strongest corner, or
  - the list of corners is exhausted
- Features are returned in descending strength order
- If **im** has more than 2 dimensions it is either a color image or a sequence
- If **im** is $N \times M \times P$ it is taken as an image sequence and **f** is a cell array whose elements are feature vectors for the corresponding image in the sequence.
- If **im** is $N \times M \times 3$ it is taken as a sequence unless the option 'color' is given

- If **im** is NxMx3xP it is taken as a sequence of color images and **f** is a cell array whose elements are feature vectors for the corresponding color image in the sequence.
- The default descriptor is a vector [Ix* Iy* Ixy*] which are the unique elements of the structure tensor, where * denotes squared and smoothed.
- The descriptor is a vector of float types to save space

References

- "A combined corner and edge detector", C.G. Harris and M.J. Stephens, Proc. Fourth Alvey Vision Conf., Manchester, pp 147-151, 1988.
- "Finding corners", J.Noble, Image and Vision Computing, vol.6, pp.121-128, May 1988.
- "Good features to track", J. Shi and C. Tomasi, Proc. Computer Vision and Pattern Recognition, pp. 593-593, IEEE Computer Society, 1994.
  - Robotics, Vision & Control, Section 13.3, P. Corke, Springer 2011.

See also

PointFeature, isurf

---

icp

Point cloud alignment

T = **icp**(**p1**, **p2**, **options**) is the homogeneous transformation that best transforms the set of points **p1** to **p2** using the iterative closest point algorithm.

[**T**,**d**] = **icp**(**p1**, **p2**, **options**) as above but also returns the norm of the error between the transformed point set **p2** and **p1**.

Options

'dplot', **d**	show the points **p1** and **p2** at each iteration, with a delay of **d** [sec].
'plot'	show the points **p1** and **p2** at each iteration, with a delay of 0.5 [sec].
'maxtheta', **T**	limit the change in rotation at each step to **T** (default 0.05 rad)
'maxiter', N	stop after N iterations (default 100)
'mindelta', **T**	stop when the relative change in error norm is less than **T** (default 0.001)
'distthresh', **T**	eliminate correspondences more than **T** x the median distance at each iteration.

Example

Create a 3D point cloud

```
p1 = randn(3,20);
```

Transform it by an arbitrary amount

```
T = transl(1,2,3)*eul2tr(0.1, 0.2, 0.3)
p2 = homtrans( T, p1);
```

Perform **icp** to determine the transformation that maps p1 to p2

```
icp(p1, p2)
```

Notes

- Does not require knowledge of correspondence between the points.
  - The point sets may have different numbers of points.
  - Points in either set may have no corresponding point.
- Points can be 2- or 3-dimensional.
- For noisy data setting distthresh and maxtheta can help to prevent the solution from diverging.

Reference

- "A method for registration of 3D shapes", P.Besl and H.McKay, IEEETrans. Pattern Anal. Mach. Intell., vol. 14, no. 2, pp. 239-256, Feb. 1992.

---

idecimate

an image

s = **idecimate**(**im**, **m**) is a decimated version of the image **im** whose size is reduced by **m** (an integer) in both dimensions. The image is smoothed with a Gaussian kernel with standard deviation **m**/2 then subsampled.

s = **idecimate**(**im**, **m**, **sd**) as above but the standard deviation of the smoothing kernel is set to **sd**.

s = **idecimate**(**im**, **m**, []) as above but no smoothing is applied prior to decimation.

Notes

- If the image has multiple planes, each plane is decimated.
- Smoothing is used to eliminate aliasing artifacts and the standard deviation should be chosen as a function of the maximum spatial frequency in the image.

See also

iscale, ismooth, ireplicate

---

idilate

Morphological dilation

out = **idilate**(**im**, **se**, **options**) is the image **im** after morphological dilation with the structuring element **se**.

out = **idilate**(**im**, **se**, **n**, **options**) as above but the structuring element **se** is applied **n** times, that is **n** dilations.

Options

'border'	the border value is replicated (default)
'none'	pixels beyond the border are not included in the window
'trim'	output is not computed for pixels where the structuring element crosses the image border, hence output image had reduced dimensions.
'wrap'	the image is assumed to wrap around, left to right, top to bottom.

Notes

- Cheaper to apply a smaller structuring element multiple times than one large one, the effective structuring element is the Minkowski sum of the structuring element with itself **n** times.
- Windowing options of IMORPH can be passed.

Reference

- Robotics, Vision & Control, Section 12.5, P. Corke, Springer 2011.

See also

ierode, iclose, iopen, imorph

---

idisp

image display tool

idisp(**im**, **options**) displays an image and allows interactive investigation of pixel values, linear profiles, histograms and zooming. The image is displayed in a figure with a toolbar across the top. If **im** is a cell array of images, they are first concatenated (horizontally).

User interface

- Left clicking on a pixel will display its value in a box at the top.
- The "line" button allows two points to be specified and a new figure displays intensity along a line between those points.
- The "histo" button displays a histogram of the pixel values in a new figure. If the image is zoomed, the histogram is computed over only those pixels in view.
- The "zoom" button requires a left-click and drag to specify a box which defines the zoomed view.
- The "colormap" button is displayed only for greyscale images, and is a popup button that allows different color maps to be selected.

Options

'nogui'	don't display the GUI
'noaxes'	don't display axes on the image
'noframe'	don't display axes or frame on the image
'plain'	don't display axes, frame or GUI
'axis', A	display the image in the axes given by handle A, the 'nogui' option is enforced.
'here'	display the image in the current axes
'title', T	put the text T in the title bar of the window
'clickfunc', F	invoke the function handle F(x,y) on a down-click in the window
'ncolors', N	number of colors in the color map (default 256)
'bar'	add a color bar to the image
'print', F	write the image to file F in EPS format
'square'	display aspect ratio so that pixels are square
'wide'	make figure full screen width, useful for displaying stereo pair
'flatten'	display image planes (colors or sequence) as horizontally adjacent images
'ynormal'	y-axis increases upward, image is inverted
'histeq'	apply histogram equalization
'cscale', C	C is a 2-vector that specifies the grey value range that spans the colormap.
'xydata', XY	XY is a cell array whose elements are vectors that span the x- and y-axes respectively.
'colormap', C	set the colormap to C ($N \times 3$)
'grey'	color map: greyscale unsigned, zero is black, maximum value is white
'invert'	color map: greyscale unsigned, zero is white, maximum value is black
'signed'	color map: greyscale signed, positive is blue, negative is red, zero is black
'invsigned'	color map: greyscale signed, positive is blue, negative is red, zero is white
'random'	color map: random values, highlights fine structure
'dark'	color map: greyscale unsigned, darker than 'grey', good for superimposed graphics
'new'	create a new figure

Notes

- Is a wrapper around the MATLAB builtin function IMAGE. See the MATLAB help on "Display Bit-Mapped Images" for details of color mapping.
- Color images are displayed in MATLAB true color mode: pixel triples map to display RGB values. (0,0,0) is black, (1,1,1) is white.
- Greyscale images are displayed in indexed mode: the image pixel value is mapped through the color map to determine the display pixel value.
- For grey scale images the minimum and maximum image values are mapped to the first and last element of the color map, which by default ('greyscale') is the range black to white. To set your own scaling between displayed grey level and pixel value use the 'cscale' option.
- The title of the figure window by default is the name of the variable passed in as the image, this can't work if the first argument is an expression.

Examples

Display 2 images side by side

```
idisp({im1, im2})
```

Display image in a subplot

```
subplot(211)
idisp(im, 'axis', gca);
```

Call a user function when you click a pixel

```
idisp(im, 'clickfunc', @(x,y) fprintf('hello %d %d\n', x,y))
```

Set a colormap, in this case a MATLAB builtin one

```
idisp(im, 'colormap', cool);
```

Display an image which contains a map of a region, perhaps an obstacle grid, that spans real world dimensions x, y in the range -10 to 10.

```
idisp(map, 'xyscale', {[-10 10], [-10 10]});
```

See also

image, caxis, colormap, iconcat

---

idisplabel

Display an image with mask

idisplabel(**im**, **labelimage**, **labels**) displays only those image pixels which belong to a specific class. **im** is a greyscale ($H \times W$) or color ($H \times W \times 3$) image, and **labelimage** ($H \times W$) contains integer pixel class labels for the corresponding pixels in **im**. The pixel classes to be displayed are given by **labels** which is either a scalar or a vector of class labels. Non-selected pixels are displayed as white by default.

idisplabel(**im**, **labelimage**, **labels**, **bg**) as above but the grey level of the non-selected pixels is specified by **bg** in the range 0 to 1 for a float image or 0 to 255 for a uint8 image..

Example

We will segment the image flowers into 7 color classes

```
cls = colorkemans(flowers, 7);
```

where the matrix cls is the same size as flowers and the elements are the corresponding pixel class, a value in the range 1 to 7. To display pixels of class 5 we use


```
idisplabel(flowers, cls, 5)
```

and to display pixels belong to class 1 or 5 we use

```
idisplabel(flowers, cls, [1 5])
```

See also

iblobs, icolorize, colorseg

---

idouble

Convert integer image to double

imd = **idouble**(**im**) is an image with double precision elements in the range 0 to 1 corresponding to the elements of **im**. The integer pixels **im** are assumed to span the range 0 to the maximum value of their integer class.

Notes

- Works for an image with arbitrary number of dimensions, eg. a color image or image sequence.
- There is a linear mapping (scaling) of the values of **imd** to **im**.

See also

iint, cast

---

iendpoint

Find end points in a binary skeleton image

out = **iendpoint**(**im**) is a binary image where pixels are set if the corresponding pixel in the binary image **im** is the end point of a single-pixel wide line such as found in an image skeleton. Computed using the hit-or-miss morphological operator.

References

- Robotics, Vision & Control, Section 12.5.3 P. Corke, Springer 2011.

See also

itriplepoint, ithin, hitormiss

---

ierode

Morphological erosion

out = **ierode**(**im**, **se**, **options**) is the image **im** after morphological erosion with the structuring element **se**.

out = **ierode**(**im**, **se**, **n**, **options**) as above but the structuring element **se** is applied **n** times, that is **n** erosions.

Options

'border'	the border value is replicated (default)
'none'	pixels beyond the border are not included in the window
'trim'	output is not computed for pixels where the structuring element crosses the image border, hence output image had reduced dimensions.
'wrap'	the image is assumed to wrap around, left to right, top to bottom.

Notes

- Cheaper to apply a smaller structuring element multiple times than one large one, the effective structuing element is the Minkowski sum of the structuring element with itself **n** times.
- Windowing options of IMORPH can be passed.

Reference

- Robotics, Vision & Control, Section 12.5, P. Corke, Springer 2011.

See also

idilate, iclose, iopen, imorph

---

igamm

correction

out = **igamm**(**im**, **gamma**) is a gamma corrected version of the image **im**. All pixels are raised to the power **gamma**. Gamma encoding can be performed with **gamma** > 1 and decoding with **gamma** < 1.

out = **igamm**(**im**, 'sRGB') is a gamma decoded version of **im** using the sRGB decoding function (JPEG images sRGB encoded).

Notes

- This function was once called igamma(), but that name taken by MATLAB method for double class objects.
- Gamma decoding should be applied to any color image prior to colometric operations.
- The exception to this is colorspace conversion using COLORSPACE which expects RGB images to be gamma encoded.
- Gamma encoding is typically performed in a camera with **gamma**=0.45.
- Gamma decoding is typically performed in the display with **gamma**=2.2.
- For images with multiple planes the gamma correction is applied to all planes.
- For images sequences the gamma correction is applied to all elements.
- For images of type double the pixels are assumed to be in the range 0 to 1.
- For images of type int the pixels are assumed in the range 0 to the maximum value of their class. Pixels are converted first to double, processed, then converted back to the integer class.

See also

iread, colorspace

---

igraphseg

Graph-based image segmentation

L = **igraphseg**(**im**, **k**, **min**) is a graph-based segmentation of the color image **im** ($H \times W \times 3$). **L** ($H \times W$) is an image where each element is the label assigned to the corresponding pixel in **im**. **k** is the scale parameter, and a larger value indicates a preference for larger regions, **min** is the minimum region size (pixels).

L = **igraphseg**(**im**, **k**, **min**, **sigma**) as above and **sigma** is the width of a Gaussian which is used to initially smooth the image (default 0.5).

[**L**,**nreg**] = **igraphseg**(**im**, **k**, **min**, **sigma**) as above but **nreg** is the number of regions found.

Example

```
im = iread('58060.jpg');
[labels,maxval] = igraphseg(im, 1500, 100, 0.5);
idisp(labels)
```

Reference

"Efficient graph-based image segmentation", P. Felzenszwalb and D. Huttenlocher, Int. Journal on Computer Vision, vol. 59, pp. 167181, Sept. 2004.

Notes

- Requires a color uint8 image.
- The hardwork is done by a MEX file in contrib/graphseg.
- With zero smoothing the number of regions can be massive and can crash MATLAB.

Author

Pedro Felzenszwalb, 2006.

See also

ithresh, imser

ihist

Image histogram

ihist(**im**, **options**) displays the image histogram. For an image with multiple planes the histogram of each plane is given in a separate subplot.

H = **ihist**(**im**, **options**) is the image histogram as a column vector. For an image with multiple planes **H** is a matrix with one column per image plane.

[**H**,**x**] = **ihist**(**im**, **options**) as above but also returns the bin coordinates as a column vector **x**.

Options

'nbins'	number of histogram bins (default 256)
'cdf'	compute a cumulative histogram
'normcdf'	compute a normalized cumulative histogram, whose maximum value is one
'sorted'	histogram but with occurrence sorted in descending magnitude order. Bin coordinates **x** reflect this sorting.

Example

```
[h,x] = ihist(im);
bar(x,h);

[h,x] = ihist(im, 'normcdf');
plot(x,h);
```

Notes

- For a uint8 image the MEX function FHIST is used (if available)
  - The histogram always contains 256 bins
  - The bins spans the greylevel range 0-255.
- For a floating point image the histogram spans the greylevel range 0-1.
- For floating point images all NaN and Inf values are first removed.

See also

hist

iint

Convert image to integer class

out = **iint**(**im**) is an image with unsigned 8-bit integer elements in the range 0 to 255 corresponding to the elements of the image **im**.

out = **iint**(**im**, **class**) as above but the output pixels belong to the integer class **class**.

Examples

Convert double precision image to 8-bit unsigned integer

```
im = rand(50, 50);
out = iint(im);
```

Convert double precision image to 16-bit unsigned integer

```
im = rand(50, 50);
out = iint(im, 'uint16');
```

Convert 8-bit unsigned integer image to 16-bit unsigned integer

```
im = randi(255, 50, 50, 'uint8');
out = iint(im, 'uint16');
```

Notes

- Works for an image with arbitrary number of dimensions, eg. a color image or image sequence.
- If the input image is floating point (single or double) the pixel values are scaled from an input range of [0,1] to a range spanning zero to the maximum positive value of the output integer class.
- If the input image is an integer class then the pixels are cast to change type but not their value.

See also

idouble

iisum

Sum of integral image

s = **iisum**(**ii**, **u1**, **v1**, **u2**, **v2**) is the sum of pixels in the rectangular image region defined by its top-left (**u1**,**v1**) and bottom-right (**u2**,**v2**). **ii** is a precomputed integral image.

See also

intgimage

---

ilabel

Label an image

L = **ilabel**(**im**) is a label image that indicates connected components within the image **im** ($H \times W$). Each pixel in **L** ($H \times W$) is an integer label that indicates which connected region the corresponding pixel in **im** belongs to. Region labels are in the range 1 to M.

[**L**,**m**] = **ilabel**(**im**) as above but returns the value of the maximum label value.

[**L**,**m**,**parents**] = **ilabel**(**im**) as above but also returns region hierarchy information. The value of **parents**(I) is the label of the parent, or enclosing, region of region I. A value of 0 indicates that the region has no single enclosing region, for a binary image this means the region touches the edge of the image, for a multilevel image it means that the region touches more than one other region.

[**L**,**maxlabel**,**parents**,**class**] = **ilabel**(**im**) as above but also returns the class of pixels within each region. The value of **class**(I) is the value of the pixels that comprise region I.

[**L**,**maxlabel**,**parents**,**class**,**edge**] = **ilabel**(**im**) as above but also returns the edge-touch status of each region. If **edge**(I) is 1 then region I touches edge of the image, otherwise it does not.

Notes

- This algorithm is variously known as region labelling, connectivity analysis, connected component analysis, blob labelling.
- All pixels within a region have the same value (or class).
- This is a "low level" function, IBLOBS is a higher level interface.
- Is a MEX file.

- The image can be binary or greyscale.
- Connectivity is only performed in 2 dimensions.
- Connectivity is performed using 4 nearest neighbours by default.
  - To use 8-way connectivity pass a second argument of 8, eg. **ilabel**(**im**, 8).
  - 8-way connectivity introduces ambiguities, a chequerboard is two blobs.

See also

iblobs, imoments

---

iline

Draw a line in an image

out = **iline**(**im**, **p1**, **p2**) is a copy of the image **im** with a single-pixel thick line drawn between the points **p1** and **p2**, each a 2-vector [U,V]. The pixels on the line are set to 1.

out = **iline**(**im**, **p1**, **p2**, **v**) as above but the pixels on the line are set to **v**.

Notes

- Uses the Bresenham algorithm.
- Only works for greyscale images.
- The line looks jagged since no anti-aliasing is performed.

See also

bresenham, iprofile, ipaste

---

im2col

Convert an image to pixel per row format

out = **im2col**(**im**) is a matrix ($N \times P$) where each row represents a single of the image **im** ($H \times W \times P$). The pixels are in image column order (ie. column 1, column 2 etc) and there are N=$W \times H$ rows.

out = **im2col**(**im**, **mask**) as above but only includes pixels if:

- the corresponding element of **mask** ($H \times W$) is non-zero
- the corresponding element of **mask** (N) is non-zero where N=$H \times W$
- the pixel index is included in the vector **mask**

See also

col2im

ImageSource

Abstract class for image sources

An abstract superclass for implementing image sources.

Methods

grab	Aquire and return the next image
close	Close the image source
iscolor	True if image is color
size	Size of image
char	Convert image source parameters to human readable string
display	Display image source parameters in human readable form

See also

AxisWebCamera, video, Movie

ImageSource.ImageSource

Image source constructor

i = ImageSource(**options**) is an ImageSource object that holds parameters related to acquisition from some particular image source.

Options

'width', W	Set image width to W
'height', H	Set image height to H
'uint8'	Return image with uint8 pixels (default)
'int16'	Return image with int16 pixels
'int32'	Return image with int32 pixels
'float'	Return image with float pixels
'double'	Return image with double precision pixels
'grey'	Return image is greyscale
'gamma', G	Apply gamma correction with gamma=G
'scale', S	Subsample the image by S in both directions.

ImageSource.display

Display value

I.display() displays the state of the image source object in human readable form.

Notes

- This method is invoked implicitly at the command line when the result of an expression is an ImageSource object and the command has no trailing semicolon.

---

imatch

Template matching

xm = imatch(**im1**, **im2**, **u**, **v**, **H**, **s**) is the position of the matching subimage of **im1** (template) within the image **im2**. The template in **im1** is centred at (**u**,**v**) and its half-width is **H**.

The template is searched for within **im2** inside a rectangular region, centred at (**u**,**v**) and whose size is a function of **s**. If **s** is a scalar the search region is [-**s**, **s**, -**s**, **s**] relative to (**u**,**v**). More generally **s** is a 4-vector **s**=[umin, umax, vmin, vmax] relative to (**u**,**v**).

The return value is **xm**=[DU,DV,CC] where (DU,DV) are the u- and v-offsets relative to (**u**,**v**) and CC is the similarity score for the best match in the search region.

[**xm**,**score**] = **imatch**(**im1**, **im2**, **u**, **v**, **H**, **s**) as above but also returns a matrix of matching score values for each template position tested. The rows correspond to horizontal positions of the template, and columns the vertical position. The centre element corresponds to (**u**,**v**).

Example

Consider a sequence of images im(:,:,N) and we find corner points in the k'th image

```
corners = icorner(im(:,:,k), 'nfeat', 20);
```

Now, for each corner we look for the 11×11 patch of surrounding pixels in the next image, by searching within a 21×21 region

```
for corner=corners

xm = imatch(im(:,:,k), im(:,:,k+1), 5, 10);
if xm(3) > 0.8

fprintf('feature (%f,%f) moved by (%f,%f) pixels)\n', ...

corner.u, corner.v, xm(1), xm(2) );

end

end
```

Notes

- Useful for tracking a template in an image sequence where **im1** and **im2** are consecutive images in a template and (**u**,**v**) is the coordinate of a corner point in **im1**.
- Is a MEX file.
- **im1** and **im2** must be the same size.
- ZNCC (zero-mean normalized cross correlation) matching is used as the similarity measure. A perfect match score is 1.0 but anything above 0.8 is typically considered to be a good match.

See also

isimilarity

imeshgrid

Domain matrices for image

[**u**,**v**] = **imeshgrid**(**im**) are matrices that describe the domain of image **im** and can be used for the evaluation of functions over the image. **u** and **v** are the same szie as **im**. The element **u**(v,u) = u and **v**(v,u) = v.

[**u**,**v**] = **imeshgrid**(**im**, **n**) as above but...

[**u**,**v**] = **imeshgrid**(**w**, **H**) as above but the domain is **w** $\times$ **H**.

[**u**,**v**] = **imeshgrid**(**size**) as above but the domain is described size which is scalar **size** $\times$ **size** or a 2-vector [**w H**].

See also

meshgrid

---

imoments

Image moments

f = **imoments**(**im**) is a RegionFeature object that describes the greyscale moments of the image **im**.

f = **imoments**(**u**, **v**) as above but the moments are computed from the pixel coordinates given as vectors **u** ($N \times 1$) and **v** ($N \times 1$). All pixels are equally weighted and is effectively a binary image.

f = **imoments**(**u**, **v**, **w**) as above but the pixels have weights given by the vector **w** and is effectively a greyscale image.

Properties

The RegionFeature object has many properties including:

uc	centroid, horizontal coordinate
vc	centroid, vertical coordinate
area	the number of pixels
a	major axis length of equivalent ellipse
b	minor axis length of equivalent ellipse
theta	angle of major ellipse axis to horizontal axis
shape	aspect ratio b/a (always <= 1.0)
moments	a structure containing moments of order 0 to 2, the elements are m00, m10, m01, m20, m02, m11.

See RegionFeature help for more details.

Notes

- For a binary image the zeroth moment is the number of non-zero pixels, or its area.
- This function does not perform connectivity it considers all non-zero pixels in the image. If connected regions are required then use IBLOBS instead.

See also

RegionFeature, iblobs

---

imono

Convert color image to monochrome

out = **imono**(**im**, **options**) is a greyscale equivalent to the color image **im**.

Options

'r601'	ITU recommendation 601 (default)
'r709'	ITU recommendation 709
'value'	HSV value component

Notes

- This function returns a greyscale image whether passed a color or a greyscale image. If a greyscale image is passed it is simply returned.
- Can convert a color image sequence (HxWx3xN) to a monochrome sequence ($H \times W \times N$).

See also

colorize, icolor, colorspace

---

imorph

Morphological neighbourhood processing

out = **imorph**(**im**, **se**, **op**) is the image **im** after morphological processing with the operator **op** and structuring element **se**.

The structuring element **se** is a small matrix with binary values that indicate which elements of the template window are used in the operation.

The operation **op** is:

'min'	minimum value over the structuring element
'max'	maximum value over the structuring element
'diff'	maximum - minimum value over the structuring element
'plusmin'	the minimum of the pixel value and the pixelwise sum of the structuring element and source neighbourhood.

out = **imorph**(**im**, **se**, **op**, **edge**) as above but performance of edge pixels can be controlled. The value of **edge** is:

'border'	the border value is replicated (default)
'none'	pixels beyond the border are not included in the window
'trim'	output is not computed for pixels where the structuring element crosses the image border, hence output image had reduced dimensions.
'wrap'	the image is assumed to wrap around, left to right, top to bottom.

Notes

- Is a MEX file.
- Performs greyscale morphology.
- The structuring element should have an odd side length.
- For binary image 'min' = EROSION, 'max' = DILATION.
- The 'plusmin' operation can be used to compute the distance transform.
- The input can be logical, uint8, uint16, float or double, the output is always double

See also

irank, ivar, hitormiss, iopen, iclose, dtransform

---

imser

Maximally stable extremal regions

label = **imser**(**im**, **options**) is a segmentation of the greyscale image **im** ($H \times W$) based on maximally stable extremal regions. **label** ($H \times W$) is an image where each element is the integer label assigned to the corresponding pixel in **im**. The labels are consecutive integers starting at zero.

[**label**,**nreg**] = **imser**(**im**, **options**) as above but **nreg** is the number of regions found, or one plus the maximum value of **label**.

Options

'dark'	looking for dark features against a light background (default)
'light'	looking for light features against a dark background

Example

```
im = iread('castle_sign2.png', 'grey', 'double');
[label,n] = imser(im, 'light');
idisp(label)
```

Notes

- Is a wrapper for vl_mser, part of VLFeat (vlfeat.org), by Andrea Vedaldi and Brian Fulkerson.
- vl_mser is a MEX file.

Reference

"Robust wide-baseline stereo from maximally stable extremal regions", J. Matas, O. Chum, M. Urban, and T. Pajdla, Image and Vision Computing, vol. 22, pp. 761-767, Sept. 2004.

See also

ithresh, igraphseg

---

inormhist

Histogram normalization

out = **inormhist**(**im**) is a histogram normalized version of the image **im**.

Notes

- Highlights image detail in dark areas of an image.
- The histogram of the normalized image is approximately uniform, that is, all grey levels ae equally likely to occur.

See also

ihist

---

intgimage

Compute integral image

out = **intimage**(**im**) is an integral image corresponding to **im**.

Integral images can be used for rapid computation of summations over rectangular regions.

Examples

Create integral images for sum of pixels over rectangular regions

```
i = intimage(im);
```

Create integral images for sum of pixel squared values over rectangular regions

```
i = intimage(im.^2);
```


See also

iisum

---

invcamcal

camera calibration

c = **invcamcal**(**C**)

Decompose, or invert, a 3x4camera calibration matrix **C**.

The result is a camera object with the following parameters set:

```
f
sx, sy  (with sx=1)
(u0, v0)  principal point

Tcam is the homog xform of the world origin wrt camera
```

Since only f.sx and f.sy can be estimated we set sx = 1.

REF: Multiple View Geometry, Hartley&Zisserman, p 163-164

SEE ALSO: camera

---

iopen

Morphological opening

out = **iopen**(**im**, **se**, **options**) is the image **im** after morphological opening with the structuring element **se**. This is a morphological erosion followed by dilation.

out = **iopen**(**im**, **se**, **n**, **options**) as above but the structuring element **se** is applied **n** times, that is **n** erosions followed by **n** dilations.

Notes

- For binary image an opening operation can be used to eliminate small white noise regions.
- It is cheaper to apply a smaller structuring element multiple times than one large one, the effective structuring element is the Minkowski sum of the structuring element with itself **n** times.

- Windowing options of IMORPH can be passed. By default output image is same size as input image.

See also

iclose, idilate, ierode, imorph

---

ipad

Pad an image with constants

out = **ipad**(**im**, **sides**, **n**) is a padded version of the image **im** with a block of NaN values **n** pixels wide on the sides of **im** as specified by **sides**.

out = **ipad**(**im**, **sides**, **n**, **v**) as above but pads with pixels of value **v**.

sides is a string containing one or more of the characters:

't'	top
'b'	bottom
'l'	left
'r'	right

Examples

Add a band of zero pixels 20 pixels high across the top of the image:

```
ipad(im, 't', 20, 0)
```

Add a band of white pixels 10 pixels wide on all sides of the image:

```
ipad(im, 'tblr', 10, 255)
```

Notes

- Not a tablet computer.

---

ipaste

Paste an image into an image

out = ipaste(**im**, **im2**, **p**, **options**) is the image **im** with the subimage **im2** pasted in at the position **p**=[U,V].

Options

'centre'	The pasted image is centred at **p**, otherwise **p** is the top-left corner of the subimage in **im** (default)
'zero'	the coordinates of **p** start at zero, by default 1 is assumed
'set'	**im2** overwrites the pixels in **im** (default)
'add'	**im2** is added to the pixels in **im**
'mean'	**im2** is set to the mean of pixel values in **im2** and **im**

Notes

- Pixels outside the pasted in region are unaffected.

See also

iline

---

ipixswitch

Pixelwise image merge

out = ipixswitch(**mask**, **im1**, **im2**) is an image where each pixel is selected from the corresponding pixel in **im1** or **im2** according to the corresponding pixel values in **mask**. If the element of **mask** is zero **im1** is selected, otherwise **im2** is selected.

im1 or **im2** can contain a color descriptor which is one of:

- A scalar value corresponding to a greyscale
- A 3-vector corresponding to a color value
- A string containing the name of a color which is found using COLORNAME.

ipixswitch(**mask**, **im1**, **im2**) as above but the result is displayed.

Example

Read a uint8 image

```
im = iread('lena.pgm');
```

and set high valued pixels to red

```
a = ipixswitch(im>120, im, uint8([255 0 0]));
```

The result is a uint8 image since both arguments are uint8 images.

```
a = ipixswitch(im>120, im, [1 0 0]);
```

The result is a double precision image since the color specification is a double.

```
a = ipixswitch(im>120, im, 'red');
```

The result is a double precision image since the result of colorname is a double precision 3-vector.

Notes

- **im1**, **im2** and **mask** must all have the same number of rows and columns.
- If **im1** and **im2** are both greyscale then **out** is greyscale.
- If either of **im1** and **im2** are color then **out** is color.
- If either one image is double and one is integer then the integer image is first converted to a double image.

See also

colorize, colorname

---

iprofile

Extract pixels along a line

v = **iprofile**(**im**, **p1**, **p2**) is a vector of pixel values extracted from the image **im** ($H \times W \times P$) between the points **p1** (2×1) and **p2** (2×1). **v** ($N \times P$) has one row for each point along the line and the row is the pixel value which will be a vector for a multi-plane image.

[**p**,**uv**] = **iprofile**(**im**, **p1**, **p2**) as above but also returns the coordinates of the pixels for each point along the line. Each row of **uv** is the pixel coordinate (u,v) for the corresponding row of **p**.

Notes

- The Bresenham algorithm is used to find points along the line.

See also

bresenham, iline

---

ipyramid

Pyramidal image decomposition

out = **ipyramid**(**im**) is a pyramid decomposition of input image **im** using Gaussian smoothing with standard deviation of 1. **out** is a cell array of images each one having dimensions half that of the previous image. The pyramid is computed down to a non-halvable image size.

out = **ipyramid**(**im**, **sigma**) as above but the Gaussian standard deviation is **sigma**.

out = **ipyramid**(**im**, **sigma**, **n**) as above but only **n** levels of the pyramid are computed.

Notes

- Works for greyscale images only.

See also

iscalespace, idecimate, ismooth

---

irank

Rank filter

out = **irank**(**im**, **order**, **se**) is a rank filtered version of **im**. Only pixels corresponding to non-zero elements of the structuring element **se** are ranked and the **order**'th value in rank becomes the corresponding output pixel value. The highest rank, the maximum, is **order**=1.

out = **irank**(**image**, **se**, **op**, **nbins**) as above but the number of histogram bins can be specified.

out = **irank**(**image**, **se**, **op**, **nbins**, **edge**) as above but the processing of edge pixels can be controlled. The value of **edge** is:

'border'	the border value is replicated (default)
'none'	pixels beyond the border are not included in the window
'trim'	output is not computed for pixels whose window crosses the border, hence output image had reduced dimensions.
'wrap'	the image is assumed to wrap around left-right, top-bottom.

Examples

5×5 median filter, 25 elements in the window, the median is the 12thn in rank

```
irank(im, 12, ones(5,5));
```

3×3 non-local maximum, find where a pixel is greater than its eight neighbours

```
se = ones(3,3); se(2,2) = 0;
im > irank(im, 1, se);
```

Notes

- The structuring element should have an odd side length.
- Is a MEX file.
- The median is estimated from a histogram with **nbins** (default 256).
- The input can be logical, uint8, uint16, float or double, the output is always double

See also

imorph, ivar, iwindow

---

iread

Read image from file

im = **iread**() presents a file selection GUI from which the user can select an image file which is returned as a matrix. On subsequent calls the initial folder is as set on the last call.

im = **iread**([], OPTIONS) as above but allows options to be specified.

im = iread(path, options) as above but the GUI is set to the folder specified by **path**. If the path is not absolute it is searched for on the MATLAB search path.

im = iread(file, options) reads the specified image file and returns a matrix. If the path is not absolute it is searched for on MATLAB search path.

The image can be greyscale or color in any of a wide range of formats supported by the MATLAB IMREAD function.

Wildcards are allowed in file names. If multiple files match a 3D or 4D image is returned where the last dimension is the number of images in the sequence.

Options

'uint8'	return an image with 8-bit unsigned integer pixels in the range 0 to 255
'single'	return an image with single precision floating point pixels in the range 0 to 1.
'double'	return an image with double precision floating point pixels in the range 0 to 1.
'grey'	convert image to greyscale, if it's color, using ITU rec 601
'grey_709'	convert image to greyscale, if it's color, using ITU rec 709
'gamma', G	apply this gamma correction, either numeric or 'sRGB'
'reduce', R	decimate image by R in both dimensions
'roi', R	apply the region of interest R to each image, where R=[umin umax; vmin vmax].

Examples

Read a color image and display it

```
>> im = iread('lena.png');
>> about im
im [uint8] : 512x512x3 (786.4 kB)
>> idisp(im);
```

Read a greyscale image sequence

```
>> im = iread('seq/*.png');
>> about im
im [uint8] : 512x512x9 (2.4 MB)
>> ianimate(im, 'loop');
```

Notes

- A greyscale image is returned as an $H \times W$ matrix
- A color image is returned as an $H \times W \times 3$ matrix
- A greyscale image sequence is returned as an $H \times W \times N$ matrix where N is the sequence length
- A color image sequence is returned as an HxWx3xN matrix where N is the sequence length

See also

idisp, ianimate, imono, igamma, imread, imwrite, path

---

irectify

Rectify stereo image pair

[**out1**,**out2**] = **irectify**(**f**, **m**, **im1**, **im2**) is a rectified pair of images corresponding to **im1** and **im2**. **f** (3×3) is the fundamental matrix relating the two views and **m** is a FeatureMatch object containing point correspondences between the images.

[**out1**,**out2**,**h1**,**h2**] = **irectify**(**f**, **m**, **im1**, **im2**) as above but also returns the homographies **h1** and **h2** that warp **im1** to **out1** and **im2** to **out2** respectively.

Notes

- The resulting image pair are epipolar aligned, equivalent to the view if the two original camera axes were parallel.
- Rectified images are required for dense stereo matching.
- The effect of lense distortion is not removed, use the camera calibration toolbox to unwarp each image prior to rectification.
- The resulting images may have negative disparity.
- Some output pixels may have no corresponding input pixels and will be set to NaN.

See also

FeatureMatch, istereo, homwarp, CentralCamera

---

ireplicate

Expand image

out = **ireplicate**(**im**, **k**) is an expanded version of the image $(H \times W)$ where each pixel is replicated into a **k** $\times$ **k** tile. If **im** is $H \times W$ the result is (KH)x(KW).

See also

idecimate, iscale

---

iroi

Extract region of interest

out = **iroi**(**im**,**rect**) is a subimage of the image **im** described by the rectangle **rect**=[umin,umax; vmin,vmax].

out = **iroi**(**im**,**C**,**s**) as above but the region is centered at **C**=(U,V) and has a size **s**. If **s** is scalar then W=H=**s** otherwise **s**=(W,H).

out = **iroi**(**im**) as above but the image is displayed and the user is prompted to adjust a rubber band box to select the region of interest.

[**out**,**rect**] = **iroi**(**im**) as above but returns the coordinates of the selected region of interest **rect**=[umin umax;vmin vmax].

[**out**,**u**,**v**] = **iroi**(**im**) as above but returns the range of **u** and **v** coordinates in the selected region of interest, as vectors.

Notes

- If no output argument is specified then the result is displayed in a new window.

See also

idisp

---

irotate

Rotate image

out = **irotate**(**im**, **angle**, **options**) is a version of the image **im** that has been rotated about its centre.

Options

'outsize', S	set size of output image to $H \times W$ where S=[W,H]
'crop'	return central part of image, same size as **im**
'scale', S	scale the image size by S (default 1)
'extrapval', V	set background pixels to V (default 0)
'smooth', S	initially smooth the image with a Gaussian of standard deviation S

Notes

- Rotation is defined with respect to a z-axis which is into the image.
- Counter-clockwise is a positive angle.
- The pixels in the corners of the resulting image will be undefined and set to the 'extrapval'.

See also

iscale

---

isamesize

Automatic image trimming

out = **isamesize**(**im1**, **im2**) is an image derived from **im1** that has the same dimensions as **im2**. This is achieved by cropping and scaling.

out = **isamesize**(**im1**, **im2**, **bias**) as above but **bias** controls which part of the image is cropped. **bias**=0.5 is symmetric cropping, **bias**<0.5 moves the crop window up or to the left, while **bias**>0.5 moves the crop window down or to the right.

See also

iscale, iroi, itrim

---

iscale

Scale an image

out = iscale(**im**, **s**) is a version of **im** scaled in both directions by **s** which is a real scalar. **s**>1 makes the image larger, **s**<1 makes it smaller.

Options

'outsize', **s**	set size of **out** to $H \times W$ where **s**=[W,H]
'smooth', **s**	initially smooth image with Gaussian of standard deviation **s** (default 1). **s**=[] for no smoothing.

See also

ireplicate, idecimate, irotate

---

iscalemax

Scale space maxima

f = iscalemax(**L**, **s**) is a vector of ScalePointFeature objects which are the maxima, in space and scale, of the Laplacian of Gaussian (LoG) scale-space image sequence **L** ($H \times W \times N$). **s** ($N \times 1$) is a vector of scale values corresponding to each plane of **L**.

If the pixels are considered as cubes in a larger volume, the maxima are those cubes greater than all their 26 neighbours.

Notes

- Features are sorted into descending feature strength.

See also

iscalespace, ScalePointFeature

---

iscalespace

Scale-space image sequence

[**g**,**L**,**s**] = **iscalespace**(**im**, **n**, **sigma**) is a scale space image sequence of length **n** derived from **im** ($H \times W$). The standard deviation of the smoothing Gaussian is **sigma**. At each scale step the variance of the Gaussian increases by **sigma**^{2}. The first step in the sequence is the original image.

g ($H \times W \times$ **n**) is the scale sequence, **L** ($H \times W \times$ **n**) is the absolute value of the Laplacian of Gaussian (LoG) of the scale sequence, corresponding to each step of the sequence, and **s** (**n** $\times$ 1) is the vector of scales.

[**g**,**L**,**s**] = **iscalespace**(**im**, **n**) as above but **sigma**=1.

Examples

Create a scale-space image sequence

```
im = iread('lena.png', 'double', 'grey');
[G,L,s] = iscalespace(im, 50, 2);
```

Then find scale-space maxima, an array of ScalePointFeature objects.

```
f = iscalemax(L, s);
```

Look at the scalespace volume

```
slice(L, [], [], 5:10:50); shading interp
```

Notes

- The Laplacian is approximated by the the difference of adjacent Gaussians.

See also

iscalemax, ismooth, ilaplace, klog

---

iscolor

Test for color image

iscolor(**im**) is true (1) if **im** is a color image, that is, it its third dimension is equal to three.

---

ishomog

Test if SE(3) homogeneous transformation

ishomog(**T**) is true (1) if the argument **T** is of dimension 4×4 or $4 \times 4 \times N$, else false (0).

ishomog(**T**, 'valid') as above, but also checks the validity of the rotation sub-matrix.

Notes

- The first form is a fast, but incomplete, test for a transform is SE(3).
- Does not work for the SE(2) case.

See also

isrot, isvec

---

ishomog2

Test if SE(2) homogeneous transformation

ishomog2(**T**) is true (1) if the argument **T** is of dimension 3×3 or $3 \times 3 \times N$, else false (0).

ishomog2(**T**, 'valid') as above, but also checks the validity of the rotation sub-matrix.

Notes

- The first form is a fast, but incomplete, test for a transform in SE(3).
- Does not work for the SE(3) case.

See also

ishomog, isrot2, isvec

---

isift

SIFT feature extractor

sf = isift(**im**, **options**) is a vector of SiftPointFeature objects representing scale and rotationally invariant interest points in the image **im**.

Options

'nfeat', N	set the number of features to return (default Inf)
'suppress', R	set the suppression radius (default 0)
'id', V	set the image_id of all features

Properties and methods

The SiftPointFeature object has many properties including:

u	horizontal coordinate
v	vertical coordinate
strength	feature strength
descriptor	feature descriptor (128×1)
sigma	feature scale
theta	feature orientation [rad]
image_id	a value passed as an option to isift

The SiftPointFeature object has many methods including:

plot	Plot feature position
plot_scale	Plot feature scale
distance	Descriptor distance
match	Match features
ncc	Descriptor similarity

See SiftPointFeature and PointFeature classes for more details.

Notes

- Greyscale images only, double or integer pixel format.
- Features are returned in descending strength order.
- Wraps a MEX file from www.vlfeat.org
- Corners are processed in order from strongest to weakest.
- If **im** is $H \times W \times N$ it is considered to be an image sequence and F is a cell array with N elements, each of which is the feature vectors for the corresponding image in the sequence.

- The SIFT algorithm is covered by US Patent 6,711,293 (March 23, 2004) held by the Univerity of British Columbia.
- ISURF is a functional equivalent.

Reference

"Distinctive image features from scale-invariant keypoints", David G. Lowe, International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.

See also

SiftPointFeature, isurf, icorner

---

isimilarity

Locate template in image

s = **isimilarity**(**T**, **im**) is an image where each pixel is the ZNCC similarity of the template **T** ($M \times M$) to the $M \times M$ neighbourhood surrounding the corresonding input pixel in **im**. **s** is same size as **im**.

s = **isimilarity**(**T**, **im**, **metric**) as above but the similarity metric is specified by the function **metric** which can be any of @sad, @ssd, @ncc, @zsad, @zssd.

Example

Load an image of Wally/Waldo (the template)

```
T = iread('wally.png', 'double');
```

then load an image of the crowd where he is hiding

```
crowd = iread('wheres-wally.png', 'double');
```

Now search for him using the ZNCC matching measure

```
S = isimilarity(T, crowd, @zncc);
```

and display the similarity

```
idisp(S, 'colormap', 'jet', 'bar')
```

The magnitude at each pixel indicates how well the template centred on that point matches the surrounding pixels. The locations of the maxima are

```
[~,p] = peak2(S, 1, 'npeaks', 5);
```

Now we can display the original scene


```
idisp(crowd)
```

and highlight the most likely places that Wally/Waldo is hiding

```
plot_circle(p, 30, 'fillcolor', 'b', 'alpha', 0.3, ...
'edgecolor', 'none')
plot_point(p, 'sequence', 'bold', 'textsize', 24, ...
'textcolor', 'k', 'Marker', 'none')
```

References

- Robotics, Vision & Control, Section 12.4, P. Corke, Springer 2011.

Notes

- For NCC and ZNCC the maximum in **s** corresponds to the most likely template location. For SAD, SSD, ZSAD and ZSSD the minimum value corresponds to the most likely location.
- Similarity is not computed for those pixels where the template crosses the image boundary, and these output pixels are set to NaN.
- The ZNCC function is a MEX file and therefore the fastest
- User provided similarity metrics can be used, the function accepts two regions and returns a scalar similarity score.

See also

imatch, sad, ssd, ncc, zsad, zssd, zncc

---

isize

Size of image

n = **isize**(**im**,**d**) is the size of the **d**'th dimension of **im**.

[**w**,**H**] = **isize**(**im**) is the image width **w** and height **H**.

wh = **isize**(**im**) is the image size **wh** = [**w H**].

[**w**,**H**,**p**] = **isize**(**im**) is the image width **w**, height **H** and and number of planes **p**. Even if the image has only two dimensions **p** will be one.

Notes

- A simple convenience wrapper on the MATLAB function SIZE.

See also

size

---

ismooth

Gaussian smoothing

out = **ismooth**(**im**, **sigma**) is the image **im** after convolution with a Gaussian kernel of standard deviation **sigma**.

out = **ismooth**(**im**, **sigma**, **options**) as above but the **options** are passed to CONV2.

Options

'full'	returns the full 2-D convolution (default)
'same'	returns **out** the same size as **im**
'valid'	returns the valid pixels only, those where the kernel does not exceed the bounds of the image.

Notes

- By default (option 'full') the returned image is larger than the passed image.
- Smooths all planes of the input image.
- The Gaussian kernel has a unit volume.
- If input image is integer it is converted to float, convolved, then converted back to integer.

See also

iconv, kgauss

---

isobel

Sobel edge detector

out = **isobel**(**im**) is an edge image computed using the Sobel edge operator applied to the image **im**. This is the norm of the vertical and horizontal gradients at each pixel. The Sobel horizontal gradient kernel is:

```
| -1   0   1|
| -2   0   2|
| -1   0   1|
```

and the vertical gradient kernel is the transpose.

[**gx**,**gy**] = **isobel**(**im**) as above but returns the gradient images rather than the gradient magnitude.

out = **isobel**(**im**,**dx**) as above but applies the kernel **dx** and **dx**' to compute the horizontal and vertical gradients respectively.

[**gx**,**gy**] = **isobel**(**im**,**dx**) as above but returns the gradient images rather than the gradient magnitude.

Notes

- Tends to produce quite thick edges.
- The resulting image is the same size as the input image.
- If the kernel **dx** is provided it can be of any size, not just 3×3, and could be generated using KDGAUSS.

See also

ksobel, kdgauss, icanny, iconv

---

isrot

Test if SO(3) rotation matrix

isrot(**R**) is true (1) if the argument is of dimension 3×3 or $3 \times 3 \times N$, else false (0).

isrot(**R**, 'valid') as above, but also checks the validity of the rotation matrix.

Notes

- A valid rotation matrix has determinant of 1.

See also

ishomog, isvec

---

isrot2

Test if SO(2) rotation matrix

isrot2(**R**) is true (1) if the argument is of dimension 2×2 or $2 \times 2 \times N$, else false (0).

isrot2(**R**, 'valid') as above, but also checks the validity of the rotation matrix.

Notes

- A valid rotation matrix has determinant of 1.

See also

ishomog2, isvec

---

istereo

Stereo matching

d = **istereo**(**left**, **right**, **range**, **H**, **options**) is a disparity image computed from the epipolar aligned stereo pair: the left image **left** (**H** $\times$ W) and the right image **right** (**H** $\times$ W). **d** (**H** $\times$ W) is the disparity and the value at each pixel is the horizontal shift of the corresponding pixel in IML as observed in IMR. That is, the disparity d=**d**(v,u) means that the pixel at **right**(v,u-d) is the same world point as the pixel at **left**(v,u).

range is the disparity search range, which can be a scalar for disparities in the range 0 to **range**, or a 2-vector [DMIN DMAX] for searches in the range DMIN to DMAX.

H is the half size of the matching window, which can be a scalar for $N \times N$ or a 2-vector [N,M] for an $N \times M$ window.

[**d**,**sim**] = **istereo**(**left**, **right**, **range**, **H**, **options**) as above but returns **sim** which is the same size as **d** and the elements are the peak matching score for the corresponding elements of **d**. For the default matching metric ZNCC this varies between -1 (very bad) to +1 (perfect).

[**d**,**sim**,**dsi**] = **istereo**(**left**, **right**, **range**, **H**, **options**) as above but returns **dsi** which is the disparity space image ($\mathbf{H} \times W \times N$) where N=DMAX-DMIN+1. The I'th plane is the similarity of IML to IMR shifted to the left by DMIN+I-1.

[**d**,**sim**,**p**] = **istereo**(**left**, **right**, **range**, **H**, **options**) if the 'interp' option is given then disparity is estimated to sub-pixel precision using quadratic interpolation. In this case **d** is the interpolated disparity and **p** is a structure with elements A, B, dx. The interpolation polynomial is s = Ad^2 + Bd + C where s is the similarity score and d is disparity relative to the integer disparity at which s is maximum. **p**.A and **p**.B are matrices the same size as **d** whose elements are the per pixel values of the interpolation polynomial coefficients. **p**.dx is the peak of the polynomial with respect to the integer disparity at which s is maximum (in the range -0.5 to +0.5).

Options

'metric', M	string that specifies the similarity metric to use which is one of 'zncc' (default), 'ncc', 'ssd' or 'sad'.
'interp'	enable subpixel interpolation and **d** contains non-integer values (default false)
'vshift', V	move the right image V pixels vertically with respect to left.

Example

Load the left and right images

```
L = iread('rocks2-l.png', 'reduce', 2);
R = iread('rocks2-r.png', 'reduce', 2);
```

then compute stereo disparity and display it

```
d = istereo(L, R, [40, 90], 3);
idisp(d);
```

References

- Robotics, Vision & Control, Section 14.3, **p**. Corke, Springer 2011.

Notes

- Images must be greyscale.
- Disparity values pixels within a half-window dimension (**H**) of the edges will not be valid and are set to NaN.
- The C term of the interpolation polynomial is not computed or returned.

- The A term is high where the disparity function has a sharp peak.
- Disparity and similarity score can be obtained from the disparity space image by [**sim**,**d**] = max(**dsi**, [], 3)

See also

irectify, stdisp

---

istretch

Image normalization

out = **istretch**(**im**, **options**) is a normalized image in which all pixel values lie in the range 0 to 1. That is, a linear mapping where the minimum value of **im** is mapped to 0 and the maximum value of **im** is mapped to 1.

Options

'max', M	Pixels are mapped to the range 0 to M
'range', R	R(1) is mapped to zero, R(2) is mapped to 1 (or max value).

Notes

- For an integer image the result is a double image in the range 0 to max value.

See also

inormhist

---

isurf

SURF feature extractor

sf = **isurf**(**im**, **options**) returns a vector of SurfPointFeature objects representing scale and rotationally invariant interest points in the image **im**.

The SurfPointFeature object has many properties including:

u	horizontal coordinate
v	vertical coordinate
strength	feature strength
descriptor	feature descriptor (64×1 or 128×1)
sigma	feature scale
theta	feature orientation [rad]

Options

'nfeat', N	set the number of features to return (default Inf)
'thresh', T	set Hessian threshold. Increasing the threshold reduces the number of features computed and reduces computation time.
'octaves', N	number of octaves to process (default 5)
'extended'	return 128-element descriptor (default 64)
'upright'	don't compute rotation invariance
'suppress', R	set the suppression radius (default 0). Features are not returned if they are within R [pixels] of an earlier (stronger) feature.

Example

Load the image

```
im = iread('lena.pgm');
```

Find the 10 strongest SURF features

```
sf = isurf(im, 'nfeat', 10);
```

and overlay them on the original image as blue circles

```
idisp(im);
sf.plot_scale()
```

Notes

- Color images, or sequences, are first converted to greyscale.
- Features are returned in descending strength order
- If **im** is $H \times W \times N$ it is considered to be an image sequence and F is a cell array with N elements, each of which is the feature vectors for the corresponding image in the sequence.
- Wraps an M-file implementation of OpenSurf by D. Kroon (U. Twente) or a MEX-file OpenCV wrapper by Petter Strandmark.
- The sign of the Laplacian is not retained.
- The SURF algorithm is covered by an extensive suite of international patents including US 8,165,401, EP 1850270 held by Toyota, KU Leuven and ETHZ. See http://www.kooaba.com/en/plans_and_pricing/ip_licensing

Reference

"SURF: Speeded Up Robust Features", Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, 2008

See also

SurfPointFeature, isift, icorner

---

isvec

Test if vector

isvec(**v**) is true (1) if the argument **v** is a 3-vector, else false (0).

isvec(**v**, **L**) is true (1) if the argument **v** is a vector of length **L**, either a row- or column-vector. Otherwise false (0).

Notes

- Differs from MATLAB builtin function ISVECTOR, the latter returns true for the case of a scalar, **isvec** does not.
- Gives same result for row- or column-vector, ie. 3×1 or 1×3 gives true.

See also

ishomog, isrot

---

ithin

Morphological skeletonization

out = **ithin**(**im**) is the binary skeleton of the binary image **im**. Any non-zero region is replaced by a network of single-pixel wide lines.

out = **ithin**(**im**,**delay**) as above but graphically displays each iteration of the skeletonization algorithm with a pause of **delay** seconds between each iteration.

References

- Robotics, Vision & Control, Section 12.5.3, P. Corke, Springer 2011.

See also

hitormiss, itriplepoint, iendpoint

---

ithresh

Interactive image threshold

ithresh(**im**) displays the image **im** in a window with a slider which adjusts the binary threshold.

ithresh(**im**, **T**) as above but the initial threshold is set to **T**.

im2 = **ithresh**(**im**) as above but returns the thresholded image after the "done" button in the GUI is pressed.

[**im2**,**T**] = **ithresh**(**im**) as above but also returns the threshold value.

Notes

- Greyscale image only.
- For a uint8 class image the slider range is 0 to 255.
- For a floating point class image the slider range is 0 to 1.0
- The GUI only displays the "done" button if output arguments are requested, otherwise the threshold window operates independently.

See also

idisp

---

itrim

Trim images

This function has two different modes of functionality.

out = **itrim**(**im**, **sides**, **n**) is the image **im** with **n** pixels removed from the image sides as specified by **sides** which is a string containing one or more of the characters:

't'	top
'b'	bottom
'l'	left
'r'	right

[**out1**,**out2**] = **itrim**(**im1**,**im2**) returns the central parts of images **im1** and **im2** as **out1** and **out2** respectively. When images are rectified or warped the shapes can become quite distorted and are embedded in rectangular images surrounded by black of NaN values. This function crops out the central rectangular region of each. It assumes that the undefined pixels in **im1** and **im2** have values of NaN. The same cropping is applied to each input image.

[**out1**,**out2**] = **itrim**(**im1**,**im2**,**T**) as above but the threshold **T** in the range 0 to 1 is used to adjust the level of cropping. The default is 0.5, a higher value will include fewer NaN value in the result (smaller region), a lower value will include more (larger region). A value of 0 will ensure that there are no NaN values in the returned region.

See also

homwarp, irectify

---

itriplepoint

Find triple points

out = **itriplepoint**(**im**) is a binary image where pixels are set if the corresponding pixel in the binary image **im** is a triple point, that is where three single-pixel wide line intersect. These are the Voronoi points in an image skeleton. Computed using the hit-or-miss morphological operator.

References

- Robotics, Vision & Control, Section 12.5.3, P. Corke, Springer 2011.

See also

iendpoint, ithin, hitormiss

---

ivar

Pixel window statistics

out = **ivar**(**im**, **se**, **op**) is an image where each output pixel is the specified statistic over the pixel neighbourhood indicated by the structuring element **se** which should have odd side lengths. The elements in the neighbourhood corresponding to non-zero elements in **se** are packed into a vector on which the required statistic is computed.

The operation **op** is one of:

'var'	variance
'kurt'	Kurtosis or peakiness of the distribution
'skew'	skew or asymmetry of the distribution

out = **ivar**(**im**, **se**, **op**, **edge**) as above but performance at edge pixels can be controlled. The value of **edge** is:

'border'	the border value is replicated (default)
'none'	pixels beyond the border are not included in the window
'trim'	output is not computed for pixels whose window crosses the border, hence output image had reduced dimensions.
'wrap'	the image is assumed to wrap around

Notes

- Is a MEX file.
- The structuring element should have an odd side length.
- The input can be logical, uint8, uint16, float or double, the output is always double

See also

irank, iwindow

---

iwindow

Generalized spatial operator

out = iwindow(**im**, **se**, **func**) is an image where each pixel is the result of applying the function **func** to a neighbourhood centred on the corresponding pixel in **im**. The neighbourhood is defined by the size of the structuring element **se** which should have odd side lengths. The elements in the neighbourhood corresponding to non-zero elements in **se** are packed into a vector (in column order from top left) and passed to the specified function handle **func**. The return value becomes the corresponding pixel value in **out**.

out = iwindow(**image**, **se**, **func**, **edge**) as above but performance of edge pixels can be controlled. The value of **edge** is:

'border'	the border value is replicated (default)
'none'	pixels beyond the border are not included in the window
'trim'	output is not computed for pixels whose window crosses the border, hence output image had reduced dimensions.
'wrap'	the image is assumed to wrap around

Example

Compute the maximum value over a 5×5 window:

```
iwindow(im, ones(5,5), @max);
```

Compute the standard deviation over a 3×3 window:

```
iwindow(im, ones(3,3), @std);
```

Notes

- Is a MEX file.
- The structuring element should have an odd side length.
- Is slow since the function **func** must be invoked once for every output pixel.
- The input can be logical, uint8, uint16, float or double, the output is always double

See also

ivar, irank

kcircle

Circular structuring element

k = **kcircle**(**R**) is a square matrix ($W \times W$) where W=2R+1 of zeros with a maximal centred circular region of radius **R** pixels set to one.

k = **kcircle**(**R**,**w**) as above but the dimension of the kernel is explicitly specified.

Notes

- If **R** is a 2-element vector the result is an annulus of ones, and the two numbers are interpretted as inner and outer radii.

See also

ones, ktriangle, imorph

---

kdgauss

Derivative of Gaussian kernel

k = **kdgauss**(**sigma**) is a 2-dimensional derivative of Gaussian kernel ($W \times W$) of width (standard deviation) **sigma** and centred within the matrix **k** whose half-width H = 3 $\times$ **sigma** and W=2 $\times$ H+1.

k = **kdgauss**(**sigma**, **H**) as above but the half-width is explictly specified.

Notes

- This kernel is the horizontal derivative of the Gaussian, dG/dx.
- The vertical derivative, dG/dy, is **k**'.
- This kernel is an effective edge detector.

See also

kgauss, kdog, klog, isobel, iconv

---

kdog

Difference of Gaussian kernel

k = kdog(**sigma1**) is a 2-dimensional difference of Gaussian kernel equal to KGAUSS(**sigma1**) - KGAUSS(SIGMA2), where **sigma1** > SIGMA2. By default SIGMA2 = 1.6***sigma1**. The kernel is centred within the matrix **k** whose half-width H = $3 \times SIGMA$ and W=$2 \times H$+1.

k = kdog(**sigma1**, **sigma2**) as above but **sigma2** is specified directly.

k = kdog(**sigma1**, **sigma2**, **H**) as above but the kernel half-width is specified.

Notes

- This kernel is similar to the Laplacian of Gaussian and is often used as an efficient approximation.

See also

kgauss, kdgauss, klog, iconv

---

kgauss

Gaussian kernel

k = kgauss(**sigma**) is a 2-dimensional Gaussian kernel of standard deviation **sigma**, and centred within the matrix **k** whose half-width is H=$2 \times$ **sigma** and W=$2 \times H$+1.

k = kgauss(**sigma**, **H**) as above but the half-width **H** is specified.

Notes

- The volume under the Gaussian kernel is one.

See also

kdgauss, kdog, klog, iconv

---

klaplace

Laplacian kernel

k = **klaplace**() is the Laplacian kernel:

```
|0   1   0|
|1  -4   1|
|0   1   0|
```

Notes

- This kernel has an isotropic response to image gradient.

See also

ilaplace, iconv

---

klog

Laplacian of Gaussian kernel

k = **klog**(**sigma**) is a 2-dimensional Laplacian of Gaussian kernel of width (standard deviation) **sigma** and centred within the matrix **k** whose half-width is H=3 $\times$ **sigma**, and W=2 $\times$ H+1.

k = **klog**(**sigma**, **H**) as above but the half-width **H** is specified.

See also

kgauss, kdog, kdgauss, iconv, zcross

---

kmeans

K-means clustering

[**L**,**C**] = **kmeans**(**x**, **k**, **options**) is a **k**-means clustering of multi-dimensional data points **x** ($D \times N$) where N is the number of points, and D is the dimension. The data is

organized into **k** clusters based on Euclidean distance from cluster centres **C** ($D \times$**k**). **L** is a vector ($N \times 1$) whose elements indicates which cluster the corresponding element of **x** belongs to.

[**L**,**C**] = **kmeans**(**x**, **k**, **c0**) as above but the initial clusters **c0** ($D \times$ **k**) is given and column I is the initial estimate of the centre of cluster I.

L = **kmeans**(**x**, **C**) is similar to above but the clustering step is not performed, it is assumed to have been completed previously. **C** ($D \times$ **k**) contains the cluster centroids and **L** ($N \times 1$) indicates which cluster the corresponding element of **x** is closest to.

Options

'random'	initial cluster centres are chosen randomly from the set of data points **x** (default)
'spread'	initial cluster centres are chosen randomly from within the hypercube spanned by **x**.

Reference

"Pattern Recognition Principles", Tou and Gonzalez, Addison-Wesley 1977, pp 94

---

ksobel

Sobel edge detector

k = **ksobel**() is the Sobel x-derivative kernel:

```
|-1  0  1|
|-2  0  2|
|-1  0  1|
```

Notes

- This kernel is an effective horizontal edge detector
- The Sobel vertical derivative is **k**'

See also

isobel

---

ktriangle

Triangular kernel

k = **ktriangle**(**w**) is a triangular kernel within a rectangular matrix **k**. The dimensions **k** are **w** $\times$ **w** if **w** is scalar or **w**(1) wide and **w**(2) high. The triangle is isocles and is full width at the bottom row of the kernel and with its apex in the top row.

Examples

```
>> ktriangle(3)
ans =
|0  1  0|
|0  1  0|
|1  1  1|
```

See also

kcircle

---

lambda2rg

RGB chromaticity coordinates

rgb = **lambda2rg**(**lambda**) is the rg-chromaticity coordinate (1×2) for illumination at the specific wavelength **lambda** [m]. If **lambda** is a vector $(N \times 1)$, then P $(N \times 2)$ is a vector whose elements are the chromaticity coordinates at the corresponding elements of **lambda**.

rgb = **lambda2rg**(**lambda**, **E**) is the rg-chromaticity coordinate (1×2) for an illumination spectrum **E** $(N \times 1)$ defined at corresponding wavelengths **lambda** $(N \times 1)$.

References

- Robotics, Vision & Control, Section 10.2, P. Corke, Springer 2011.

See also

cmfrgb, lambda2xy

---

lambda2xy

= LAMBDA2XY(LAMBDA) is the xy-chromaticity coordinate (1×2) for

illumination at the specific wavelength LAMBDA [metres]. If LAMBDA is a vector ($N \times 1$), then P ($N \times 2$) is a vector whose elements are the luminosity at the corresponding elements of LAMBDA.

xy = **lambda2xy**(**lambda**, **E**) is the rg-chromaticity coordinate (1×2) for an illumination spectrum **E** ($N \times 1$) defined at corresponding wavelengths **lambda** ($N \times 1$).

References

- Robotics, Vision & Control, Section 10.2, P. Corke, Springer 2011.

See also

cmfxyz, lambda2rg

LineFeature

Line feature class

This class represents a line feature.

Methods

plot	Plot the line segment
seglength	Determine length of line segment
display	Display value
char	Convert value to string

Properties

rho	Offset of the line
theta	Orientation of the line
strength	Feature strength
length	Length of the line

Properties of a vector of LineFeature objects are returned as a vector. If L is a vector

($N \times 1$) of LineFeature objects then L.rho is an $N \times 1$ vector of the rho element of each feature.

Note

- LineFeature is a reference object.
- LineFeature objects can be used in vectors and arrays

See also

Hough, RegionFeature, PointFeature

LineFeature.LineFeature

Create a line feature object

L = **LineFeature**() is a line feature object with null parameters.

L = **LineFeature**(**rho**, **theta**, **strength**) is a line feature object with the specified properties. LENGTH is undefined.

L = **LineFeature**(**rho**, **theta**, **strength**, **length**) is a line feature object with the specified properties.

L = **LineFeature**(**l2**) is a deep copy of the line feature **l2**.

LineFeature.char

Convert to string

s = L.**char**() is a compact string representation of the line feature. If L is a vector then the string has multiple lines, one per element.

LineFeature.display

Display value

L.**display**() displays a compact human-readable representation of the feature. If L is a vector then the elements are printed one per line.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a LineFeature object and the command has no trailing semicolon.

See also

LineFeature.char

LineFeature.plot

Plot line

L.**plot**() overlay the line on current **plot**.

L.**plot**(**ls**) as above but the optional line style arguments **ls** are passed to **plot**.

Notes

- If L is a vector then each element is plotted.

LineFeature.points

Return points on line segments

p = L.**points**(**edge**) is the set of **points** that lie along the line in the edge image **edge** are determined.

See also

icanny

LineFeature.seglength

Compute length of line segments

The Hough transform identifies lines but cannot determine their length. This method examines the edge pixels in the original image and determines the longest stretch of non-zero pixels along the line.

l2 = L.**seglength**(**edge**, **gap**) is a copy of the line feature object with the property length updated to the length of the line (pixels). Small gaps, less than **gap** pixels are tolerated.

l2 = L.**seglength**(**edge**) as above but the maximum allowable gap is 5 pixels.

See also

icanny

---

loadspectrum

Load spectrum data

s = **loadspectrum**(**lambda**, **filename**) is spectral data ($N \times D$) from file **filename** interpolated to wavelengths [metres] specified in **lambda** ($N \times 1$). The spectral data can be scalar (D=1) or vector (D>1) valued.

[**s**,**lambda**] = **loadspectrum**(**lambda**, **filename**) as above but also returns the passed wavelength **lambda**.

Notes

- The file is assumed to have its first column as wavelength in metres, the remaining columns are linearly interpolated and returned as columns of **s**.
- The files are kept in the private folder inside the MVTB folder.

References

- Robotics, Vision & Control, Section 14.3, P. Corke, Springer 2011.

---

luminos

Photopic luminosity function

p = **luminos**(**lambda**) is the photopic luminosity function for the wavelengths in **lambda** [m]. If **lambda** is a vector ($N \times 1$), then **p** ($N \times 1$) is a vector whose elements are the luminosity at the corresponding elements of **lambda**.

Luminosity has units of lumens which are the intensity with which wavelengths are perceived by the light-adapted human eye.

References

- Robotics, Vision & Control, Section 10.1, **p**. Corke, Springer 2011.

See also

rluminos

---

mkcube

Create cube

p = **mkcube**(**s**, **options**) is a set of points (3×8) that define the vertices of a cube of side length **s** and centred at the origin.

[**x**,**y**,**z**] = **mkcube**(**s**, **options**) as above but return the rows of **p** as three vectors.

[**x**,**y**,**z**] = **mkcube**(**s**, 'edge', **options**) is a mesh that defines the edges of a cube.

Options

'facepoint'	Add an extra point in the middle of each face, in this case the returned value is 3×14 (8 vertices + 6 face centres).
'centre', C	The cube is centred at C (3×1) not the origin
'T', T	The cube is arbitrarily transformed by the homogeneous transform T
'edge'	Return a set of cube edges in MATLAB mesh format rather than points.

See also

cylinder, sphere

---

mkgrid

Create grid of points

p = **mkgrid**(**d**, **s**, **options**) is a set of points (3 x $\mathbf{d}^2$) that define a **d** $\times$ **d** planar grid of points with side length **s**. The points are the columns of **p**. If **d** is a 2-vector the grid is **d**(1)xD(2) points. If **s** is a 2-vector the side lengths are **s**(1)xS(2).

By default the grid lies in the XY plane, symmetric about the origin.

Options

'T', T	the homogeneous transform T is applied to all points, allowing the plane to be translated or rotated.

---

mlabel

for mplot style graph

mlabel({lab1 lab2 lab3})

---

morphdemo

Demonstrate morphology using animation

morphdemo(**im**, **se**, **options**) displays an animation to show the principles of the mathematical morphology operations dilation or erosion. Two windows are displayed side by side, input binary image on the left and output image on the right. The structuring element moves over the input image and is colored red if the result is zero, else blue. Pixels in the output image are initially all grey but change to black or white as the structuring element moves.

out = **morphdemo**(**im**, **se**, **options**) as above but returns the output image.

Options

'dilate'	Perform morphological dilation
'erode'	Perform morphological erosion
'delay'	Time between animation frames (default 0.5s)
'scale', S	Scale factor for output image (default 64)
'movie', M	Write image frames to the folder M

Notes

- This is meant for small images, say 10×10 pixels.

See also

imorph, idilate, ierode

Movie

Class to read movie file

A concrete subclass of ImageSource that acquires images from a web camera built by Axis Communications (www.axis.com).

Methods

grab	Aquire and return the next image
size	Size of image
close	Close the image source
char	Convert the object parameters to human readable string

Properties

curFrame	The index of the frame just read
totalDuration	The running time of the movie (seconds)

See also

ImageSource, video

SEE ALSO: Video

Movie.Movie

Image source constructor

m = Movie(**file**, **options**) is an Movie object that returns frames from the movie file **file**.

Options

'uint8'	Return image with uint8 pixels (default)
'float'	Return image with float pixels
'double'	Return image with double precision pixels
'grey'	Return greyscale image
'gamma', G	Apply gamma correction with gamma=G
'scale', S	Subsample the image by S in both directions
'skip', S	Read every S'th frame from the movie

Movie.char

Convert to string

M.char() is a string representing the state of the movie object in human readable form.

Movie.close

Close the image source

M.close() closes the connection to the movie.

Movie.grab

Acquire next frame from movie

im = M.grab() acquires the next image from the movie

im = M.grab(**options**) as above but allows the next frame to be specified.

Options

'skip', S	Skip frames, and return current+S frame
'frame', F	Return frame F within the movie

Notes

- If no output argument given the image is displayed using IDISP.

---

mplot

time series data

mplot(**y**, **options**) plots the time series data **y**($N \times M$) in multiple subplots. The first column is assumed to be time, so M-1 plots are produced.

mplot(**T**, **y**, **options**) plots the time series data **y**($N \times M$) in multiple subplots. Time is provided explicitly as the first argument so M plots are produced.

mplot(**s**, **options**) as above but **s** is a structure. Each field is assumed to be a time series which is plotted. Time is taken from the field called 't'.

mplot(**w**, **options**) as above but **w** is a structure created by the Simulink write to workspace block where the save format is set to "Structure with time". Each field in the signals substructure is plotted.

mplot(**R**, **options**) as above but **R** is a Simulink.SimulationOutput object returned by the Simulink sim() function.

Options

'col', C	Select columns to plot, a boolean of length M-1 or a list of column indices in the range 1 to M-1
'label', L	Label the axes according to the cell array of strings L
'date'	Add a datestamp in the top right corner

Notes

- In all cases a simple GUI is created which is invoked by a right click anywhere on one of the plots. The supported options are:
  -

---

mpq

Image moments

m = **mpq**(**im**, **p**, **q**) is the PQ'th moment of the image **im**. That is, the sum of I(x,y).$x^{\mathbf{p}}.y^{\mathbf{q}}$.

See also

mpq_poly, npq, upq

---

mpq_poly

Polygon moments

m = **mpq_poly**(**v**, **p**, **q**) is the PQ'th moment of the polygon with vertices described by the columns of **v**.

Notes

- The points must be sorted such that they follow the perimeter in sequence (counter-clockwise).
- If the points are clockwise the moments will all be negated, so centroids will be still be correct.
- If the first and last point in the list are the same, they are considered to be a single vertex.

See also

mpq, npq_poly, upq_poly, Polygon

---

mtools

simple/useful tools to all windows in figure

---

multidfprintf

Print formatted text to multiple streams

COUNT = MULTIDFPRINTF(IDVEC, FORMAT, A, ...) performs formatted output to multiple streams such as console and files. FORMAT is the format string as used by sprintf and fprintf. A is the array of elements, to which the format will be applied similar to sprintf and fprint.

IDVEC is a vector ($1 \times N$) of file descriptors and COUNT is a vector ($1 \times N$) of the number of bytes written to each file.

Notes

- To write to the consolde use the file identifier 1.

Example

```
% Create and open a new example file:
fid = fopen('exampleFile.txt','w+');
% Write something to the file and the console simultaneously:
multidfprintf([1 FID],'% s % d % d % d% Close the file:
fclose(FID);
```

Authors

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

fprintf, sprintf

---

ncc

Normalized cross correlation

m = **ncc**(**i1**, **i2**) is the normalized cross-correlation between the two equally sized image patches **i1** and **i2**. The result **m** is a scalar in the interval -1 (non match) to 1 (perfect match) that indicates similarity.

Notes

- A value of 1 indicates identical pixel patterns.
- The **ncc** similarity measure is invariant to scale changes in image intensity.

See also

zncc, sad, ssd, isimilarity

---

niblack

Adaptive thresholding

T = **niblack**(**im**, **k**, **w2**) is the per-pixel (local) threshold to apply to image **im**. **T** has the same dimensions as **im**. The threshold at each pixel is a function of the mean and standard deviation computed over a $W \times W$ window, where W=2*w2+1.

[**T**,**m**,**s**] = **niblack**(**im**, **k**, **w2**) as above but returns the per-pixel mean **m** and standard deviation **s**.

Example

```
t = niblack(im, -0.2, 20);
idisp(im >= t);
```

Notes

- This is an efficient algorithm very well suited for binarizing text.
- **w2** should be chosen to be half the "size" of the features to be segmented, for example, in text segmentation, the height of a character.
- A common choice of k=-0.2

Reference

An Introduction to Digital Image Processing, W. **niblack**, Prentice-Hall, 1986.

See also

otsu, ithresh

---

npq

Normalized central image moments

m = **npq**(**im**, **p**, **q**) is the PQ'th normalized central moment of the image **im**. That is UPQ(**im**,**p**,**q**)/MPQ(**im**,0,0).

Notes

- The normalized central moments are invariant to translation and scale.

See also

npq_poly, mpq, upq

---

npq_poly

Normalized central polygon moments

m = **npq_poly**(**v**, **p**, **q**) is the PQ'th normalized central moment of the polygon with vertices described by the columns of **v**.

Notes

- The points must be sorted such that they follow the perimeter in sequence (counter-clockwise).
- If the points are clockwise the moments will all be negated, so centroids will be still be correct.
- If the first and last point in the list are the same, they are considered as a single vertex.
- The normalized central moments are invariant to translation and scale.

See also

mpq_poly, mpq, npq, upq, Polygon

---

numcols

Number of columns in matrix

nc = **numcols**(**m**) is the number of columns in the matrix **m**.

Notes

- Readable shorthand for SIZE(**m**,2);

See also

numrows, size

---

numrows

Number of rows in matrix

nr = **numrows**(**m**) is the number of rows in the matrix **m**.

Notes

- Readable shorthand for SIZE(**m**,1);

See also

numcols, size

---

optparse

Standard option parser for Toolbox functions

[**optout**,**args**] = **optparse**(**opt**, **arglist**) is a generalized option parser for Toolbox functions. It supports options that have an assigned value, boolean or enumeration types (string or int).

The software pattern is:

```
function(a, b, c, varargin)
opt.foo = true;
opt.bar = false;
opt.blah = [];
opt.choose = {'this', 'that', 'other'};
opt.select = {'#no', '#yes'};
opt = tb_optparse(opt, varargin);
```

Optional arguments to the function behave as follows:

'foo'	sets opt.foo <- true
'nobar'	sets opt.foo <- false
'blah', 3	sets opt.blah <- 3
'blah', {x,y}	sets opt.blah <- {x,y}
'that'	sets opt.choose <- 'that'
'yes'	sets opt.select <- 2 (the second element)

and can be given in any combination.

If neither of 'this', 'that' or 'other' are specified then opt.choose <- 'this'. Alternatively if:

```
opt.choose = {[], 'this', 'that', 'other'};
```

then if neither of 'this', 'that' or 'other' are specified then opt.choose <- []

If neither of 'no' or 'yes' are specified then opt.select <- 1.

Note:

- That the enumerator names must be distinct from the field names.
- That only one value can be assigned to a field, if multiple values

```
are required they must be converted to a cell array.
```

- To match an option that starts with a digit, prefix it with 'd_', so the field 'd_3d' matches the option '3d'.

The allowable options are specified by the names of the fields in the structure opt. By default if an option is given that is not a field of opt an error is declared.

Sometimes it is useful to collect the unassigned options and this can be achieved using a second output argument

```
[opt,arglist] = tb_optparse(opt, varargin);
```

which is a cell array of all unassigned arguments in the order given in varargin.

The return structure is automatically populated with fields: verbose and debug. The following options are automatically parsed:

'verbose'	sets opt.verbose <- true
'verbose=2'	sets opt.verbose <- 2 (very verbose)
'verbose=3'	sets opt.verbose <- 3 (extremeley verbose)
'verbose=4'	sets opt.verbose <- 4 (ridiculously verbose)
'debug', N	sets opt.debug <- N
'setopt', S	sets opt <- S
'showopt'	displays opt and arglist

otsu

Threshold selection

T = otsu(**im**) is an optimal threshold for binarizing an image with a bimodal intensity histogram. **T** is a scalar threshold that maximizes the variance between the classes of pixels below and above the thresold **T**.

Example

```
t = otsu(im);
idisp(im >= t);
```

Options

'levels', N	Number of grey levels to use if image is float (default 256)
'valley', S	Standard deviation for the Gaussian weighted valley emphasis option

Notes

- Performance for images with non-bimodal histograms can be quite poor.

Reference

A Threshold Selection Method from Gray-Level Histograms, N. otsu IEEE Trans. Systems, Man and Cybernetics Vol SMC-9(1), Jan 1979, pp 62-66

An improved method for image thresholding on the valley-emphasis method H-F Ng, D. Jargalsaikhan etal Signal and Info Proc. Assocn. Annual Summit and Conf (APSIPA) 2013 pp 1-4

See also

niblack, ithresh

---

peak

Find peaks in vector

yp = **peak**(**y**, **options**) are the values of the maxima in the vector **y**.

[**yp**,**i**] = **peak**(**y**, **options**) as above but also returns the indices of the maxima in the vector **y**.

[**yp**,**xp**] = **peak**(**y**, **x**, **options**) as above but also returns the corresponding x-coordinates of the maxima in the vector **y**. **x** is the same length as **y** and contains the corresponding x-coordinates.

Options

'npeaks', N	Number of peaks to return (default all)
'scale', S	Only consider as peaks the largest value in the horizontal range +/- S points.
'interp', M	Order of interpolation polynomial (default no interpolation)
'plot'	Display the interpolation polynomial overlaid on the point data

Notes

- A maxima is defined as an element that larger than its two neighbours. The first and last element will never be returned as maxima.
- To find minima, use **peak**(-V).
- The interp options fits points in the neighbourhood about the **peak** with an M'th order polynomial and its **peak** position is returned. Typically choose M to be odd. In this case **xp** will be non-integer.

See also

peak2

---

peak2

Find peaks in a matrix

zp = peak2(**z**, **options**) are the peak values in the 2-dimensional signal **z**.

[**zp**,**ij**] = peak2(**z**, **options**) as above but also returns the indices of the maxima in the matrix **z**. Use SUB2IND to convert these to row and column coordinates

Options

'npeaks', N	Number of peaks to return (default all)
'scale', S	Only consider as peaks the largest value in the horizontal and vertical range +/- S points.
'interp'	Interpolate peak (default no interpolation)
'plot'	Display the interpolation polynomial overlaid on the point data

Notes

- A maxima is defined as an element that larger than its eight neighbours. Edges elements will never be returned as maxima.
- To find minima, use peak2(-V).
- The interp options fits points in the neighbourhood about the peak with a paraboloid and its peak position is returned. In this case **ij** will be non-integer.

See also

peak, sub2ind

PGraph

Graph class

g = PGraph()	create a 2D, planar embedded, directed graph
g = PGraph(n)	create an n-d, embedded, directed graph

Provides support for graphs that:

- are directed
- are embedded in a coordinate system
- have symmetric cost edges (A to B is same cost as B to A)

- have no loops (edges from A to A)
- have vertices that are represented by integers VID
- have edges that are represented by integers EID

Methods

Constructing the graph

g.add_node(coord)	add vertex, return vid
g.add_edge(v1, v2)	add edge from v1 to v2, return eid
g.setcost(e, c)	set cost for edge e
g.setdata(v, u)	set user data for vertex v
g.data(v)	get user data for vertex v
g.clear()	remove all vertices and edges from the graph

Information from graph

g.edges(v)	list of edges for vertex v
g.cost(e)	cost of edge e
g.neighbours(v)	neighbours of vertex v
g.component(v)	component id for vertex v
g.connectivity()	number of edges for all vertices

Display

g.plot()	set goal vertex for path planning
g.highlight_node(v)	highlight vertex v
g.highlight_edge(e)	highlight edge e
g.highlight_component(c)	highlight all nodes in component c
g.highlight_path(p)	highlight nodes and edge along path p

g.pick(coord) vertex closest to coord

g.char() convert graph to string
g.display() display summary of graph

Matrix representations

g.adjacency()	adjacency matrix
g.incidence()	incidence matrix
g.degree()	degree matrix
g.laplacian()	Laplacian matrix

Planning paths through the graph

g.Astar(s, g)	shortest path from s to g
g.goal(v)	set goal vertex, and plan paths
g.path(v)	list of vertices from v to goal

Graph and world points

g.coord(v)	coordinate of vertex v
g.distance(v1, v2)	distance between v1 and v2
g.distances(coord)	return sorted distances from coord to all vertices
g.closest(coord)	vertex closest to coord

Object properties (read only)

g.n	number of vertices
g.ne	number of edges
g.nc	number of components

Examples

```
g = PGraph();
g.add_node([1 2]');  % add node 1
g.add_node([3 4]');  % add node 1
g.add_node([1 3]');  % add node 1
g.add_edge(1, 2);    % add edge 1-2
g.add_edge(2, 3);    % add edge 2-3
g.add_edge(1, 3);    % add edge 1-3
g.plot()
```

Notes

- Graph connectivity is maintained by a labeling algorithm and this is updated every time an edge is added.
- Nodes and edges cannot be deleted.
- Support for edge direction is rudimentary.

PGraph.PGraph

Graph class constructor

g=PGraph(d, options) is a graph object embedded in **d** dimensions.

Options

'distance', M	Use the distance metric M for path planning which is either 'Euclidean' (default) or 'SE2'.
'verbose'	Specify verbose operation

Notes

- Number of dimensions is not limited to 2 or 3.
- The distance metric 'SE2' is the sum of the squares of the difference in position and angle modulo 2pi.
- To use a different distance metric create a subclass of PGraph and override the method distance_metric().

PGraph.add_edge

Add an edge

E = G.**add_edge**(**v1**, **v2**) adds a directed edge from vertex id **v1** to vertex id **v2**, and returns the edge id **E**. The edge cost is the distance between the vertices.

E = G.**add_edge**(**v1**, **v2**, **C**) as above but the edge cost is **C**.

Notes

- Distance is computed according to the metric specified in the constructor.
- Graph connectivity is maintained by a labeling algorithm and this is updated every time an edge is added.

See also

PGraph.add_node, PGraph.edgedir

PGraph.add_node

Add a node

v = G.**add_node**(**x**) adds a node/vertex with coordinate **x** ($D \times 1$) and returns the integer node id **v**.

v = G.**add_node**(**x**, **v2**) as above but connected by a directed edge from vertex **v** to vertex **v2** with cost equal to the distance between the vertices.

v = G.**add_node**(**x**, **v2**, **C**) as above but the added edge has cost **C**.

Notes

- Distance is computed according to the metric specified in the constructor.

See also

PGraph.add_edge, PGraph.data, PGraph.getdata

PGraph.adjacency

Adjacency matrix of graph

a = G.**adjacency**() is a matrix ($N \times N$) where element **a**(i,j) is the cost of moving from vertex i to vertex j.

Notes

- Matrix is symmetric.
- Eigenvalues of **a** are real and are known as the spectrum of the graph.
- The element **a**(I,J) can be considered the number of walks of one edge from vertex I to vertex J (either zero or one). The element (I,J) of $\mathbf{a}^N$ are the number of walks of length N from vertex I to vertex J.

See also

PGraph.degree, PGraph.incidence, PGraph.laplacian

PGraph.Astar

path finding

path = G.**Astar**(**v1**, **v2**) is the lowest cost path from vertex **v1** to vertex **v2**. **path** is a list of vertices starting with **v1** and ending **v2**.

[**path**,**C**] = G.**Astar**(**v1**, **v2**) as above but also returns the total cost of traversing **path**.

Notes

- Uses the efficient A* search algorithm.

References

- Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". Hart, P. E.; Nilsson, N. J.; Raphael, B. SIGART Newsletter 37: 28-29, 1972.

See also

PGraph.goal, PGraph.path

PGraph.char

Convert graph to string

s = G.**char**() is a compact human readable representation of the state of the graph including the number of vertices, edges and components.

PGraph.clear

Clear the graph

G.**clear**() removes all vertices, edges and components.

PGraph.closest

Find closest vertex

v = G.**closest**(**x**) is the vertex geometrically **closest** to coordinate **x**.

[**v**,**d**] = G.**closest**(**x**) as above but also returns the distance **d**.

See also

PGraph.distances

PGraph.component

Graph component

C = G.**component**(**v**) is the id of the graph **component** that contains vertex **v**.

PGraph.connectivity

Graph connectivity

C = G.**connectivity**() is a vector ($N \times 1$) with the number of edges per vertex.

The average vertex **connectivity** is

```
mean(g.connectivity())
```

and the minimum vertex **connectivity** is

```
min(g.connectivity())
```

PGraph.coord

Coordinate of node

x = G.**coord**(**v**) is the coordinate vector ($D \times 1$) of vertex id **v**.

PGraph.cost

Cost of edge

C = G.**cost**(**E**) is the **cost** of edge id **E**.

PGraph.data

Get user data for node

u = G.**data**(**v**) gets the user **data** of vertex **v** which can be of any type such as a number, struct, object or cell array.

See also

PGraph.setdata

PGraph.degree

Degree matrix of graph

d = G.**degree**() is a diagonal matrix ($N \times N$) where element **d**(i,i) is the number of edges connected to vertex id i.

See also

PGraph.adjacency, PGraph.incidence, PGraph.laplacian

PGraph.display

Display graph

G.**display**() displays a compact human readable representation of the state of the graph including the number of vertices, edges and components.

See also

PGraph.char

PGraph.distance

Distance between vertices

d = G.**distance**(**v1**, **v2**) is the geometric **distance** between the vertices **v1** and **v2**.

See also

PGraph.distances

PGraph.distances

Distances from point to vertices

d = G.**distances**(**x**) is a vector $(1 \times N)$ of geometric distance from the point **x** (**d** $\times$ 1) to every other vertex sorted into increasing order.

[**d**,**w**] = G.**distances**(**p**) as above but also returns **w** $(1 \times N)$ with the corresponding vertex id.

Notes

- Distance is computed according to the metric specified in the constructor.

See also

PGraph.closest

PGraph.edgedir

Find edge direction

d = G.**edgedir**(**v1**, **v2**) is the direction of the edge from vertex id **v1** to vertex id **v2**.

If we add an edge from vertex 3 to vertex 4

```
g.add_edge(3, 4)
```

then

```
g.edgedir(3, 4)
```

is positive, and

```
g.edgedir(4, 3)
```

is negative.

See also

PGraph.add_node, PGraph.add_edge

PGraph.edges

Find edges given vertex

E = G.**edges**(**v**) is a vector containing the id of all **edges** connected to vertex id **v**.

See also

PGraph.edgedir

PGraph.get.n

Number of vertices

G.n is the number of vertices in the graph.

See also

PGraph.ne

PGraph.get.nc

Number of components

G.nc is the number of components in the graph.

See also

PGraph.component

PGraph.get.ne

Number of edges

G.ne is the number of **edges** in the graph.

See also

PGraph.n

PGraph.goal

Set goal node

G.**goal**(**vg**) computes the cost of reaching every vertex in the graph connected to the **goal** vertex **vg**.

Notes

- Combined with G.path performs a breadth-first search for paths to the **goal**.

See also

PGraph.path, PGraph.Astar, astar

PGraph.highlight_component

Highlight a graph component

G.**highlight_component**(**C**, **options**) highlights the vertices that belong to graph component **C**.

Options

'NodeSize', S	Size of vertex circle (default 12)
'NodeFaceColor', **C**	Node circle color (default yellow)
'NodeEdgeColor', **C**	Node circle edge color (default blue)

See also

PGraph.highlight_node, PGraph.highlight_edge, PGraph.highlight_component

---

PGraph.highlight_edge

Highlight a node

G.**highlight_edge**(**v1**, **v2**) highlights the edge between vertices **v1** and **v2**.

G.**highlight_edge**(**E**) highlights the edge with id **E**.

Options

'EdgeColor', C	Edge edge color (default black)
'EdgeThickness', T	Edge thickness (default 1.5)

See also

PGraph.highlight_node, PGraph.highlight_path, PGraph.highlight_component

---

PGraph.highlight_node

Highlight a node

G.**highlight_node**(**v**, **options**) highlights the vertex **v** with a yellow marker. If **v** is a list of vertices then all are highlighted.

Options

'NodeSize', S	Size of vertex circle (default 12)
'NodeFaceColor', C	Node circle color (default yellow)
'NodeEdgeColor', C	Node circle edge color (default blue)

See also

PGraph.highlight_edge, PGraph.highlight_path, PGraph.highlight_component

---

PGraph.highlight_path

Highlight path

G.**highlight_path**(**p**, **options**) highlights the path defined by vector **p** which is a list of vertex ids comprising the path.

Options

'NodeSize', S	Size of vertex circle (default 12)
'NodeFaceColor', C	Node circle color (default yellow)
'NodeEdgeColor', C	Node circle edge color (default blue)
'EdgeColor', C	Node circle edge color (default black)

See also

PGraph.highlight_node, PGraph.highlight_edge, PGraph.highlight_component

---

PGraph.incidence

Incidence matrix of graph

in = G.**incidence**() is a matrix ($N \times NE$) where element **in**(i,j) is non-zero if vertex id i is connected to edge id j.

See also

PGraph.adjacency, PGraph.degree, PGraph.laplacian

PGraph.laplacian

Laplacian matrix of graph

L = G.**laplacian**() is the Laplacian matrix ($N \times N$) of the graph.

Notes

- **L** is always positive-semidefinite.
- **L** has at least one zero eigenvalue.
- The number of zero eigenvalues is the number of connected components in the graph.

See also

PGraph.adjacency, PGraph.incidence, PGraph.degree

PGraph.neighbours

Neighbours of a vertex

n = G.**neighbours**(**v**) is a vector of ids for all vertices which are directly connected **neighbours** of vertex **v**.

[**n**,**C**] = G.**neighbours**(**v**) as above but also returns a vector **C** whose elements are the edge costs of the paths corresponding to the vertex ids in **n**.

PGraph.neighbours_d

Directed neighbours of a vertex

n = G.**neighbours_d**(**v**) is a vector of ids for all vertices which are directly connected neighbours of vertex **v**. Elements are positive if there is a link from **v** to the node, and negative if the link is from the node to **v**.

[**n**,**C**] = G.**neighbours_d**(**v**) as above but also returns a vector **C** whose elements are the edge costs of the paths corresponding to the vertex ids in **n**.

PGraph.path

Find path to goal node

p = G.**path**(**vs**) is a vector of vertex ids that form a **path** from the starting vertex **vs** to the previously specified goal. The **path** includes the start and goal vertex id.

To compute **path** to goal vertex 5

```
g.goal(5);
```

then the **path**, starting from vertex 1 is

```
p1 = g.path(1);
```

and the **path** starting from vertex 2 is

```
p2 = g.path(2);
```

Notes

- Pgraph.goal must have been invoked first.
- Can be used repeatedly to find paths from different starting points to the goal specified to Pgraph.goal().

See also

PGraph.goal, PGraph.Astar

PGraph.pick

Graphically select a vertex

v = G.**pick**() is the id of the vertex closest to the point clicked by the user on a plot of the graph.

See also

PGraph.plot

PGraph.plot

Plot the graph

G.**plot**(**opt**) plots the graph in the current figure. Nodes are shown as colored circles.

Options

'labels'	Display vertex id (default false)
'edges'	Display edges (default true)
'edgelabels'	Display edge id (default false)
'NodeSize', S	Size of vertex circle (default 8)
'NodeFaceColor', C	Node circle color (default blue)
'NodeEdgeColor', C	Node circle edge color (default blue)
'NodeLabelSize', S	Node label text sizer (default 16)
'NodeLabelColor', C	Node label text color (default blue)
'EdgeColor', C	Edge color (default black)
'EdgeLabelSize', S	Edge label text size (default black)
'EdgeLabelColor', C	Edge label text color (default black)
'componentcolor'	Node color is a function of graph component

PGraph.setcost

Set cost of edge

G.**setcost**(**E**, **C**) set cost of edge id **E** to **C**.

PGraph.setdata

Set user data for node

G.**setdata**(**v**, **u**) sets the user data of vertex **v** to **u** which can be of any type such as a number, struct, object or cell array.

See also

PGraph.data

PGraph.vertices

Find vertices given edge

v = G.**vertices**(**E**) return the id of the **vertices** that define edge **E**.

---

pickregion

Pick a rectangular region of a figure using mouse

[**p1**,**p2**] = **pickregion**() initiates a rubberband box at the current click point and animates it so long as the mouse button remains down. Returns the first and last coordinates in axis units.

Options

'axis', A	The axis to select from (default current axis)
'ls', LS	Line style for foreground line (default ':y');
'bg'LS,	Line style for background line (default '-k');
'width', W	Line width (default 2)

Notes

- Effectively a replacement for the builtin rbbox function which draws the box in the wrong location on my Mac's external monitor.

Author

Based on rubberband box from MATLAB Central written/Edited by Bob Hamans (B.C.Hamans@student.tue.nl) 02-04-2003, in turn based on an idea of Sandra Martinka's Rubberline.

---

plot2

Plot trajectories

plot2(**p**) plots a line with coordinates taken from successive rows of **p**. **p** can be $N \times 2$ or $N \times 3$.

If **p** has three dimensions, ie. $N \times 2 \times M$ or $N \times 3 \times M$ then the M trajectories are overlaid in the one plot.

plot2(**p**, **ls**) as above but the line style arguments **ls** are passed to plot.

See also

plot

---

plot_arrow

Draw an arrow

plot_arrow(**p**, **options**) draws an arrow from P1 to P2 where **p**=[P1; P2].

Options

All options are passed through to arrow3. Pass in a single character MATLAB colorspec (eg. 'r') to set the color.

See also

arrow3

---

plot_box

a box

plot_box(**b**, **ls**) draws a box defined by **b**=[XL XR; YL YR] on the current plot with optional MATLAB linestyle options **ls**.

plot_box(**x1**,**y1**, **x2**,**y2**, **ls**) draws a box with corners at (**x1**,**y1**) and (**x2**,**y2**), and optional MATLAB linestyle options **ls**.

plot_box('centre', P, 'size', W, **ls**) draws a box with center at P=[X,Y] and with dimensions W=[WIDTH HEIGHT].

plot_box('topleft', P, 'size', W, **ls**) draws a box with top-left at P=[X,Y] and with dimensions W=[WIDTH HEIGHT].

Notes

- The box is added to the current plot.
- Additional options **ls** are MATLAB LineSpec options and are passed to PLOT.

See also

plot_poly, plot_circle, plot_ellipse

---

plot_circle

Draw a circle

plot_circle(**C**, **R**, **options**) draws a circle on the current plot with centre **C**=[X,Y] and radius **R**. If **C**=[X,Y,Z] the circle is drawn in the XY-plane at height Z.

H = **plot_circle**(**C**, **R**, **options**) as above but return handles. For multiple circles **H** is a vector of handles, one per circle.

If **C** $(2 \times N)$ then N circles are drawn and **H** is $N \times 1$. If **R** (1×1) then all circles have the same radius or else **R** $(1 \times N)$ to specify the radius of each circle.

Options

'edgecolor'	the color of the circle's edge, Matlab color spec
'fillcolor'	the color of the circle's interior, Matlab color spec
'alpha'	transparency of the filled circle: 0=transparent, 1=solid
'alter', **H**	alter existing circles with handle **H**

For an unfilled ellipse any MATLAB LineProperty **options** can be given, for a filled ellipse any MATLAB PatchProperty **options** can be given.

Notes

- The circle(s) is added to the current plot.

See also

plot_ellipse, plot_box, plot_poly

---

plot_ellipse

Draw an ellipse or ellipsoid

plot_ellipse(**a**, **options**) draws an ellipse defined by X'AX = 0 on the current plot, centred at the origin.

plot_ellipse(**a**, **C**, **options**) as above but centred at **C**=[X,Y]. If **C**=[X,Y,Z] the ellipse is parallel to the XY plane but at height Z.

H = **plot_ellipse**(**a**, **C**, **options**) as above but return graphic handle.

Options

'edgecolor'	the color of the circle's edge, Matlab color spec
'fillcolor'	the color of the circle's interior, Matlab color spec
'alpha'	transparency of the filled circle: 0=transparent, 1=solid
'alter', **H**	alter existing circles with handle **H**

Notes

- If **a** (2×2) draw an ellipse, else if **a**(3×3) draw an ellipsoid.
- The ellipse is added to the current plot.

See also

plot_ellipse_inv, plot_circle, plot_box, plot_poly

---

plot_ellipse_inv

Draw an ellipse or ellipsoid

plot_ellipse_inv(**a**, **options**) draws an ellipse defined by X'.inv(**a**).X = 0 on the current plot, centred at the origin.

plot_ellipse_inv(**a**, **C**, **options**) as above but centred at **C**=[X,Y]. If **C**=[X,Y,Z] the ellipse is parallel to the XY plane but at height Z.

H = **plot_ellipse_inv**(**a**, **C**, **options**) as above but return graphic handle.

Options

'edgecolor'	the color of the circle's edge, Matlab color spec
'fillcolor'	the color of the circle's interior, Matlab color spec
'alpha'	transparency of the filled circle: 0=transparent, 1=solid
'alter', **H**	alter existing circles with handle **H**

Notes

- For the case where the inverse of ellipse parameters are known, perhaps an inverse covariance matrix.
- If **a** (2×2) draw an ellipse, else if **a**(3×3) draw an ellipsoid.
- The ellipse is added to the current plot.

See also

plot_ellipse, plot_circle, plot_box, plot_poly

---

plot_homline

Draw a homogeneous

plot_homline(**L**, **ls**) draws a line in the current plot **L**.X = 0 where **L** (3×1). The current axis limits are used to determine the endpoints of the line. MATLAB line specification **ls** can be set. If **L** $(3 \times N)$ then N lines are drawn, one per column.

H = **plot_homline**(**L**, **ls**) as above but returns a vector of graphics handles for the lines.

Notes

- The line(s) is added to the current plot.

See also

plot_box, plot_poly, homline

---

plot_point

a feature point

plot_point(**p**, **options**) adds point markers to the current plot, where **p** $(2 \times N)$ and each column is the point coordinate.

Options

'textcolor', colspec	Specify color of text
'textsize', size	Specify size of text
'bold'	Text in bold font.
'printf', {fmt, data}	Label points according to printf format string and corresponding element of data
'sequence'	Label points sequentially

Additional options are passed through to PLOT for creating the marker.

Examples

Simple point plot

```
P = rand(2,4);
plot_point(P);
```

Plot points with markers


```
plot_point(P, '*');
```

Plot points with square markers and labels 1 to 4

```
plot_point(P, 'sequence', 's');
```

Plot points with circles and annotations P1 to P4

```
data = [1 2 4 8];
plot_point(P, 'printf', {' P%d', data}, 'o');
```

Notes

- The point(s) is added to the current plot.
- 2D points only.

See also

plot, text

---

plot_poly

Draw a polygon

plot_poly(**p**, **options**) draws a polygon defined by columns of **p** ($2 \times N$), in the current plot.

options

'fill', F	the color of the circle's interior, MATLAB color spec
'alpha', A	transparency of the filled circle: 0=transparent, 1=solid.

Notes

- If **p** ($3 \times N$) the polygon is drawn in 3D
- The line(s) is added to the current plot.

See also

plot_box, patch, Polygon

---

plot_sphere

Draw sphere

plot_sphere(**C**, **R**, **ls**) draws spheres in the current plot. **C** is the centre of the sphere (3×1), **R** is the radius and **ls** is an optional MATLAB color spec, either a letter or a 3-vector.

H = **plot_sphere**(**C**, **R**, **color**) as above but returns the handle(s) for the spheres.

H = **plot_sphere**(**C**, **R**, **color**, **alpha**) as above but **alpha** specifies the opacity of the sphere were 0 is transparant and 1 is opaque. The default is 1.

If **C** ($3 \times N$) then N sphhere are drawn and **H** is $N \times 1$. If **R** (1×1) then all spheres have the same radius or else **R** ($1 \times N$) to specify the radius of each sphere.

Example

Create four spheres

```
plot_sphere( mkgrid(2, 1), .2, 'b')
```

and now turn on a full lighting model

```
lighting gouraud
light
```

NOTES

- The sphere is always added, irrespective of figure hold state.
- The number of vertices to draw the sphere is hardwired.

---

plotp

Plot trajectories

plotp(**p**) plots a set of points **p**, which by Toolbox convention are stored one per column. **p** can be $N \times 2$ or $N \times 3$. By default a linestyle of 'bx' is used.

plotp(**p**, **ls**) as above but the line style arguments **ls** are passed to plot.

See also

plot, plot2

Plucker

Plucker coordinate class

Concrete class to represent a line in Plucker coordinates.

Methods

line	Return Plucker line coordinates (1×6)
side	Side operator

Operators

*	Multiple Plucker matrix by a general matrix
—	Side operator

Notes

- This is reference class object
- Link objects can be used in vectors and arrays

Plucker.Plucker

Create Plucker object

p = **Plucker**(**p1**, **p2**) create a **Plucker** object that represents the line joining the 3D points **p1** (3×1) and **p2** (3×1).

Plucker.char

Convert to string

s = P.**char**() is a string showing **Plucker** parameters in a compact single line format.

See also

Plucker.display

Plucker.display

Display parameters

P.**display**() displays the **Plucker** parameters in compact single line format.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a Plucker object and the command has no trailing semicolon.

See also

Plucker.char

Plucker.line

Plucker liner coordinates

P.**line**() is a 6-vector representation of the **Plucker** coordinates of the **line**.

Plucker.mtimes

Plucker composition

P * M is the product of the **Plucker** matrix and M ($4 \times N$).

M * P is the product of M ($N \times 4$) and the **Plucker** matrix.

Plucker.or

— P2 is the side operator which is zero whenever

the lines P1 and P2 intersect or are parallel.

Plucker.side

Side operator

SIDE(**p1**, **p2**) is the side operator which is zero whenever the lines **p1** and **p2** intersect or are parallel.

---

pnmfilt

Pipe image through PNM utility

out = **pnmfilt**(**cmd**) runs the external program given by the string **cmd** and the output (assumed to be PNM format) is returned as **out**.

out = **pnmfilt**(**cmd**, **im**) pipes the image **im** through the external program given by the string **cmd** and the output is returned as **out**. The external program must accept and return images in PNM format.

Examples

```
im = pnmfilt('ppmforge -cloud');
im = pnmfilt('pnmrotate 30', lena);
```

Notes

- Provides access to a large number of Unix command line utilities such as ImageMagick and netpbm.
- The input image is passed as stdin, the output image is assumed to come from stdout.
- MATLAB doesn't support i/o to pipes so the image is written to a temporary file, the command run to another temporary file, and that is read into MATLAB.

See also

pgmfilt, iread

PointFeature

PointCorner feature object

A superclass for image corner features.

Methods

plot	Plot feature position
distance	Descriptor distance
ncc	Descriptor similarity
uv	Return feature coordinate
display	Display value
char	Convert value to string

Properties

u	horizontal coordinate
v	vertical coordinate
strength	feature strength
descriptor	feature descriptor (vector)

Properties of a vector of PointFeature objects are returned as a vector. If F is a vector ($N \times 1$) of PointFeature objects then F.u is a $2 \times N$ matrix with each column the corresponding point coordinate.

See also

ScalePointFeature, SurfPointFeature, SiftPointFeature

PointFeature.PointFeature

Create a point feature object

f = **PointFeature**() is a point feature object with null parameters.

f = **PointFeature**(**u**, **v**) is a point feature object with specified coordinates.

f = **PointFeature**(**u**, **v**, **strength**) as above but with specified strength.

PointFeature.char

Convert to string

s = F.**char**() is a compact string representation of the point feature. If F is a vector then the string has multiple lines, one per element.

PointFeature.display

Display value

F.**display**() displays a compact human-readable representation of the feature. If F is a vector then the elements are printed one per line.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a PointFeature object and the command has no trailing semicolon.

See also

PointFeature.char

PointFeature.distance

Distance between feature descriptors

d = F.**distance**(**f1**) is the **distance** between feature descriptors, the norm of the Euclidean **distance**.

If F is a vector then **d** is a vector whose elements are the **distance** between the corresponding element of F and **f1**.

PointFeature.match

Match point features

m = F.**match**(**f2**, **options**) is a vector of FeatureMatch objects that describe candidate matches between the two vectors of point features F and **f2**.

[**m**,**C**] = F.**match**(**f2**, **options**) as above but returns a correspodence matrix where each row contains the indices of corresponding features in F and **f2** respectively.

Options

'thresh', T	**match** threshold (default 0.05)
'median'	Threshold at the median distance

See also

FeatureMatch

PointFeature.ncc

Feature descriptor similarity

s = F.**ncc**(**f1**) is the similarty between feature descriptors which is a scalar in the interval -1 to 1, where 1 is perfect match.

If F is a vector then D is a vector whose elements are the distance between the corresponding element of F and **f1**.

PointFeature.plot

Plot feature

F.**plot**() overlay a white square marker at the feature position.

F.**plot**(**ls**) as above but the optional line style arguments **ls** are passed to **plot**.

If F is a vector then each element is plotted.

---

polydiff

Differentiate a polynomial

pd = **polydiff**(**p**) is a vector of coefficients of a polynomial ($1 \times N$-1) which is the derivative of the polynomial **p** ($1 \times N$).

See also

polyval

Polygon

Polygon class

A general class for manipulating polygons and vectors of polygons.

Methods

plot	Plot polygon
area	Area of polygon
moments	Moments of polygon
centroid	Centroid of polygon
perimeter	Perimter of polygon
transform	Transform polygon
inside	Test if points are inside polygon
intersection	Intersection of two polygons
difference	Difference of two polygons
union	Union of two polygons
xor	Exclusive or of two polygons
display	print the polygon in human readable form
char	convert the polgyon to human readable string

Properties

vertices	List of polygon vertices, one per column
extent	Bounding box [minx maxx; miny maxy]
n	Number of vertices

Notes

- This is reference class object
- Polygon objects can be used in vectors and arrays

Acknowledgement

The methods: inside, intersection, difference, union, and xor are based on code written by:

Kirill K. Pankratov, kirill@plume.mit.edu, http://puddle.mit.edu/ glenn/kirill/saga.html

and require a licence. However the author does not respond to email regarding the licence, so use with care, and modify with acknowledgement.

Polygon.Polygon

Polygon class constructor

p = **Polygon**(**v**) is a polygon with vertices given by **v**, one column per vertex.

p = **Polygon**(**C**, **wh**) is a rectangle centred at **C** with dimensions **wh**=[WIDTH, HEIGHT].

Polygon.area

Area of polygon

a = P.**area**() is the **area** of the polygon.

See also

Polygon.moments

Polygon.centroid

Centroid of polygon

x = P.**centroid**() is the **centroid** of the polygon.

See also

Polygon.moments

Polygon.char

String representation

s = P.**char**() is a compact representation of the polgyon in human readable form.

Polygon.difference

Difference of polygons

d = P.**difference**(**q**) is polygon P minus polygon **q**.

Notes

- If polygons P and **q** are not intersecting, returns coordinates of P.
- If the result **d** is not simply connected or consists of several polygons, resulting vertex list will contain NaNs.

Polygon.display

Display polygon

P.**display**() displays the polygon in a compact human readable form.

See also

Polygon.char

Polygon.inside

Test if points are inside polygon

in = **p**.**inside**(**p**) tests if points given by columns of **p** ($2 \times N$) are **inside** the polygon. The corresponding elements of **in** ($1 \times N$) are either true or false.

Polygon.intersect

Intersection of polygon with list of polygons

i = P.**intersect**(**plist**) indicates whether or not the **Polygon** P intersects with

i(**j**) = 1 if p intersects polylist(**j**), else 0.

Polygon.intersect_line

Intersection of polygon and line segment

i = P.**intersect_line**(**L**) is the intersection points of a polygon P with the line segment **L**=[x1 x2; y1 y2]. **i** ($2 \times N$) has one column per intersection, each column is [x y]'.

Polygon.intersection

Intersection of polygons

i = P.**intersection**(**q**) is a **Polygon** representing the **intersection** of polygons P and **q**.

Notes

- If these polygons are not intersecting, returns empty polygon.
- If **intersection** consist of several disjoint polygons (for non-convex P or **q**) then vertices of **i** is the concatenation of the vertices of these polygons.

Polygon.moments

Moments of polygon

a = P.**moments**(**p**, **q**) is the pq'th moment of the polygon.

See also

Polygon.area, Polygon.centroid, mpq_poly

Polygon.perimeter

Perimeter of polygon

L = P.**perimeter**() is the **perimeter** of the polygon.

Polygon.plot

Draw polygon

P.**plot**() draws the polygon P in the current **plot**.

P.**plot**(**ls**) as above but pass the arguments **ls** to **plot**.

Notes

- The polygon is added to the current **plot**.

Polygon.transform

Transform polygon vertices

p2 = P.**transform**(**T**) is a new **Polygon** object whose vertices have been transformed by the SE(2) homgoeneous transformation **T** (3×3).

Polygon.union

Union of polygons

i = P.**union**(**q**) is a polygon representing the **union** of polygons P and **q**.

Notes

- If these polygons are not intersecting, returns a polygon with vertices of both polygons separated by NaNs.
- If the result P is not simply connected (such as a polygon with a "hole") the resulting contour consist of counter- clockwise "outer boundary" and one or more clock-wise "inner boundaries" around "holes".

Polygon.xor

Exclusive or of polygons

i = P.**union**(**q**) is a polygon representing the exclusive-or of polygons P and **q**.

Notes

- If these polygons are not intersecting, returns a polygon with vertices of both polygons separated by NaNs.
- If the result P is not simply connected (such as a polygon with a "hole") the resulting contour consist of counter- clockwise "outer boundary" and one or more clock-wise "inner boundaries" around "holes".

---

radgrad

Radial gradient

[**gr**,**gt**] = **radgrad**(**im**) is the radial and tangential gradient of the image **im**. At each pixel the image gradient vector is resolved into the radial and tangential directions.

[**gr**,**gt**] = **radgrad**(**im**, **centre**) as above but the centre of the image is specified as **centre**=[X,Y] rather than the centre pixel of **im**.

radgrad(**im**) as above but the result is displayed graphically.

See also

isobel

---

randinit

Reset random number generator

RANDINIT resets the defaul random number stream.

See also

randstream

---

ransac

Random sample and consensus

m = **ransac**(**func**, **x**, **T**, **options**) is the **ransac** algorithm that robustly fits data **x** to the model represented by the function **func**. **ransac** classifies Points that support the model as inliers and those that do not as outliers.

x typically contains corresponding point data, one column per point pair. **ransac** determines the subset of points (inliers) that best fit the model described by the function **func** and the parameter **m**. **T** is a threshold on how well a point fits the estimated, if the fit residual is aboe the the threshold the point is considered an outlier.

[**m**,**in**] = **ransac**(**func**, **x**, **T**, **options**) as above but returns the vector **in** of column indices of **x** that describe the inlier point set.

[**m**,**in**,**resid**] = **ransac**(**func**, **x**, **T**, **options**) as above but returns the final residual of applying **func** to the inlier set.

Options

'maxTrials', N	maximum number of iterations (default 2000)
'maxDataTrials', N	maximum number of attempts to select a non-degenerate data set (default 100)

Model function

out = **func**(**R**) is the function passed to RANSAC and it must accept a single argument **R** which is a structure:

R.cmd	the operation to perform which is either (string)
R.debug	display what's going on (logical)
R.**x**	data to work on, N point pairs ($6 \times N$)
R.t	threshold (1×1)
R.theta	estimated quantity to test (3×3)
R.misc	private data (cell array)

The function return value is also a structure:

out.s	sample size (1×1)
out.**x**	conditioned data ($2D \times N$)
out.misc	private data (cell array)
out.inliers	list of inliers ($1 \times$ **m**)
out.valid	if data is valid for estimation (logical)
out.theta	estimated quantity (3×3)
out.resid	model fit residual (1×1)

The values of **R**.cmd are:

'size'	**out**.s is the minimum number of points required to compute an estimate to **out**.s
'condition'	**out**.**x** = CONDITION(**R**.**x**) condition the point data
'decondition'	**out**.theta = DECONDITION(**R**.theta) decondition the estimated model data
'valid'	**out**.valid is true if a set of points is not degenerate, that is they will produce a model. This is used to discard random samples that do not result in useful models.
'estimate'	[**out**.theta,**out**.resid] = EST(**R**.**x**) returns the best fit model and residual for the subset of points **R**.**x**. If this function cannot fit a model then **out**.theta = []. If multiple models are found **out**.theta is a cell array.
'error'	[**out**.inliers,**out**.theta] = ERR(**R**.theta,**R**.**x**,**T**) evaluates the distance from the model(s) **R**.theta to the points **R**.**x** and returns the best model **out**.theta and the subset of **R**.**x** that best supports (most inliers) that model.

Notes

- For some algorithms (eg. fundamental matrix) it is necessary to condition the data to improve the accuracy of model estimation. For efficiency the data is conditioned once, and the data transform parameters are kept in the .misc element. The inverse conditioning operation is applied to the model to transform the estimate based on conditioned data to a model applicable to the original data.
- The functions FMATRIX and HOMOG are written so as to be callable from RANSAC, that is, they detect a structure argument.

References

- **m**.A. Fishler and **R**.C. Boles. "Random sample concensus: A paradigm for model fitting with applications to image analysis and automated cartography". Comm. Assoc. Comp, Mach., Vol 24, No 6, pp 381-395, 1981
- Richard Hartley and Andrew Zisserman. "Multiple View Geometry in Computer Vision". pp 101-113. Cambridge University Press, 2001

Author

Peter Kovesi School of Computer Science & Software Engineering The University of Western Australia pk at csse uwa edu au http://www.csse.uwa.edu.au/ pk

See also

fmatrix, homography

Ray3D

Ray in 3D space

This object represents a ray in 3D space, defined by a point on the ray and a direction unit-vector.

Methods

intersect	Intersection of ray with plane or ray
closest	Closest distance between point and ray
char	Ray parameters as human readable string
display	Display ray parameters in human readable form

Properties

P0	A point on the ray (3×1)
d	Direction of the ray, unit vector (3×1)

Notes

- Ray3D objects can be used in vectors and arrays

Ray3D.Ray3D

Ray constructor

R = Ray3D(**p0**, **d**) is a new Ray3D object defined by a point on the ray **p0** and a direction vector **d**.

Ray3D.char

Convert to string

s = R.**char**() is a compact string representation of the **Ray3D**'s value. If R is a vector then the string has multiple lines, one per element.

Ray3D.closest

Closest distance between point and ray

x = R.**closest**(**p**) is the point on the ray R **closest** to the point **p**.

[**x**,**E**] = R.**closest**(**p**) as above but also returns the distance **E** between **x** and **p**.

Ray3D.display

Display value

R.**display**() displays a compact human-readable representation of the **Ray3D**'s value. If R is a vector then the elements are printed one per line.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a Ray3D object and the command has no trailing semicolon.

See also

Ray3D.char

Ray3D.intersect

Intersetion of ray with line or plane

x = R.**intersect**(**r2**) is the point on R that is closest to the ray **r2**. If R is a vector then then **x** has multiple columns, corresponding to the intersection of R(i) with **r2**.

[**x**,**E**] = R.**intersect**(**r2**) as above but also returns the closest distance between the rays.

x = R.**intersect**(**p**) returns the point of intersection between the ray R and the plane **p**=(a,b,c,d) where aX + bY + cZ + d = 0. If R is a vector then **x** has multiple columns, corresponding to the intersection of R(i) with **p**.

RegionFeature

Region feature class

This class represents a region feature.

Methods

boundary	Return the boundary as a list
box	Return the bounding box
plot	Plot the centroid
plot_boundary	Plot the boundary
plot_box	Plot the bounding box
plot_ellipse	Plot the equivalent ellipse
display	Display value
char	Convert value to string
pick	Return the index of the blob that is clicked

Properties

uc*	centroid, horizontal coordinate
vc*	centroid, vertical coordinate
p	centroid (uc, vc)
umin	bounding box, minimum horizontal coordinate
umax	bounding box, maximum horizontal coordinate
vmin	bounding box, minimum vertical coordinate
vmax	bounding box, maximum vertical coordinate
area*	the number of pixels
class*	the value of the pixels forming this region
label*	the label assigned to this region
children	a list of indices of features that are children of this feature
edgepoint	coordinate of a point on the perimeter
edge	a list of edge points $2 \times N$ matrix
perimeter*	edge length (pixels)
touch*	true if region touches edge of the image
a	major axis length of equivalent ellipse
b	minor axis length of equivalent ellipse
theta*	angle of major ellipse axis to horizontal axis
shape*	aspect ratio b/a (always <= 1.0)
circularity*	1 for a circle, less for other shapes
moments	a structure containing moments of order 0 to 2
bbox*	the bounding box, 2×2 matrix [umin umax; vmin vmax]
bboxarea*	bounding box area

Note

- Properties indicated with a * can be determined for a vector of RegionFeatures and the result will be a vector of those properties (not a list) with elements corresponding to the original vector of RegionFeatures.
- RegionFeature is a reference object.
- RegionFeature objects can be used in vectors and arrays
- This class behaves differently to LineFeature and PointFeature when getting properties of a vector of RegionFeature objects. For example R.u_ will be a list not a vector.

See also

iblobs, imoments

RegionFeature.RegionFeature

Create a region feature object

R = **RegionFeature**() is a region feature object with null parameters.

RegionFeature.boundary

Boundary in polar form

[**d**,**th**] = R.**boundary**() is a polar representation of the **boundary** with respect to the centroid. **d**(i) and **th**(i) are the distance to the **boundary** point and the angle respectively. These vectors have 400 elements irrespective of region size.

RegionFeature.box

Return bounding box

b = R.**box**() is the bounding **box** in standard Toolbox form [xmin,xmax; ymin, ymax].

RegionFeature.char

Convert to string

s = R.**char**() is a compact string representation of the region feature. If R is a vector then the string has multiple lines, one per element.

RegionFeature.contains

Test if coordinate is contained within region bounding box

R.**contains**(**coord**) true if the coordinate COORD lies within the bounding box of the region feature R. If R is a vector, return a vector of logical values, one per input region.

RegionFeature.display

Display value

R.**display**() is a compact string representation of the region feature. If R is a vector then the elements are printed one per line.

Notes

- this method is invoked implicitly at the command line when the result of an expression is a RegionFeature object and the command has no trailing semicolon.

See also

RegionFeature.char

RegionFeature.pick

Select blob from mouse click

i = R.**pick**() is the index of the region feature within the vector of RegionFeatures R to which the clicked point corresponds. Since regions can overlap of be contained in other regions, the region with the smallest area of bounding box that contains the selected point is returned.

See also

ginput, RegionFeature.inbox

RegionFeature.plot

Plot centroid

R.**plot**() overlay the centroid on current **plot**. It is indicated with overlaid o- and x-markers.

R.**plot**(**ls**) as above but the optional line style arguments **ls** are passed to **plot**.

If R is a vector then each element is plotted.

RegionFeature.plot_boundary

plot boundary

R.**plot_boundary**() overlay perimeter points on current plot.

R.**plot_boundary**(**ls**) as above but the optional line style arguments **ls** are passed to plot.

Notes

- If R is a vector then each element is plotted.

See also

boundmatch

RegionFeature.plot_box

Plot bounding box

R.**plot_box**() overlay the the bounding box of the region on current plot.

R.**plot_box**(**ls**) as above but the optional line style arguments **ls** are passed to plot.

If R is a vector then each element is plotted.

RegionFeature.plot_ellipse

Plot equivalent ellipse

R.**plot_ellipse**() overlay the the equivalent ellipse of the region on current plot.

R.**plot_ellipse**(**ls**) as above but the optional line style arguments **ls** are passed to plot.

If R is a vector then each element is plotted.

rg_addticks

Label spectral locus

rg_addticks() adds wavelength ticks to the spectral locus.

See also

xycolourspace

---

rgb2xyz

RGB to XYZ color space

[**x**, **y**, **z**] = **rgb2xyz**(**r**, **g**, **b**) xyz = **rgb2xyz**(rgb)

convert (**R**,**g**,**b**) coordinates to (X,Y,Z) color space. If RGB (or **R**, **g**, **b**) have more than one row, then computation is

done row wise.

SEE ALSO: ccxyz cmfxyz

---

rluminos

Relative photopic luminosity function

p = **rluminos**(**lambda**) is the relative photopic luminosity function for the wavelengths in **lambda** [m]. If **lambda** is a vector ($N \times 1$), then **p** ($N \times 1$) is a vector whose elements are the luminosity at the corresponding elements of **lambda**.

Relative luminosity lies in the interval 0 to 1 which indicate the intensity with which wavelengths are perceived by the light-adapted human eye.

References

- Robotics, Vision & Control, Section 10.1, **p**. Corke, Springer 2011.

See also

luminos

---

runscript

Run an M-file in interactive fashion

runscript(**fname**, **options**) runs the M-file **fname** and pauses after every executable line in the file until a key is pressed. Comment lines are shown without any delay between lines.

Options

'delay', D	Don't wait for keypress, just delay of D seconds (default 0)
'cdelay', D	Pause of D seconds after each comment line (default 0)
'begin'	Start executing the file after the comment line %%begin (default false)
'dock'	Cause the figures to be docked when created
'path', P	Look for the file **fname** in the folder P (default .)
'dock'	Dock figures within GUI

Notes

- If no file extension is given in **fname**, .m is assumed.
- If the executable statement has comments immediately afterward (no blank lines) then the pause occurs after those comments are displayed.
- A simple '-' prompt indicates when the script is paused, hit enter.
- If the function cprintf() is in your path, the display is more colorful, you can get this file from MATLAB Central.
- If the file has a lot of boilerplate, you can skip over and not display it by giving the 'begin' option which searchers for the first line starting with %%begin and commences execution at the line after that.

See also

eval

---

rvcpath

Install location of RVC tools

p = RVCPATH is the path of the top level folder for the installed RVC tools.

---

sad

Sum of absolute differences

m = **sad**(**i1**, **i2**) is the sum of absolute differences between the two equally sized image patches **i1** and **i2**. The result **m** is a scalar that indicates image similarity, a value of 0 indicates identical pixel patterns and is increasingly positive as image dissimilarity increases.

See also

zsad, ssd, ncc, isimilarity

ScalePointFeature

ScalePointCorner feature object

A subclass of PointFeature for features with scale.

Methods

plot	Plot feature position
plot_scale	Plot feature scale
distance	Descriptor distance
ncc	Descriptor similarity
uv	Return feature coordinate
display	Display value
char	Convert value to string

Properties

u	horizontal coordinate
v	vertical coordinate
strength	feature strength
scale	feature scale
descriptor	feature descriptor (vector)

Properties of a vector of ScalePointFeature objects are returned as a vector. If F is a vector ($N \times 1$) of ScalePointFeature objects then F.u is a $2 \times N$ matrix with each column the corresponding point coordinate.

See also

PointFeature, orientedscalepointfeature, SurfPointFeature, SiftPointFeature

---

ScalePointFeature.ScalePointFeature

Create a scale point feature object

f = **ScalePointFeature**() is a point feature object with null parameters.

f = **ScalePointFeature**(**u**, **v**) is a point feature object with specified coordinates.

f = **ScalePointFeature**(**u**, **v**, **strength**) as above but with specified strength.

f = **ScalePointFeature**(**u**, **v**, **strength**, **scale**) as above but with specified feature scale.

ScalePointFeature.plot

Plot feature

F.**plot**(**options**) overlay a marker at the feature position. The default is a point marker.

F.**plot**(**options**, **ls**) as above but the optional line style arguments **ls** are passed to **plot**.

If F is a vector then each element is plotted.

Options

'circle'	Indicate scale by a circle
'disk'	Indicate scale by a translucent disk
'color', C	Color of circle or disk (default green)
'alpha', A	Transparency of disk, 1=opaque, 0=transparent (default 0.2)
'scale', S	Scale factor for drawing circles and arrows.

Examples

Mark the feature coordinates with a white asterisk

```
f.plot('w*')
```

Mark each feature with a blue translucent disk

```
f.plot('disk', 'color', 'b', 'alpha', 0.3);
```

Mark each feature with a green circle and with exagerated scale

```
f.plot('circle', 'color', 'g', 'scale', 2)
```

See also

PointFeature.plot, plot

---

showpixels

Show low resolution image

Displays a low resolution image in detail as a grid with colored lines between pixels and numeric display of pixel values at each pixel. Useful for illustrating principles in teaching.

Options

'fmt', F	Format string (defaults to %d or %.2f depending on image type)
'label'	Display axis labels (default true)
'color', C	Text color (default 'b')
'fontsize', S	Font size (default 12)
'pixval'	Display pixel numeric values (default true)
'tick'	Display axis tick marks (default true)
'cscale', C	Color map scaling [min max] (defaults [0 1] or [0 255])
'uv', UV	UV={u,v} vectors of u and v coordinates

Notes

- This is meant for small images, say 10×10 pixels.

SiftPointFeature

SIFT point corner feature object

A subclass of OrientedScalePointFeature for SIFT features.

Methods

plot	**plot** feature position
plot_scale	**plot** feature scale
distance	Descriptor distance
ncc	Descriptor similarity
match	Match features
ncc	Descriptor similarity
uv	Return feature coordinate
display	Display value
char	Convert value to string

Properties

u	horizontal coordinate
v	vertical coordinate
strength	feature strength
theta	feature orientation [rad]
scale	feature scale
descriptor	feature descriptor (vector)
image_id	index of image containing feature

Properties of a vector of SiftCornerFeature objects are returned as a vector. If F is a vector ($N \times 1$) of SiftCornerFeature objects then F.u is a $2 \times N$ matrix with each column the corresponding u coordinate.

Notes

- SiftCornerFeature is a reference object.
- SiftCornerFeature objects can be used in vectors and arrays
- The SIFT algorithm is patented and not distributed with this toolbox. You can download a SIFT implementation which this class can utilize. See README.SIFT.

References

"Distinctive image features from scale-invariant keypoints", D.Lowe, Int. Journal on Computer Vision, vol.60, pp.91-110, Nov. 2004.

See also

isift, PointFeature, ScalePointFeature, orientedscalepointfeature, SurfPointFeature

---

SiftPointFeature.SiftPointFeature

Create a SIFT point feature object

f = **SiftPointFeature**() is a point feature object with null parameters.

f = **SiftPointFeature**(**u**, **v**) is a point feature object with specified coordinates.

f = **SiftPointFeature**(**u**, **v**, **strength**) as above but with specified strength.

f = **SiftPointFeature**(**u**, **v**, **strength**, **scale**) as above but with specified feature scale.

f = **SiftPointFeature**(**u**, **v**, **strength**, **scale**, **theta**) as above but with specified feature orientation.

See also

isift

SiftPointFeature.match

Match SIFT point features

m = F.**match**(**f2**, **options**) is a vector of FeatureMatch objects that describe candidate matches between the two vectors of SIFT features F and **f2**. Correspondence is based on descriptor similarity.

SiftPointFeature.support

Support region of feature

out = F.**support**(**im**, **w**) is an image of the **support** region of the feature F, extracted from the image **im** in which the feature appears. The **support** region is scaled to $\mathbf{w} \times \mathbf{w}$ and rotated so that the feature's orientation axis is upward.

out = F.**support**(**images**, **w**) as above but if the features were extracted from an image sequence **images** then the feature is extracted from the appropriate image in the same sequence.

[**out**,**T**] = F.**support**(**images**, **w**) as above but returns the pose of the feature as a 3×3 homogeneous transform in SE(2) that comprises the feature position and orientation.

F.**support**(**im**, **w**) as above but the **support** region is displayed.

See also

SiftPointFeature

---

simulinkext

Return file extension of Simulink block diagrams.

str = **simulinkext**() is either

- '.mdl' if Simulink version number is less than 8
- '.slx' if Simulink version numberis larger or equal to 8

Notes

The file extension for Simulink block diagrams has changed from Matlab 2011b to Matlab 2012a. This function is used for backwards compatibility.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

symexpr2slblock, doesblockexist, distributeblocks

SphericalCamera

Spherical camera class

A concrete class a spherical-projection camera.

Methods

project	project world points
plot	plot/return world point on image plane
hold	control hold for image plane
ishold	test figure hold for image plane
clf	clear image plane
figure	figure holding the image plane
mesh	draw shape represented as a mesh
point	draw homogeneous points on image plane
line	draw homogeneous lines on image plane
plot_camera	draw camera
rpy	set camera attitude
move	copy of Camera after motion
centre	get world coordinate of camera centre
delete	object destructor
char	convert camera parameters to string
display	display camera parameters

Properties (read/write)

npix	image dimensions in pixels (2×1)
pp	intrinsic: principal point (2×1)
rho	intrinsic: pixel dimensions (2×1) in metres
T	extrinsic: camera pose as homogeneous transformation

Properties (read only)

nu	number of pixels in u-direction
nv	number of pixels in v-direction

Note

- SphericalCamera is a reference object.
- SphericalCamera objects can be used in vectors and arrays

See also

Camera

SphericalCamera.SphericalCamera

Create spherical projection camera object

C = **SphericalCamera**() creates a spherical projection camera with canonic parameters: f=1 and name='canonic'.

C = **CentralCamera**(**options**) as above but with specified parameters.

Options

'name', N	Name of camera
'pixel', S	Pixel size: $S \times S$ or S(1)xS(2)
'pose', T	Pose of the camera as a homogeneous transformation

See also

Camera, CentralCamera, fisheyecamera, CatadioptricCamera

SphericalCamera.project

Project world points to image plane

pt = C.**project**(**p**, **options**) are the image plane coordinates for the world points **p**. The columns of **p** ($3 \times N$) are the world points and the columns of **pt** ($2 \times N$) are the corresponding spherical projection points, each column is phi (longitude) and theta (colatitude).

Options

'Tobj', T	Transform all points by the homogeneous transformation T before projecting them to the camera image plane.
'Tcam', T	Set the camera pose to the homogeneous transformation T before projecting points to the camera image plane. Overrides the current camera pose C.T.

See also

SphericalCamera.plot

SphericalCamera.sph

Implement spherical IBVS for point features

results = **sph**(**T**) **results** = **sph**(**T**, params)

Simulate IBVS with for a square target comprising 4 points is placed in the world XY plane. The camera/robot is initially at pose **T** and is driven to the orgin.

Two windows are shown and animated:

1. The camera view, showing the desired view (*) and the

```
current view (o)
```

2. The external view, showing the target points and the camera

The **results** structure contains time-history information about the image plane, camera pose, error, Jacobian condition number, error norm, image plane size and desired feature locations.

The params structure can be used to override simulation defaults by providing elements, defaults in parentheses:

target_size - the side length of the target in world units (0.5)

target_center - center of the target in world coords (0,0,2)

niter - the number of iterations to run the simulation (500)
eterm - a stopping criteria on feature error norm (0)
lambda - gain, can be scalar or diagonal 6×6 matrix (0.01)
ci - camera intrinsic structure (camparam)
depth - depth of points to use for Jacobian, scalar for

```
all points, of 4-vector.  If null take actual value
from simulation      ([])
```

SEE ALSO: ibvsplot

SphericalCamera.sph2

Implement spherical IBVS for point features

results = **sph**(**T**) **results** = **sph**(**T**, params)

Simulate IBVS with for a square target comprising 4 points is placed in the world XY plane. The camera/robot is initially at pose **T** and is driven to the orgin.

Two windows are shown and animated:

1. The camera view, showing the desired view (*) and the

```
current view (o)
```


2. The external view, showing the target points and the camera

The **results** structure contains time-history information about the image plane, camera pose, error, Jacobian condition number, error norm, image plane size and desired feature locations.

The params structure can be used to override simulation defaults by providing elements, defaults in parentheses:

target_size - the side length of the target in world units (0.5)

target_center - center of the target in world coords (0,0,3)

niter - the number of iterations to run the simulation (500)
eterm - a stopping criteria on feature error norm (0)
lambda - gain, can be scalar or diagonal 6×6 matrix (0.01)
ci - camera intrinsic structure (camparam)
depth - depth of points to use for Jacobian, scalar for

```
all points, of 4-vector.  If null take actual value
from simulation      ([])
```

SEE ALSO: ibvsplot

SphericalCamera.visjac_p

Visual motion Jacobian for point feature

J = C.**visjac_p**(**pt**, **z**) is the image Jacobian ($2N \times 6$) for the image plane points **pt** ($2 \times N$) described by phi (longitude) and theta (colatitude). The depth of the points from the camera is given by **z** which is a scalar, for all points, or a vector ($N \times 1$) for each point.

The Jacobian gives the image-plane velocity in terms of camera spatial velocity.

Reference

"Spherical image-based visual servo and structure estimation", P. I. Corke, in Proc. IEEE Int. Conf. Robotics and Automation, (Anchorage), pp. 5550-5555, May 3-7 2010.

See also

CentralCamera.visjac_p_polar, CentralCamera.visjac_l, CentralCamera.visjac_e

ssd

Sum of squared differences

m = ssd(**i1**, **i2**) is the sum of squared differences between the two equally sized image patches **i1** and **i2**. The result **m** is a scalar that indicates image similarity, a value of 0 indicates identical pixel patterns and is increasingly positive as image dissimilarity increases.

See also

zsdd, sad, ncc, isimilarity

---

stdisp

Display stereo pair

stdisp(**L**, **R**) displays the stereo image pair **L** and **R** in adjacent windows.

Two cross-hairs are created. Clicking a point in the left image positions black cross hair at the same pixel coordinate in the right image. Clicking the corresponding world point in the right image sets the green crosshair and displays the disparity [pixels].

See also

idisp, istereo

SurfPointFeature

SURF point corner feature object

A subclass of OrientedScalePointFeature for SURF features.

Methods

plot	Plot feature position
plot_scale	Plot feature scale
distance	Descriptor distance
ncc	Descriptor similarity
match	Match features
uv	Return feature coordinate
display	Display value
char	Convert value to string

Properties

u	horizontal coordinate
v	vertical coordinate
strength	feature strength
scale	feature scale
theta	feature orientation [rad]
descriptor	feature descriptor (vector)
image_id	index of image containing feature

Properties of a vector of SurfCornerFeature objects are returned as a vector. If F is a vector ($N \times 1$) of SurfCornerFeature objects then F.u is a $2 \times N$ matrix with each column the corresponding u coordinate.

Notes

- SurfCornerFeature is a reference object.
- SurfCornerFeature objects can be used in vectors and arrays

Reference

"SURF: Speeded Up Robust Features", Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, 2008

See also

isurf, PointFeature, ScalePointFeature, orientedscalepointfeature, SiftPointFeature

SurfPointFeature.SurfPointFeature

Create a SURF point feature object

f = **SurfPointFeature**() is a point feature object with null parameters.

f = **SurfPointFeature**(**u**, **v**) is a point feature object with specified coordinates.

f = **SurfPointFeature**(**u**, **v**, **strength**) as above but with specified strength.

f = **SurfScalePointFeature**(**u**, **v**, **strength**, **scale**) as above but with specified feature scale.

f = **SurfPointFeature**(**u**, **v**, **strength**, **scale**, **theta**) as above but with specified feature orientation.

See also

isurf, orientedscalepointfeature

SurfPointFeature.match

Match SURF point features

m = F.**match**(**f2**, **options**) is a vector of FeatureMatch objects that describe candidate matches between the two vectors of SURF features F and **f2**. Correspondence is based on descriptor similarity.

[**m**,**C**] = F.**match**(**f2**, **options**) as above but returns a correspodence matrix where each row contains the indices of corresponding features in F and **f2** respectively.

Options

'thresh', T	**match** threshold (default 0.05)
'median'	Threshold at the median distance

Notes

- for no threshold set to [].

See also

FeatureMatch

SurfPointFeature.support

Support region of feature

out = F.**support**(**im**, **w**) is an image of the **support** region of the feature F, extracted from the image **im** in which the feature appears. The **support** region is scaled to $\mathbf{w} \times \mathbf{w}$ and rotated so that the feature's orientation axis is upward.

out = F.**support**(**images**, **w**) as above but if the features were extracted from an image sequence **images** then the feature is extracted from the appropriate image in the same sequence.

[**out**,**T**] = F.**support**(**images**, **w**) as above but returns the pose of the feature as a 3×3 homogeneous transform in SE(2) that comprises the feature position and orientation.

F.**support**(**im**, **w**) as above but the **support** region is displayed.

See also

SurfPointFeature

---

symexpr2slblock

Create symbolic embedded MATLAB Function block

symexpr2slblock(**varargin**) creates an Embedded MATLAB Function block from a symbolic expression. The input arguments are just as used with the functions emlBlock or matlabFunctionBlock.

Notes

- In Symbolic Toolbox versions prior to V5.7 (2011b) the function to create Embedded Matlab Function blocks from symbolic expressions is 'emlBlock'.
- Since V5.7 (2011b) there is another function named 'matlabFunctionBlock' which replaces the old function.
- **symexpr2slblock** is a wrapper around both functions, which checks for the installed Symbolic Toolbox version and calls the required function accordingly.

Authors

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

emlblock, matlabfunctionblock

---

tb_optparse

Standard option parser for Toolbox functions

optout = **tb_optparse**(**opt**, **arglist**) is a generalized option parser for Toolbox functions. **opt** is a structure that contains the names and default values for the options, and **arglist** is a cell array containing option parameters, typically it comes from VARARGIN. It supports options that have an assigned value, boolean or enumeration types (string or int).

The software pattern is:

```
function(a, b, c, varargin)
opt.foo = false;
opt.bar = true;
opt.blah = [];
opt.choose = {'this', 'that', 'other'};
opt.select = {'#no', '#yes'};
opt = tb_optparse(opt, varargin);
```

Optional arguments to the function behave as follows:

'foo'	sets opt.foo := true
'nobar'	sets opt.foo := false
'blah', 3	sets opt.blah := 3
'blah', {x,y}	sets opt.blah := {x,y}
'that'	sets opt.choose := 'that'
'yes'	sets opt.select := (the second element)

and can be given in any combination.

If neither of 'this', 'that' or 'other' are specified then opt.choose := 'this'. Alternatively if:

```
opt.choose = {[], 'this', 'that', 'other'};
```

then if neither of 'this', 'that' or 'other' are specified then opt.choose := []

If neither of 'no' or 'yes' are specified then opt.select := 1.

Note:

- That the enumerator names must be distinct from the field names.
- That only one value can be assigned to a field, if multiple values are required they must placed in a cell array.
- To match an option that starts with a digit, prefix it with 'd_', so the field 'd_3d' matches the option '3d'.

- **opt** can be an object, rather than a structure, in which case the passed options are assigned to properties.

The return structure is automatically populated with fields: verbose and debug. The following options are automatically parsed:

'verbose'	sets opt.verbose := true
'verbose=2'	sets opt.verbose := 2 (very verbose)
'verbose=3'	sets opt.verbose := 3 (extremeley verbose)
'verbose=4'	sets opt.verbose := 4 (ridiculously verbose)
'debug', N	sets opt.debug := N
'showopt'	displays opt and arglist
'setopt', S	sets opt := S, if S.foo=4, and opt.foo is present, then opt.foo is set to 4.

The allowable options are specified by the names of the fields in the structure opt. By default if an option is given that is not a field of opt an error is declared.

[**optout**,**args**] = **tb_optparse**(**opt**, **arglist**) as above but returns all the unassigned options, those that don't match anything in **opt**, as a cell array of all unassigned arguments in the order given in **arglist**.

[**optout**,**args**,**ls**] = **tb_optparse**(**opt**, **arglist**) as above but if any unmatched option looks like a MATLAB LineSpec (eg. 'r:') it is placed in **ls** rather than in **args**.

---

testpattern

Create test images

im = **testpattern**(**type**, **w**, **args**) creates a test pattern image. If **w** is a scalar the image is **w** $\times$ **w** else **w**(2)xW(1). The image is specified by the string **type** and one or two (type specific) arguments:

'rampx'	intensity ramp from 0 to 1 in the x-direction. **args** is the number of cycles.
'rampy'	intensity ramp from 0 to 1 in the y-direction. **args** is the number of cycles.
'sinx'	sinusoidal intensity pattern (from -1 to 1) in the x-direction. **args** is the number of cycles.
'siny'	sinusoidal intensity pattern (from -1 to 1) in the y-direction. **args** is the number of cycles.
'dots'	binary dot pattern. **args** are dot pitch (distance between centres); dot diameter.
'squares'	binary square pattern. **args** are pitch (distance between centres); square side length.
'line'	a line. **args** are theta (rad), intercept.

Examples

A 256 $\times$ 256 image with 2 cycles of a horizontal sawtooth intensity ramp:

```
testpattern('rampx', 256, 2);
```


A 256×256 image with a grid of dots on 50 pixel centres and 20 pixels in diameter:

```
testpattern('dots', 256, 50, 25);
```

Notes

- With no output argument the **testpattern** in displayed using idisp.

See also

idisp

Tracker

Track points in image sequence

This class assigns each new feature a unique identifier and tracks it from frame to frame until it is lost. A complete history of all tracks is maintained.

Methods

plot	Plot all tracks
tracklengths	Length of all tracks

Properties

track	A vector of structures, one per active track.
history	A vector of track history structures with elements id and uv which is the path of the feature.

See also

PointFeature

Tracker.Tracker

Create new Tracker object

T = Tracker(**im**, **C**, **options**) is a new tracker object. **im** ($H \times W \times S$) is an image sequence and **C** ($S \times 1$) is a cell array of vectors of PointFeature subclass objects. The elements of the cell array are the point features for the corresponding element of the image sequence.

During operation the image sequence is animated and the point features are overlaid along with annotation giving the unique identifier of the track.

Options

'radius', R	Search radius for feature in next frame (default 20)
'nslots', N	Maximum number of tracks (default 800)
'thresh', **T**	Similarity threshold (default 0.8)
'movie', M	Write the frames as images into the folder M as with sequential filenames.

Notes

- The 'movie' options saves frames as files NNNN.png.
- When using 'movie' option ensure that the window is fully visible.
- To convert frames to a movie use a command like:

```
ffmpeg -r 10 -i %04d.png out.avi
```

See also

PointFeature

Tracker.char

Convert to string

s = T.char() is a compact string representation of the Tracker parameters and status.

Tracker.display

Display value

T.**display**() displays a compact human-readable string representation of the **Tracker** object

Notes

- This method is invoked implicitly at the command line when the result of an expression is a Tracker object and the command has no trailing semicolon.

See also

Tracker.char

Tracker.plot

Show feature trajectories

T.**plot**() overlays the tracks of all features on the current **plot**.

Tracker.tracklengths

Length of all tracks

T.**tracklengths**() is a vector containing the length of every track.

tristim2cc

Tristimulus to chromaticity coordinates

cc = **tristim2cc**(**tri**) is the chromaticity coordinate (1×2) corresponding to the tristimulus **tri** (1×3). If **tri** is RGB then **cc** is rg, if **tri** is XYZ then **cc** is xy. Multiple tristimulus values can be given as rows of **tri** $(N \times 3)$ in which case the chromaticity coordinates are the corresponding rows of **cc** $(N \times 2)$.

[**c1**,**C2**] = **tristim2cc**(**tri**) as above but the chromaticity coordinates are returned in separate vectors, each $N \times 1$.

out = **tristim2cc**(**im**) is the chromaticity coordinates corresponding to every pixel in the tristimulus image **im** ($H \times W \times 3$). **out** ($H \times W \times 2$) has planes corresponding to r and g, or x and y.

[**o1**,**o2**] = **tristim2cc**(**im**) as above but the chromaticity is returned as separate images ($H \times W$).

---

upq

Central image moments

m = **upq**(**im**, **p**, **q**) is the PQ'th central moment of the image **im**. That is, the sum of I(x,y).(x-x0)$^{\mathbf{p}}$.(y-y0)$^{\mathbf{q}}$ where (x0,y0) is the centroid.

Notes

- The central moments are invariant to translation.

See also

upq_poly, mpq, npq

---

upq_poly

Central polygon moments

m = **upq_poly**(**v**, **p**, **q**) is the PQ'th central moment of the polygon with vertices described by the columns of **v**.

Notes

- The points must be sorted such that they follow the perimeter in sequence (counter-clockwise).

- If the points are clockwise the moments will all be negated, so centroids will be still be correct.
- If the first and last point in the list are the same, they are considered as a single vertex.
- The central moments are invariant to translation.

See also

upq, mpq_poly, npq_poly

VideoCamera

Abstract class to read from local video camera

A concrete subclass of ImageSource that acquires images from a local camera using the MATLAB Image Acquisition Toolbox (imaq). This Toolbox provides a multiplatform interface to a range of cameras, and this class provides a simple wrapper.

This class is not intended to be used directly, instead use the factory method Video which will return an instance of this class if the Image Acquisition Toolbox is installed, for example

```
vid = VideoCamera();
```

Methods

grab	Aquire and return the next image
size	Size of image
close	Close the image source
char	Convert the object parameters to human readable string

See also

VideoCamera, ImageSource, AxisWebCamera, Movie

VideoCamera_fg

Class to read from local video camera

A concrete subclass of ImageSource that acquires images from a local camera using a simple open-source frame grabber interface.

This class is not intended to be used directly, instead use the factory method VideoCamera.which will return an instance of this class if the interface is supported on your platform (Mac or Linux), for example

```
vid = VideoCamera.amera();
```

Methods

grab	Aquire and return the next image
size	Size of image
close	Close the image source
char	Convert the object parameters to human readable string

See also

ImageSource, AxisWebCamera, Movie

VideoCamera_fg.VideoCamera_fg

Video camera constructor

V = **VideoCamera_fg**.CAMERA, OPTIONS) is a **VideoCamera_fg**.object that acquires images from the local video camera specified by the string CAMERA.

If CAMERA is '?' a list of available cameras, and their characteristics is displayed.

Options

'uint8'	Return image with uint8 pixels (default)
'float'	Return image with float pixels
'double'	Return image with double precision pixels
'grey'	Return greyscale image
'gamma', G	Apply gamma correction with gamma=G
'scale', S	Subsample the image by S in both directions.
'resolution', S	Obtain an image of size S=[W H].
'id', I	ID of camera

Notes:

- The specified 'resolution' must match one that the camera is capable of, otherwise the result is not predictable.

VideoCamera_fg.char

Convert to string

V.**char**() is a string representing the state of the camera object in human readable form.

VideoCamera_fg.close

Close the image source

V.**close**() closes the connection to the camera.

VideoCamera_fg.grab

Acquire image from the camera

im = V.**grab**() acquires an image from the camera.

Notes

- the function will block until the next frame is acquired.

VideoCamera_IAT

Class to read from local video camera

A concrete subclass of ImageSource that acquires images from a local camera using the MATLAB Image Acquisition Toolbox (imaq). This Toolbox provides a multiplatform interface to a range of cameras, and this class provides a simple wrapper.

This class is not intended to be used directly, instead use the factory method Video which will return an instance of this class if the Image Acquisition Toolbox is installed, for example

```
vid = VideoCamera();
```


Methods

grab	Aquire and return the next image
size	Size of image
close	Close the image source
char	Convert the object parameters to human readable string

See also

VideoCamera, ImageSource, AxisWebCamera, Movie

VideoCamera_IAT.VideoCamera_IAT

Video camera constructor

v = **Video_IAT**(**camera**, **options**) is a Video object that acquires images from the local video camera specified by the string **camera**.

Options

'uint8'	Return image with uint8 pixels (default)
'float'	Return image with float pixels
'double'	Return image with double precision pixels
'grey'	Return greyscale image
'gamma', G	Apply gamma correction with gamma=G
'scale', S	Subsample the image by S in both directions.
'resolution', S	Obtain an image of size S=[W H].
'id', I	ID of camera

Notes:

- The specified 'resolution' must match one that the camera is capable of, otherwise the result is not predictable.

VideoCamera_IAT.char

Convert to string

V.**char**() is a string representing the state of the camera object in human readable form.

VideoCamera_IAT.close

Close the image source

V.**close**() closes the connection to the camera.

VideoCamera_IAT.grab

Acquire image from the camera

im = V.**grab**() acquires an image from the camera.

Notes

- the function will block until the next frame is acquired.

VideoCamera_IAT.list

available adaptors and cameras

VideoCamera_IAT.preview

Control image preview

V.**preview**(**true**) enables camera **preview** in a separate window

---

xaxis

Set X-axis scaling

xaxis(**max**) set x-axis scaling from 0 to **max**.

xaxis(**min**, **max**) set x-axis scaling from **min** to **max**.

xaxis([**min max**]) as above.

xaxis restore automatic scaling for x-axis.

See also

yaxis

---

xycolorspace

Display spectral locus

xycolorspace() display a fully colored spectral locus in terms of CIE x and y coordinates.

xycolorspace(**p**) as above but plot the points whose xy-chromaticity is given by the columns of **p**.

[**im**,**ax**,**ay**] = **xycolorspace**() as above returns the spectral locus as an image **im**, with corresponding x- and y-axis coordinates **ax** and **ay** respectively.

Notes

- The colors shown within the locus only approximate the true colors, due to the gamut of the display device.

See also

rg_addticks

---

xyzlabel

Label X, Y and Z axes

XYZLABEL label the x-, y- and z-axes with 'X', 'Y', and 'Z' respectiveley

---

yaxis

Y-axis scaling

yaxis(**max**) set y-axis scaling from 0 to **max**.

yaxis(**min**, **max**) set y-axis scaling from **min** to **max**.

yaxis([**min max**]) as above.

yaxis restore automatic scaling for y-axis.

See also

yaxis

YUV

Class to read YUV4MPEG file

A concrete subclass of ImageSource that returns images from a YUV4MPEG format uncompressed video file.

Methods

grab	Aquire and return the next image
size	Size of image
close	Close the image source
char	Convert the object parameters to human readable string

Properties

curFrame	The index of the frame just read

See also

ImageSource, video

SEE ALSO: Video

YUV.YUV

YUV4MPEG sequence constructor

y = YUV(**file**, **options**) is a YUV4MPEG object that returns frames from the yuv4mpeg format file **file**. This file contains uncompressed color images in 4:2:0 format, with a full resolution luminance plane followed by U and V planes at half resolution both directions.

Options

'uint8'	Return image with uint8 pixels (default)
'float'	Return image with float pixels
'double'	Return image with double precision pixels
'grey'	Return greyscale image
'gamma', G	Apply gamma correction with gamma=G
'scale', S	Subsample the image by S in both directions
'skip', S	Read every S'th frame from the movie

YUV.char

Convert to string

M.**char**() is a string representing the state of the movie object in human readable form.

YUV.close

Close the image source

M.**close**() closes the connection to the movie.

YUV.grab

Acquire next frame from movie

im = Y.**grab**(**options**) is the next frame from the file.

[**y**,**u**,**v**] = **y**.**grab**(**options**) is the next frame from the file

Options

'skip', S	Skip frames, and return current+S frame (default 1)
'rgb'	Return as an RGB image, **y** image is downsized by two (default).
'rgb2'	Return as an RGB image, **u** and **v** images are upsized by two.
'yuv'	Return **y**, **u** and **v** images.

Notes

- If no output argument given the image is displayed using IDISP.
- For the 'yuv' option three output arguments must be given.

---

zcross

Zero-crossing detector

iz = zcross(**im**) is a binary image with pixels set where the corresponding pixels in the signed image **im** have a zero crossing, a positive pixel adjacent to a negative pixel.

Notes

- Can be used in association with a Lapalacian of Gaussian image to determine edges.

See also

ilog

---

zncc

Normalized cross correlation

m = zncc(**i1**, **i2**) is the zero-mean normalized cross-correlation between the two equally sized image patches **i1** and **i2**. The result **m** is a scalar in the interval -1 to 1 that indicates similarity. A value of 1 indicates identical pixel patterns.

Notes

- The **zncc** similarity measure is invariant to affine changes in image intensity (brightness offset and scale).

See also

ncc, sad, ssd, isimilarity

---

zsad

Sum of absolute differences

m = **zsad**(**i1**, **i2**) is the zero-mean sum of absolute differences between the two equally sized image patches **i1** and **i2**. The result **m** is a scalar that indicates image similarity, a value of 0 indicates identical pixel patterns and is increasingly positive as image dissimilarity increases.

Notes

- The **zsad** similarity measure is invariant to changes in image brightness offset.

See also

sad, ssd, ncc, isimilarity

---

zssd

Sum of squared differences

m = **zssd**(**i1**, **i2**) is the zero-mean sum of squared differences between the two equally sized image patches **i1** and **i2**. The result **m** is a scalar that indicates image similarity, a value of 0 indicates identical pixel patterns and is increasingly positive as image dissimilarity increases.

Notes

- The **zssd** similarity measure is invariant to changes in image brightness offset.

See also

sdd, sad, ncc, isimilarity

