Imago Documentation

Release 0.1

Tomáš Musil

November 03, 2016

Contents

Contents:

CHAPTER 1

Introduction

Imago is a program for automatic processing of Go images. It takes an image (or a set of images), finds the board-grid and stones and produces an abstract representation of the game situation (or a game record).

It will supports JPEG, BMP, TIFF (and other) image formats on input. It is capable of output to ASCII and SGF format. As the process is fully automatic, the program is operated from command line.

CHAPTER 2

Installation

Just run *make* in the imago folder.

CHAPTER 3

Usage

Go image recognition.

```
usage: imago [-h] [-w W] [-m] [-d] [-s] [-c] [-S] [-v] [--rng-seed RNG_SEED]
             file [file ...]
```

Positional arguments:

files image to analyse

Options:

-w=640 scale image to the specified width before analysis

-m=False, --manual=False manual grid selection

-d=False, --debug=False show every step of the computation

-s=False, --save=False save images instead of displaying them

-c=False, --cache=False use cached lines

-S=False, --sgf=False output in SGF

-v=False, --verbose=False report progress

--rng-seed Specify random number generator seed, for consistent test results.

CHAPTER 4

Modules

4.1 imago

Go image recognition.

imago.**main**()
Main function of the program.

4.2 hough

Hough transform module.

class hough.**Hough**(*size*, *dt*, *init_angle*)
Hough transform.

This class stores the parameters of the transformation.

angle_distance(*point*)
Take a point from the transformed image and return the corresponding line in the original as (angle, distance) tuple.

classmethod default(*image*)
Default parameters for Hough transform of the *image*.

lines_from_list(*p_list*)
Take a list of transformed points and return a list of corresponding lines as (angle, distance) tuples.

4.3 ransac.py

RANSAC estimation.

class ransac.**Linear_model**(*data*)
Linear model for RANSAC.

ransac.**estimate**(*data*, *dist*, *k*, *modelClass=<class ransac.Linear_model>*, *model=None*)
Estimate model from data with RANSAC.

ransac.**filter_near**(*data*, *line*, *distance*)
Find points in *data* that are closer than *distance* to *line*.

ransac.**least_squares**(*data*)
The least squares method.

ransac.**points_to_line**(*(x1, y1), (x2, y2)*)
Take two points, return coefficitiens for line that connects them.

4.4 geometry

Imago geometry module.

class geometry.**V**(*x*, *y*)
Class for vector manipulation.

geometry.**intersection**(*p*, *q*)
Return intersection of two lines.

geometry.**l2ad**(*(a, b), size*)
Represent line as (angle, distance).

Take a line (represented by two points) and image size. Return the line represented by its angle and distance from the center of the image.

geometry.**line**(*x*, *y*)
Given two points *x* and *y*, compute the line joining them.

x and *y* are 2-tuples of the (x, y) coordinates of the points.

The function returns a 3-tuple (a, b, c) such that the line through the points *x* and *y* is described by the equation $ax + by = c$.

4.5 filters

Image filters module.

All filters return a filtered copy of the image, the original image is preserved.

filters.**color_enhance**(*image*)
Stretch all color channels to their full range.

filters.**edge_detection**(*image*)
Edge detection (on BW images).

filters.**high_pass**(*image*, *height*)
High pass filter (on BW images).

filters.**peaks**(*image*)
Peak filter (on BW images).

4.6 linef

Lines finding module.

linef.**find_lines**(*image*, *show_image*, *logger*)
Find lines in the *image*.

linef.**line_from_angl_dist**(*(angle*, *distance)*, *size*)
Take *angle* and *distance* (from the center of the image) of a line and size of the image. Return the line represented by two points.

linef.**transform**(*image*, *hough*, *show_image*)
Produces a simplified Hough transformation of the input image.

4.7 gridf_new

Imago grid fitting module

RANSAC inspired method.

class gridf_new.**Diagonal_model**(*data*)
Ransac model for finding diagonals.

class gridf_new.**Line**(*(a*, *b*, *c)*)
Line with a list of important points that lie on it.

This and the Point class in this module serves to implement a model of perspective plain – a line has a list of intersections with other lines and each intersection has two lines that go through it.

class gridf_new.**Point**(*(x*, *y)*)
Class that represents a point in 2D.

gridf_new.**dst**(*p*, *l*)
Distance from a point to a line.

gridf_new.**find**(*lines*, *size*, *l1*, *l2*, *bounds*, *hough*, *show_all*, *do_something*, *logger*)
Find the best grid given the *lines* and *size* of the image.

lines: A list containing two lists of lines detected in the image. Each list of lines contains lines that are more-or-less parallel to each other. The lines in the two lists are more-or-less perpendicular to each other.

size: A list of (width, height) of the input image, in pixels.

Last three parameters serves for debugging, *l1*, *l2*, *bounds* and *hough* are here for compatibility with older version of gridf, so they can be easily exchanged, tested and compared.

gridf_new.**gen_corners**(*d1*, *d2*, *min_size*)
Generate candidates on corner positions from the diagonals.

gridf_new.**intersection**(*(a1*, *b1*, *c1)*, *(a2*, *b2*, *c2)*)
Intersection of two lines, given by coefficients in their equations.

gridf_new.**plot_line**(*line*, *c*, *size*)
Plot a *line* with pyplot.

4.8 intrsc

Imago intersections module.

intrsc.**b_intersects**(*image*, *lines*, *show_all*, *do_something*, *logger*)
Compute intersections.

intrsc.**board**(*image*, *intersections*, *show_all*, *do_something*, *logger*)
Find stone colors and return board situation.

intrsc.**dst**(*line*)
Return normalized line.

intrsc.**dst_sort**(*lines*)
Return lines sorted by distance.

intrsc.**intersections_from_angl_dist**(*lines*, *size*, *get_all=True*)
Take grid-lines and size of the image. Return intersections.

intrsc.**mean_luma**(*cluster*)
Return mean luminanace of the *cluster* of points.

intrsc.**rgb2lumsat**(*color*)
Convert RGB to luminance and HSI model saturation.

intrsc.**stone_color_raw**(*image*, *(x, y)*)
Given image and coordinates, return stone color.

4.9 k_means

K-means module.

k_means.**centroid**(*cluster*)
Find the centroid of the *cluster*.

k_means.**cluster**(*k*, *d*, *data*, *i_centers=None*)
Find *k* clusters on *d* dimensional *data*.

k_means.**delta**(*c1*, *c2*)
Find the absolute distance between two lists of points.

k_means.**nearest**(*centers*, *point*)
Find the nearest cluster *center* for *point*.

k_means.**next_step**(*centers*, *data*)
Compute new clusters and centers.

4.10 manual

Manual grid selection module

4.11 output

Imago output module.

class output.**Board**(*size*, *stones*)
Represents the state of the board.

addMove(*m*)
Add move to the board.

asSGFsetPos()
Returns SGF (set position) representation of the position.

getMoveCandidates(*board*)
Take the next board in game and return a list of moves that are new.

class output.**Game**(*size*, *board=None*, *debug=True*)
Represents a game.

addMove(*board*)
Add next move to the game.

asSGF()
Return the game representation as SGF string.

class output.**Move**(*color*, *y*, *x*, *comment=None*)
Repsresents a move.

sgf_coords()
Return coordinates of the move in SGF.

4.12 im_debug

Debugging image display.

This is a simple module that shows images on screen using PyGame. It is not used anywhere in the standard UI, serves only for debugging.

im_debug.**show**(*image*, *caption=''*, *name=None*)
Initialize PyGame and show the *image*.

4.13 camera

Camera module.

This module handles various backends (different for every OS) for streaming the video from a (web)camera.

class camera.**Camera**(*vid=0*, *res=None*)
Implement basic camera capabilities

This class has different implementations for different OS. On posix systems it calls to opencv, on Windows to VideoCapture.

get_image()
Get a new image from the camera.

4.14 capture

Go image capture.

This module defines a UI for capturing images with a (web)camera and imediately proccessing them with Imago.

class capture.**Capture**(*device*, *res*)
This object maintains communication between the camera, the PyGame screen and Imago.

auto(*interval*)
Take new image every *interval* seconds.

live(*q*)
Run live preview on the screen.

manual()
Take images manualy by pressing a key.

take()
Take a new image from the camera.

class capture.**Screen**(*res*)
Basic PyGame setup.

display_picture(*im*)
Display image on PyGame screen.

capture.**main**()
Parse the argument and setup the UI.

4.15 timer

Go timer.

This module defines a UI that combines game clock with automatic game recording. When player presses the clock, an image of the board is taken. The sequence of images can later be analysed by Imago to produce a game record in abstract notation.

class timer.**Timer**(*main_time*, *byop*, *byot*)
Keeps track of time remaining one player.

get_time()
Return string representation of remaining time.

is_running()
Return True if the clock is running.

start()
Start the clock.

stop()
Stop the clock.

timer.**main**()
Parse the arguments and run the clock.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

c

f

g

h

i

k

l

m

o

r

t

N

O

P

R

S

T

V