PyCelegans Documentation

Release 0

Nicholas P. Labello

May 11, 2012

CONTENTS

CHAPTER

ONE

BACKGROUND

This executable builds a composite background image by taking the brightest pixel from a series of images. By compositing multiple images the worm, which blocks light and always darkens a pixel, is effectively removed leaving a background which can be subtracted from individual images. Subtracting a background image results in much cleaner segmentation and subsequent identification of worm properties (head, tail, sides, etc.).

usage: background.py [-h] [–imgdir IMGDIR] [–Nimg NIMG]

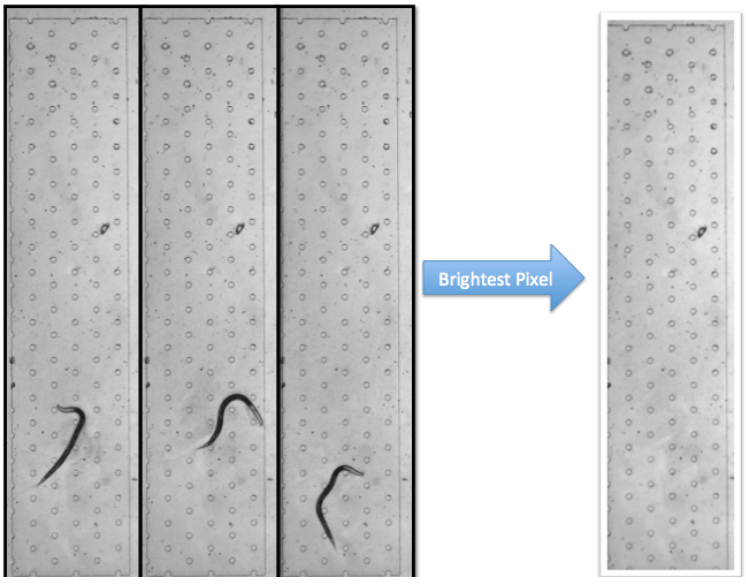
optional arguments:

-h, --help show this help message and exit

--imgdir IMGDIR directory where input images are stored

--Nimg NIMG Number of images to use to construct the background. The images are spaced as far apart as possible. For example, if 100 images are available and Nimg=10, then image 1,11,21,31...91 would be used.

--windowsize WINDOWSIZE Width of window used for background image generation. This is the number of sequential frames that will use the same background image. For example, if there are 10,000 frames, and –windowsize=1000, ten background images will be generated. (default: 1)



CHAPTER

TWO

CREATEDIFFIMAGE

This stand alone executable builds a new image with the following operation.

((background - image) > THRESH) * image

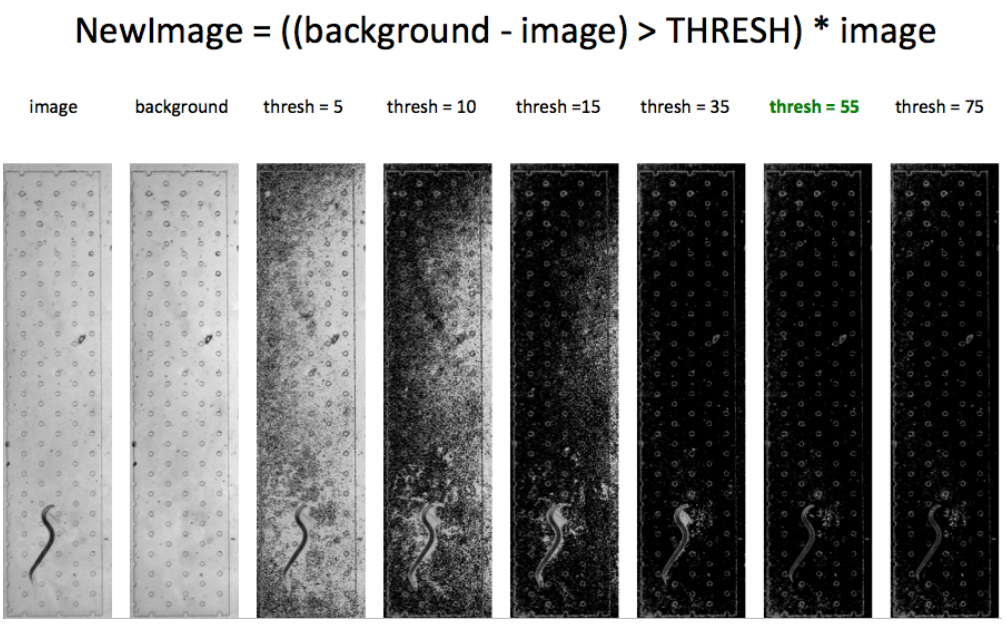
background will be approximately equally bright as *image* on every pixel that the worm does not occupy in *image* (roughly +/- 10 based on 0-255 range possible with 8 bit grayscale images).

Pixels that the worm occupies will be substantially darker (lower) in *image* than background, by an intensity of 20 or more. The image that is segmented and used for analysis is generated by the following steps.

1. **subtract *image* from *background*, result 8 bit array** tempimage = background - image
2. **generate binary array where *tempimage* is greater than THRESH** temp2image = tempimage > THRESH
3. **multiply *temp2image* by *image*.** finalimage = temp2image * image

As a result, if THRESH is chosen well, *finalimage* will contain only one major object, the worm. Segmentation and analysis of *finalimage* results in vastly cleaner identification of worm head/tail/border than segmentation/analysis of

- Usage: creatediffimage.py [background.jpg] [image.jpg] [THRESH]



CHAPTER

THREE

PREPROCESSAJP

This executable splits AJP movies into individual jpeg images. Given a root directory preprocessajp will traverse down the file system recursively to find all .ajp files. Images are named with the movie prefix and a unique frame number.

usage: preprocessajp.py [-h] [–np NPROCS] [–imgdir IMGDIR] –ajpdir AJPDIR [–prefix PREFIX] [–nzerom NZEROM] [–nzeroa NZEROA]

Convert AJP movies to images. Usage: preprocessajp.py [Directory of AJP Files]

optional arguments:

-h, --help	show this help message and exit
--np NPROCS	number of processors
--imgdir IMGDIR	directory where output images are stored
--ajpdir AJPDIR	directory where input AJP movies are found
--prefix PREFIX	prefix added to jpg file name
--nzerom NZEROM	width of zero padding on the frame numbers
--nzeroa NZEROA	width of zero padding on the total frame number, continuous from first movie to last

CHAPTER

FOUR

COLLECTOUTPUT

This executable compiles output from the image processing runs.

CHAPTER

FIVE

UTIL

util contains functions that operate on the file system, read data from the file system, and operate on the aggregate data. These functions are outside of the scope of the operations applied to the C. elegans image and have more to do with the practical aspects of building a cohesive program.

libutil.**get_data_chunks**(*FileNames*, *nchunks*)
Split a list into *nchunks*, return list of lists.

libutil.**get_file_names**(*dir*, *filter='.jpg'*)
Get all file names that contain the string indicated by *filter*. This function will recursively search down the directory tree starting at *dir*.

libutil.**makedir**(*dirname*)
Make a directory on the file system.

libutil.**split_list**(*l*, *n=1*)
split a list l into many lists of size n the last item will be less than n if len(l) not evenly divisible

CHAPTER

SIX

MODULE PYCELEGANS

pycelegans is a collection of tools built from gray-scale and binary morphological functions intended to address image segmentation and image analysis problems involving the study of C. elegans.

class libcelegans.**Point**
Point(x, y)

x
Alias for field number 0

y
Alias for field number 1

class libcelegans.**WormBorderPoint**
WormBorderPoint(x, y, score)

score
Alias for field number 2

x
Alias for field number 0

y
Alias for field number 1

libcelegans.**fill_object**(*Arr*, *i=2*)

- **Usage** NewArr = get_border(Arr,i=[INTEGER])
- **Input** Arr: Binary Image stored as Numpy Array
- **Output** NewArr: Binary Image
- **Description** Intended for use with a binary array that contains only one object. Takes a binary array, returns a binary array of the same dimensions. The returned array has been dilated by *i* iterations, had a binary_fill_holes morphological operation applied, and then been eroded by *i* iterations. As a result any holes that may have resulted from initial segmentation are filled. Holes are most common in the lighter regions of the interior of the worm (head/neck region).

libcelegans.**find_true_neighbors**(*Arr*, *point*)

- **Usage** ListOfPoints = find_true_neighbors(Arr,point)
- **Input** Arr: Numpy Array, point: Point class of coordinates in the array
- **Output** a list of **Point**s
- **Description** Return a list of points that contains all of the True neighbors that exist in the binary array for the point passed in.

libcelegans.**get_absolute_pixel_neighbors**(*P*, *k*)

- **Usage** ListOfPoints = get_absolute_pixel_neighbors(Arr)
- **Input** k: Any positive integer
- **Output** ListOfPoints: a list of :Point:s
- **Description** Returns the the *k*th generation of neighbors around a pixel. k = 1 returns the 8-connected neighbors. k = 2 returns the union of k = 1 and the 8-connected neighbors of all points in the k = 1 set.

libcelegans.**get_biggest_object**(*Arr*)

- **Usage** NewArr = get_border(Arr)
- **Input** Arr: Binary Image stored as Numpy Array
- **Output** NewArr: Binary Image of largest object
- **Description** Takes a binary array, returns a binary array of the same dimensions. The returned array will contain only the largest single object that could be identified in the image. This is ALWAYS the worm if the background image has been correctly substracted.

libcelegans.**get_border**(*Arr*)

- **Usage** NewArr = get_border(Arr)
- **Input** Arr: Binary Image stored as Numpy Array
- **Output** NewArr: Binary Image
- **Description** Intended for use with a binary array that contains only one object. Takes a binary array, returns a binary array of the same dimensions. The returned array will contain the one-pixel thick border of the object in the array passed in. Array passed in should have one object only: the worm.

libcelegans.**get_distance**(*p1*, *p2*)

- Input p1,p2: Point class
- Output floating point value, two dimensional distance between p1 and p2
- Description: Returns the distance between two pixels

libcelegans.**get_image**(*file_pathname*)

- **Usage** NewArr = get_image(file_pathname)
- **Input** file_pathname: Full path and name to the image on your file system
- **Output** NewArr: the intensity values of the image loaded as integers in a Numpy array
- **Description** Thin wrapper around scipy.misc.imread.

libcelegans.**get_midline**(*SideOnePath*, *SideTwoPath*, *jrange=12*)

- **Usage** ListOfPoints = get_midline(SideOnePath,SideTwoPath)
- **Input** SideOnePath and SideTwoPath: Both are lists of ordered pairs that represent the a path through the perimeter pixels (worm border) of the array, from head to tail. len(SideOnePath) must == len(SideTwoPath). jrange: integer
- **Output** ListOfPoints: a list of :Point:s
- **Description** This function finds the midline of the worm based on the two ordered paths passed in. Each point in SideOnePath is paired to a point in SideTwoPath. Point *i* in SideOnePath is compared to point [i-jrange, i-jrange+1, ... i+jrange] points on SideTwoPath. The closest points in two-dimensional space is used as the match. Midline-Point[i] is taken as the average of these two points.

Note: A larger jrange will sample more points and generally be more accurate, though exceptions are possible. Generally this should be based on the number of points that is chosen to represent the length of the of worm, len(SideOnePath) or len(SideTwoPath). jrange of 5-20% of this length is probably reasonable though experimentation may be necessary.

libcelegans.**get_point_list**(*Arr*)

- **Usage** ListOfPoints = get_point_list(Arr)
- **Input** Arr: Binary Image stored as Numpy Array
- **Output** ListOfPoints
- **Description** Takes a binary array, returns a Python list of the True or Nonzero elements in the array. Each item in the list is an (x,y) tuple

libcelegans.**get_relative_pixel_neighbors**(*k*)

- **Usage** ListOfPoints = get_relative_pixel_neighbors(Arr)
- **Input** k: Any positive integer
- **Output** ListOfPoints: a list of **Point**s
- **Description** Returns the transforms necessary to generate the *k*th generation of neighbors around a pixel. For example, if k = 1, ListOfPoints returns the transforms [(-1,-1), (-1,0), ... (1,1)] necessary to get the 8 neighboring connected pixels. (0,0) is excluded.

libcelegans.**get_side_paths**(*HeadArr*, *TailArr*, *BorderRoute*)

- Usage

ListOfPoints = get_midline(SideOnePath,SideTwoPath) * Input SideOnePath and SideTwoPath: Both are lists of ordered pairs that represent the a path through the perimeter pixels (worm border) of the array, from head to tail. len(SideOnePath) must == len(SideTwoPath). jrange: integer * Output ListOfPoints: a list of :Point:s * Description This function finds the midline of the worm based on the two ordered paths passed in. Each point in SideOnePath is paired to a point in SideTwoPath. Point *i* in SideOnePath is compared to point [i-jrange, i-jrange+1, ... i+jrange] points on SideTwoPath. The closest points in two-dimensional space is used as the match. Midline-Point[i] is taken as the average of these two points.

Note: A larger jrange will sample more points and generally be more accurate, though exceptions are possible. Generally this should be based on the number of points that is chosen to represent the length of the of worm, len(SideOnePath) or len(SideTwoPath). jrange of 5-20% of this length is probably reasonable though experimentation may be necessary.

libcelegans.**get_thresh**(*Arr*, *thresh*)

- **Usage** NewArr = get_thresh(Arr,thresh)
- **Input** Arr: greyscale image stored as Numpy Array thresh: integer that should be between 0-255 is Arr is generated from 8-bit grayscale
- **Output** NewArr: image with original Arr values where Arr > thresh.
- Description thresh refers to the pixel intensity. Assuming 8-bit grayscale values between 0 - 255 are logical.

libcelegans.**perform_head_tail_intensity_test**(*HeadArr*, *TailArr*, *WormArr*)

- **Usage** bool_result = perform_head_tail_intensity_test(HeadArr,TailArr,WormArr)
- **Input** Head, Tail, and Worm Arrays
- **Output** True or False. If Head region is more intense, True, test passed.

- **Description** Check regions in worm around what has been identified as Head and Tail. This test dilates the head and tail points and averages the intensity of the points that fall within the worm body. Since the Head is brighter than the tail, it will both have an average higher intensity. Thus, the value computed for the Head should be greater. If not, the identification fails this test.

libcelegans.**perform_head_tail_volume_test**(*HeadArr*, *TailArr*, *WormArr*)

- **Usage** bool_result = perform_head_tail_volume_test(HeadArr,TailArr,WormArr)
- **Input** Head, Tail, and Worm Arrays
- **Output** True or False. If Head region has a greater volume, test passed.
- **Description** Check regions in worm around what has been identified as Head and Tail. This test dilates the head and tail points and counts the number of points that fall within the worm body. Since the Head is thicker than the tail, it will it should have more pixels. Thus, the value computed for the Head should be greater. If not, the identification fails this test.

libcelegans.**save_wormviz_image**(*vizdir*, *imname*, *Image*, *red*, *green=None*, *blue=None*, *magneto=None*, *yellow=None*, *cyan=None*)

Save an image with color overlays. vizdir: directory where image will be saved imname: name of saved image, include ".jpg" Image: The base grayscale image. red,green,blue,magneto,yellow,cyan: binary arrays that should be

overlayed. Pass "None" to select a color out of order. e.g., with one binary array only that should be blue, output = save_wormviz_image(vizdir,imname,Image,None,None,BlueArr)

PYTHON MODULE INDEX

INDEX