PINT
R Function Documentation

Bhupendra A. Raut

September 21, 2020

PINT: a Prototype for Integrated Non-physical Tracking of Convective Features in Meteorological Data

PINT is an R implementation of the efficient and adaptive algorithm for tracking convection in radar, satellite, and simulated data. It uses the Fourier phase shift method to compute the first guess of the motion and the Hungarian method for optimized matching. The algorithm also checks merging and splitting of the objects based on their size and overlap with the other objects. The algorithm works with bi-level images. Therefore, the tracks are estimated independent of the physical properties of the objects, such as brightness temperature or reflectivity. "TINT is not Titan" is the python versin based on the PINT. https://github.com/openradar/TINT

The package is made from the R script located at https://github.com/RBhupi/Darwin-Rscripts/blob/master/echo_tracking.R

R topics documented:

`attach_xyDist` *Attaches y and x distance from radar in km to object location indices*

Description

Attaches y and x distance from radar in km to object location indices

Usage

```
attach_xyDist(obj_props, xdist, ydist)
```

`attach_xyheads` *saves last x y movements of the objects.*

Description

Attaches last xyheads to current objects for future use.

Usage

```
attach_xyheads(frame1, frame2, current_objects)
```

check_bigOverlap *checks overlapping regoin for big objects in both the frames.*

Description

Checks overlapping area in pixels, size of the object and return if overlapping is considerable.

Usage

```
check_bigOverlap(obj_extend, target_extend)
```

check_merging *Checks possible merging of the dead objects.*

Description

This function takes in two R-lists containing information about current objects in the frame1 and their properties, such as center location and area. If the I am using an arbitrary crieterion for merging. If euclidean distance between centers of the two objects c_dist < or = to r=sqrt(area), then merging is considered. Here, if we assume square objects, then the r is length of a sides of the square. If all objects are dead in frame2 then no merging happened.

Usage

```
check_merging(dead_obj_id1, current_objects, frame1, frame2)
```

check_searchBox *checks that the search box is in the domain.*

Description

Returns NA if search box outside the image or search box is very small.

Usage

```
check_searchBox(search_box, img_dims)
```

clear_smallEchoes *Removed objects smaller than the given size.*

Description

Takes in labeled image removes objects smaller than min_size and returns re-labeled image. Pixels attached diagonally are not considered as continuouse part of the object.

Usage

```
clear_smallEchoes(label_image, min_size)
```

correct_shift *FFT shifts are corrected using last headings.*

Description

takes in flow vector based shift and current_object dataframe which has last headings, and check if they are resonably close if not rejects or modify shift and return. Note: frame2 of last timestep is now frame1, but current_objects still has it as frame2. So id2 in the last frame2 are actually ids related to frame1 now.

Usage

```
correct_shift(this_shift, current_objects, object_id1)
```

create_outNC_track *Creates output netcdf file for radar echo trajectories.*

Description

This is the longest function.

Usage

```
create_outNC_track(ofile, max_obs)
```

Arguments

ofile	name and path of the output file.
max_obs	longest possible track record. Deafult maximum 65 observation per track.

`euclidean_dist` *standard Euclidean distance.*

Description

Returns Euclidean distance between two vectors or matrices.

Usage

```
euclidean_dist(vec1, vec2)
```

`fft_crossCov` *Computes cross-covariance using FFT method, returns shifted covariance image*

Description

Computes cross-covariance using FFT method, returns shifted covariance image

Usage

```
fft_crossCov(img1, img2)
```

`fft_flowVectors` *Estimates flow vectors in two images using cross covariance of the images.*

Description

Leese, John A., Charles S. Novak, and Bruce B. Clark. "An automated technique for obtaining cloud motion from geosynchronous satellite data using cross correlation." Journal of applied meteorology 10.1 (1971): 118-132.

Usage

```
fft_flowVectors(im1, im2)
```

`fft_shift` *Rearranges the crossCov matrix*

Description

Rearranges the crossCov matrix so that 'zero' frequency or DC component is in the middle of the matrix. Taken from stackoverflow Que. 30630632

Usage

```
fft_shift(fft_mat)
```

`filterFrame` *filters frame for small objects when frame is already available.*

Description

filters frame for small objects when frame is already available.

Usage

```
filterFrame(frame, min_size = 2)
```

`find_objects` *Returns vector of objects ids in the given reion.*

Description

Given the search box and image2, returns objects in the region.

Usage

```
find_objects(search_box, image2)
```

`find_origin` *Checks for parent in the vicinity.*

Description

This function checks near by objects in the frame for the given new-born object. origin is an object which existed before the new born objects, has comparable or larger size and is close enough to the offspring.

Usage

```
find_origin(id1_newObj, frame1, old_frame1)
```

`fitEllipse` *Try to fit optimum ellipse for the object circularity, given the object index. The ellipse fitted with this method does not enclosed the object but it provide optimum ellipse parameters. The ratio of major and minor axis and eccentricity are correctly estimated for tracking purpose.*

Description

Try to fit optimum ellipse for the object circularity, given the object index. The ellipse fitted with this method does not enclosed the object but it provide optimum ellipse parameters. The ratio of major and minor axis and eccentricity are correctly estimated for tracking purpose.

Usage

```
fitEllipse(object_index)
```

`get_disparity` *Actually computes desparity for a single object.*

Description

Check how it is computed for detail. This parameter has most effect on the acccuracy of the tracks.

Usage

```
get_disparity(obj_found, image2, search_box, obj1_extent)
```

`get_disparity_all` *Returns Disparity of all the objects in the region.*

Description

Returns disparities of all the objects found within the search box or NA if no object is present.

Usage

```
get_disparity_all(obj_found, image2, search_box, obj1_extent)
```

get_filteredFrame *Returns a single radar scan from the input netcdf file.*

Description

Smaller objects are removed and the rest are lebeled.

Usage

```
get_filteredFrame(ncfile, var_id, scan_num, min_size = 2)
```

Arguments

ncfile	input netcdf file object from ncdf4 package.
var_id	string name of the varibale in the file.
scan_num	index of frame to be read from the file.
min_size	in pixels objects smaller than this will be removed. Default 2.

get_matchPairs *returns pairs of object ids from both frames.*

Description

Given two images, the function identifies the matching objects and pair them appropriatly using Hungarian method and desparity.

Usage

```
get_matchPairs(image1, image2)
```

See Also

codedisparity function.

get_objAmbientFlow *Computes flow in the vicinity of the object.*

Description

Takes in object info (major axis and center) and two images to estimate ambient flow using FFT phase correlation. Margin is the additional region arround the object used to comput the flow vectors.

Usage

```
get_objAmbientFlow(obj_extent, img1, img2, margin)
```

get_objectCenter *return center indices of the object.*

Description

Returns indices of center pixel of the given object id from a labeled image. This may be done in better way for non-oval objects.

Usage

```
get_objectCenter(obj_id, labeled_image)
```

get_objectProp *Return all the object's size, location and classification info.*

Description

xyDist should be a list of x_dist and y_dist in km.

Usage

```
get_objectProp(image1, xyDist)
```

get_objExtent *Returns object extent properties.*

Description

Takes in a labeled image and finds the radius, area and the center of the given object.

Usage

```
get_objExtent(labeled_image, obj_label)
```

get_origin_uid *Find id of the parent of the new born object.*

Description

returns unique id of the origin (or zero) for given object in frame1. Also remember that old object id2 is actual id1 in frame1, as we still have to update the object_ids.

Usage

```
get_origin_uid(obj, frame1, old_objects, old_frame1)
```

`get_sizeChange` *returns size change between the frames.*

Description

Returns change in size of the echo.

Usage

```
get_sizeChange(x, y)
```

`get_std_flowVector` *Alternative to get_objAmbientFlow.*

Description

Flow vectors magnitude is clipped to given magnitude to controll erratic output from FFT flow.

Usage

```
get_std_flowVector(obj_extent, img1, img2, margin, magnitude)
```

`init_uids` *Returns a dataframe for objects with unique ids and their corresponding ids in frame1 and frame2.*

Description

This function is called when new rainy scan is seen after the period of no rain or the first time.

Usage

```
init_uids(first_frame, second_frame, pairs)
```

Arguments

`first_frame`	First image for tracking.
`second_frame`	Second image for tracking.
`pairs`	output of `get_match_pairs()`

`locate_allObjects` *Matches all the obejects in image1 to the objects in image2.*

Description

This is the main function to be called on two sets of radar images, for tracking.

Usage

```
locate_allObjects(image1, image2)
```

`match_pairs` *Matches objects into pairs and removes bad matching.*

Description

The bad matching is when disparity is more than the set value.

Usage

```
match_pairs(obj_match)
```

`next_uid` *Returns sequence of next unique ids and increament the uid_counter.*

Description

Returns sequence of next unique ids and increament the uid_counter.

Usage

```
next_uid(count = 1)
```

`plot_objects_label` *Plots image with objects labels.*

Description

This is used in development stage to test images when processed 1-by-1.

Usage

```
plot_objects_label(labeled_image, xvalues, yvalues)
```

predict_searchExtent *predict search region.*

Description

Predicts search extent/region for the object in image2 given the image shift.

Usage

```
predict_searchExtent(obj1_extent, shift)
```

save_objMatch *Corrects and saves disparity for the object matching stage.*

Description

If disparity is large then it saves a large number for the value to reduce the chances of this pairing to zero, else it save the value in the obj_match array.

Usage

```
save_objMatch(obj_id1, obj_found, disparity, obj_match)
```

survival_stats *Returns a list with number of objects lived, died and born between the current and the previousstep.*

Description

Returns a list with number of objects lived, died and born between the current and the previousstep.

Usage

```
survival_stats(pairs, num_obj2)
```

`update_current_objects` *Removes dead objects, updates living objects and assign new uids to new born objects.*

Description

Also, updates number of valid observations for each echo. This function is called when rain continues from the last frame. This is a complicated function to understand.

Usage

```
update_current_objects(old_frame1, frame1, frame2, pairs, old_objects)
```

Details

See how the pairs vector looks like for a real case. The pairs shows mapping of the current frame1 and frame2. This shows that frame2 has 4 objects. The objects [1, 2, 3, 4] in current frame2 are mapped with objects [0, 1, 2, 3] in current frame1. Thus, object 1 in frame2 is new born. Others can be traced back to frame1.

pairs»

0, 1, 2, 3

Now check old_objects and remember that at this instant, id2 (in the old_objects) correspond to the objects in current frame1 which was frame2 in the earlier time-step, and that they are the same frame.

old_objects»

id1, uid, id2, obs_num, xhead, yhead

1, 1, 1, 2, 1, 0

2, 12, 3, 2, 0, -1

So the object 1 and 3 in current frame1 (earlier it was frame2 with id2) existed before and has "uid" (11 and 12). We will copy their "uid" to our object_matrix and increament the observation number (obs_num). For object 2 and 4 in current frame2 which do not exist in frame1, we will ask for new uids. This information will be written in current_objects and return for writting in to the output file.

`write_duration` *Write duration of dead objects in output NC file.*

Description

Writes number of observations for dead objects. Duration is in time-steps.

Usage

```
write_duration(outNC, current_objects, frame1, frame2)
```

write_settingParms_toNC

Writes all the setting parameters (as attributes) for the tracking.

Description

These parameters affect the sensitivity of the tracks, mergers and split definitions etc.

Usage

```
write_settingParms_toNC(outNC)
```

write_survival

Write survival stats

Description

write number of lived, dead and born objects to the file for each scan.

Usage

```
write_survival(outNC, survival_stat, time, scan)
```

write_update

Writes properties and uids for all objects into output netcdf file.

Description

Writes properties and uids for all objects into output netcdf file.

Usage

```
write_update(outNC, current_objects, obj_props, obs_time, frame1, frame2)
```

Arguments

outNC	output netcdf file object from function create_outNC_track
current_objects	output of update_current_objects
obj_props	output of get_object_prop()
obs_time	time of first scan in POSIX format. units="seconds since 1970-01-01".

Index