Supplementary materials of this thesis

Ting He

January 8, 2015

1 Introduction

Matlab functions for the proposed rainfall model were presented in this supplementary materials document. These functions were orgnized into two parts: 'Rain cluster identification module' and 'Rain cluster tracking module'. All the functions were developed under the matlab enviroment (version 2013a) with the '***Image Processing Toolbox***' was implemented. Before the main contents of this document, Python codes for pre-processing raw radar images by **Wradlib** package was introduced firstly.

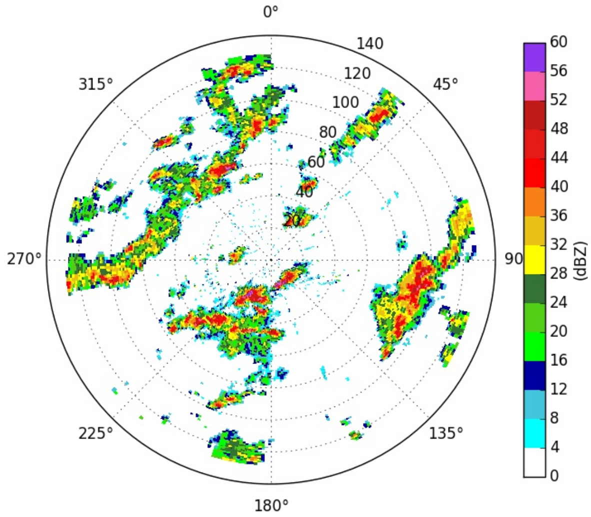
1.1 Python code for pre-process raw radar image

The python code was development under the Python enviorment (version 2.7.6). The following contents presented the developed python codes.

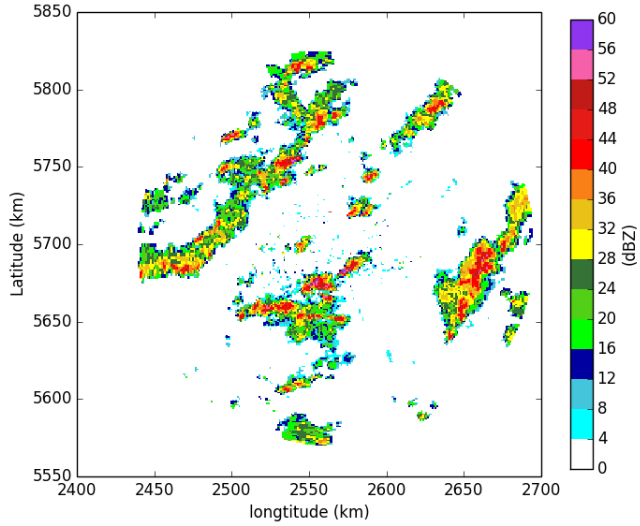
```
# importing needed python packages
import wradlib.vis
import wradlib.georef
import wradlib.comp
import wradlib.io
import wradlib.zr
import wradlib.trafo
import numpy as np
import matplotlib as mpl

# defining reflectivity color for the radar reflectivity threshold
def radar_colormap():
reflectivity_colors = [
"#FFFFFF", # 0 dBZ
"#00FFFF", # 4 dBZ
"#43C6DB", # 8 dBZ
"#0000A0", # 12 dBZ
"#00FF00", # 16 dBZ
"#52D017", # 20 dBZ
"#347235", # 24 dBZ
"#FFFF00", # 28 dBZ
"#EAC117", # 32 dBZ
"#F88017", # 36 dBZ
"#FF0000", # 40 dBZ
"#E41B17", # 44 dBZ
"#C11B17", # 48 dBZ
"#F660AB", # 52 dBZ
"#8E35EF", # 56 dBZ
"#000000", #60 dBZ
]
cmap = mpl.colors.ListedColormap(reflectivity_colors)
return cmap
```

```
radarclasses = [0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60]
colormaps = radar_colormap()
# opening the raw radar data which is in the ess format
f = open('G:/070526/ein_bild.dat')
lines = f.readlines()
files = []
for lines2 in lines:
line = lines2.split('\n')
line1 = lines2.split('.ess\n')
#read the meta data of DWD DX format data
data_dBZ, metadata = wradlib.io.readDX("G:/080719/" + line[0])
#transfer radar reflectivity to rainfall intensity
data_Z = wradlib.trafo.idecibel(data_dBZ)
# using Z-R relationship defined by DWD
Rainfallintensity = wradlib.zr.z2r(data_Z, a = 256., b=1.42)
files = line[0]
print files
# present raw radar data in polar coordinate map
wradlib.vis.polar_plot(data_dBZ, R=128, theta0=0, colormap = colormaps,
classes = radarclasses, unit = 'dBZ', saveto = "D:/080719/" + line1[0]+".jpg")
#reprojection data in cartesian cooedinates system
radar_location = (51.40643, 6.96454, 152) # latitude, longtitude, elevation
elevation = 0.8 # radar scanning elevation
# radar scanning angle range in polar coordinates
azimuths = np.arange(0,360)
#radar scanning distance range in polar coordinates
ranges = np.arange(0, 128000., 1000.)
polargrid = np.meshgrid(ranges, azimuths)
lat, lon, alt = wradlib.georef.polar2latlonalt(polargrid[0], polargrid[1],
 elevation, radar_location)#cartesian coordinates of radar pixel
# projection the cartesian coordinates intoto Gauss Krueger zone 2
gk3 = wradlib.georef.create_projstr("gk", zone=2)
x, y = wradlib.georef.project(lat, lon, gk3)
xy = np.vstack((x.ravel(), y.ravel())).transpose()
#transfer the north-east sector to a 1km x 1km grid
xgrid = np.linspace(x.min(), x.max(), 256)
ygrid = np.linspace(y.min(), y.max(), 256)
grid_xy = np.meshgrid(xgrid, ygrid)
grid_xy = np.vstack((grid_xy[0].ravel(), grid_xy[1].ravel())).transpose()
np.savetxt("G:/outputbywradlib0807191/coords.txt", grid_xy)
gridded = wradlib.comp.togrid(xy, grid_xy, 128000., np.array([x.mean(), ...
    y.mean()]),
data_dBZ.ravel(), wradlib.ipol.Nearest)
gridded = np.ma.masked_invalid(gridded).reshape((len(xgrid), len(ygrid)))
#generating radar image in projected coordinate system
wradlib.vis.cartesian_plot(gridded, x=xgrid, y=ygrid,
colormap = colormaps, classes = radarclasses, unit= 'dBZ',
saveto = "D:/080719/cartesian_"+line1[0]+".png")
#output projected radar images as inputs into matlab functions
np.savetxt("D:/080719/" + line[0], gridded)
f.close()
```

Figure 1 presented an example of raw radar image pre-process by **Wradlib** package.



(a) Raw radar image with polar coordinates



(b) Porjected raw radar image

Figure 1: An example for raw radar image pre-process by Wradlib package

2 Functions for Rain Cluster Identification Module

Figure 2 presented the flow chart of matlab functions for rain cluster identification module.

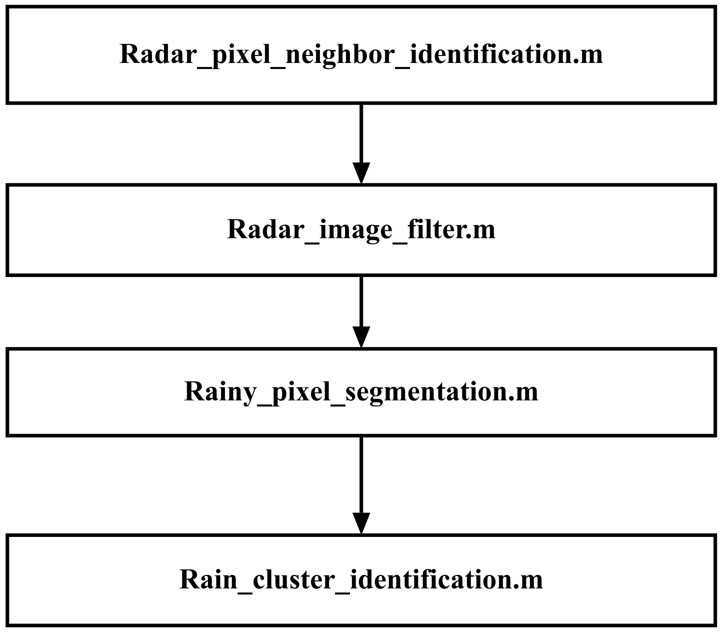


Figure 2: Flow chart of matlab functions for rain cluster identification procedure

(1) ***Radar pixel neighbor identification.m*** This function is used to searching the 8-neighbor pixels of radar pixel in the cartesian coordinate system.

(2) ***Radar image filter.m*** This function is used to eliminating noise pixels in radar image by the Median Filter algorithm.

(3) ***Rainy pixel segmentation.m*** A function which is used to identify rain clusters and derive characters of them from single radar image.

(4) ***Rain cluster identification.m*** The main function for rain cluster identification module.

3 Functions for Rain cluster Tracking Module

Functions for rain cluster tracking module were orgnized in two parts: functions for global motion vectors generation by PIV method and functions for most matched child rain cluster identification. Figure 3 presented the flow chart of functions for rain clutser tracking moudle.

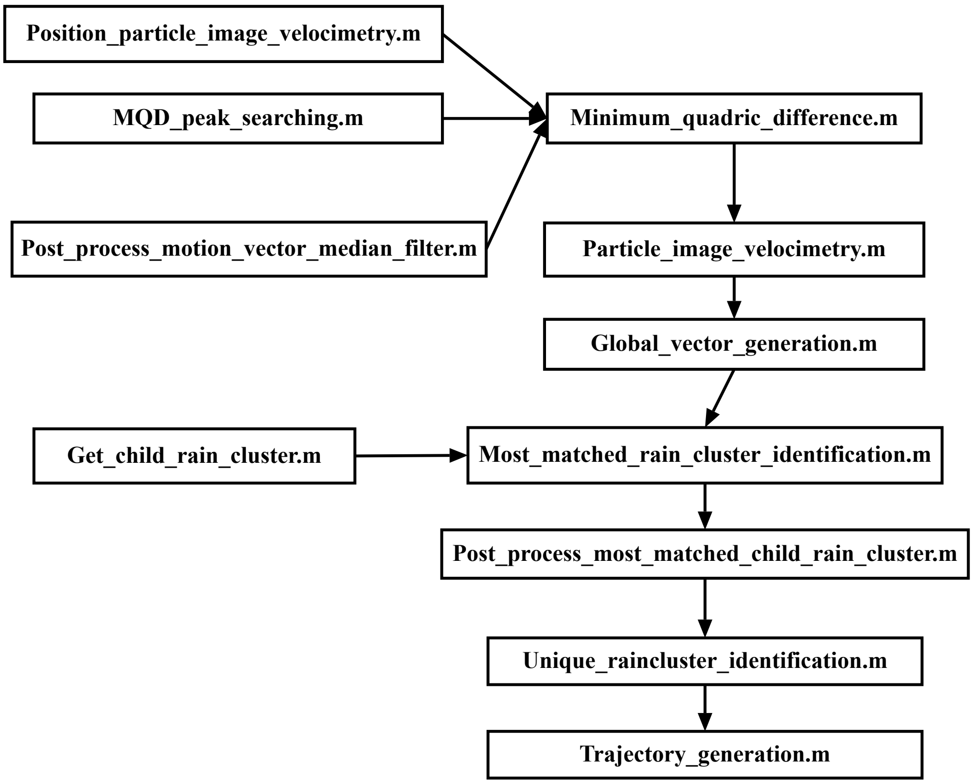


Figure 3: Figure caption.

Functions for mean global motion vector identification were listed in the following:

(1) ***Position particle image velocimetry.m*** A function which is used to search the position of peak value from MQD(Minimum Quadric Difference) result for each sub-window in the radar image.

(2) ***MQD peak searching.m*** A function which is used to calculate the MQD peak value by Gaussian sub-pixel fitting algorithm.

(3) ***Post process motion vector median filter.m*** a function for post-process for the generated motion vectors at each sub-window in radar image by median filter way. The object this function is to fill the empty position or eliminate the unusual motion vectors.

(4) ***Minimum quadric difference.m*** It is the main function of MQD method.

(5) ***Particle image velocimetry.m*** Main function of Particle Image Velocimetry method.

(6) ***Global vector generation.m*** Main function for generating mean global motion vector at each moment.

Functions for most matched child rain cluster identification were listed in the following:

(1) ***Get child rain cluster.m*** A function for searching all the child rain clutsers within the bounding box of parent rain cluster.

(2) ***Most matched rain cluster identification.m*** Main function for searching the most matched child rain cluster.

(3) ***Post process most matched child rain cluster.m*** A function for post-processing the identified most matched child rain cluster. By appyling this function, child rain clusters with too large distance of weighted center to the parent rain cluster are eliminated.

(4) ***Unique raincluster identification.m*** A function for getting id of rain clusters and their parent/child rain clusters.

(5) ***Trajectory generation.m*** function for generating trajectories of rain clutsers, output of this function is the trajectories which are linked by the ids of rain cluster at successive moments. For the rainfall statistical analysis (e.g. temporal development of rain clutser) should combing this output with the characters which stored in the relational database.

4 Demo code

The matlab demo code for excuating proposed rainfall mode was presented in the following:

```
%% load workspace
load('Inputs.mat');

%% Light Rain cluster identification (threshold > 19dBZ)
[raincells_19dBZ, raincellsfeatures_19dBZ] = ...
Rain_cluster_identification(288, files, 7, 19, 37, 5, 9, x_coordinates, ...
    y_coordinates);

%% Storing raincellsfratures_19dBZ into a relational database(PostgreSQL)

%% Tracking procedure for light rain cluster (threshold > 19dBZ)
[global_vector, mean_x_vector, mean_y_vector] = ...
Global_vector_generation(288, originalreflectivities);

[clusterboundingboxwithsearchbox_19dbz, clusterboundingboxes_19dbz, ...
clustercentroids_19dbz, clustercoords_19dbz, ...
clusterareas_19dbz, clustercenterofmass_19dbz] = ...
Pre_process_tracking_procedure(288, files, 5, 9, 7, 19, ...
x_coordinates, y_coordinates);

[all_potential_centroids_19dbz, all_potential_coords_19dbz, ...
all_potential_areas_19dbz] = Get_child_rain_cluster(288, ...
clusterboundingboxes_19dbz, clusterboundingboxwithsearchbox_19dbz, ...
clustercenterofmass_19dbz, clustercoords_19dbz, clusterareas_19dbz);

[overlaps_19dbz, themostlikelysuccessors_19dbz, ...
distances_19dbz, directions_19dbz, clusterdatabase_19dbz] = ...
Most_matched_rain_cluster_identification(288, clustercenterofmass_19dbz, ...
clustercoords_19dbz, clusterareas_19dbz, all_potential_centroids_19dbz, ...
all_potential_coords_19dbz, all_potential_areas_19dbz, global_vector, 3, 40, ...
    20);

resultss_19dbz = ...
Post_process_most_matched_child_rain_cluster(clusterdatabase_19dbz, cellid, ...
    30, 0.16);

%% storing resultss_19dbz into relational database and retriving the
%% id of rain cluster including parent/child rain cluster.
%% in this demo, the ids of parent/child rain clusters have been
%% presented in the worksapce with name of precursor/successor
%% cellid is the first column of matrix raincellsfeatures_19dBZ
precursors_19dbz = Unique_raincluster_identification(precursor, cellid);
successors_19dbz = Unique_raincluster_identification(successor, cellid);

```

```
tracks_19dbz = Trajectory_generation(precursors_19dbz , cellid , ...
    successors_19dbz ,288);
```