Documentation for LAAFU System

Author: Robert ZHAO Ziqi, Version: 2019.1.10

This is the documentation of the LAAFU system, which contains two parts: **fingerprint database construction** and **altered AP detection**. Localization part and fast detection part is not included, since they are not the most difficult part in LAAFU.

Most of the codes are converted from **LIN Wenbin's C++ version** with some improvement. One of the amazing features is that this program provides **UI** and it enables user to **visualize** data through lots of methods. Besides, it provides a separate **setting file** for easy debugging.

This system is built under **Python 3.6.6**, and it only needs some basic libraries to work: **PyQt5**, **NumPy**, **SciPy**, **MatPlotLib**. Please set up all environments before using it.

This documentation will be divided into **3** parts. The first part is the instruction of demonstration, and the second part is the detailed explanation of every component of the code. For the third part, some known issues will be listed here, and it needs a little work to fix them.

Contents

Part I Demonstration Instruction

I.1 Some Terminologies

To avoid being messed up, we will define some of the words with specific meanings. When mentioned below, their meanings should be the same as follows:

- Paths File: the file that will be used for fingerprint database construction. Data in it may not be the latest one.
- Target File: the file that will be used for detecting altered AP. Data in it should be newer than that in paths file.
- RP: Reference point
- Root folder (or Root path): the folder containing data that will be processed by the program. Refer to data_folder1, data_folder2 below.

Back to contents

I.2 Project Structure

Please follow the project structure below for better using experience.

I.2.1 Initial Environment

> LAAFU
>
> |-> Documentation for LAAFU System.md (with latex formula)
>
> |-> Documentation for LAAFU System.pdf
>
> |-> [code]laafu_cpp (LIN Wenbin's C++ code with documentations)
>
> |-> [code]laafu_python (Python Code)
>
> |-> construct_fp_db.py (Used for constructing fingerprint databae)
>
> |-> find_altered_ap.py (Used for detecting altered APs)
>
> |-> gaussian_process.py (Provide GP as a class with training and estimating abilities)
>
> |-> heatmap_plotter.py (Provide multiple options of visualizing data)
>
> |-> icon.png (Used for application icon)
>
> |-> main.py (Functions invoked by main program)
>
> |-> settings.py (All parameters that can be adjusted)
>
> |-> test_gp.py (Provide functions that generate RSSI for all RP)
>
> **|-> ui.py (Main program with UI. Provide full functionalities of the system)**
>
> |-> utils.py (Some frequently used functions)
>
> |-> [Top]original_data (Source Data Backup)
>
> |-> original_data_folder1
>
> |-> p_all.txt (Paths File)
>
> |-> target_xxx.txt (Target File)
>
> |-> original_data_folder2
>
> |-> ...

> |-> ...
>
> **|-> data_folder1 (Used for testing)**
>
> **|-> p_all.txt (Paths File)**
>
> **|-> target_xxx.txt (Target File)**
>
> |-> data_folder2 (Used for testing)
>
> |-> ...
>
> |-> ...

Back to contents

I.2.2 Generated folders and files

Please note that not all generated folders and files are listed. Only important files are introduced here.

- db.txt: fingerprint database file
- loc.txt: location of all RPs
- resX: folder containing all APs' map from points to RSSIs. They will be used in visualizing
- paraX: folder containing all APs' GP parameters. They are currently used for restoring APs' location
- output_X\all_xxxxx.txt: file of one AP containing AP location, RSSI error among all RPs and corresponding std
- output_X\xxxxx.txt: file containing all RP points that deviate a lot from estimation.

Back to contents

I.3 Basic Operations

I.3.1 Main window

- Data folder: choose root folder, otherwise all functions below won't work.
- Paths file: choose paths file. It will be needed for most of the functions below.
- Target file: choose target file.
- Construct Fingerprint Database: use paths file and root folder to construct fingerprint database
- **View APs' Information**: will lead to a new window visualizing APs in database
- Detect target file: detect altered APs using target file
- **Show Alternation Level**: will lead to a new window visualizing altered APs.
- Quit: exit the whole program.

Note: Please first **choose Data Folder, Paths file and target file** before any operations!

Back to contents

I.3.2 View APs' Information

- plot 2D heatmap: plot 2D heatmap based on one AP. Should **copy one AP's path from the left to the input box on the right**, and then click "Plot 2D Heatmap"
- plot 3D heatmap: plot 3D heatmap based on one AP. Should **copy one AP's path from the left to the input box on the right**, and then click "Plot 3D Heatmap"
- Plot AP location: Plot all APs' location. For easy visualizing, mac address has been removed from the plot.

- Back: return to main window

Back to contents

I.3.3 Show Alternation Level

Copy one AP's path from the left to the input box on the right, and then click "select". Altered points are labeled red.

Back to contents

I.4 Brief Description of parameters in settings.py

- **min_size_to_process (5)**: This controls when to discard AP with small number of corresponding positions
- **separate_parts (4)**: This controls how many parts will the data be divided into
- **with_z (1)**: This determines whether to consider z value of AP's location or not
- **ptm_ratio (13.3)**: This is the conversion ratio from pixel to meter
- **n_sigma (7)**: This is used for tolerance of alternation
- **interval (65)**: This is used for half heighbour filtering interval
- **min_length (15)**: minimum length of series to be processed
- **neighbour_bound (3)**: This is used for half neighbour filtering minimum count of neighbour to be counted as altered
- **min_rssi (-100)**: Minimum value of RSSI

Back to contents

Part II Detailed Documentation of code

In this part, detailed documentation will be given to all code except those in ui.py and settings.py, since they don't need to document or they have been described above.

II.1 main.py

- **construct_fp_database(root_path,pre_paths)**: Construct fingerprint database using *pre_paths* file in *root_path*.
- **detect(target,path)**: (**Abandoned**) use database in *path* and merged target file generated by *target* to detect altered APs
- **detect_original(target,path)**: use database in *path* and target file *target* to detect altered APs

Back to contents

II.2 heatmap_plotter.py

Back to contents

II.3 gaussian_process.py

constructor arguments:

- position: list of position points (gN*2 list)
- rssi: list of rssi (gN*1 list)

- with_z: set to 1 if considering location of AP as 3D point (x_{ap},y_{ap},z_{ap})

data members ("g" for the meaning of global):

- gN(long): size of training data
- gKy(gN*gN Matrix): inverse of K_y, or temp $(x_p - x_q)^T \cdot (x_p - x_q)$
- gT(gN*gN Matrix): temp matrix
- gX(gN*2 Matrix): training input data X
- gY(gN*1 Matrix): training input data Y, or temp$(K_y - 1) \cdot (Y - m(X))$
- g_pGP(3*1 Col. Vector): GP parameters: σ_n, σ_f, l
- g_pLDPL(4(5)*1 Col. Vector): LDPL (log-distance path loss) parameters: A, B, x_{ap},y_{ap},(z_{ap})

public:

- train(): This function will train two groups of hyper-parameters.
- estimate_gp(x, y, sd_mode=False): estimate RSSI value at point (x,y), return estimated **mean** and **std**(when sd_mode=True)/**0.0**(when sd_mode=False)
- paramters(): return all parameters [A, B, x_{ap},y_{ap},z_{ap}(if with_z=1), σ_n, σ_f, l]

private:

- _obj_gp(m): Objective function of gp, with input as [σ_n, σ_f, l]
- _deriv_gp(m): Derivative function of gp, with input as [σ_n, σ_f, l]
- _obj_ldpl(m): Objective function of ldpl, with input as [A, B, x_{ap},y_{ap},z_{ap}(if with_z=1)]
- _deriv_ldpl(m): Derivative function of ldpl, with input as [A, B, x_{ap},y_{ap},z_{ap}(if with_z=1)]
- _estimate_ldpl(x,y,m=None): Estimate mean value at point (x,y) based on m [A, B, x_{ap},y_{ap},z_{ap}(if with_z=1)]
- _train_ldpl(): Train ldpl model. Use L-BFGS-B algorithm to find minimum of objective function
- _train_gp(): Train gp model. Use L-BFGS-B algorithm to find minimum of objective function
- _estimate_ky(m, Ky): Estimate K_y

Back to contents

II.4 utils.py

- **mkdir(path)**: create new folder if it doesn't exist
- **sqd_exp(sigma_f, l, sqd_sum_v)**: return $\sigma_f^2 \cdot e^{-\frac{sqd\_sum\_v}{2l^2}}$
- **sqd_sum(x1,y1,x2,y2)**: return $(x_1 - x_2)^2 + (y_1 - y_2)^2$

Back to contents

II.5 test_gp.py

- **ap_map_rp(file_path)**: run through all data in *file_path* and generate map between location and RSSI for all APs (no std option)
- **ap_map_rp_std(file_path)**: run through all data in *file_path* and generate map between location and (RSSI, std) for all APs
- **test_gp_all_ap(in_folder, loc_file_path, out_folder, para_folder)**: for every AP file in *in_folder*, train GP model and estimate RSSI for all points in *loc_file_path*, keep the ones with reasonable RSSI values and output them to files in *out_folder*. GP parameters will be output to corresponding files in *para_folder*.
- **get_loc(file_path, loc_name="loc")**: run through all data in *file_path* and extract all location points to the file *loc_name*.

Back to contents

II.6 construct_fp_db.py

- **separate(file_path, n)**: Split the data in f*ile_path* into *n* parts
- **construct_fing(root_folder,loc_file_path, out_file_path, n, pre_dir, pre_res="res", pre_para="para")**: Construct fingerprint database to *out_file_path*, with RPs' location in *loc_file_path* and root folder in *root_folder*. Paths file has been split into *n* parts, and name of paths file is *pre_dir*. It will group all RSSI of each AP at all points in paths file in *pre_res*X folders and generate GP parameters in *pre_para*X folder.
- **combine_fingerprint(in_folder_path, out_folder_path, pre_res)**: Generate database file to *out_folder_path* using folders *pre_res*X in *in_folder_path*. Structure in this file should be the same as that of paths file and target file.

Back to contents

II.7 find_altered_ap.py

- **merge_targets(in_file_path, out_file_path)**: (**Abandoned**) merge every 5 data points
- **detect_alt_at_rp(target_file, rp_file, out_folder)**: run through all APs in *target_file* using database file *rp_file* and check whether it is an altered AP. Output will be in *out_folder*.

Back to contents

Part III Known Issues

1. Sometimes the plot will cause an error *Tcl_AsyncDelete: async handler deleted by the wrong thread*
2. When using Scipy.optimize.minimize() function, 'Powell' and 'CG' options will lead to unknown error but 'l-bfgs-b' won't

Back to contents