Course Project 1 - Smartphone-based Indoor Localization

AI6128 Urban Computing

Kishore Rajasekar
Nanyang Technological University
G2101949G
Singapore
kishore004@e.ntu.edu.sg

Seah Xiu Xuan Sherbelle
Nanyang Technological University
G2102418J
Singapore
sseah013@e.ntu.edu.sg

Sean Goh Ann Ray
Nanyang Technological University
G2212190G
Singapore
sean0057@e.ntu.edu.sg

Sita Rajagopal
Nanyang Technological University
G2202266H
Singapore
sita001@e.ntu.edu.sg

***Abstract*— The smartphone-based indoor localization technique uses sensor-rich smartphones to collect readings that aid in device-based localization. In this report, the team analyses data collected by Microsoft as part of their ongoing efforts in conducting the Indoor Localization 2.0 Competition [1]. The dataset released comprises of dense indoor signatures of Wi-Fi, geomagnetic field, Bluetooth iBeacon and ground truth data collected by surveyors. The team visualized the dataset to plot a heatmap. Subsequently, the team proposed a few deep-learning models to predict the indoor location of a device given its indoor signatures.**

I. INTRODUCTION

Indoor localization enables the locating of objects or individuals in indoor spaces. The Global Positioning System (GPS) in interior spaces is rendered ineffective in locating persons or objects of interest due to its signal's impermeability into interior spaces. As such, it is crucial that an implement for indoor localization is in place to track targets in indoor spaces. Such an implement can be broadly classified into two categories: device-free localization or device-based localization. Device-free localization refers to tracking a target without the use of a device that is tagged to it. Device-based localization, on the other hand, involves a wireless device that is tagged to the target and computes its location through cooperation with other installed devices. In this report, the team will examine the Indoor Location Competition 2.0, a device-based localization competition's dataset.

The objective of this report is twofold: to visualize the dataset appropriately and to introduce a few approaches to the deep learning-based location fingerprint model. The first entails parsing the smartphone-generated signatures and waypoints to visualize heatmaps and trajectories respectively. As the team notes, when associating smartphone signatures with the waypoints collected, the signatures collected are far denser than the waypoints that had been surveyed. As such, data augmentation was performed during the preprocessing phase to remedy this issue. This provides more information-rich heatmaps for Wi-Fi, geomagnetic field and iBeacon signatures. The second contains proposals for some deep-learning fingerprint models to predict the location of a target given its respective signatures as inputs to the model.

II. DATASET

A. Background

The Indoor Localization 2.0 Competition dataset used in this project consists of indoor signatures across Xixi Yintai (site1) and Joy City (site2). The first is a building with 5 floors, from Basement 1 to the 4th floor. The second building consists of 9 floors, from Basement 1 to the 8th floor. The dataset contains raw trace files. Each trace consists of an indoor path walked by a surveyor. While on this path, the surveyor is tagged with an Android-phone at the front of his body which collects 10 different types of data, with readings that consists of, but not limited to, Inertial Measurement Unit (IMU), geomagnetic field, Wi-Fi, and Bluetooth iBeacon scanning results. TABLE I provides a brief description. Each data entry also features a Unix timestamp which specifies the time at which the reading was recorded.

Floor plan metadata for each floor was also provided. It includes a raster image, size of the image in height and width, as well as a GeoJSON map file.

TABLE I. DATA TYPE VALUE DESCRIPTIONS

Data Type	Description
TYPE_WAYPOINT	Values of points in meters collected through (x,y) coordinates.
TYPE_ACCELEROMETER	Provides readings along the X, Y and Z axes, along with the accuracy measurement.
TYPE_GYROSCOPE	
TYPE_MAGNETIC_FIELD	
TYPE_ROTATION VECTOR	
TYPE_ACCELEROMETER _UNCALIBRATED	An uncalibrated sensor reports the acceleration/gyroscope/magnetic field readings of the device along the three sensor axes (X,Y,Z) without any bias correction. Accuracy measurement is included.
TYPE_GYROSCOPE _UNCALIBRATED	
TYPE_MAGNETIC_FIELD _UNCALIBRATED	
TYPE_WIFI	Provides ssid, bssid, RSSI, frequency and last seen Unix timestamp readings
TYPE_BEACON	Provides UUID, MajorID, MinorID, Tx Power, RSSI, Distance, MAC Address, Unix Timestamp readings

TABLE II. NUMBER OF READINGS FOR EACH DATA TYPE.

Site	Floor	Magnetic Field	Wi-Fi	iBeacon	Waypoints
1	B1	265122	333724	31814	1034
	F1	290966	868647	31477	975
	F2	382553	778674	52499	1049
	F3	475461	702449	47882	1012
	F4	356808	703488	20619	1042
2	B1	132112	201667	2049	534
	F1	272351	795937	3900	1006
	F2	85189	228229	311	362
	F3	69756	177006	254	278
	F4	54967	122219	231	215
	F5	67935	163887	868	298
	F6	130772	272565	2164	565
	F7	67491	178423	1878	273
	F8	66601	147718	697	265

B. Preprocessing

The data types that were of interest for the essential tasks were TYPE_WAYPOINT, TYPE_MAGNETIC_FIELD, TYPE_WIFI and TYPE_BEACON. The number of readings for each data type recorded across the floors can be found in TABLE II.

It is evident that the number of readings collected for the smartphone signatures of interest were significantly more than the ground truth data, or waypoints, collected. As such, when associating the signatures with the waypoints based on similar Unix timestamps, the waypoints serve as a limiting factor. The heatmaps plotted as a result will contain sparse points in the coordinate space, despite collecting rich signatures from the smartphone. Data augmentation is needed to generate more waypoint estimates. This can be achieved through the compute_step_position function provided by Microsoft.

The function takes in inputs of the following data types: TYPE_ACCELEROMETER, TYPE_WAYPOINTS and TYPE_ROTATION_VECTOR. The function makes use of each input as such:

- TYPE_ROTATION_VECTOR – The rotation vector readings collected will be used to calculate the orientation of the surveyor. The orientation was calculated using the compute_headings function.
- TYPE_ACCELEROMETER – The steps are detected according to acceleration magnitudes. A valid peak or valley of acceleration magnitudes are identified. A peak in magnitude is identified as a step and the length of the stride is calculated by taking the difference between two valley points. The estimated timestamps of these steps were also calculated. Step indices were also generated to identify steps with their magnitudes as well as its estimated orientation. These were achieved through the compute_steps, compute_stride_length functions.
- TYPE_WAYPOINTS – The relative positions of the surveyor's movement was calculated through the compute_rel_positions function using the length of the strides and the orientation of the surveyor. Thereafter, the waypoints were used as reference positions to calculate the final step positions, using the correct_positions function.

TABLE III. TYPE_ WAYPOINT DATA DESCRIPTION

Field	Description
Coordinate X(meter)	X coordinate of the location surveyor labelled on the map
Coordinate Y(meter)	Y coordinate of the location surveyor labelled on the map

III. ESSENTIAL TASKS

In this section, the tasks of visualizing the way points, geomagnetic heat map, Wi-Fi RSS heat map, and iBeacon RSS heat map are described.

A. Visualisation of Waypoints

1) Identifying the Location of Each Sensor Reading

The way points are the ground truth locations of the sensors labelled by the surveyor.

2) Results

The way points represent ground truth locations. The blue marker represents the start point, the dotted line represents the trajectory and the red marker represents the end point. This represents the visualization of the waypoints in level F1. visualize_trajectory function in the visualize.py file was used to plot the visualization of the ground truth waypoints. Fig. 1 represents the visualization of way points in level F1 of site1.

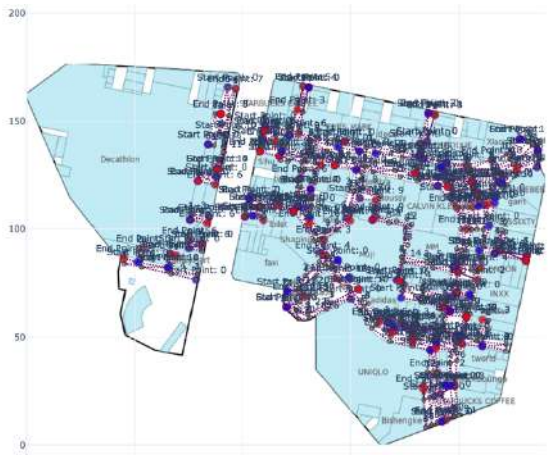


Fig. 1. Visualization of trajectory of ground truth waypoints

B. Visualise Geomagnetic Heat Map

The files provided by Microsoft's Indoor Location competition includes the readings from the magnetic field sensors (of TYPE_MAGNETIC_FIELD), which are sampled every 20ms as the site surveyor moves around the building.

1) Identifying the Location of Each Sensor Reading

As each sensor reading only contains the ambient magnetic field reading in the X, Y and Z axis, without the corresponding location value, there is a need to map the location of each magnetic field reading so that the heat map can be plotted out.

The most straightforward way is to map the magnetic field reading to the nearest waypoint data that is provided together with the dataset. However, as can be seen from Fig. 2, this provides a very sparse heat map of the geomagnetic field strength.

Fig. 2. Geomagnetic Heat Map for Site 1, First Floor, with provided waypoints only.

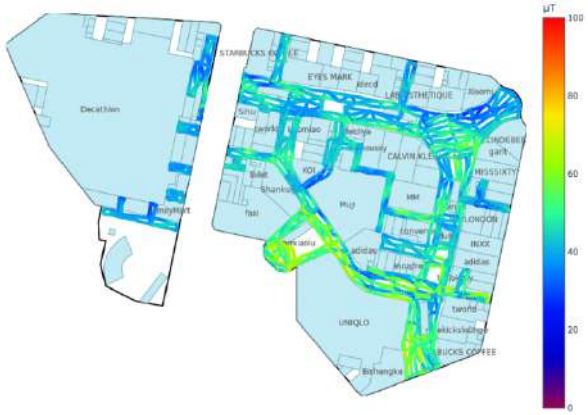


Fig. 3. Geomagnetic Heat Map for Site 1, First Floor, with added augmented waypoints.

To add in more waypoints, the compute_step_position function provided by Microsoft interpolates the location of the surveyor (in x and y coordinate values) at each timestamp, based on the accelerometer and rotation vector readings. This is then mapped to the readings from the magnetic field sensor.

Instead of associating with the exact timestamp values, the timestamp from each magnetic field sensor reading is mapped to the location of the surveyor at the nearest timestamp value. This can be done, as each reading is sampled every 20ms, and the surveyor is unlikely to have travelled over large distances in that period. This results in a mapping of magnetic field strength readings to the nearest x and y coordinate positions.

After obtaining the mappings of all surveyed readings for the same floor, the mappings are then concatenated together, giving multiple magnetic field strength readings across all possible positions taken by the surveyors. The average magnetic field strength at each position is taken by:

- First, summing the squared values of the reading in each axis and taking the squared root. This gives the magnetic field strength of each reading.
- Then, taking the average of all readings at each location to give a single magnetic field strength value for each x and y coordinate position.

From here, the average magnetic field strength for each x and y coordinate position are then plotted, with the magnitude of the magnetic field strength being represented by the color of the heat map. To ensure consistency in comparison across different floors, the range of the heatmap values is fixed to 0 to 100, with smaller values being represented by blues and purples, and larger values being represented by oranges and reds.

2) Results

Due to the number of heatmaps generated, only the maps with augmented waypoints for the first floor for Site 1 (Fig. 3) and Site 2 (Fig. 4) will be shown.

Fig. 3 shows the final geomagnetic heat map for the first floor of site one, with the augmented waypoints. Each dot represents the average magnetic strength in μT for that position in time. The closer the color of the dot to purple, the lower the magnitude, hence indicating a weaker magnetic field strength. On the other hand, if the color of the heat map tends to red, it indicates a higher magnitude and hence a stronger magnetic field strength.

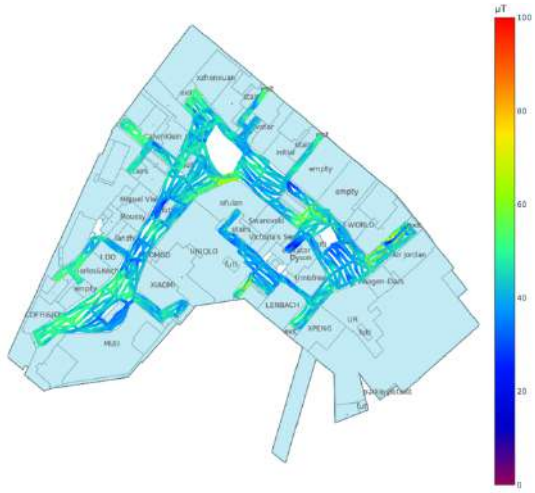


Fig. 4. Geomagnetic Heat Map for Site 2, First Floor, with added augmented waypoints.

With the added interpolated waypoints, the heat map that is generated (Fig. 3) is much richer in information than the one without (Fig. 2).

The rest of the heatmaps can be found in Appendix B – Visualization of Geomagnetic Field Strengths.

C. Visualise Wi-Fi RSS Heat Map

1) Identifying the Location of Each Sensor Reading

The files provided by Microsoft's Indoor Location competition includes Wi-Fi signals captured by the smartphones from the available Wi-Fi Access Points (APs) (of TYPE_WIFI) and their respective data, as shown in TABLE IV.

Each instance of TYPE_WIFI data comes with a timestamp, which would be used to compare with that of the step positions produced from compute_step_positions, and the x and y coordinate position of the nearest timestamp produced would be the x and y coordinate for that instance of TYPE_WIFI data.

TABLE IV. TYPE_WIFI DATA DESCRIPTION

Field	Description
SSID	Service Set Identifier, it is the name of the specific Wi-Fi AP
BSSID	Basic Service Set Identifier, it is the unique identifier of the specific Wi-Fi AP
RSSI	Received Signal Strength Indicator, it is a measurement of how strong a signal, from a specific Wi-Fi AP, is seen from the device, in this case a smartphone
Frequency	Frequency of the Wi-Fi signal, usually 2.4GHz or 5GHz

A nested dictionary is then made such that the keys of the external dictionary are the Wi-Fi BSSID, where their respective values internal dictionaries where the keys are the x and y coordinates of the step positions that sees that Wi-Fi AP, and the corresponding value is the RSSI value, averaged over repeated step positions, if any.

With the augmented step positions and their respective Wi-Fi data, a heatmap is produced, based on the site, floor, and Wi-Fi AP chosen. The code loops through each floor of both sites and randomly chooses 3 Wi-Fi APs for visualization.

2) Results

Due to the large number of heatmaps produced, only 3 selected heatmaps shall be described here. Each individual coloured dot represents the step position while the colour of the dot represents the RSSI value. Larger values (e.g. -65dBm) means that the step position sees a stronger signal strength from that Wi-Fi AP, while smaller values (e.g. -90dBm) means that the step position sees a weaker signal strength from that Wi-Fi AP. Larger values thus indicate that the step is closer to the Wi-Fi AP.

Fig. 5 and Fig. 6 shows the heatmaps of 2 different APs in site 1, 1st floor. As observed, some step positions in Fig. 5 are not shown in Fig. 6, and vice-versa. This simply means that at a specific step position, one Wi-Fi AP can be seen (and therefore have a RSSI value) while the other cannot.

Fig. 6 and Fig. 7 shows the heatmaps of the same Wi-Fi AP, but on 1st and 2nd floor respectively. This suggests that the same Wi-Fi AP can be seen from multiple floors if the transmission signal is strong enough. It can also be seen in Fig. 5 that the AP emits a strong enough signal such that even step positions in another building can detect the AP.

Fig. 5. Wi-Fi Heatmap of Site 1, 1st Floor, Wi-Fi AP: ec:56:23:f8:dc:74

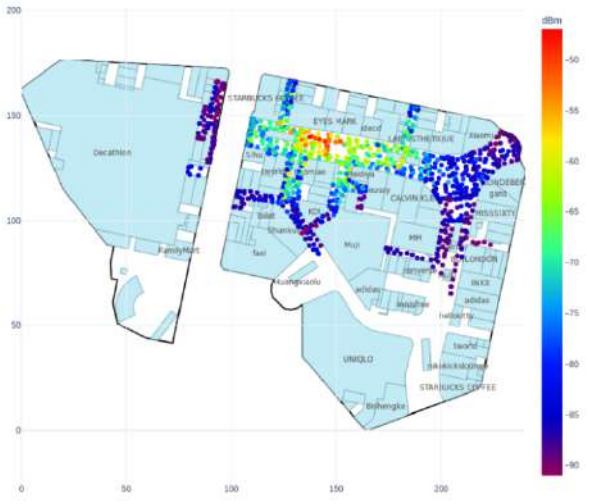


Fig. 6. Wi-Fi Heatmap of Site 1, 1st Floor, Wi-Fi AP: 1e:74:9c:2b:3a:37

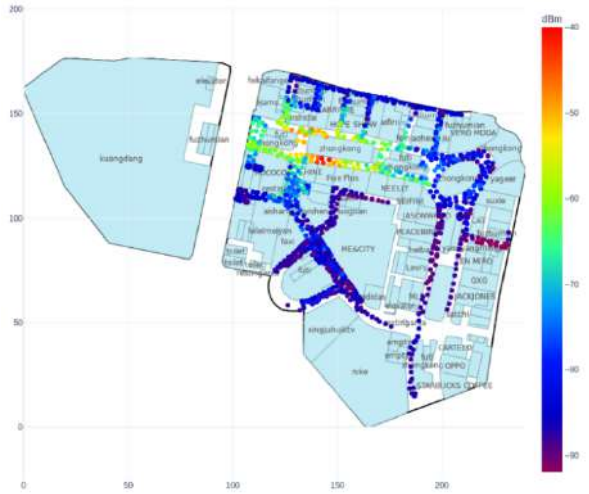


Fig. 7. Wi-Fi Heatmap of Site 1, 2nd Floor, Wi-Fi- AP: 1e:74:9c:2b:3a:37

D. Visualise iBeacon RSS Heat Map

Bluetooth Low Energy (BLE) is a power-efficient transmission wireless protocol that facilitates communication between smart devices that exist within limited range. Beacons are BLE transmitters [2]. When smartphones are within the range of these transmitters, signals can be transmitted to these smartphones, paving the way for location-based services.

The files provided by Microsoft's Indoor Location competition includes the signals captured by the smartphones from beacons (of TYPE_BEACON). The data collected consists of the following:

TABLE V. TYPE_ BEACON DATA DESCRIPTION

Field	Description
UUID	The Universally Unique Identifier. is a standard that generates and assigns unique numbers to a device, such as an iBeacon.
MajorID	Assigned to recognize or separate a group. E.g., Assigned to a room or floor
MinorID	Allows further subdivision of region or use case.
Tx Power	The transmit power of the beacon
RSSI	The BLE signal strength
Distance	Distance between smartphone and beacons, as calculated by surveyors.

1) Identifying the Location of Each Sensor Reading

The beacon data provided does not contain a corresponding location value, only the RSSI and calculated distance from the beacon. Therefore, to localize each iBeacon sensor reading, we compared t_i, the timestamp in which the instance was generated, with timestamps of all the waypoints. The nearest waypoint data based on timestamp is mapped to the iBeacon reading. Such mapping was performed for data collected by each iBeacon and a sample heatmap, Fig. 8, is shown below:

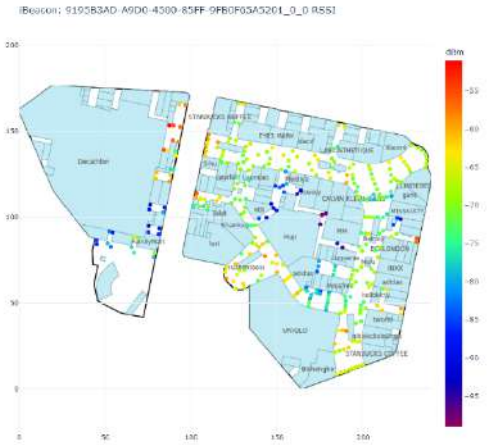


Fig. 8. iBeacon heatmap for Site 1, 1st floor. Readings were collected from iBeacon with UUID 9195B3AD-A9D0-4500-85FF-9FB0F65A5201, MajorID 0 and MinorID 0

As discussed in previous sections, data augmentation using the accelerometer and rotation vector readings was needed to add in more waypoints. With the data augmentation, visualization of the iBeacon data from the same iBeacon shows a denser map, as reflected in Fig. 9 below.

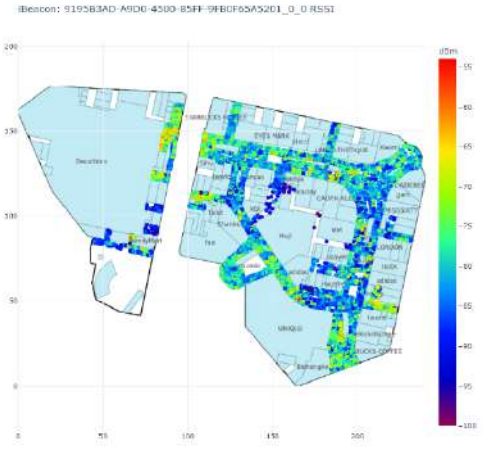


Fig. 9. iBeacon heatmap for Site 1, 1st floor. Readings were collected from iBeacon with UUID 9195B3AD-A9D0-4500-85FF-9FB0F65A5201, MajorID 0 and MinorID 0

After determining the positions of all TYPE_BEACON entries provided, it was evident that there was multiple TYPE_BEACON RSSI values, s_i, for a single position (P_x, P_y). As such, the average RSSI value was calculated for each given point. For each unique iBeacon, where its unique ID can be derived through a concatenation of its UUID, MajorID and MinorID, the following formula can be used to calculate the average RSSI value:

$$s^b_{rss_k} = \frac{1}{N}\sum_{iBeacon^b_k} s_i, \quad (1)$$

Where $s^b_{rss_k}$ is the RSSI of an iBeacon with a unique ID b at a particular waypoint position and N is the total number of instances where a TYPE_BEACON record of timestamp k is linked to a particular waypoint position.

2) Discussion of Results

Due to many heatmaps generated, only Fig. 8 and Fig. 9 will be discussed in this section. The colour of each dot in the heatmap represents the average signal strength received calculated for the waypoint position. Cooler toned dots, closer to purple, indicate a low magnitude of signal strength received in dBM while warmer toned dots, closer to red, indicate higher magnitude of signal strength received in dBM.

With the added interpolated waypoints, the heat map that is generated (Fig. 9) is much richer in information than the one without (Fig. 8).

IV. Bonus Tasks

A. Deep Learning Based Fingerprinting Model based on Linear Regression using Wi-Fi RSSI

For the DL fingerprinting model, the approach was to use the Wi-Fi RSSI readings as the input feature and to output a location within the chosen site and floor. The difficulty here was that compiling the input features for every step position observed would mean that some features had a null value, whereby the Wi-Fi AP was not observed. Since DL models do not work with null values, 2 approaches were used. The 1st approach was to introduce an arbitrary value smaller than any of the observed RSSI readings. The 2nd approach was to impute the null using the mean value of that column.

The first floor of Site 1 was chosen as the dataset, and the model was a simple linear regression model with 1 hidden layer. Additionally, the model was tested using the same input data, where the model's predicted step position of each sample was compared to their labels, and the accuracy was calculated using the Root Mean Square Error (RMSE) of the Euclidean distance.

However, with over 2000 data samples with over 2500 features, the model took about an hour to train for 10000 epochs, where it achieved an accuracy of approximately 50m.

Due to the computational limitations, a 2nd model was built using the same approach. However, it utilized only 3 Wi-Fi APs and had an increased number of hidden layers. Parameters were played around, and it was observed that using 3 hidden layers and 100000 epochs, the training took about 2 minutes while achieving an improved accuracy of approximately 23m. TABLE VI shows the parameters and performance of some of the DL models, where the bolded row shows the best performing model.

While the accuracy of the models is not very good, this can be improved on by implementing a more complicated model which would have more computational requirements. This simple model is limited due to the complicated input data, whereby several different step positions may have the same RSSI readings from the same set of Wi-Fi APs (same input

TABLE VI. PARAMETERS AND PERFORMANCE OF LINEAR REGRESSION MODELS FOR WI-FI RSSI FINGERPRINTING

Replace Null Values with?	No. of Input Features	No. of Hidden Layers	Total No. of Neurons	Epoch	Training Time (s)	RMSE Euclidean Distance Accuracy (m)
-100	2524	1	3788	10000	3782	52.634
-120	2524	1	3788	10000	3905	58.193
Mean	2524	1	3788	10000	3830	65.948
-100	3	1	11	100000	47	34.923
-120	3	1	11	100000	51	37.209
Mean	3	1	11	100000	52	38.522
-100	3	2	23	100000	85	29.346
-100	3	3	47	100000	125	**22.727**
-120	3	3	47	100000	131	24.591
Mean	3	3	47	100000	128	25.092
-100	3	4	59	100000	487	27.837

features but different 'labels'). Accuracy could also be improved on by using more Wi-Fi APs instead of just 3, but that would largely increase the computational requirements and the performance improvement may not be justified by the increased computational cost.

B. *Deep Learning Based Fingerprinting Model based on CNNs using Geomagnetic Field Strength*

The approach taken to build a deep learning fingerprinting model using geomagnetic field strength values is based off the 2018 paper by Al-homayani and Mahoor [3].

In their paper, the authors found that a deep neural network with two convolutional layers followed by two fully connected layers performed the best on their dataset (Fig. 10). However, in this paper, the authors were building a classification model (hence the fully connected layer with 317 neurons and a Softmax layer), and not a regression model, which would give the exact x and y coordinates of the unknown point.

In this case, to alter the model for the use case of a regression problem, the fully connected layer is amended to use output a sample of size 2, representing the x and y coordinate. The Softmax layer is also removed, to prevent a "selection" of whether the x or y coordinate is more important.

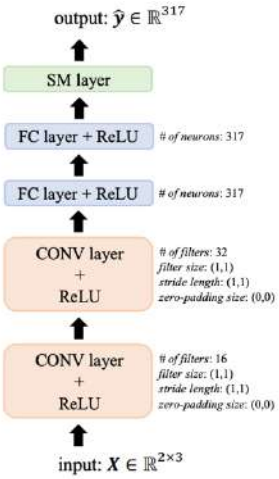


Fig. 10. Network architecture proposed by Al-homayani and Mahoor.

The readings from site 1 was chosen as the dataset, and it was split into training and validation on an 80-20% split (1416728 training points, 354182 validation points). The original data set was used, to avoid introducing any errors in the interpolation process.

The model was trained for 300 epochs with early stopping implemented. The criterion for early stopping is when the validation error (RMSE – same as what was used in the previous section) does not improve over 20 epochs.

This gives a final validation error of 76.10m. The performance of this model is significantly worse than reported in the paper, which might be due to the change in model objective (from classification to regression), necessitating other optimization steps for tuning the model performance.

C. *Experimentation with Deep Learning Based Fingerprinting Model based on Available Encoder-Decoder Network*

During our research, we also carried out experiments with an available deep learning fingerprinting model [4] that used Encoder-Decoder architecture . The inputs for this model are the sensor readings of Geomagnetic, Wi-Fi and iBeacon. The ground truth waypoints are used as labels to train the model. Individual layers are used to encode the representation of data from the three sensors. Several skip connections are present in the model from the raw input data into the merging layer to allow the network to preserve certain important raw sensors' information. All the combinations of inputs contain at least the Geomagnetic sensor data in order to ensure that feature values are not too sparse to produce satisfactory results. These feature representations are then concatenated and used as input to the Decoder network to produce predictions. The MSE mean-squared error function is used as the loss function for the model. The Euclidean distance between the ground truth location and the predicted positions is used as the evaluation metric. We experimented with varying combinations of the features and analysed the results produced by training the model for all the floors in the two given sites in TABLE VII. The values in TABLE VII hold the Euclidean distance in meters between the ground truth and predicted location in the test set.

It is evident from the table above that the best performance is achieved by Geomagnetic + Wi-Fi input to the model with average Euclidean distance of 10.86 meter.

TABLE VII. DEPICTION OF THE AVERAGE EUCLIDEAN DISTANCE (IN METERS) BETWEEN THE GROUNDTRUTH DATA IN THE TEST SETS AND PREDICTIONS ARE SHOWN IN THE TABLE BELOW.

SITE/ FLOOR	GEOMAGNETIC ONLY	GEOMAGNETIC + WIFI	GEOMAGNETIC + iBeacon	GEOMAGNETIC + WIFI + iBeacon
SITE 1/B1	55.40	12.13	31.47	10.43
SITE 1/F1	54.21	12.47	53.33	12.82
SITE 1/F2	46.41	11.17	39.44	11.48
SITE 1/F3	50.11	9.30	40.45	9.08
SITE 1/F4	49.07	11.40	38.41	11.44
SITE 2/B1	40.55	8.01	23.87	7.71
SITE 2/F1	44.49	12.22	33.26	11.92
SITE 2/F2	48.74	11.89	42.26	13.00
SITE 2/F3	40.33	11.62	41.14	12.57
SITE 2/F4	39.07	10.15	43.04	17.74
SITE 2/F5	46.59	13.08	44.64	12.97
SITE 2/F6	55.06	9.78	40.14	9.14
SITE 2/F7	46.95	10.12	36.64	10.72
SITE 2/F8	39.60	8.81	40.04	9.30
Average	46.89	**10.86**	39.15	11.45

ACKNOWLEDGMENT

Kishore Rajasekar worked on the visualization of the waypoints data, as well as experimenting with the deep learning model based on the encoder-decoder network. Seah Xiu Xuan Sherbelle worked on the visualization of the geomagnetic field strengths, as well as developing the deep learning model based on CNNs. Sean Goh Ann Ray worked on the visualization of Wi-Fi RSSI data, as well as developing the deep learning model based on linear regression. Sita Rajagopal worked on visualization of the iBeacon data, and the write-up on the abstract, introduction and dataset section.

REFERENCES

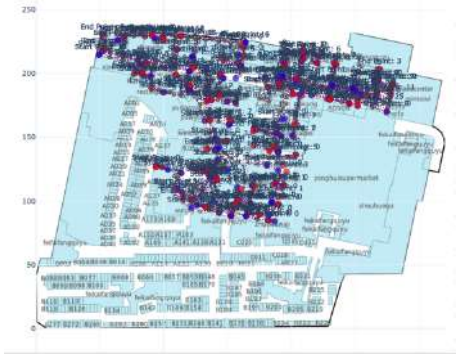
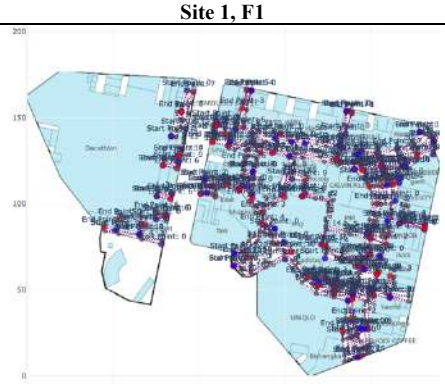
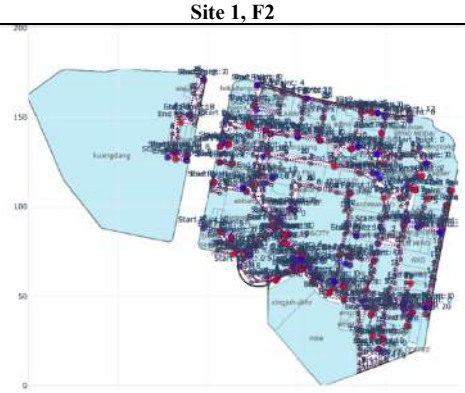
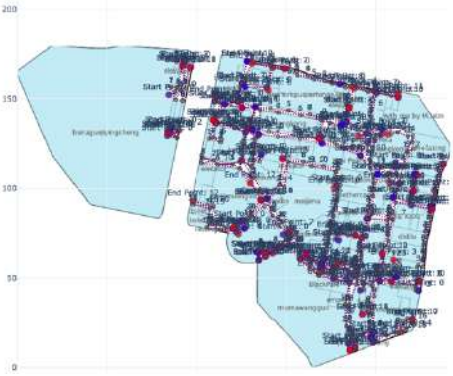
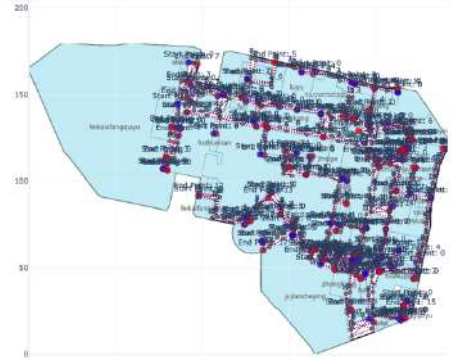
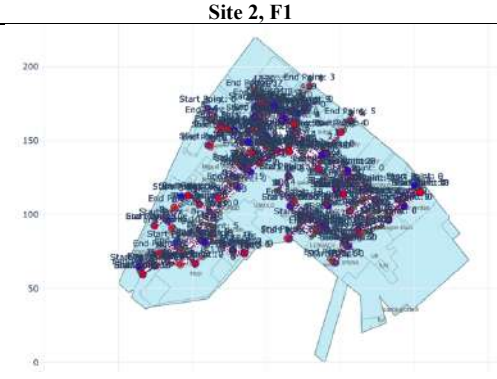
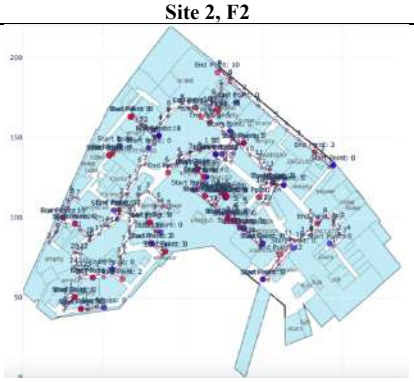
[1] M. I. L. Competition, "Indoor Location Competition 2.0 (Sample Data and Code)," 2021. [Online]. Available: https://github.com/location-competition/indoor-location-competition-20.

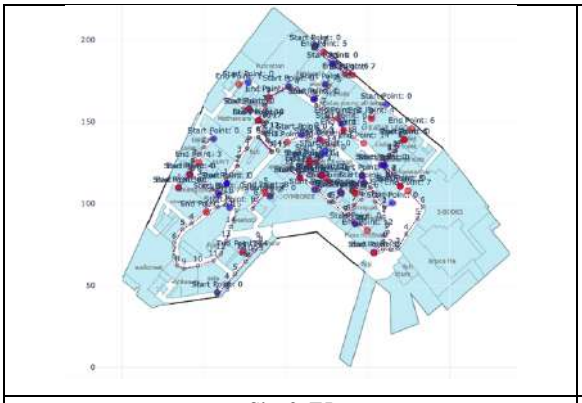
[2] MOKO BLUE, "A Detailed Guide to iBeacon," 19 March 2021. [Online]. Available: https://www.mokoblue.com/a-detailed-guide-to-ibeacon/.

[3] F. Al-homayani and M. Mahoor, "Improved Indoor Geomagnetic Field Fingerprinting for Smartwatch Localization Using Deep Learning," Nantes, 2018.

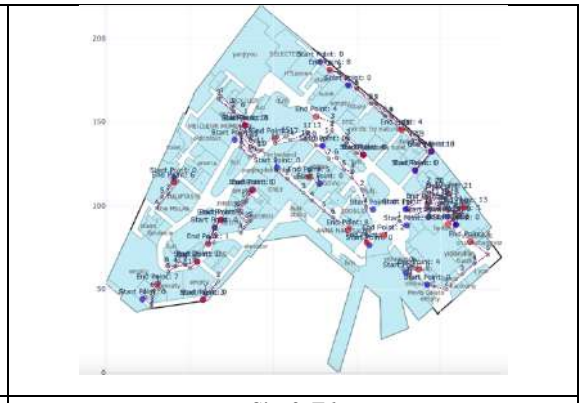
[4] J. H. Ong, M. Tan and Z. Zhu, "Smartphone-based Indoor Localization," 11 December 2020. [Online]. Available: https://github.com/kkaryl/AI6128-Urban_Computing/tree/master/course-project-1.

APPENDIX A – VISUALIZATION OF WAYPOINTS

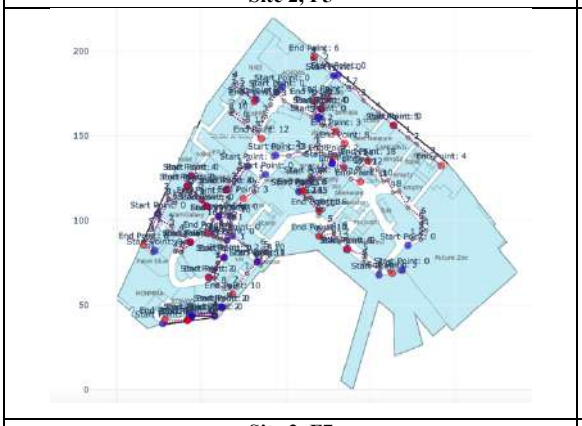
Site 1, B1	Site 1, F1
	
Site 1, F2	**Site 1, F3**
	
Site 1, F4	**Site 2, B1**
	
Site 2, F1	**Site 2, F2**
	
Site 2, F3	**Site 2, F4**



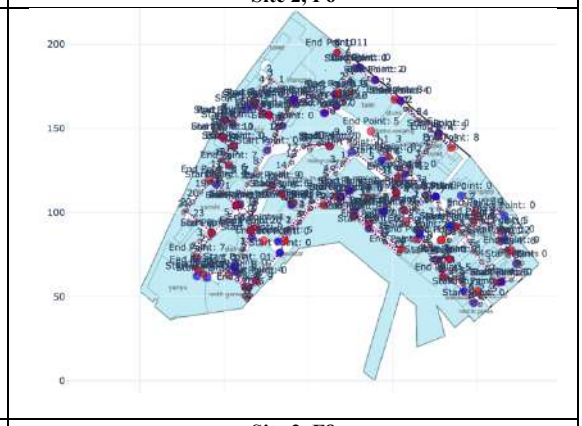
Site 2, F5



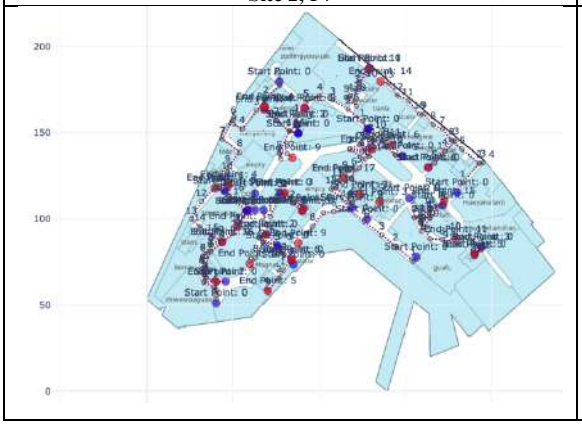
Site 2, F6

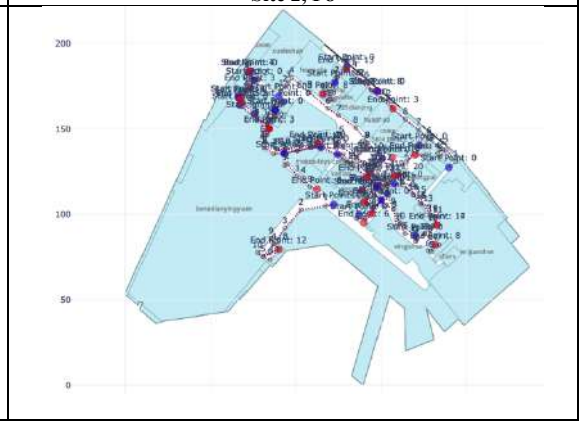


Site 2, F7

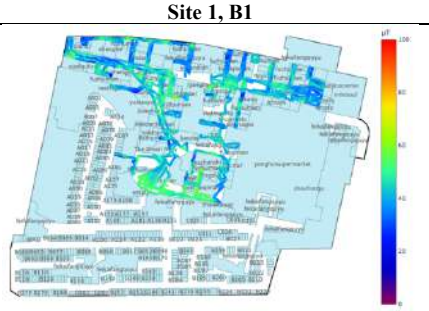
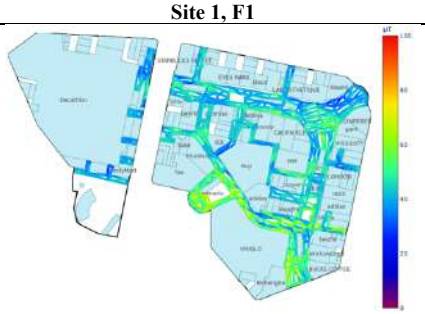
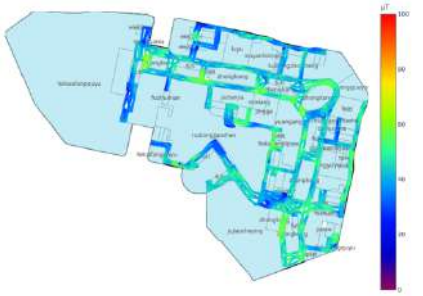
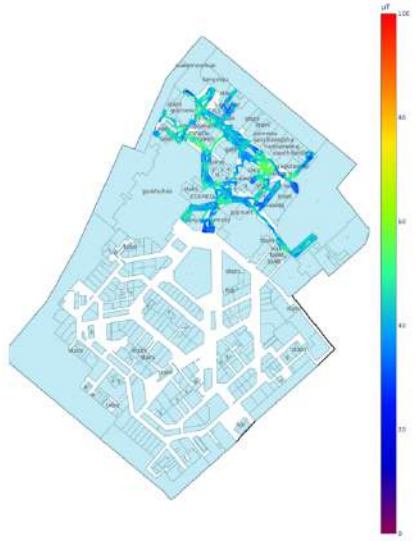
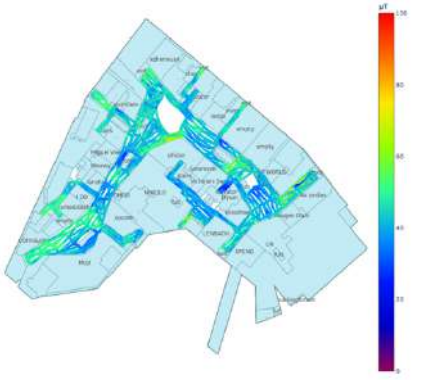
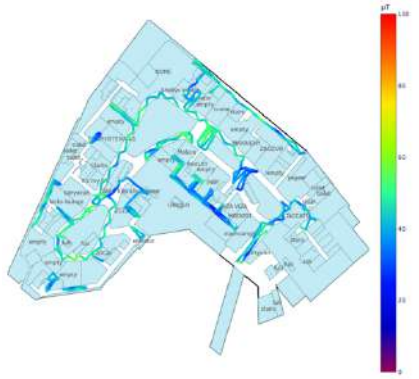


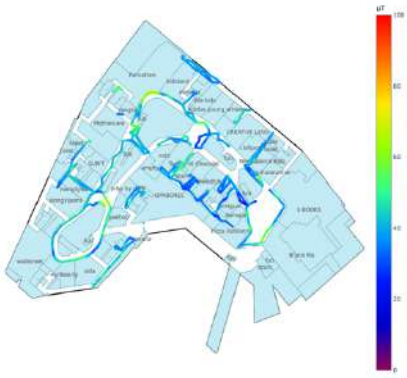
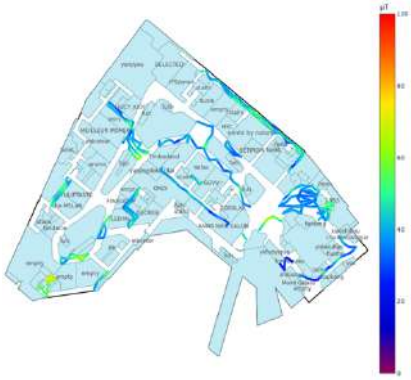
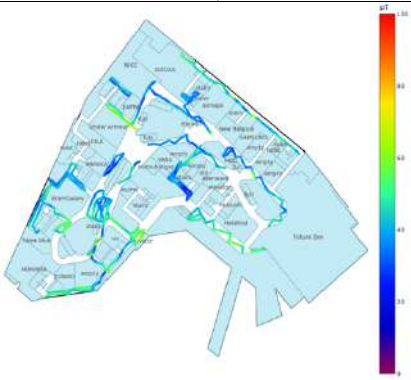
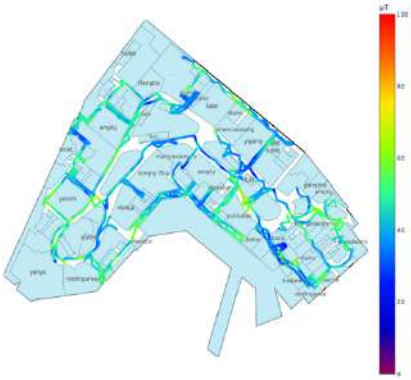
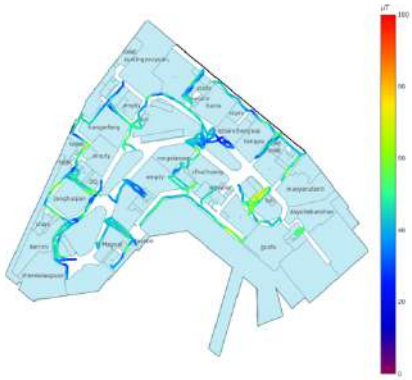
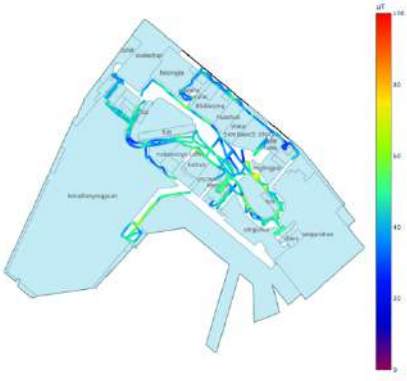
Site 2, F8



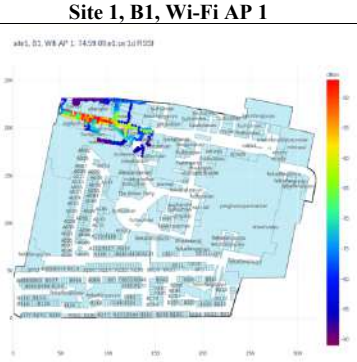
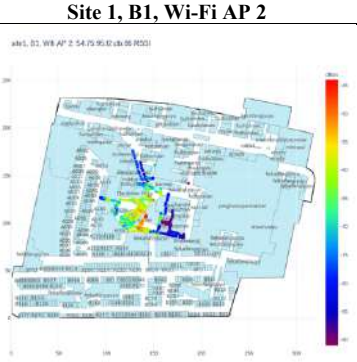
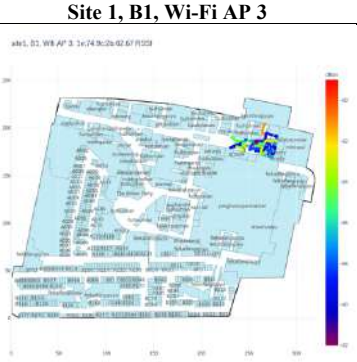
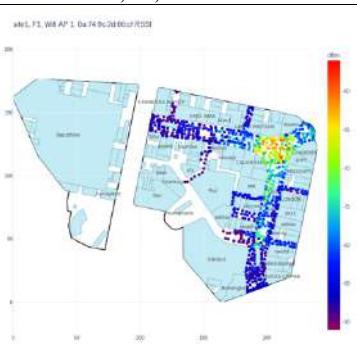
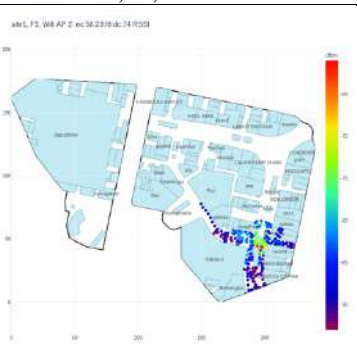
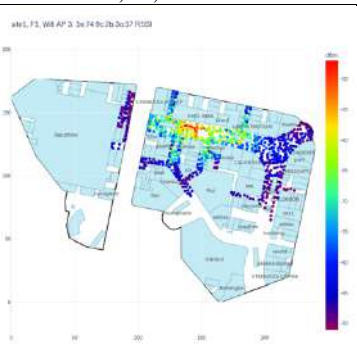
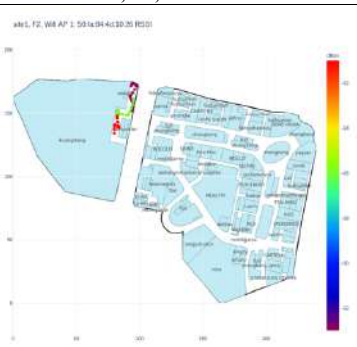
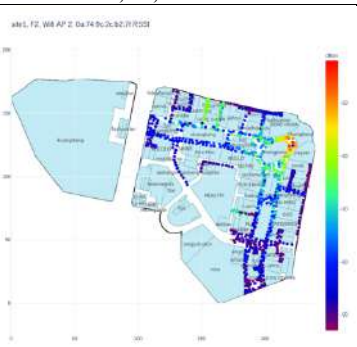
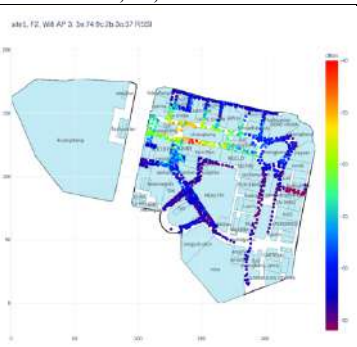
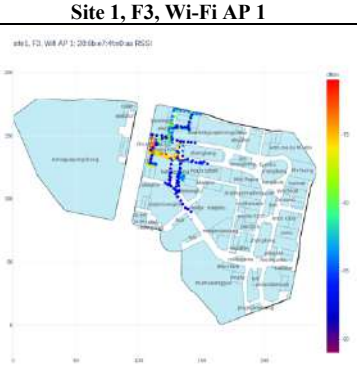
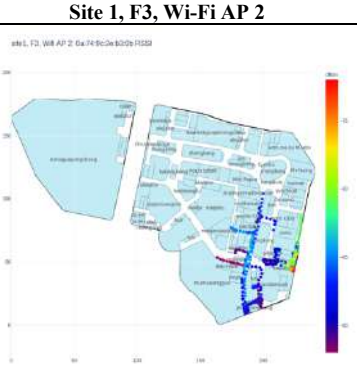
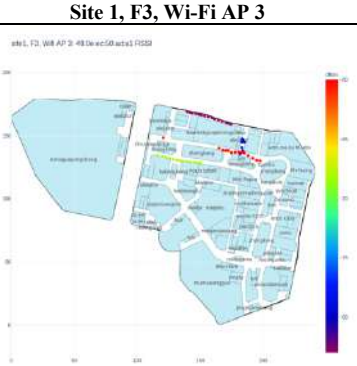
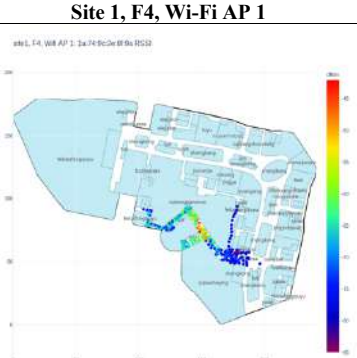
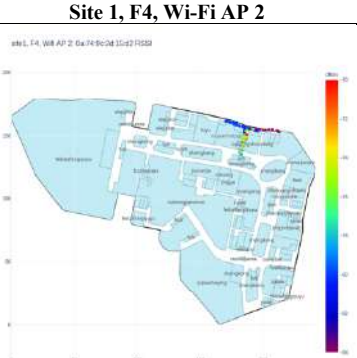
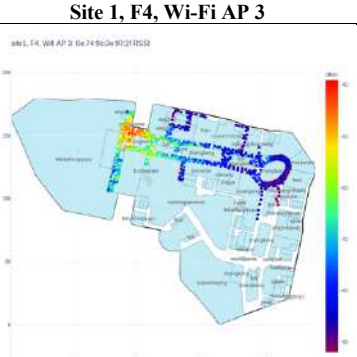


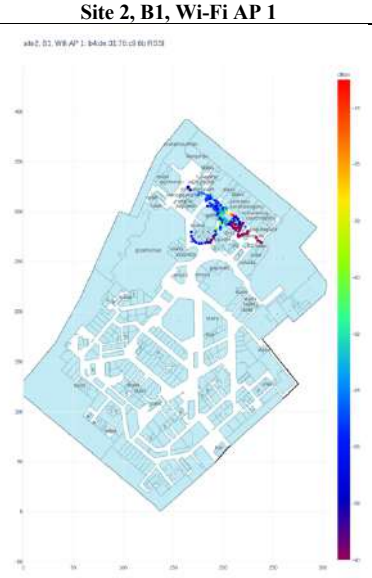
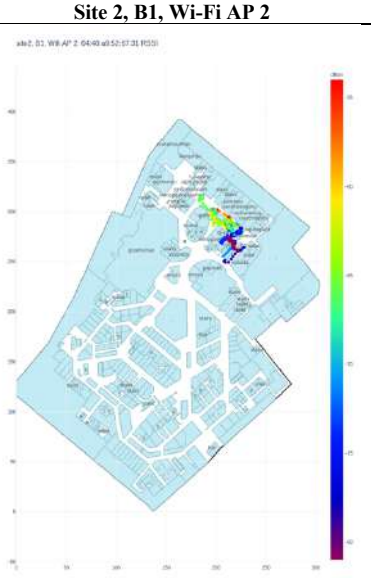
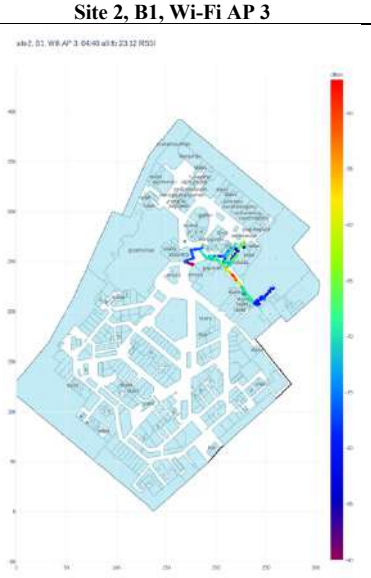
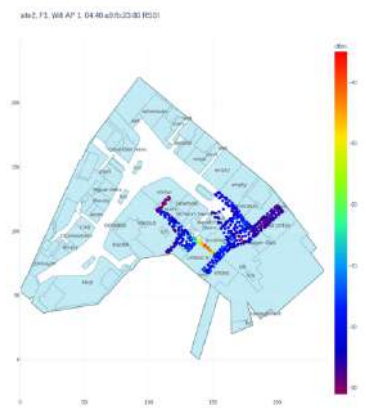
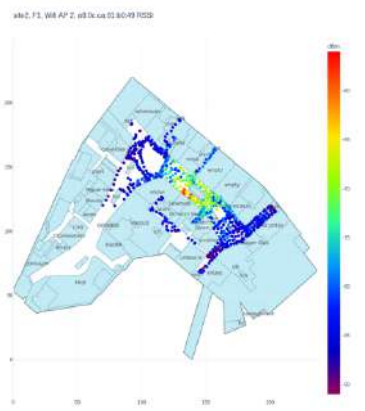
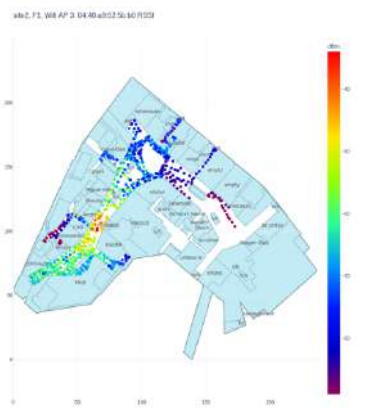
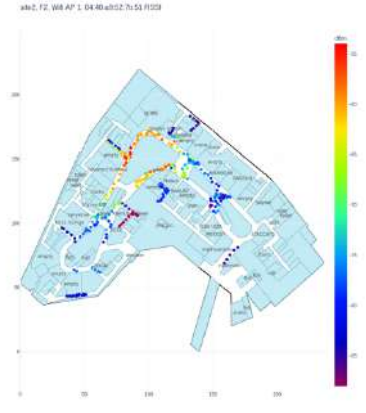
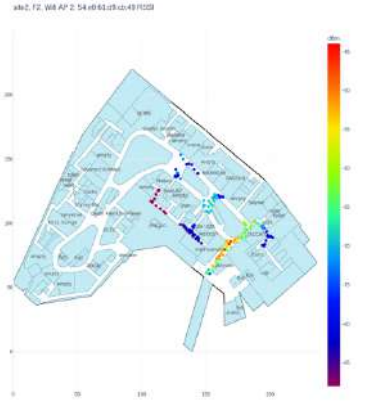
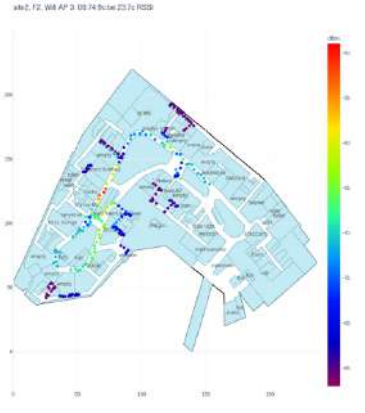
APPENDIX B – VISUALIZATION OF GEOMAGNETIC FIELD STRENGTHS

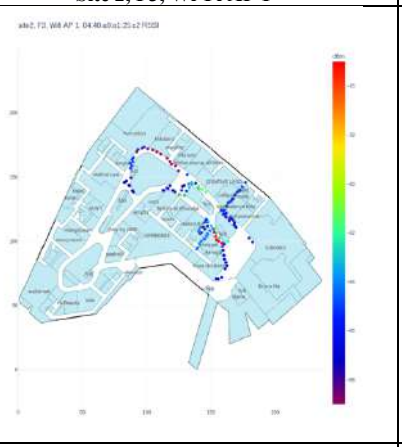
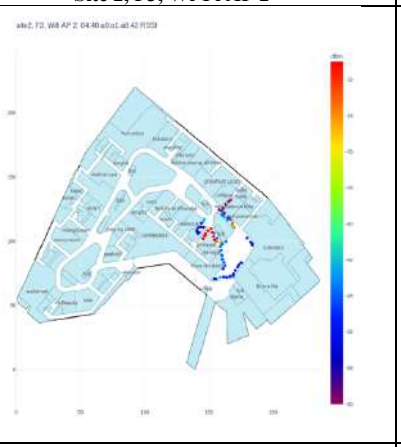
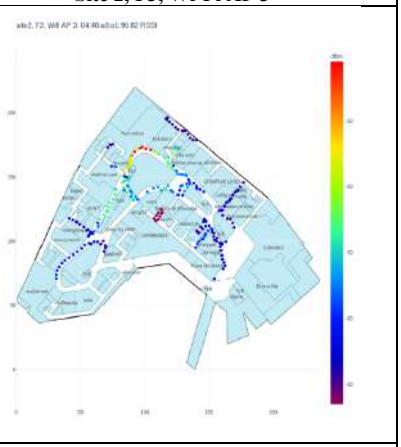
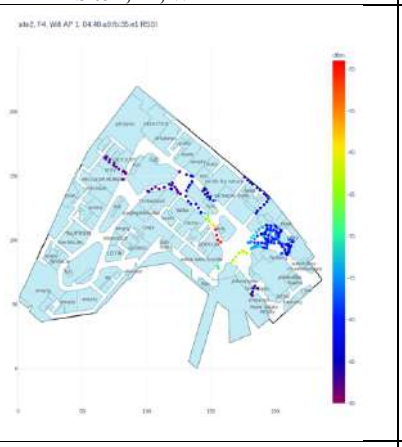
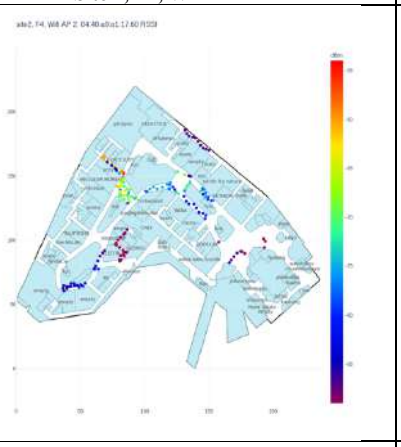
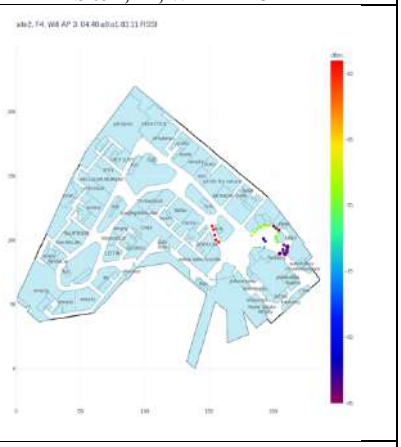
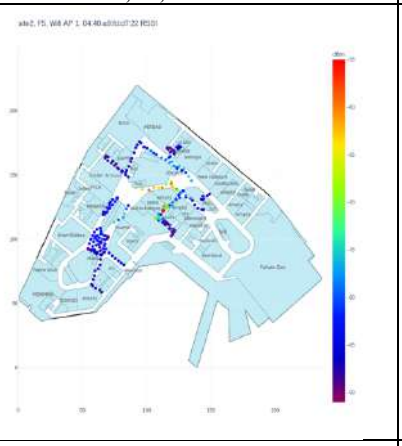
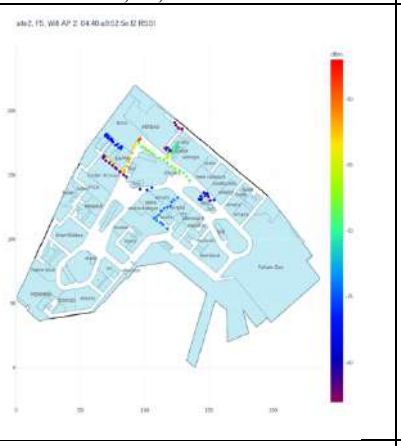
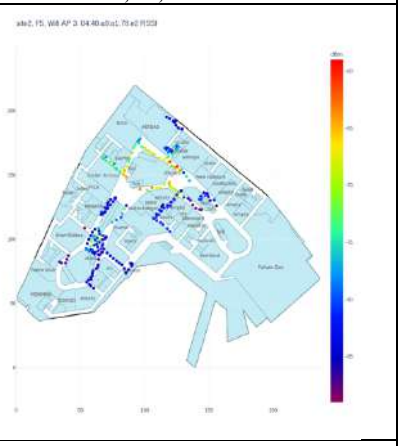
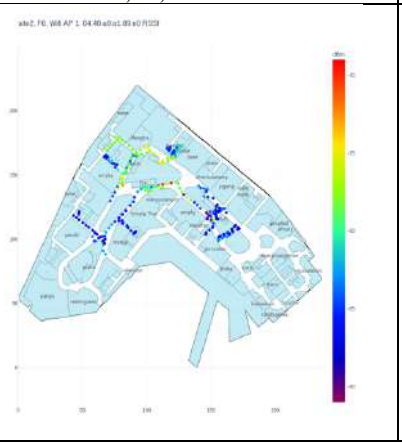
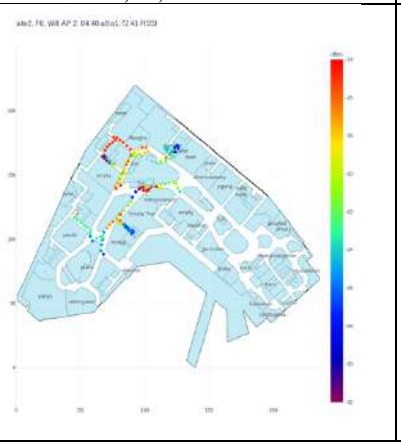
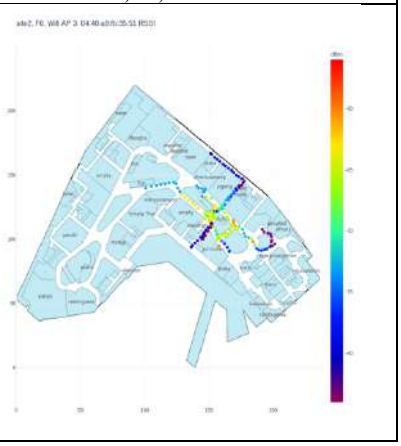
Site 1, B1	Site 1, F1
	
Site 1, F2	**Site 1, F3**
	
Site 1, F4	**Site 2, B1**
	
Site 2, F1	**Site 2, F2**
	

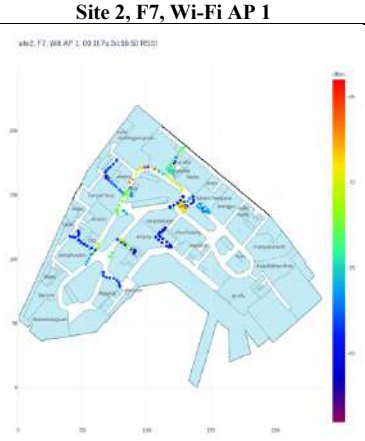
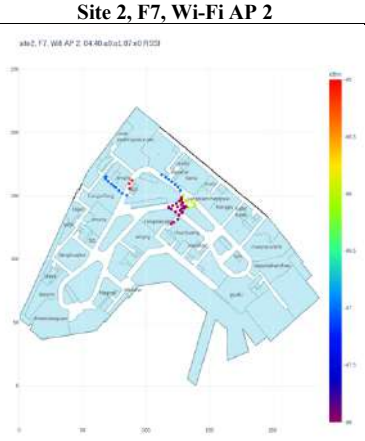
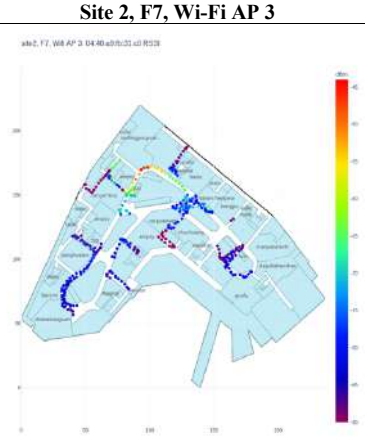
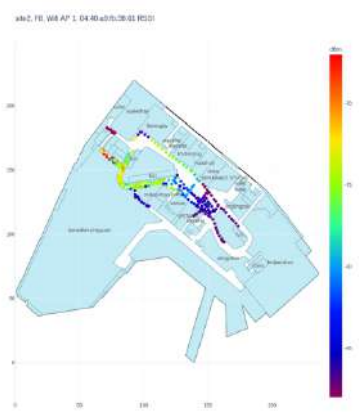
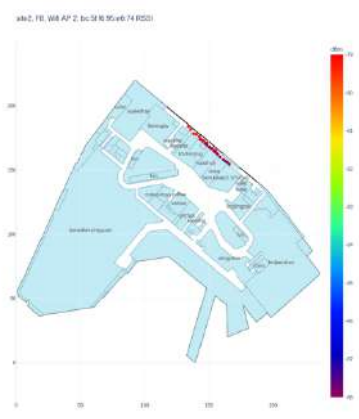
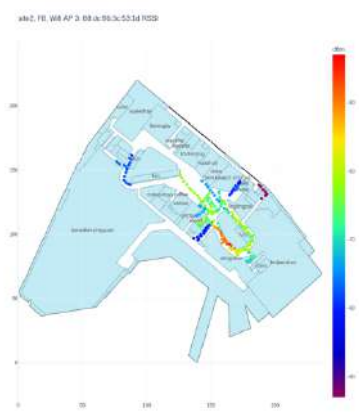
Site 2, F3	Site 2, F4
	
Site 2, F5	**Site 2, F6**
	
Site 2, F7	**Site 2, F8**
	

APPENDIX C – VISUALIZATION OF WIFI RSS

Site 1, B1, Wi-Fi AP 1	Site 1, B1, Wi-Fi AP 2	Site 1, B1, Wi-Fi AP 3
		
Site 1, F1, Wi-Fi AP 1	**Site 1, F1, Wi-Fi AP 2**	**Site 1, F1, Wi-Fi AP 3**
		
Site 1, F2, Wi-Fi AP 1	**Site 1, F2, Wi-Fi AP 2**	**Site 1, F2, Wi-Fi AP 3**
		
Site 1, F3, Wi-Fi AP 1	**Site 1, F3, Wi-Fi AP 2**	**Site 1, F3, Wi-Fi AP 3**
		
Site 1, F4, Wi-Fi AP 1	**Site 1, F4, Wi-Fi AP 2**	**Site 1, F4, Wi-Fi AP 3**
		

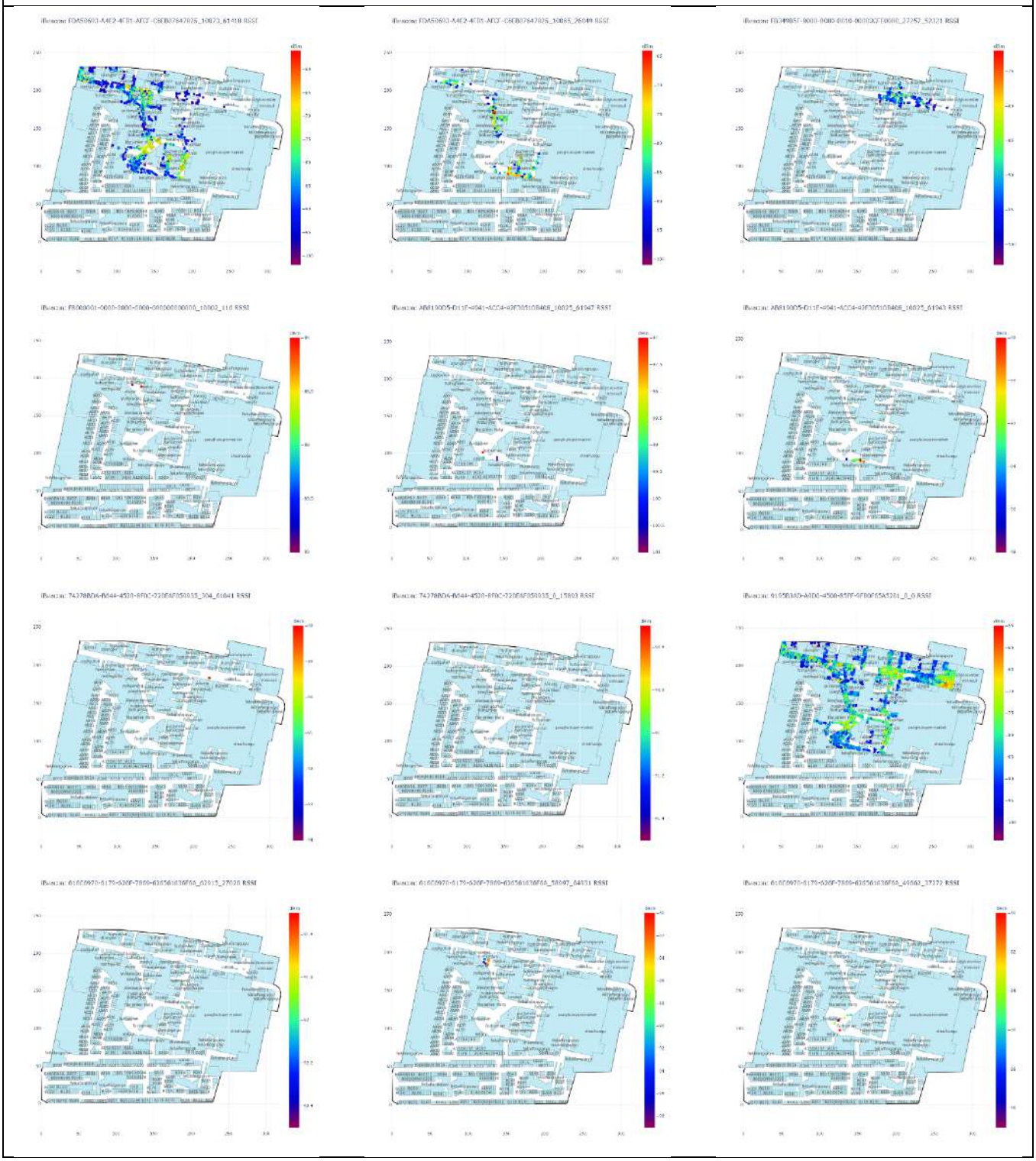
Site 2, B1, Wi-Fi AP 1	Site 2, B1, Wi-Fi AP 2	Site 2, B1, Wi-Fi AP 3
		
Site 2, F1, Wi-Fi AP 1	**Site 2, F1, Wi-Fi AP 2**	**Site 2, F1, Wi-Fi AP 3**
		
Site 2, F2, Wi-Fi AP 1	**Site 2, F2, Wi-Fi AP 2**	**Site 2, F2, Wi-Fi AP 3**
		

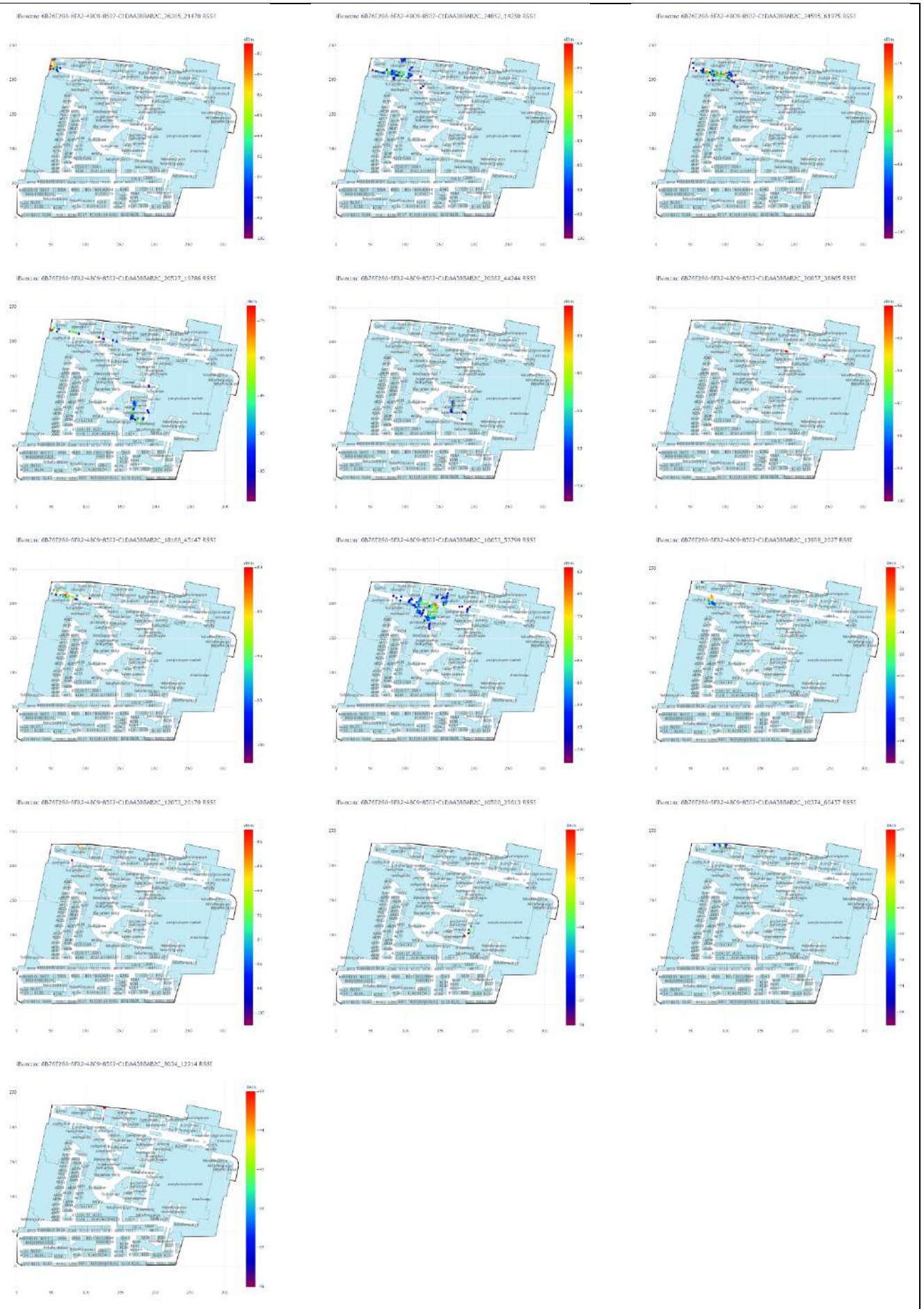
Site 2, F3, Wi-Fi AP 1	Site 2, F3, Wi-Fi AP 2	Site 2, F3, Wi-Fi AP 3
		
Site 2, F4, Wi-Fi AP 1	**Site 2, F4, Wi-Fi AP 2**	**Site 2, F4, Wi-Fi AP 3**
		
Site 2, F5, Wi-Fi AP 1	**Site 2, F5, Wi-Fi AP 2**	**Site 2, F5, Wi-Fi AP 3**
		
Site 2, F6, Wi-Fi AP 1	**Site 2, F6, Wi-Fi AP 2**	**Site 2, F6, Wi-Fi AP 3**
		

Site 2, F7, Wi-Fi AP 1	Site 2, F7, Wi-Fi AP 2	Site 2, F7, Wi-Fi AP 3
		
Site 2, F8, Wi-Fi AP 1	**Site 2, F8, Wi-Fi AP 2**	**Site 2, F8, Wi-Fi AP 3**
		

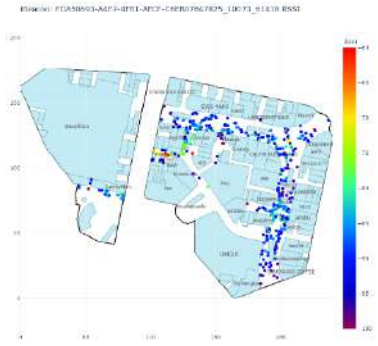
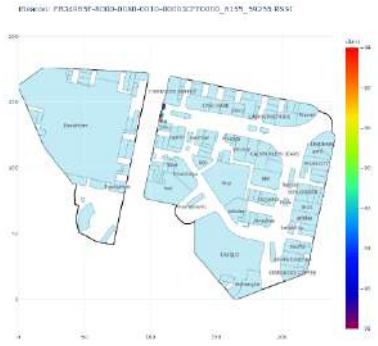
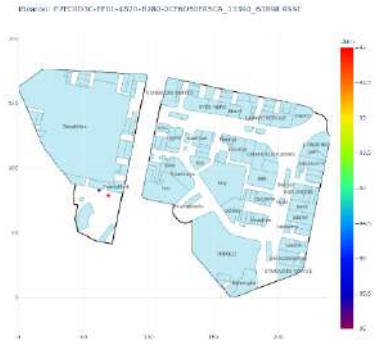
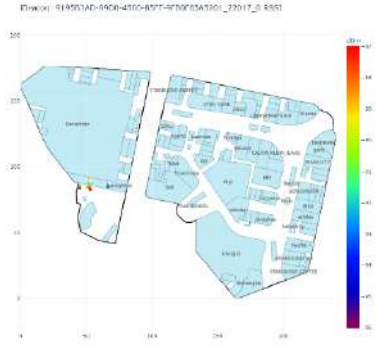
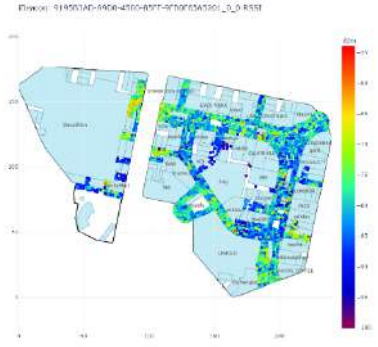
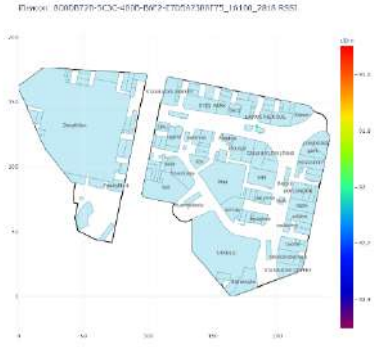
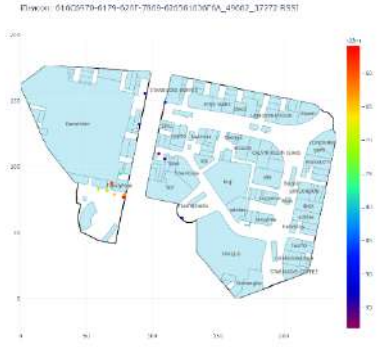
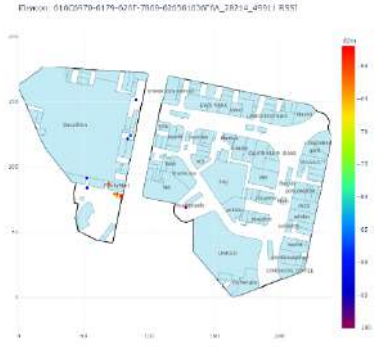
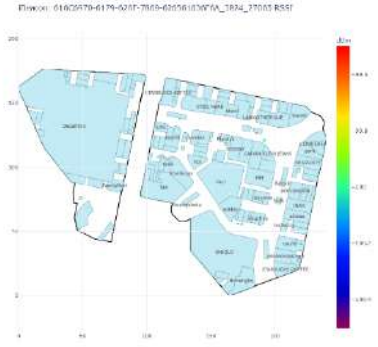
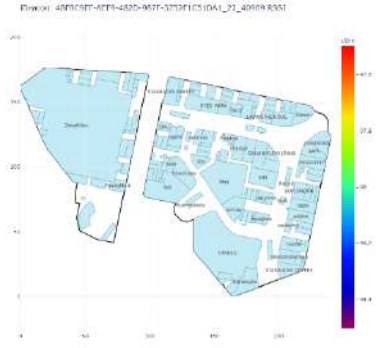
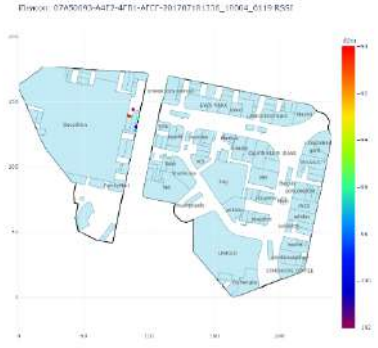
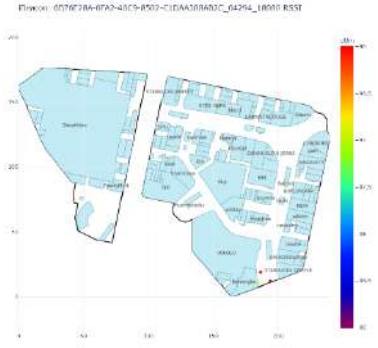
APPENDIX D – VISUALIZATION OF IBEACON RSS

Site 1, B1



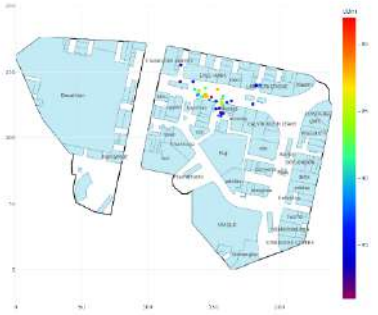


Site 1, F1

Beacon: 6B76E2BA-6FA2-48C9-B502-C1DAA38BAB2C_31635_15211 RSSI

Beacon: 6B76E2BA-6FA2-48C9-B502-C1DAA38BAB2C_47237_15305 RSSI

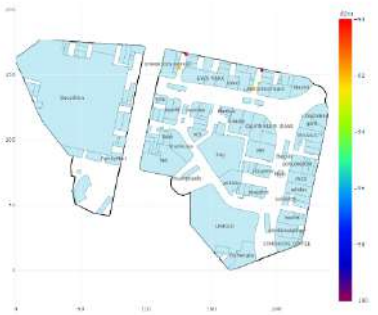


Beacon: 6B76E2BA-6FA2-48C9-B502-C1DAA38BAB2C_45858_14031 RSSI

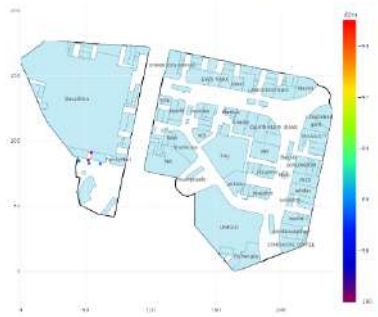
Beacon: 6B76E2BA-6FA2-48C9-B502-C1DAA38BAB2C_45161_52732 RSSI

Beacon: 6B76E2BA-6FA2-48C9-B502-C1DAA38BAB2C_25448_53473 RSSI

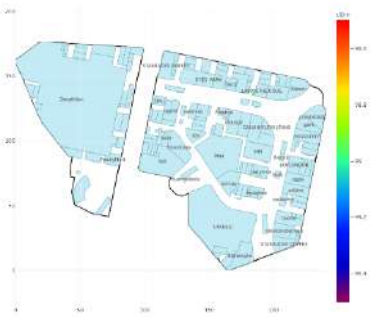
Beacon: 6B76E2BA-6FA2-48C9-B502-C1DAA38BAB2C_19018_53278 RSSI



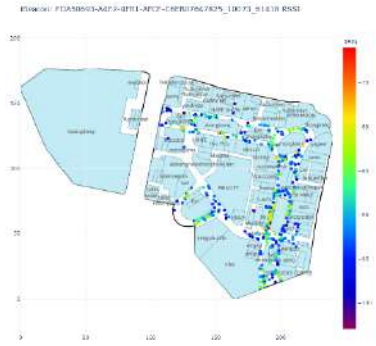
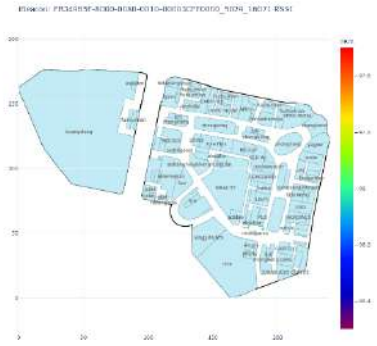
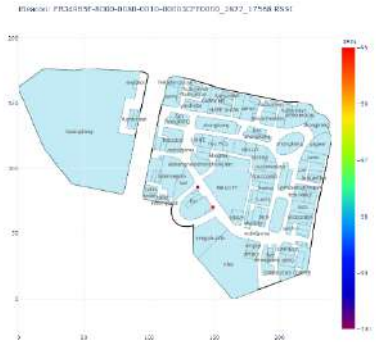
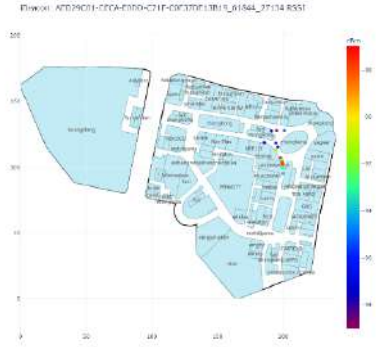
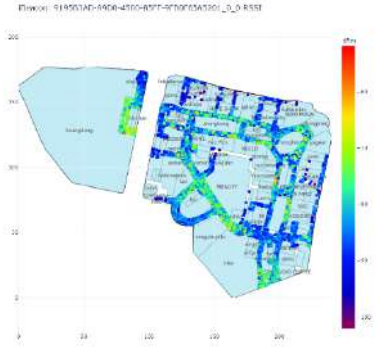
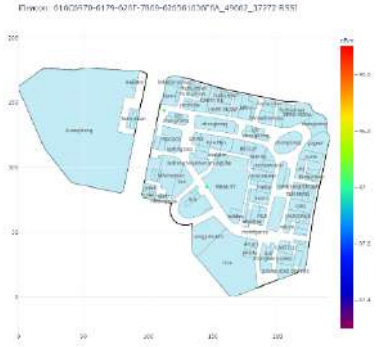
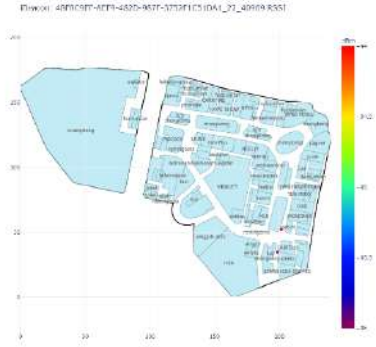
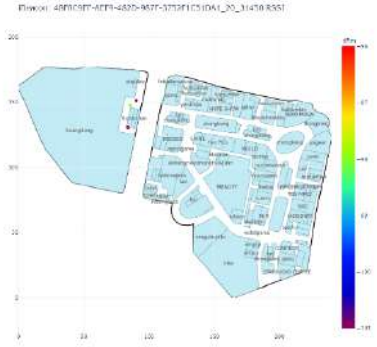
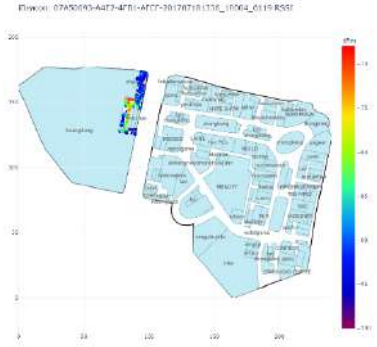
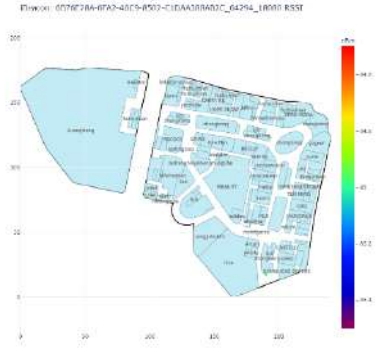
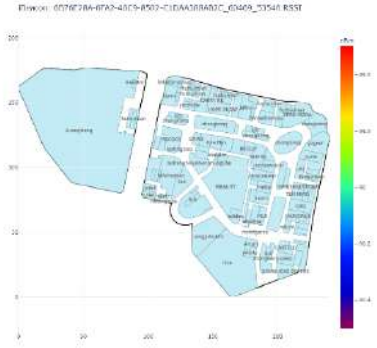
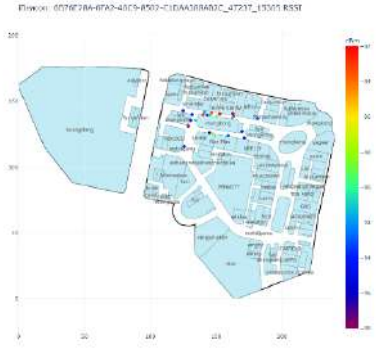
Beacon: 6B76E2BA-6FA2-48C9-B502-C1DAA38BAB2C_6209_6410 RSSI

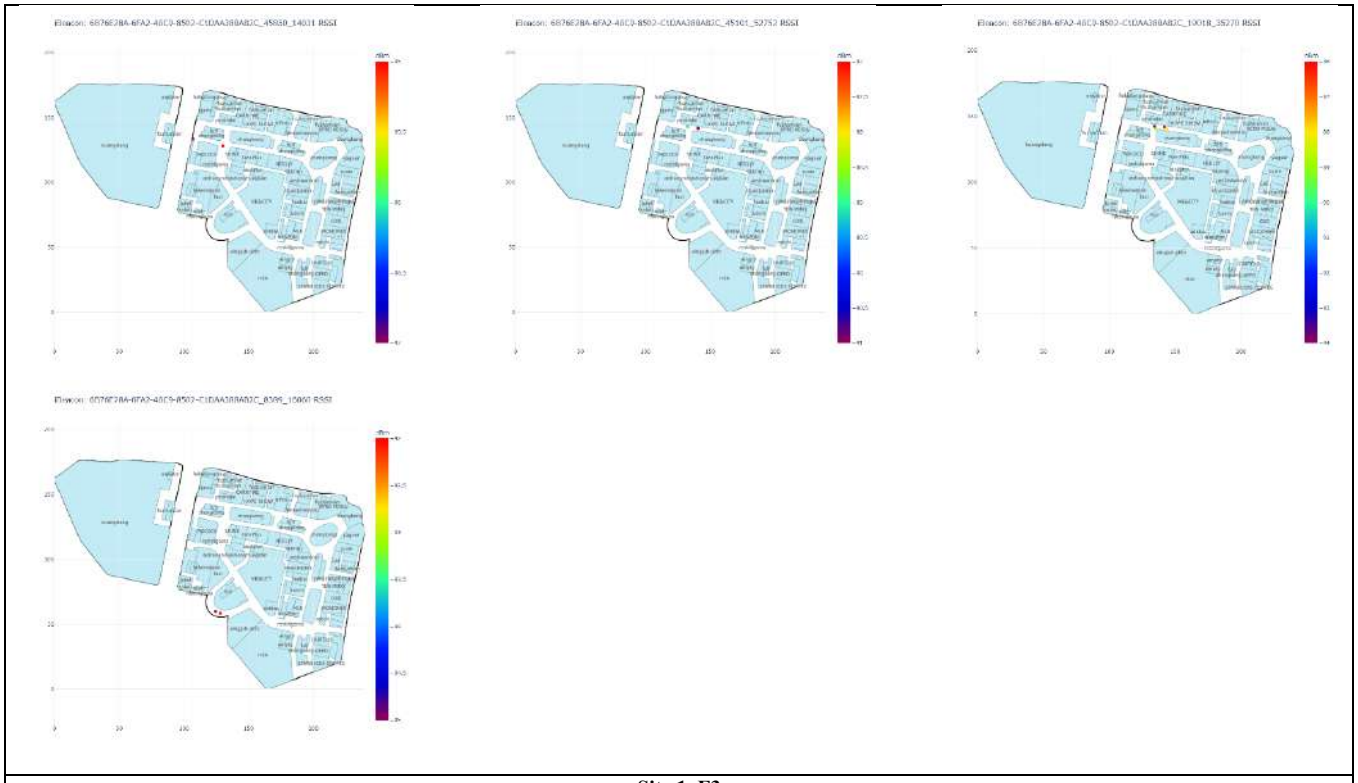


Beacon: 3C3491F1-997A-D187-9AD9-F03A8D4FFA03_20347_6415 RSSI

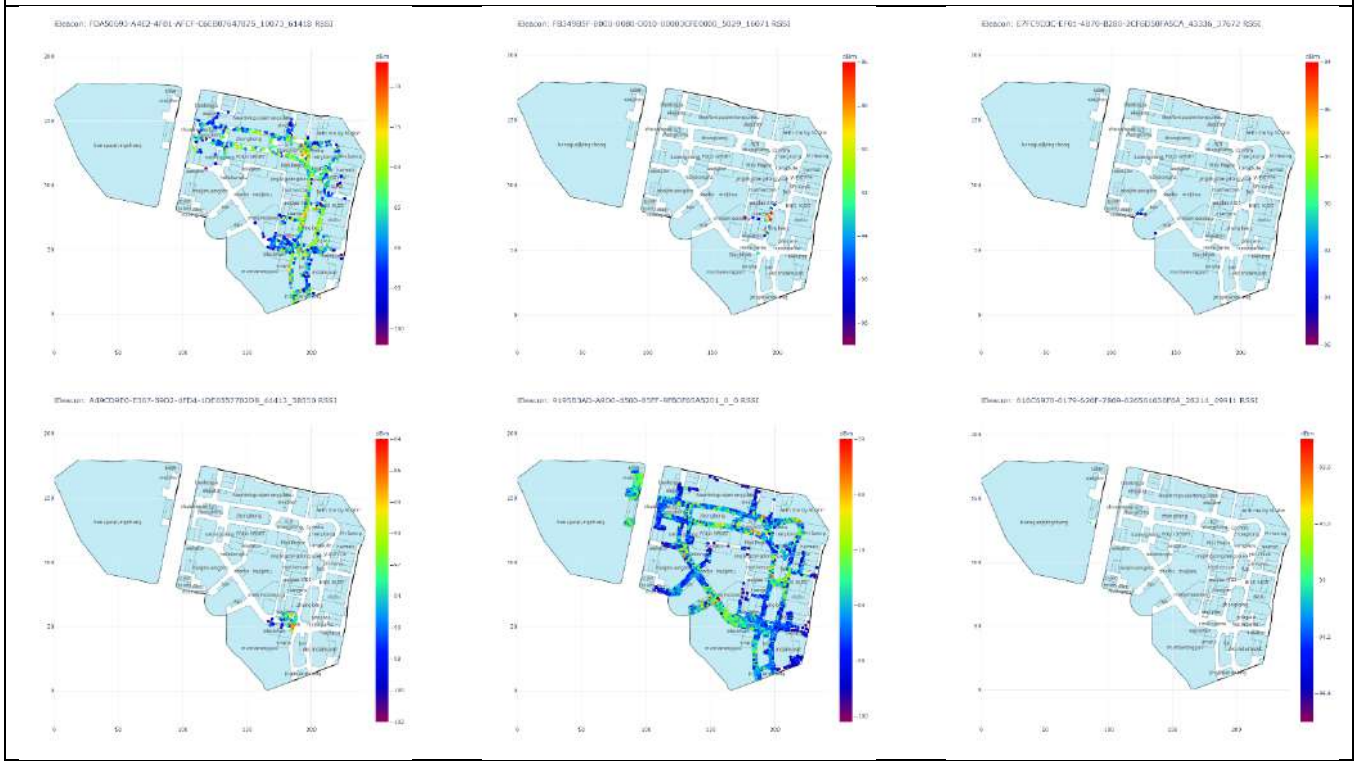


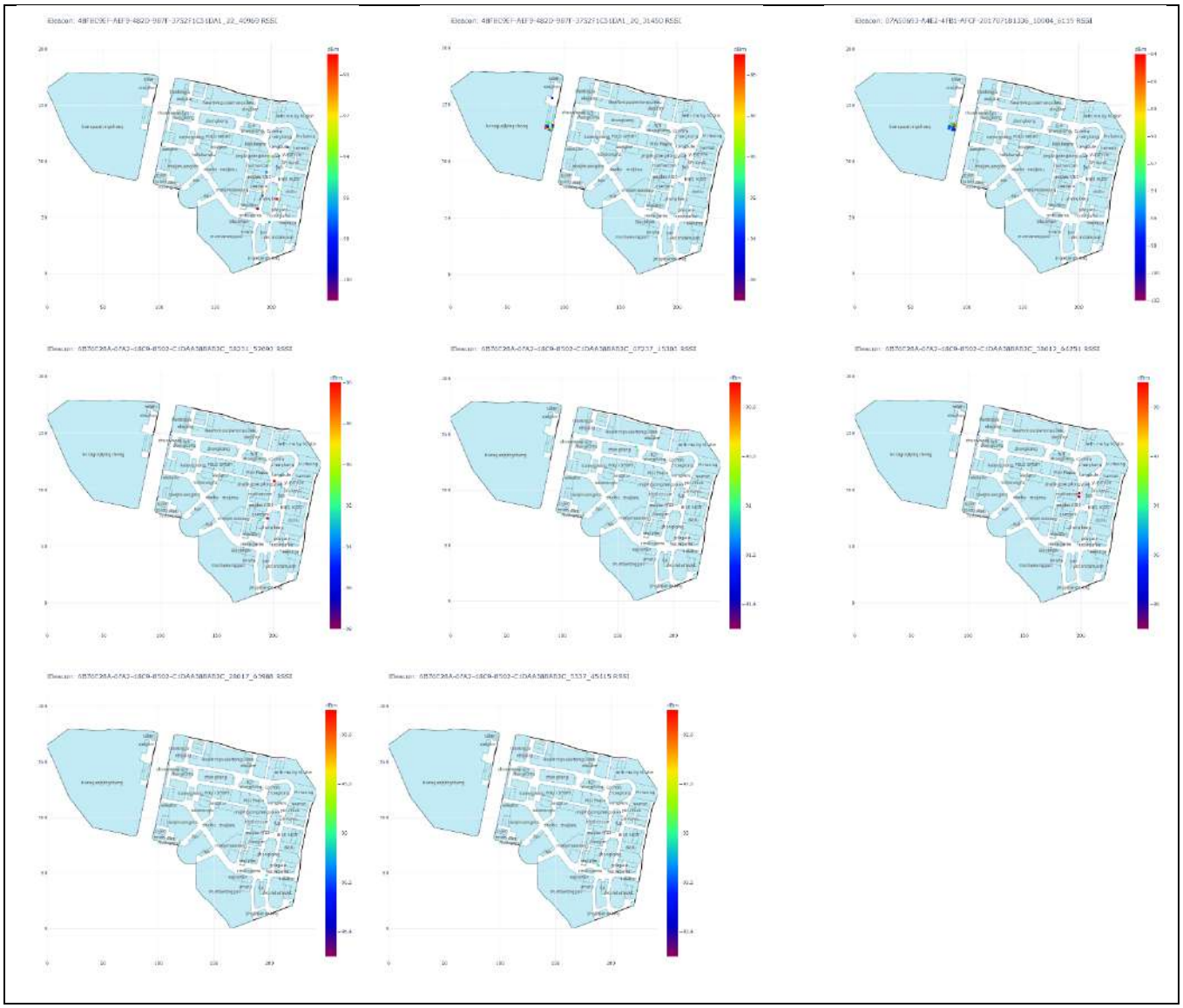
Site 1, F2

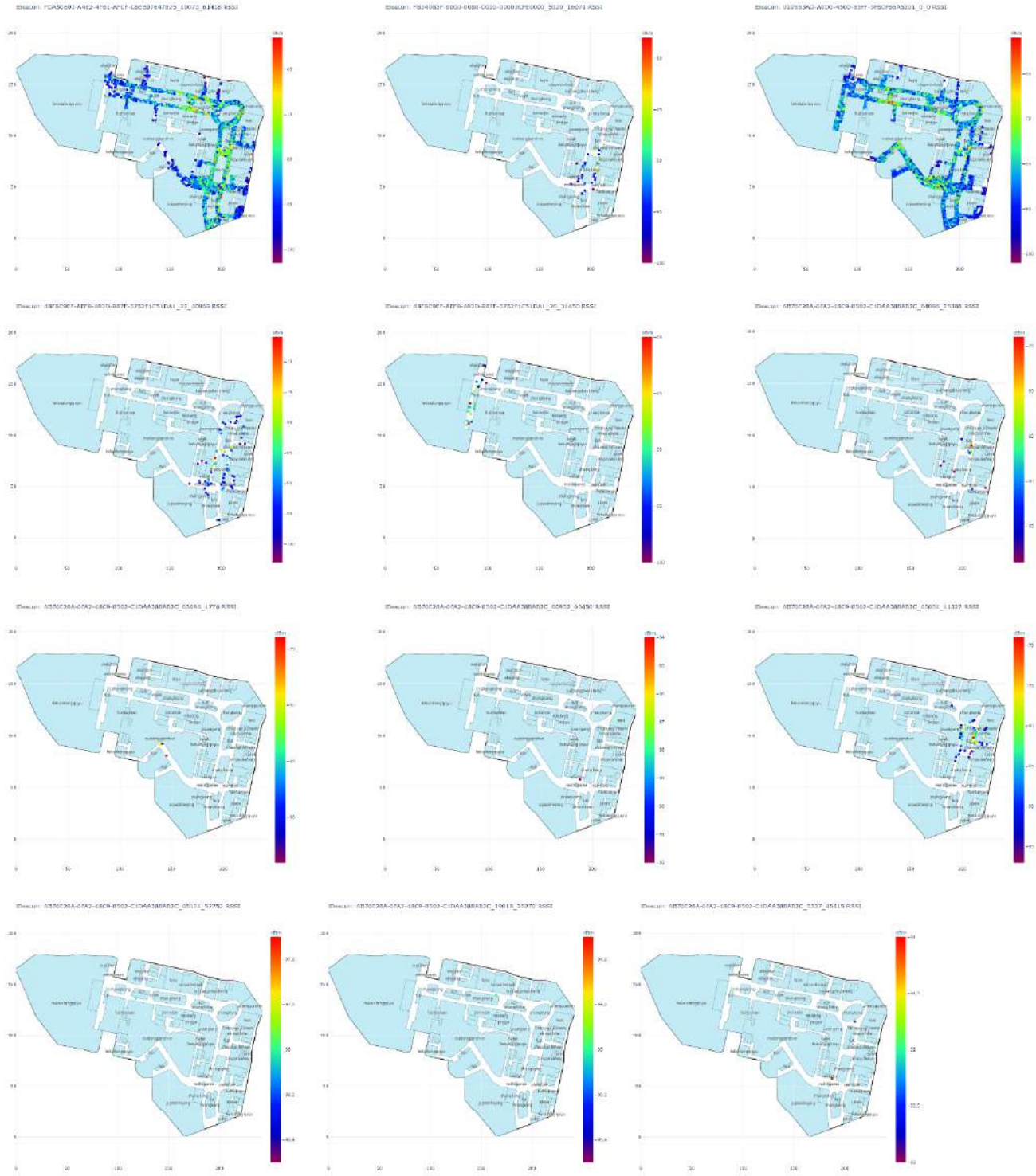


Site 1, F3

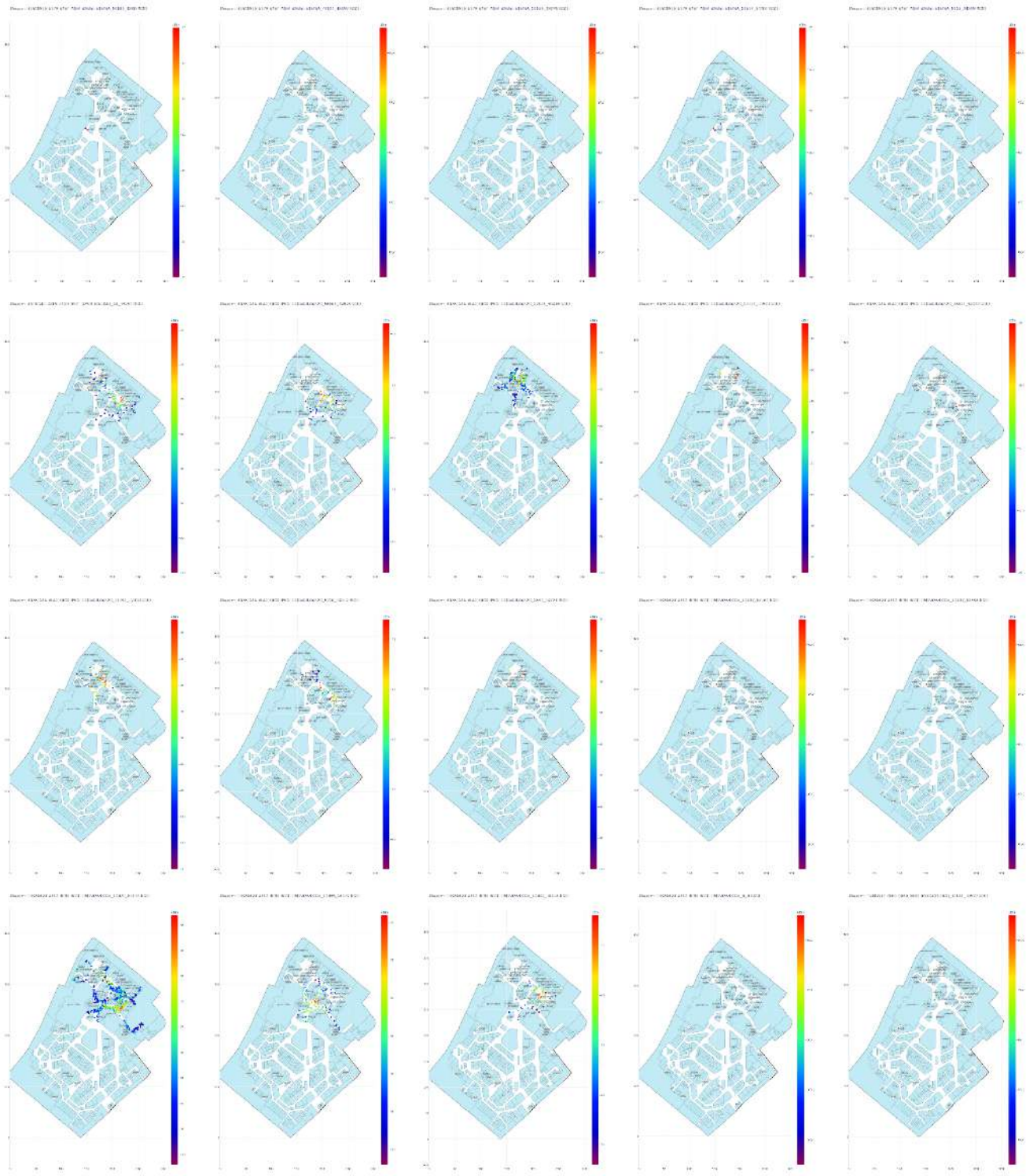


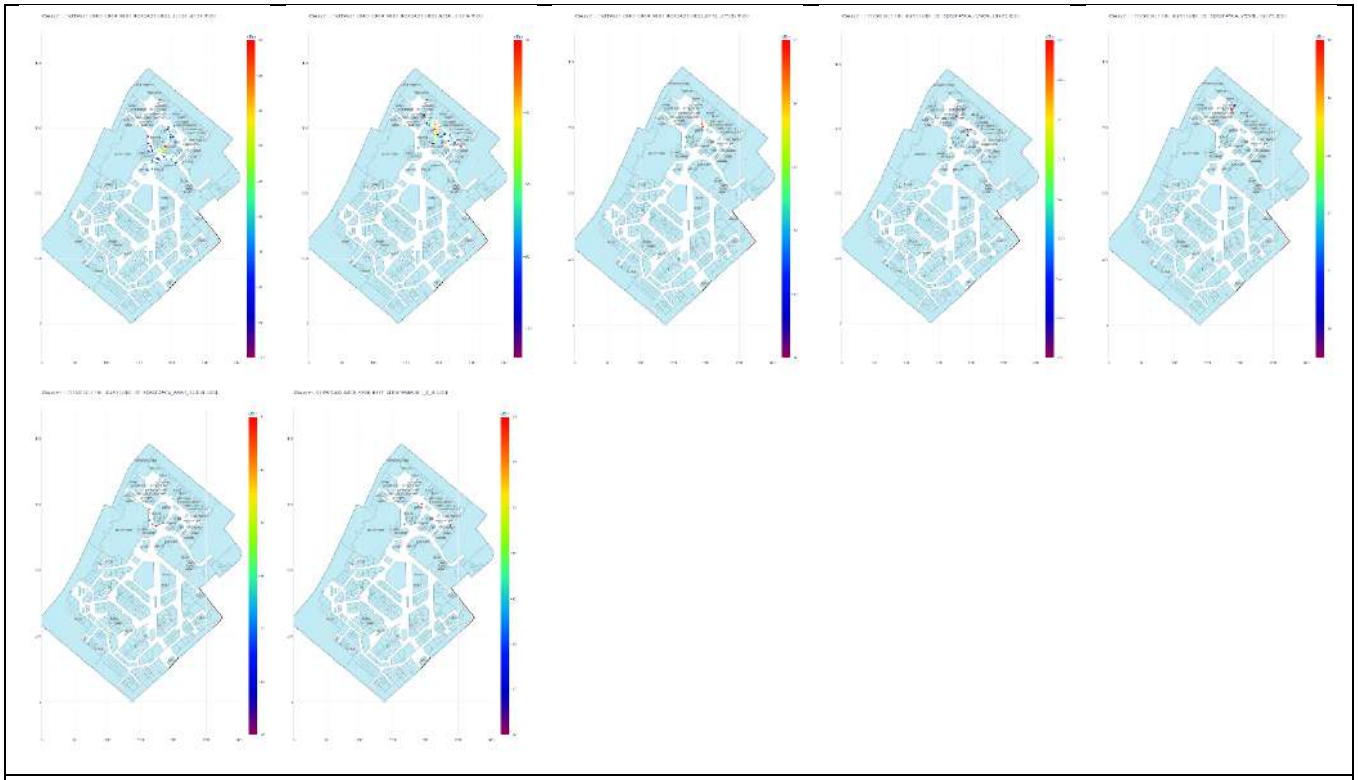


Site 1, F4

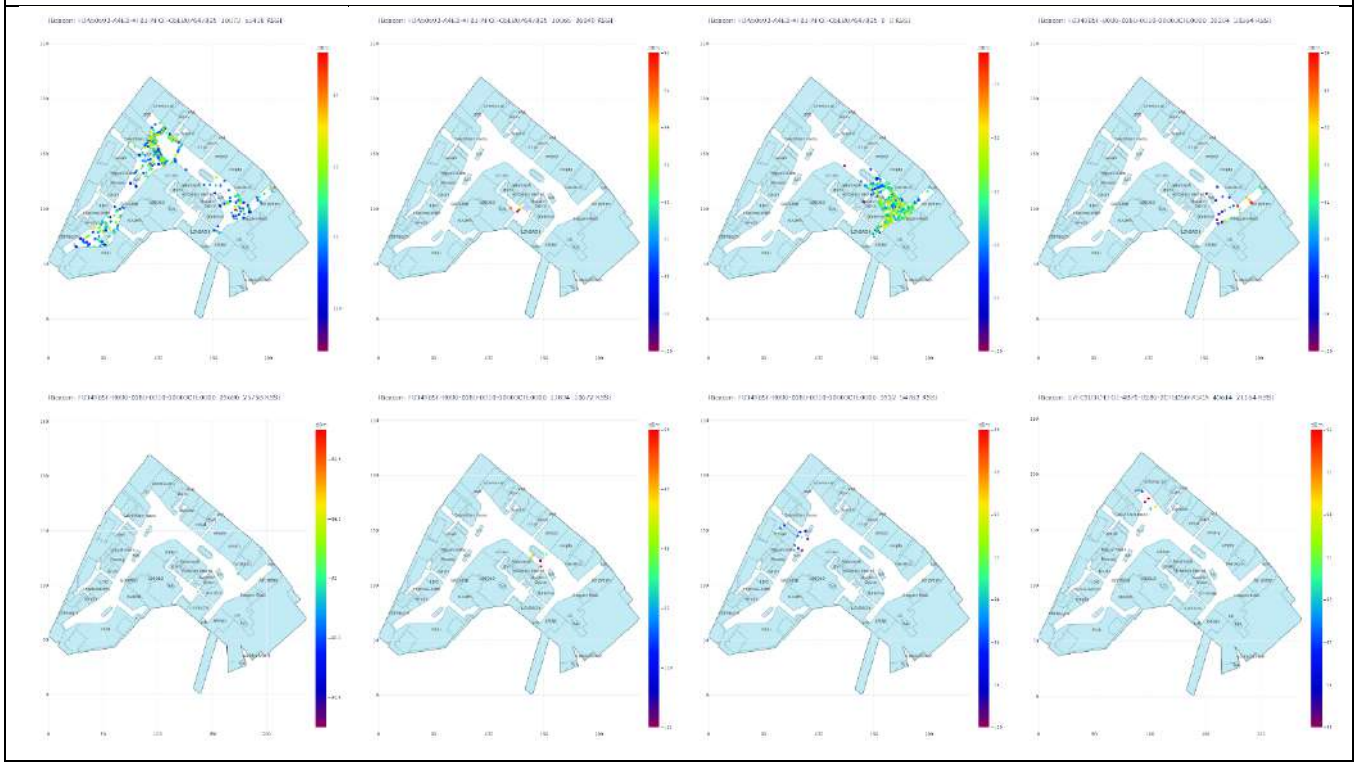


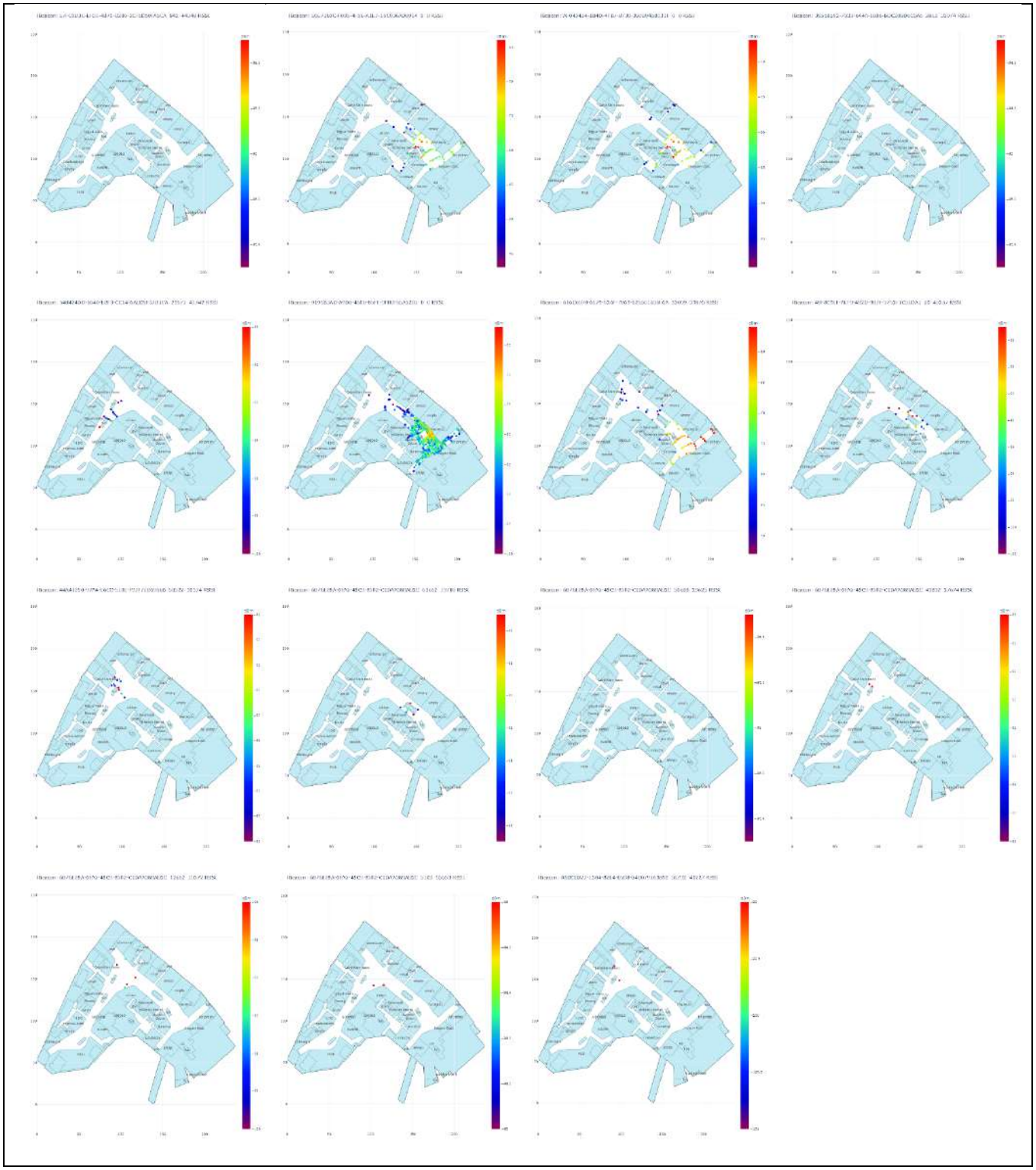
Site 2, B1



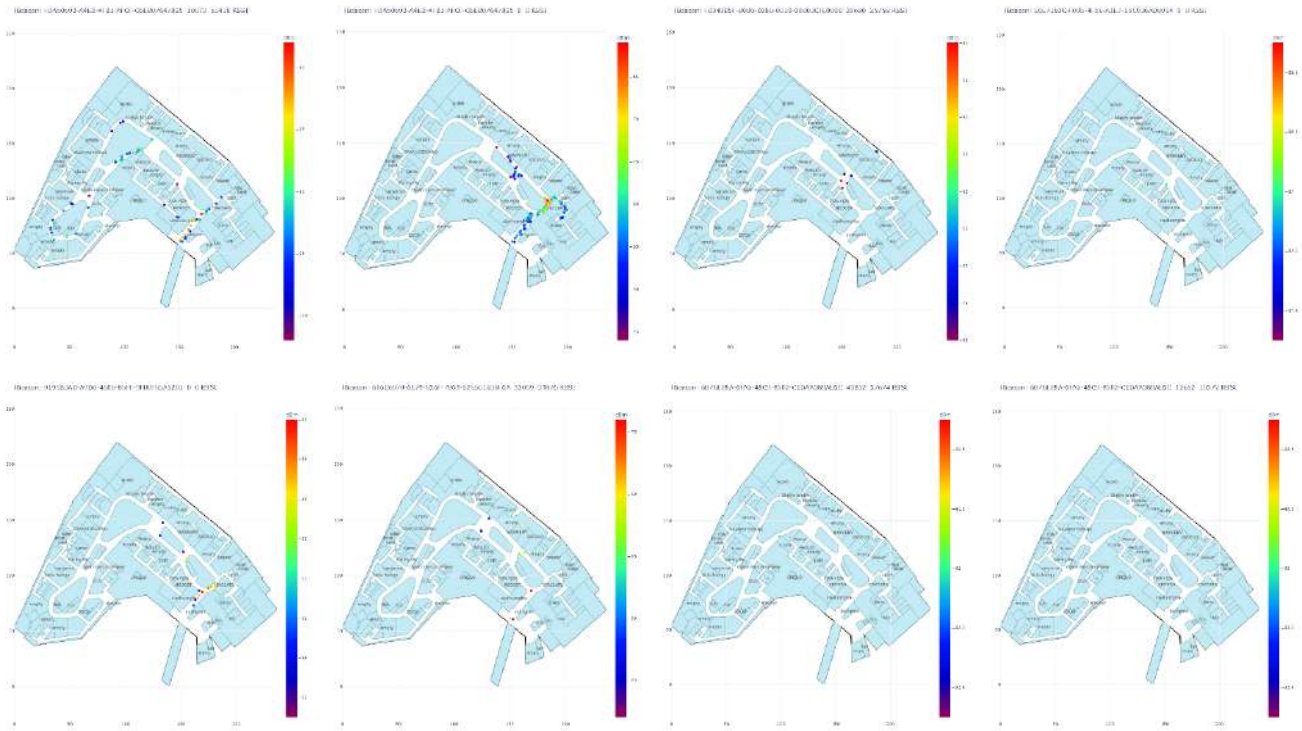


Site 2, F1

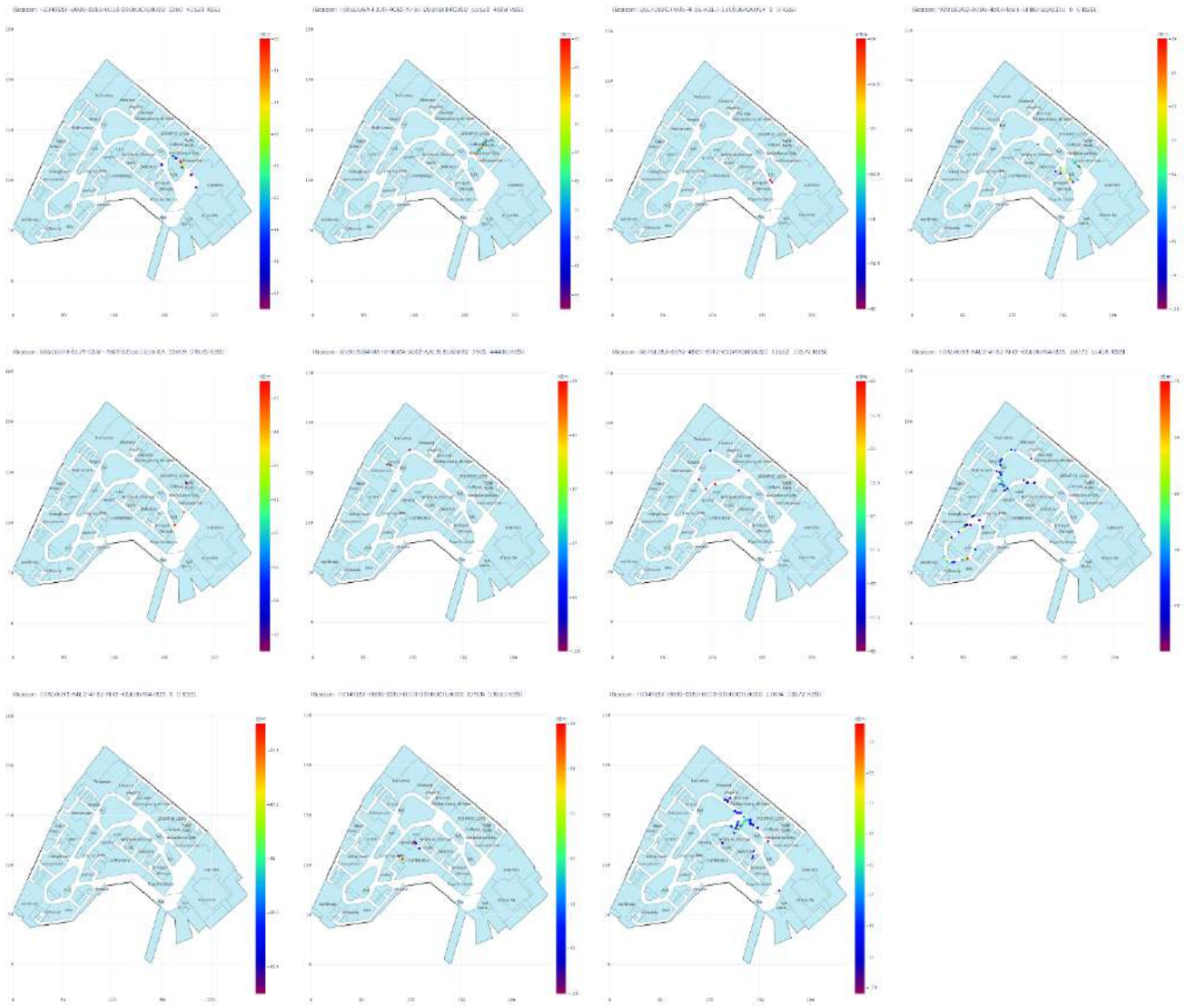




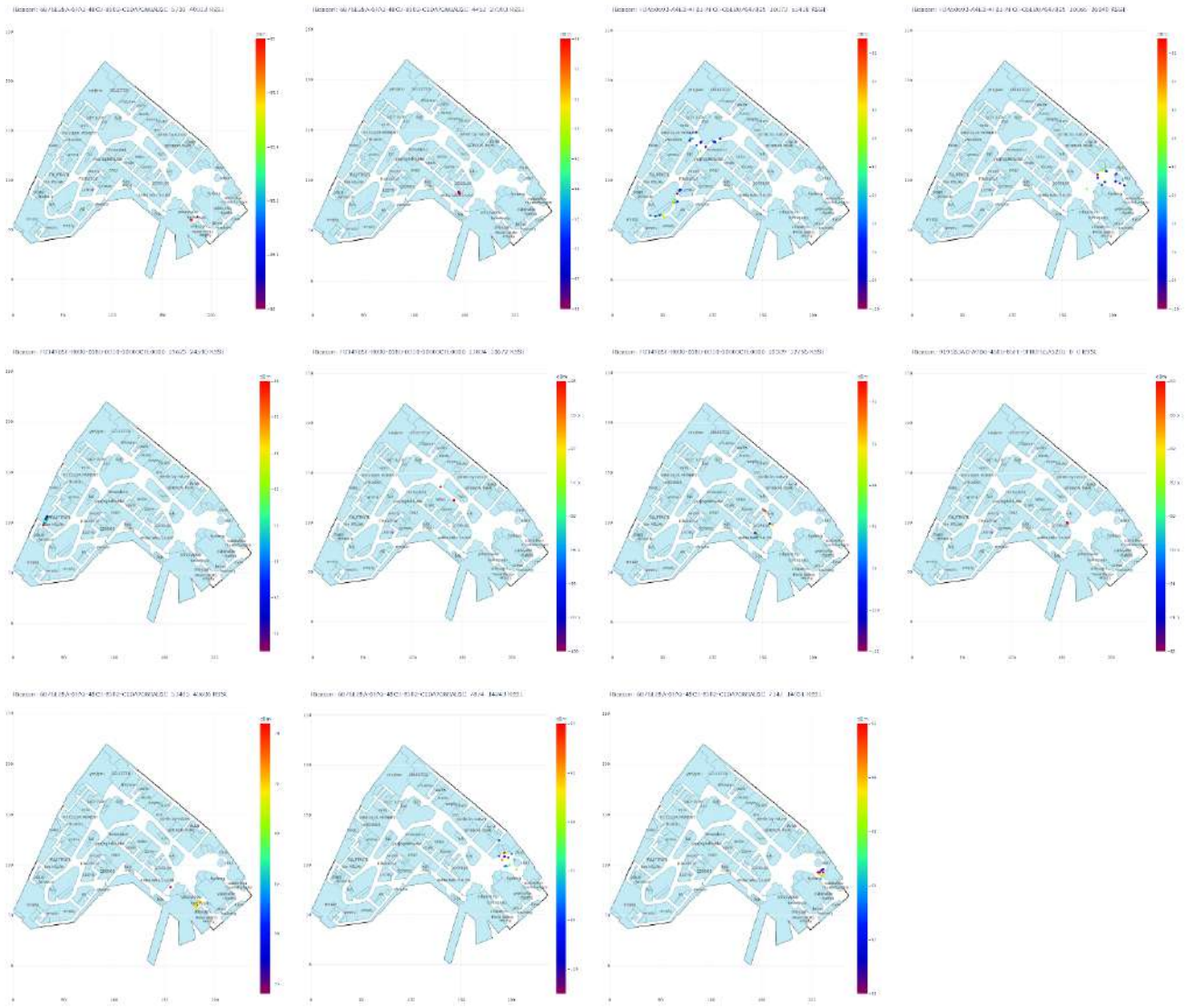
Site 2, F2



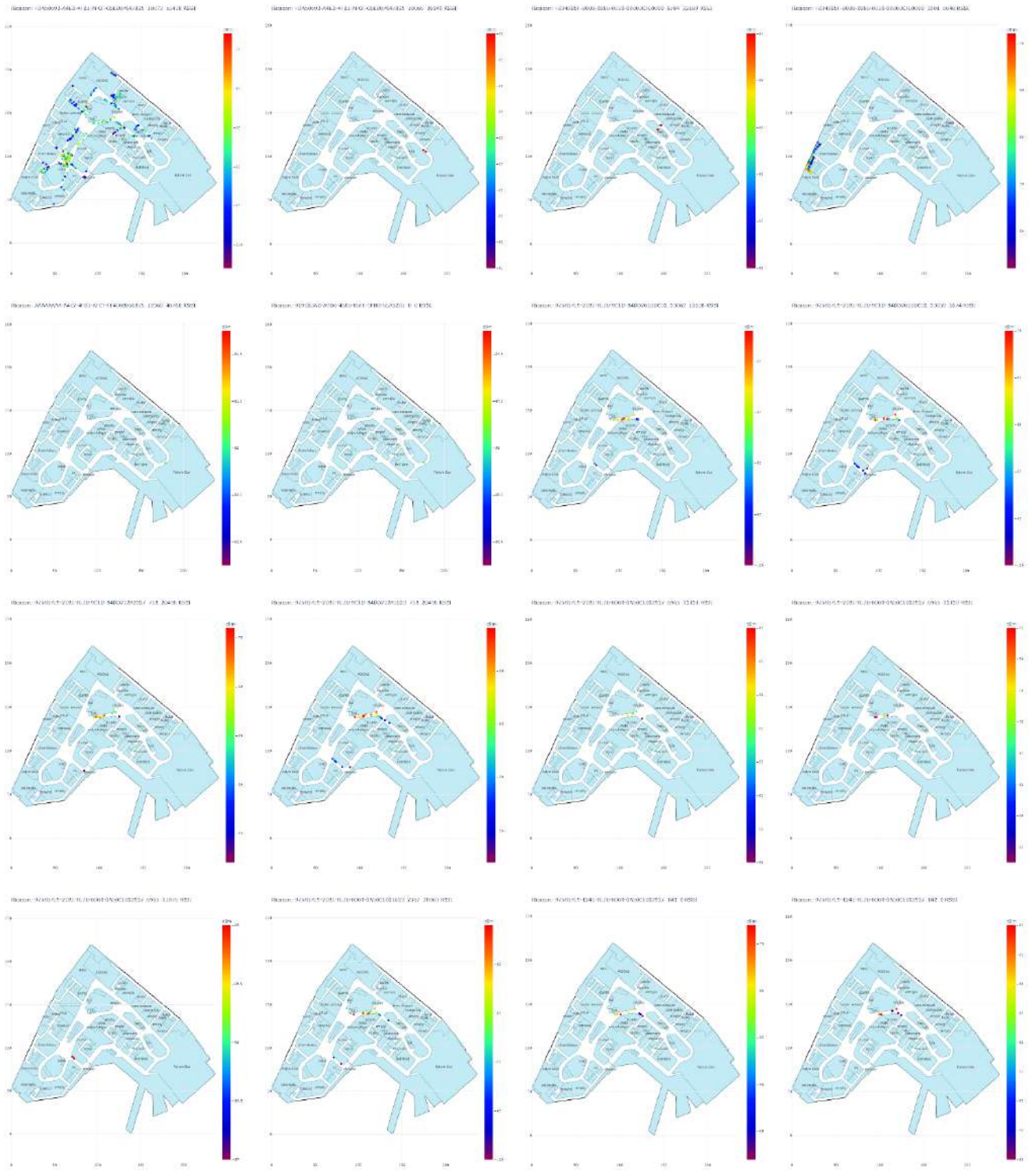
Site 2, F3

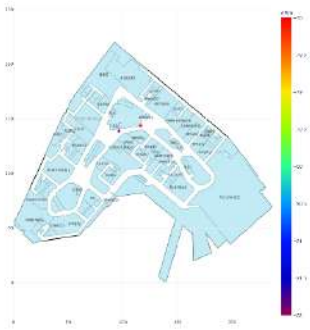
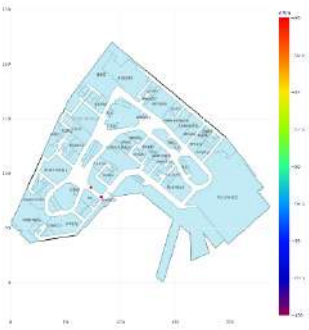
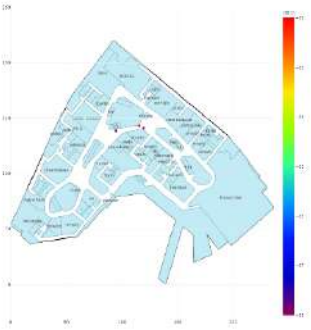
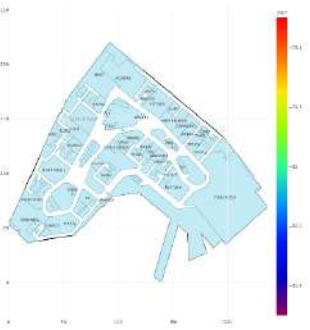
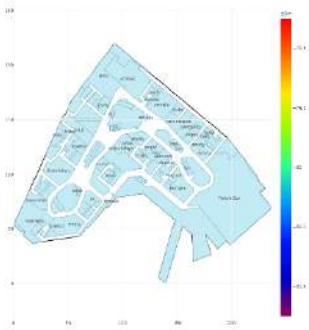
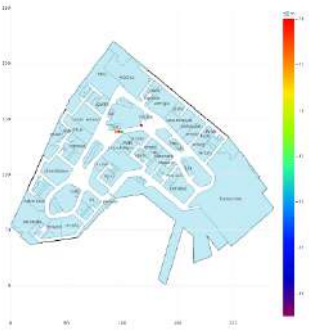
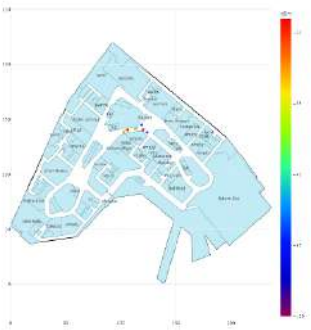
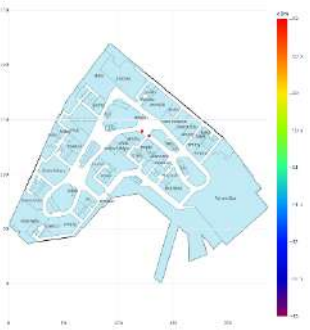
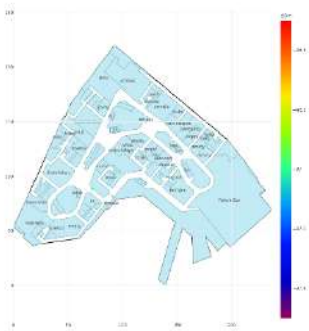
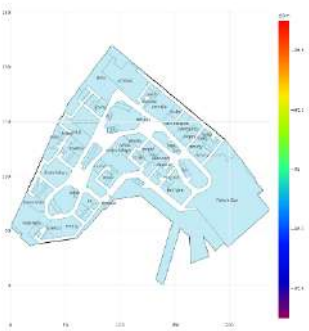
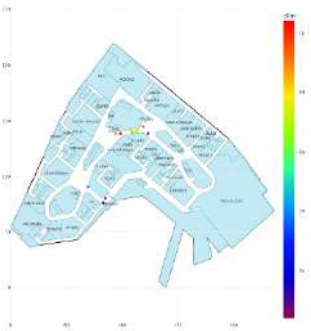
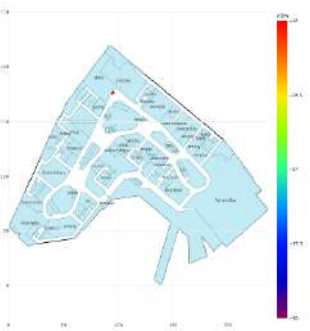
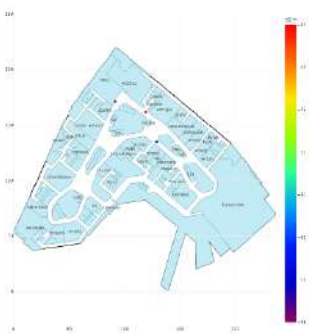
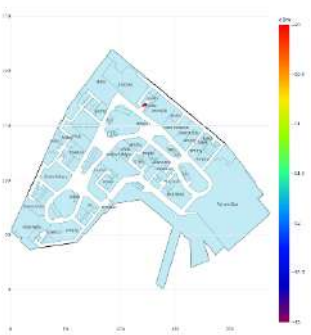


Site 2, F4

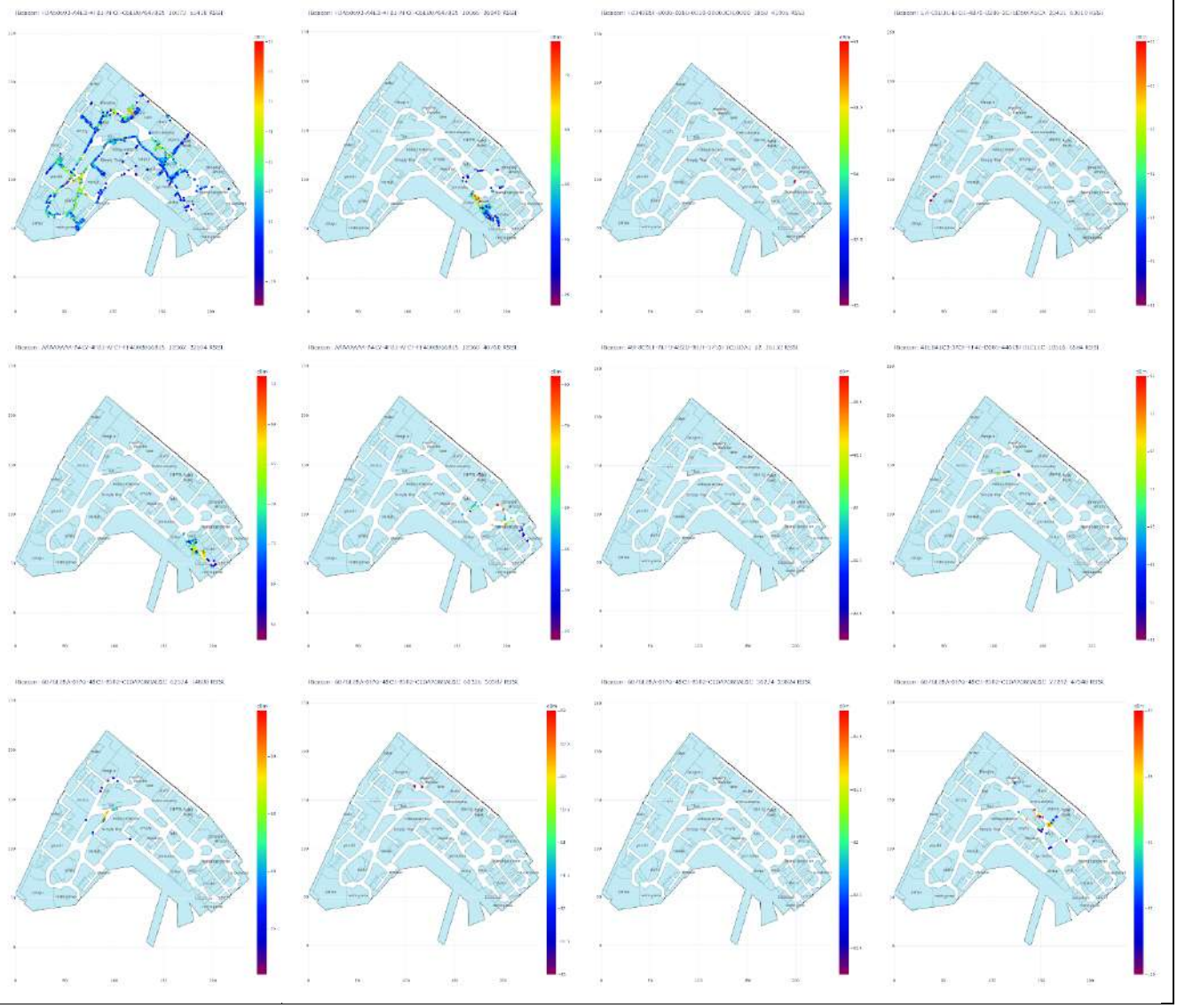


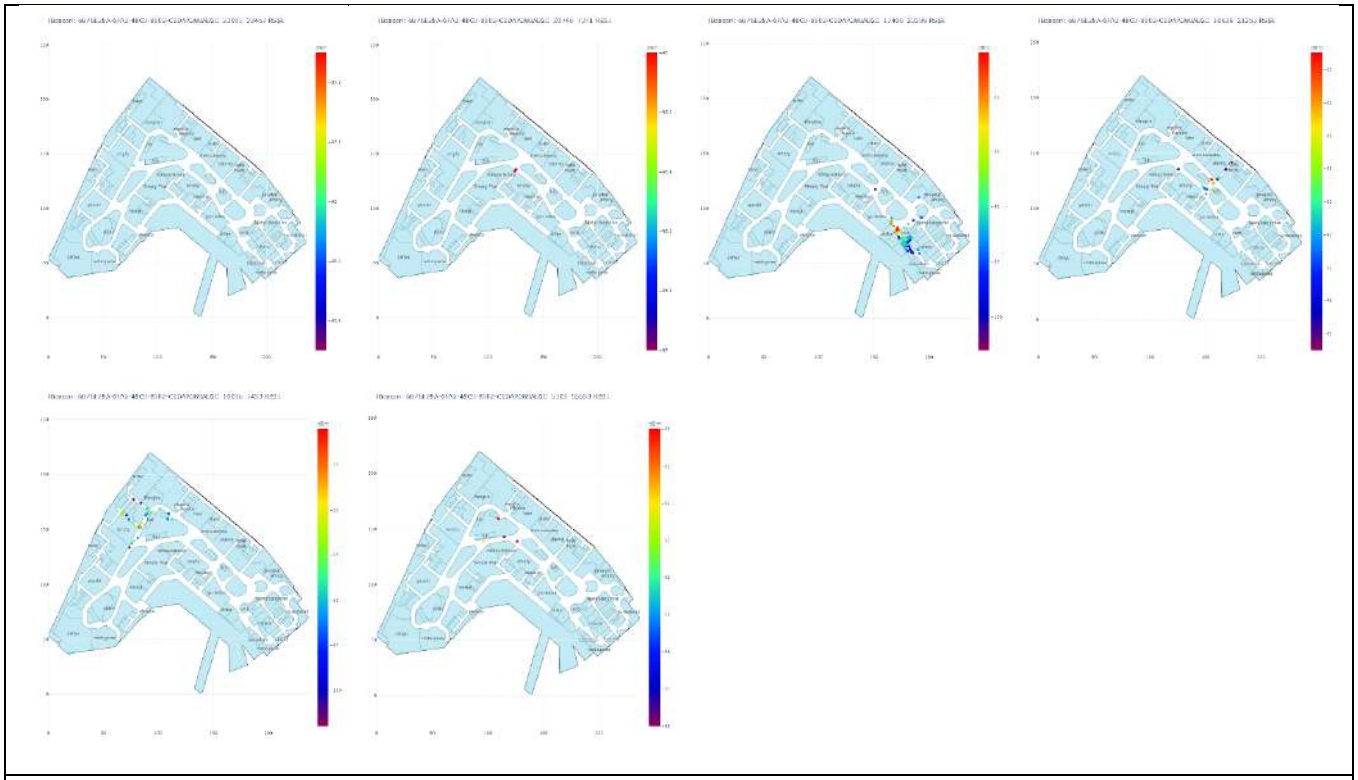
Site 2, F5



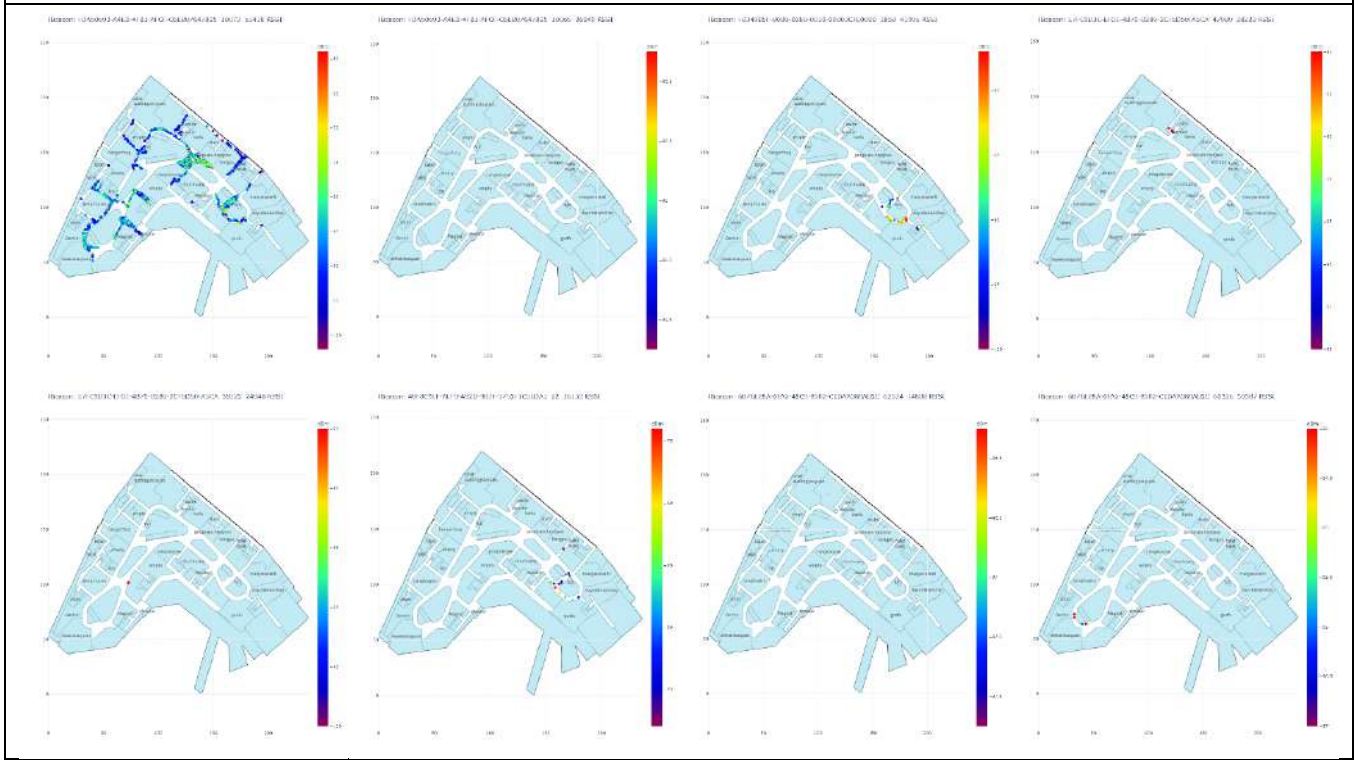















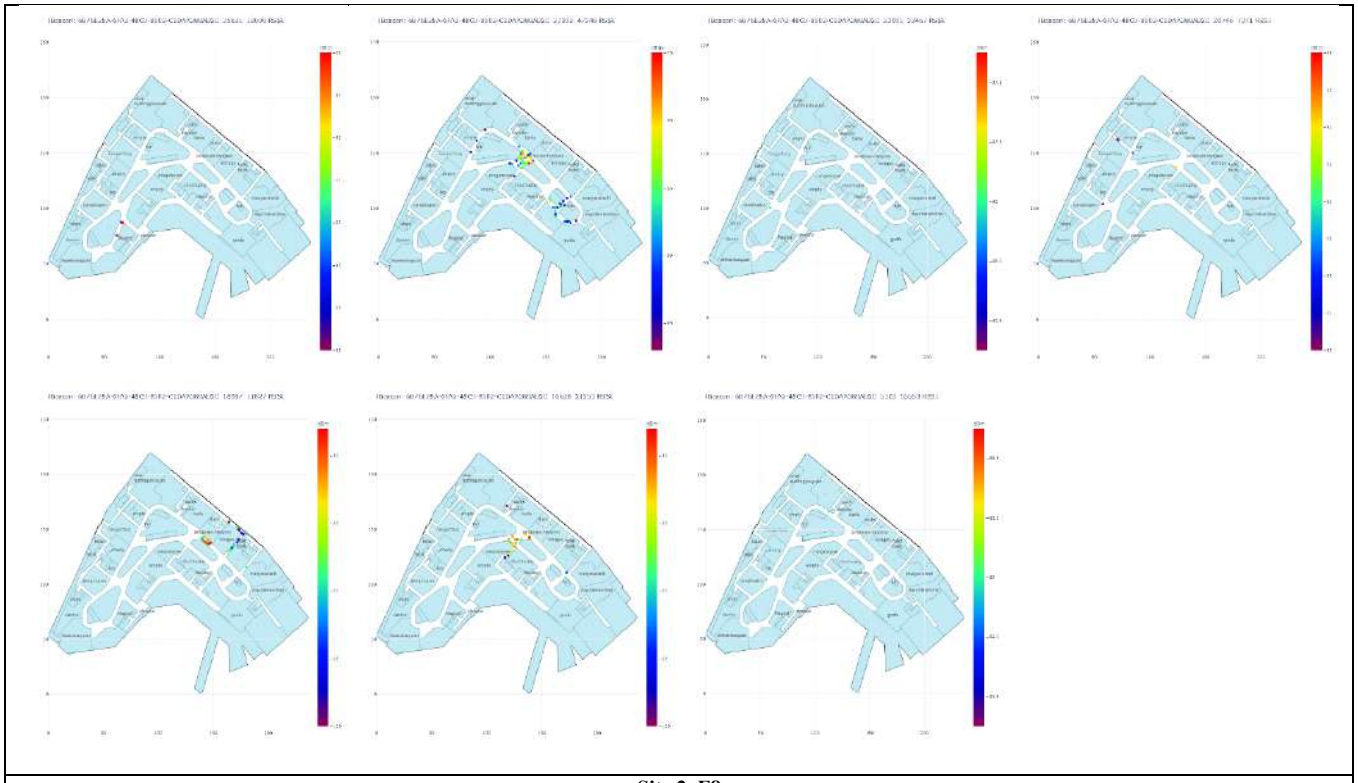
Site 2, F6



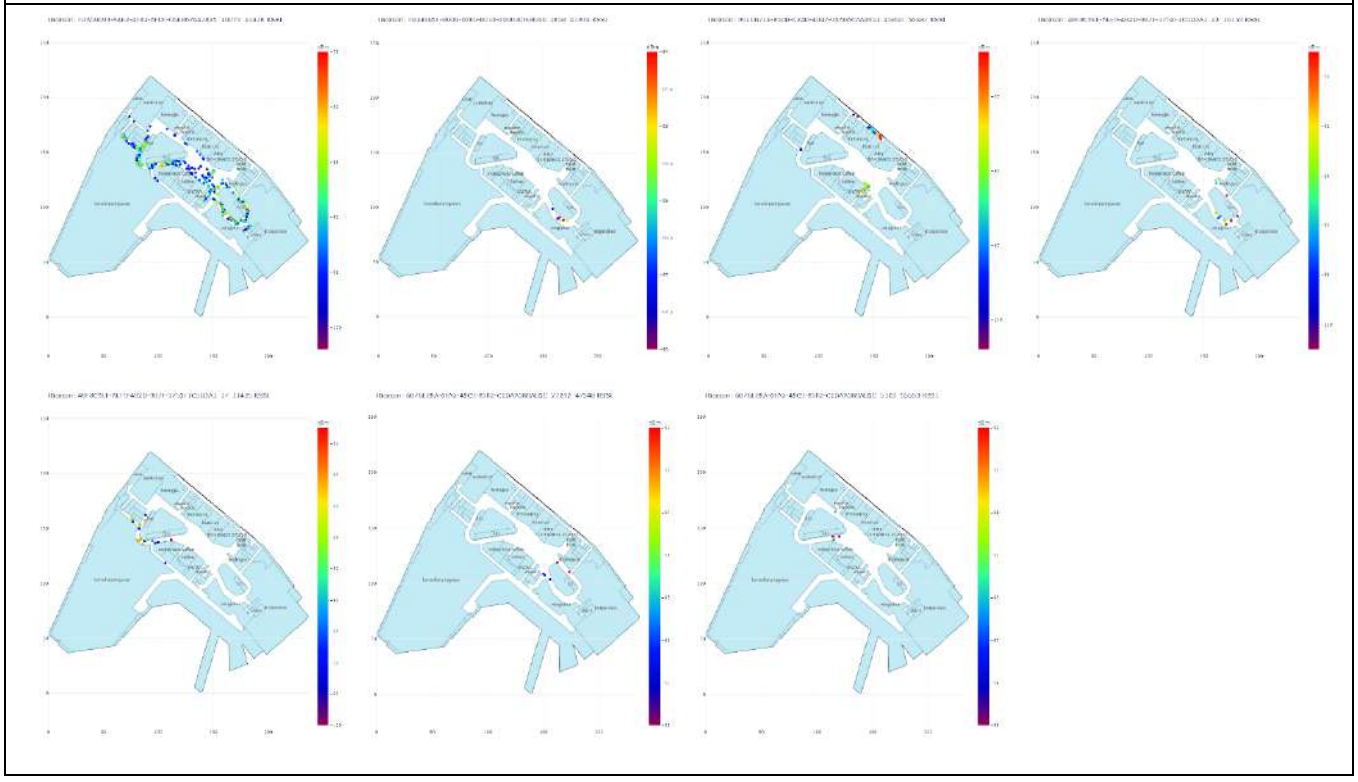


Site 2, F7





Site 2, F8



APPENDIX E – VISUALIZATION OF WAYPOINTS (SOURCE CODE)

```
# this cell contains the content from main.py
import json
import os
from pathlib import Path


import numpy as np

## commented out because already included in previous cells
# from compute_f import split_ts_seq, compute_step_positions
# from io_f import read_data_file
# from visualize_f import visualize_trajectory, visualize_heatmap, save_figure_to_html

floor_data_dir = './data/site1/F1'
bl_floor = './data/site1/B1'
f2_floor = './data/site1/F2'
f3_floor = './data/site1/F3'
f4_floor = './data/site1/F4'

s2_b1_floor = './data/site2/B1'
s2_f1_floor = './data/site2/F1'
s2_f2_floor = './data/site2/F2'
s2_f3_floor = './data/site2/F3'
s2_f4_floor = './data/site2/F4'
s2_f5_floor = './data/site2/F5'
s2_f6_floor = './data/site2/F6'
s2_f7_floor = './data/site2/F7'
s2_f8_floor = './data/site2/F8'

path_data_dir_b1 = bl_floor + '/path_data_files'
path_data_dir_f1 = floor_data_dir + '/path_data_files'
path_data_dir_f2 = f2_floor + '/path_data_files'
path_data_dir_f3 = f3_floor + '/path_data_files'
path_data_dir_f4 = f4_floor + '/path_data_files'
path_data_dir_s2b1 = s2_b1_floor + '/path_data_files'
path_data_dir_s2f1 = s2_f1_floor + '/path_data_files'
path_data_dir_s2f2 = s2_f2_floor + '/path_data_files'
path_data_dir_s2f3 = s2_f3_floor + '/path_data_files'
path_data_dir_s2f4 = s2_f4_floor + '/path_data_files'
path_data_dir_s2f5 = s2_f5_floor + '/path_data_files'
path_data_dir_s2f6 = s2_f6_floor + '/path_data_files'
path_data_dir_s2f7 = s2_f7_floor + '/path_data_files'
path_data_dir_s2f8 = s2_f8_floor + '/path_data_files'

floor_plan_filename = s2_f8_floor + '/floor_image.png'
floor_info_filename = s2_f8_floor + '/floor_info.json'

save_dir = './output/site1/B1'
path_image_save_dir = save_dir + '/path_images'
```

```
step_position_image_save_dir = save_dir
magn_image_save_dir = save_dir
wifi_image_save_dir = save_dir + '/wifi_images'
ibeacon_image_save_dir = save_dir + '/ibeacon_images'
wifi_count_image_save_dir = save_dir


Path(path_image_save_dir).mkdir(parents=True, exist_ok=True)
Path(magn_image_save_dir).mkdir(parents=True, exist_ok=True)
Path(wifi_image_save_dir).mkdir(parents=True, exist_ok=True)
Path(ibeacon_image_save_dir).mkdir(parents=True, exist_ok=True)

with open(floor_info_filename) as f:
    floor_info = json.load(f)
width_meter = floor_info["map_info"]["width"]
height_meter = floor_info["map_info"]["height"]

path_filenames_b1 = list(Path(path_data_dir_b1).resolve().glob("*.txt"))
path_filenames_f1 = list(Path(path_data_dir_f1).resolve().glob("*.txt"))
path_filenames_f2 = list(Path(path_data_dir_f2).resolve().glob("*.txt"))
path_filenames_f3 = list(Path(path_data_dir_f3).resolve().glob("*.txt"))
path_filenames_f4 = list(Path(path_data_dir_f4).resolve().glob("*.txt"))
path_filenames_s2b1 = list(Path(path_data_dir_s2b1).resolve().glob("*.txt"))
path_filenames_s2f1 = list(Path(path_data_dir_s2f1).resolve().glob("*.txt"))
path_filenames_s2f2 = list(Path(path_data_dir_s2f2).resolve().glob("*.txt"))
path_filenames_s2f3 = list(Path(path_data_dir_s2f3).resolve().glob("*.txt"))
path_filenames_s2f4 = list(Path(path_data_dir_s2f4).resolve().glob("*.txt"))
path_filenames_s2f5 = list(Path(path_data_dir_s2f5).resolve().glob("*.txt"))
path_filenames_s2f6 = list(Path(path_data_dir_s2f6).resolve().glob("*.txt"))
path_filenames_s2f7 = list(Path(path_data_dir_s2f7).resolve().glob("*.txt"))
path_filenames_s2f8 = list(Path(path_data_dir_s2f8).resolve().glob("*.txt"))

# this cell contains the content from visualize_f.py
import plotly.graph_objs as go
from PIL import Image


def save_figure_to_html(fig, filename):
    fig.write_html(filename)


def visualize_trajectory(trajectory, floor_plan_filename,width_meter, height_meter,fig, title=None, mode='lines +
markers + text', show=False):

    # add trajectory
    size_list = [6] * trajectory.shape[0]
    size_list[0] = 10
    size_list[-1] = 10

    color_list = ['rgba(4, 174, 4, 0.5)'] * trajectory.shape[0]
```

```
color_list[0] = 'rgba(12, 5, 235, 1)'
color_list[-1] = 'rgba(235, 5, 5, 1)'

position_count = {}
text_list = []
for i in range(trajectory.shape[0]):
    if str(trajectory[i]) in position_count:
        position_count[str(trajectory[i])] += 1
    else:
        position_count[str(trajectory[i])] = 0
    text_list.append('        ' * position_count[str(trajectory[i])] + f'{i}')
text_list[0] = 'Start Point: 0'
text_list[-1] = f'End Point: {trajectory.shape[0] - 1}'

fig.add_trace(
    go.Scattergl(
        x=trajectory[:, 0],
        y=trajectory[:, 1],
        mode=mode,
        marker=dict(size=size_list, color=color_list),
        line=dict(shape='linear', color='rgb(100, 10, 100)', width=2, dash='dot'),
        text=text_list,
        textposition="top center",
        name='trajectory',
    ))

# add floor plan
floor_plan = Image.open(floor_plan_filename)
fig.update_layout(images=[
    go.layout.Image(
        source=floor_plan,
        xref="x",
        yref="y",
        x=0,
        y=height_meter,
        sizex=width_meter,
        sizey=height_meter,
        sizing="contain",
        opacity=1,
        layer="below",
    )
])

# configure
fig.update_xaxes(autorange=False, range=[0, width_meter])
fig.update_yaxes(autorange=False, range=[0, height_meter], scaleanchor="x", scaleratio=1)
fig.update_layout(
    title=go.layout.Title(
        text=title or "No title.",
        xref="paper",
```

```
            x=0,
        ),
        autosize=True,
        width=900,
        height=200 + 900 * height_meter / width_meter,
        template="plotly_white",
    )

    if show:
        fig.show()

    return fig

# 1. visualize ground truth positions
print('Visualizing ground truth positions...')
i=0
# print(len(path_filenames))
fig = go.Figure()

for path_filename in path_filenames_b1:
    i+=1
    # print(f'Processing file: {path_filename}...')
    path_data = read_data_file(path_filename)
    path_id = path_filename.name.split(".")[0]
    visualize_trajectory(path_data.waypoint[:, 1:3], floor_plan_filename,width_meter, height_meter,
fig,title=path_id, show=False)
    # if i ==len(path_filenames_b1):
    if i ==len(path_filenames_b1):
      fig = visualize_trajectory(path_data.waypoint[:, 1:3],floor_plan_filename, width_meter,height_meter,fig,
title=path_id, show=True)

      break
    html_filename = f'{path_image_save_dir}/{path_id}.html'
    html_filename = str(Path(html_filename).resolve())
    save_figure_to_html(fig, html_filename)
```

APPENDIX F – VISUALIZATION OF GEOMAGNETIC FIELD STRENGTH (SOURCE CODE)

```
import base64
import json
import os
from pathlib import Path
from typing import Tuple

import numpy as np
import pandas as pd
import plotly.graph_objects as go

from compute_f import compute_step_positions

def read_data_file(data_filename: str, verbose: bool = False
                   ) -> Tuple[np.array, np.array, np.array, np.array]:
    """Function to read txt file provided and extract 4 key pieces of information.

    Args:
        data_filename: path to file holding raw data
        verbose: flag on whether to print helper messages

    Returns:
        A tuple (acce_data, magn_data, ahrs_data, wayp_data) where:
        - acce_data: ` timestamp | x | y | z ` data of `TYPE_ACCELEROMETER`
        - magn_data: ` timestamp | x | y | z ` data of `TYPE_MAGNETIC_FIELD`
        - ahrs_data: ` timestamp | x | y | z ` data of `TYPE_ROTATION_VECTOR`
        - wayp_data: ` timestamp | x | y ` data of `TYPE_WAYPOINT`
    """
    with open(data_filename, 'r', encoding='utf-8') as file:
        lines = file.readlines()

    acce_data = []
    magn_data = []
    ahrs_data = []
    wayp_data = []

    for line_data in lines:
        line_data = line_data.strip()
        if not line_data or line_data[0] == '#':
            continue

        line_data = line_data.split('\t')

        if line_data[1] == 'TYPE_ACCELEROMETER':
            # timestamp | x | y | z
            acce_data.append(
                [int(line_data[0]), float(line_data[2]), float(line_data[3]), float(line_data[4])])
            continue

        if line_data[1] == 'TYPE_MAGNETIC_FIELD':
            # timestamp | x | y | z
            magn_data.append(
                [int(line_data[0]), float(line_data[2]), float(line_data[3]), float(line_data[4])])
            continue

        if line_data[1] == 'TYPE_ROTATION_VECTOR':
            # timestamp | x | y | z
            ahrs_data.append(
                [int(line_data[0]), float(line_data[2]), float(line_data[3]), float(line_data[4])])
            continue

        if line_data[1] == 'TYPE_WAYPOINT':
            # timestamp | x | y
            wayp_data.append(
                [int(line_data[0]), float(line_data[2]), float(line_data[3])])

    if verbose:
        print(f"# of accelerometer data: {len(acce_data)}")
        print(f"# of geomagnetic data  : {len(magn_data)}")
        print(f"# of rotation vect data: {len(ahrs_data)}")
        print(f"# of waypoints data    : {len(wayp_data)}")

    acce_data = np.array(acce_data)
    magn_data = np.array(magn_data)
    ahrs_data = np.array(ahrs_data)
```

```
    wayp_data = np.array(wayp_data)

    return acce_data, magn_data, ahrs_data, wayp_data

def get_nearest_position_to_magn_data(step_pos: np.array, magn_data: np.array) -> pd.DataFrame:
    """Maps magnetic strength data to its nearest position based on timestamp values.

    Args:
        step_pos: maps timestamp to x,y coordinates
        magn_data: maps timestamp to x,y,z magnetic values

    Returns:
        A pandas dataframe holding mapping of step position coordinates to magnetic strength values
    """
    # Convert to pandas dataframe
    magn_data_df = pd.DataFrame(data=magn_data, columns=['timestamp', 'x', 'y', 'z'])
    step_pos_df = pd.DataFrame(data=step_pos, columns=['timestamp', 'x', 'y'])

    # Ensure no duplicated timestamp
    magn_data_df = magn_data_df.sort_values(by=['timestamp']).drop_duplicates(keep='first')

    # Get join both dataframes based on nearest timestamp
    mag_pos_datas = pd.merge_asof(
        magn_data_df, step_pos_df, on="timestamp", direction='nearest', suffixes=('', '_pos'))

    return mag_pos_datas

def calculate_magnetic_strength(mag_pos_datas: pd.DataFrame) -> pd.DataFrame:
    """Calculates average magnetic strength at each position.

    Args:
        mag_pos_datas: holds magnetic strength readings for each positions.

    Returns:
        A pandas dataframe holding the average magnetic strength reading for each position
    """
    mag_pos_datas['magn_strength'] = (
        mag_pos_datas['x'] ** 2 + mag_pos_datas['y'] ** 2 + mag_pos_datas['z']**2)
    mag_pos_datas['magn_strength'] = np.sqrt(mag_pos_datas['magn_strength'])

    avg_magn_strength = mag_pos_datas.groupby(['x_pos', 'y_pos'], as_index=False)['magn_strength'].mean()
    return avg_magn_strength

def plot_and_save_geomagnetic_heatmap(mag_strength_pos_data: pd.DataFrame, path_dir: str, site_id: int,
                                      floor_id: str, augment: bool = True, save_img: bool = False):
    """Plot the average geomagnetic stregnth as a heatmap, overlaid on the site floorplan.

    Args:
        mag_strength_pos_data: holds the average magnetic strength reading for each position
        path_dir: directory to read floorplan and metadata from
        site_id: ID of site
        floor_id: floor level
        augment: flag on whether augmented waypoints was used
        save_img: flag on whether to save the heatmap as a png file
    """
    # Get data for plotting
    x_pos = mag_strength_pos_data['x_pos'].values
    y_pos = mag_strength_pos_data['y_pos'].values
    strength = mag_strength_pos_data['magn_strength'].values

    data = []
    # Plot heatmap
    data.append(go.Scatter(
        x=x_pos, y=y_pos,
        mode='markers',
        marker=dict(
            color=strength,
            colorbar=dict(title="µT"),
            colorscale="Rainbow",
            cmin=0,
            cmax=100,
            size=5,
        )
    ))

    # Add floor plan
```

```
    with open(os.path.join(path_dir, 'floor_info.json')) as f:
        floor_info = json.load(f)
    height_meter = floor_info['map_info']['height']
    width_meter = floor_info['map_info']['width']
    with open(os.path.join(path_dir, 'floor_image.png'), 'rb') as image_file:
        floor_plan = base64.b64encode(image_file.read()).decode('utf-8')
    img_bg = {
        'source': f'{floor_plan}',
        'xref': 'x', 'yref': 'y',
        'x': 0, 'y': height_meter,
        'sizex': width_meter, 'sizey': height_meter,
        'sizing': 'contain', 'opacity': 1, 'layer': "below",
    }

    layout = go.Layout(
        xaxis=dict(autorange=False, range=[0, width_meter], showticklabels=False, showgrid=False, zeroline=False),
        yaxis=dict(autorange=False, range=[0, height_meter], showticklabels=False, showgrid=False, zeroline=False,
                   scaleanchor='x', scaleratio=1),
        autosize=True,
        width=900,
        height=100 + 900 * height_meter / width_meter,
        template='plotly_white',
        images=[img_bg]
    )

    fig = go.Figure(data=data, layout=layout)
    # fig.show()

    if save_img:
        output_filepath = f"./outputs/site{site_id}"
        Path(output_filepath).mkdir(parents=True, exist_ok=True)
        file_name = f'{floor_id}.png' if augment else f'{floor_id}_unaugmented.png'
        fig.write_image(os.path.join(output_filepath, file_name))

def get_geomagnetic_heatmap(site_id: int, floor_id: str, augment: bool = True, save_img: bool = False):
    """Function to get a heatmap of the average geomagnetic strength of a given floor in a given site.

    Args:
        site_id: ID of site
        floor_id: floor level
        augment: flag on whether to add extra waypoints
        save_img: flag on whether to save the heatmap as a png file
    """
    path_dir = f'./indoor-location-competition-20/data/site{site_id}/{floor_id}'
    data_files_path_dir = os.path.join(path_dir, 'path_data_files')
    data_filenames = [f for f in os.listdir(data_files_path_dir)
                      if os.path.isfile(os.path.join(data_files_path_dir, f))]

    mag_pos_datas = []
    for data_filename in data_filenames:
        data_filename = os.path.join(data_files_path_dir, data_filename)
        acce_data, magn_data, ahrs_data, wayp_data = read_data_file(data_filename)
        if augment:
            wayp_data = compute_step_positions(acce_data, ahrs_data, wayp_data)
        mag_pos_datas.append(get_nearest_position_to_magn_data(wayp_data, magn_data))

    mag_pos_data_df = pd.concat(mag_pos_datas, ignore_index=True)

    mag_strength_pos_data = calculate_magnetic_strength(mag_pos_data_df)

    plot_and_save_geomagnetic_heatmap(mag_strength_pos_data, path_dir, site_id, floor_id, augment, save_img)

if __name__ == "__main__":
    places_to_map = {
        1: ['B1', 'F1', 'F2', 'F3', 'F4'],
        2: ['B1', 'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8'],
    }

    for site_id, floor_ids in places_to_map.items():
        for floor_id in floor_ids:
            get_geomagnetic_heatmap(site_id, floor_id, augment=True, save_img=True)
            print(f"Site {site_id} {floor_id} done!")
```

APPENDIX G – VISUALIZATION OF WIFI RSS (SOURCE CODE)

```
import json
import os
from dataclasses import dataclass
from pathlib import Path
import numpy as np
import plotly.graph_objs as go
from PIL import Image

from compute_f import split_ts_seq, compute_step_positions

@dataclass
class ReadData:
    acce: np.ndarray
    ahrs: np.ndarray
    wifi: np.ndarray
    waypoint: np.ndarray

def read_data_file(data_filename):
    acce = []
    ahrs = []
    wifi = []
    waypoint = []

    with open(data_filename, 'r', encoding='utf-8') as file:
        lines = file.readlines()

    for line_data in lines:
        line_data = line_data.strip()
        if not line_data or line_data[0] == '#':
            continue

        line_data = line_data.split('\t')

        if line_data[1] == 'TYPE_ACCELEROMETER':
            acce.append([int(line_data[0]), float(line_data[2]), float(line_data[3]), float(line_data[4])])
            # timestamp | x | y | z
            continue

        if line_data[1] == 'TYPE_ROTATION_VECTOR':
            ahrs.append([int(line_data[0]), float(line_data[2]), float(line_data[3]), float(line_data[4])])
            # timestamp | x | y | z
            continue

        if line_data[1] == 'TYPE_WIFI':
            sys_ts = line_data[0]
            ssid = line_data[2]
            bssid = line_data[3]
            rssi = line_data[4]
            lastseen_ts = line_data[6]
            wifi_data = [sys_ts, ssid, bssid, rssi, lastseen_ts]
            wifi.append(wifi_data)
            # timestamp | ssid (wifi name) | bssid (wifi AP address) | RSSI | last seen timestamp
            continue

        if line_data[1] == 'TYPE_WAYPOINT':
            waypoint.append([int(line_data[0]), float(line_data[2]), float(line_data[3])])
            # timestamp | x location | y location

    acce = np.array(acce)
    ahrs = np.array(ahrs)
    wifi = np.array(wifi)
    waypoint = np.array(waypoint)

    return ReadData(acce, ahrs, wifi, waypoint)


def get_wifis_by_position(path_file_list):
    pos_wifi_datas = {}
    # keys : xy positions based on step
    # values : wifi data
    for path_filename in path_file_list:
      print(f'Processing {path_filename}...')
      path_datas = read_data_file(path_filename)
      acce_datas = path_datas.acce
```

```
      ahrs_datas = path_datas.ahrs
      wifi_datas = path_datas.wifi
      posi_datas = path_datas.waypoint
      step_positions = compute_step_positions(acce_datas, ahrs_datas, posi_datas)
      # timestamp | x location | y location
      if wifi_datas.size != 0:
        sep_tss = np.unique(wifi_datas[:, 0].astype(float))
        wifi_datas_list = split_ts_seq(wifi_datas, sep_tss)
        for wifi_ds in wifi_datas_list:
          diff = np.abs(step_positions[:, 0] - float(wifi_ds[0, 0]))
          index = np.argmin(diff)
          target_xy_key = tuple(step_positions[index, 1:3])
          if target_xy_key in pos_wifi_datas:
              pos_wifi_datas[target_xy_key] = np.append(pos_wifi_datas[target_xy_key], wifi_ds, axis=0)
          else:
              pos_wifi_datas[target_xy_key] = wifi_ds

    return pos_wifi_datas


def extract_wifi_rssi(pos_wifi_datas):
    wifi_rssi = {}
    # keys : bssid
    # nested keys : x location, y location
    # nested value : rssi
    for position_key in pos_wifi_datas:
        wifi_data = pos_wifi_datas[position_key]
        for wifi_d in wifi_data:
            bssid = wifi_d[2]
            rssi = int(wifi_d[3])
            if bssid in wifi_rssi:
                position_rssi = wifi_rssi[bssid]
                if position_key in position_rssi:
                    old_rssi = position_rssi[position_key][0]
                    old_count = position_rssi[position_key][1]
                    position_rssi[position_key][0] = (old_rssi * old_count + rssi) / (old_count + 1)
                    position_rssi[position_key][1] = old_count + 1
                else:
                    position_rssi[position_key] = np.array([rssi, 1])
            else:
                position_rssi = {}
                position_rssi[position_key] = np.array([rssi, 1])
            wifi_rssi[bssid] = position_rssi

    return wifi_rssi


def visualize_heatmap(position, value, floor_plan_filename, width_meter, height_meter, title, show=False):
    fig = go.Figure()
    # add heat map
    fig.add_trace(
        go.Scatter(x=position[:, 0],
                   y=position[:, 1],
                   mode='markers',
                   marker=dict(size=7,
                               color=value,
                               colorbar=dict(title="dBm"),
                               colorscale="Rainbow"),
                   text=value,
                   name=title))
    # add floor plan
    floor_plan = Image.open(floor_plan_filename)
    fig.update_layout(images=[
        go.layout.Image(
            source=floor_plan,
            xref="x",
            yref="y",
            x=0,
            y=height_meter,
            sizex=width_meter,
            sizey=height_meter,
            sizing="contain",
            opacity=1,
            layer="below",
        )
    ])
```

```
    # configure
    fig.update_xaxes(autorange=False, range=[0, width_meter])
    fig.update_yaxes(autorange=False, range=[0, height_meter], scaleanchor="x", scaleratio=1)
    fig.update_layout(
        title=go.layout.Title(
            text=title or "No title.",
            xref="paper",
            x=0,
        ),
        autosize=True,
        width=900,
        height=200 + 900 * height_meter / width_meter,
        template="plotly_white",
    )

    if show:
        fig.show()

    return fig


def save_figure_as_png(fig, filename):
    fig.write_image(filename)


if __name__ == "__main__":
  data_dir = {
      'site1': ['B1', 'F1', 'F2', 'F3', 'F4'],
      'site2': ['B1', 'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8']
  }

  for site_dir, floor_dirs in data_dir.items():
    for floor_dir in floor_dirs:

      # get floor data directory, floor plan image and info, and create output save directory
      floor_data_dir = './data/' + site_dir + '/' + floor_dir
      path_data_dir = floor_data_dir + '/path_data_files'
      floor_plan_filename = floor_data_dir + '/floor_image.png'
      floor_info_filename = floor_data_dir + '/floor_info.json'
      wifi_image_save_dir = './output/' + site_dir + '/' + floor_dir
      Path(wifi_image_save_dir).mkdir(parents=True, exist_ok=True)

      # get floor info
      with open(floor_info_filename) as f:
          floor_info = json.load(f)
      width_meter = floor_info["map_info"]["width"]
      height_meter = floor_info["map_info"]["height"]

      # get wifi data based on step positions
      path_filenames = list(Path(path_data_dir).resolve().glob("*.txt"))
      pos_wifi_datas = get_wifis_by_position(path_filenames)
      step_positions = np.array(list(pos_wifi_datas.keys()))
      wifi_rssi = extract_wifi_rssi(pos_wifi_datas)

      # choose random 3 wifi AP to see RSSI fingerprints
      rng = np.random.default_rng()
      for i in range(3):
        target_wifi_index = rng.integers(low=0, high=len(wifi_rssi.keys()))
        target_wifi = list(wifi_rssi.keys())[target_wifi_index]

        # get all step locations (fingerprints) that sees the chosen wifi AP
        heat_positions = np.array(list(wifi_rssi[target_wifi].keys()))

        # get corresponding RSSI value
        heat_values = np.array(list(wifi_rssi[target_wifi].values()))[:, 0]
        fig = visualize_heatmap(heat_positions, heat_values, floor_plan_filename, width_meter, height_meter,
title=f'{site_dir}, {floor_dir}, Wifi AP {i+1}: {target_wifi} RSSI', show=True)

        # save fig as png file
        png_filename = f'{wifi_image_save_dir}/{target_wifi.replace(":", "-")}.png'
        png_filename = str(Path(png_filename).resolve())
        save_figure_as_png(fig, png_filename)
```

APPENDIX H – VISUALIZATION OF IBEACON RSS (SOURCE CODE)

```
import json
import os
from pathlib import Path
import plotly.graph_objs as go
from PIL import Image


import numpy as np
from compute_f import split_ts_seq, compute_step_positions

def parse_file_ibeacon(path_data_files):
        ibeacons = []
        acceleration = []
        positions = []
        ahrs = []
        with open(path_data_files, 'r', encoding='utf-8') as file:
                lines = file.readlines()
        for line in lines:
                line = line.strip()
                if line[0] == "#" or not line:
                        continue
                line = line.split("\t")
                if line[1] == "TYPE_ACCELEROMETER":
                        acceleration.append([int(line[0]), float(line[2]), float(line[3]), float(line[4])])
                        continue
                if line[1] == "TYPE_BEACON":
                        ts = line[0]
                        uuid = line[2]
                        major = line[3]
                        minor = line[4]
                        rssi = line[6]
                        ibeacon_entry = [ts, f"{uuid}_{major}_{minor}", rssi]
                        ibeacons.append(ibeacon_entry)
                        continue
                if line[1] == "TYPE_WAYPOINT":
                        positions.append([int(line[0]), float(line[2]), float(line[3])])
                        continue
                if line[1] == "TYPE_ROTATION_VECTOR":
                        ahrs.append([int(line[0]), float(line[2]), float(line[3]), float(line[4])])
                        continue
        ibeacons = np.array(ibeacons)
        acceleration = np.array(acceleration)
        positions = np.array(positions)
        ahrs = np.array(ahrs)
        return {"ibeacon": ibeacons , "acce": acceleration , "waypoint": positions , "ahrs": ahrs}

def read_file_ibeacon(path_data_files):
        print(f"Readiing {path_data_files}...")
        path_datas = parse_file_ibeacon(path_data_files)
        ibeacon_datas = path_datas["ibeacon"]
        acce_datas = path_datas["acce"]
        ahrs_datas = path_datas["ahrs"]
        posi_datas = path_datas["waypoint"]
        return ibeacon_datas , acce_datas , ahrs_datas , posi_datas

def getibeacon_to_position(ibeacon_datas , acce_datas , ahrs_datas, posi_datas,  augment = False):
        ibeacon_extraction = {}
        step_positions = compute_step_positions(acce_datas, ahrs_datas, posi_datas)
        if augment == False:
                step_positions = posi_datas
        if ibeacon_datas.size > 0:
                ts_list = np.unique(ibeacon_datas[:, 0].astype(float))
                ibeacon_data_list = split_ts_seq(ibeacon_datas , ts_list)
                for ibeacon_data in ibeacon_data_list:
                        diff = np.abs(step_positions[:, 0] - float(ibeacon_data[0, 0]))
                        index = np.argmin(diff)
                        target_xy_key = tuple(step_positions[index, 1:3])
                        if target_xy_key in ibeacon_extraction:
                                ibeacon_extraction[target_xy_key] = np.append(ibeacon_extraction[target_xy_key],
ibeacon_data, axis=0)
                        else:
                                ibeacon_extraction[target_xy_key] = ibeacon_data
        return ibeacon_extraction
```

```
def getAxesValues(ibeacon_extraction):
        ibeacon_rssi = {}
        for key in ibeacon_extraction.keys():
                for ibeacon_d in ibeacon_extractions[key]:
                        ummid = ibeacon_d[1]
                        rssi = int(ibeacon_d[2])
                        if ummid in ibeacon_rssi:
                                position_rssi = ibeacon_rssi[ummid]
                                if key in position_rssi:
                                        old_rssi = position_rssi[key][0]
                                        old_count = position_rssi[key][1]
                                        position_rssi[key][0] = (old_rssi * old_count + rssi) / (old_count + 1)
                                        position_rssi[key][1] = old_count + 1
                                else:
                                        position_rssi[key] = np.array([rssi, 1])
                        else:
                                position_rssi = {}
                                position_rssi[key] = np.array([rssi, 1])
                        ibeacon_rssi[ummid] = position_rssi
        return ibeacon_rssi

def draw_heatmap(position , value , directory , width , height , title , filename):
        fig = go.Figure()
        # add heat map
        fig.add_trace(
                go.Scatter(x=position[:, 0],
                        y=position[:, 1],
                        mode='markers',
                        marker=dict(size=7,
                                color=value,
                                colorbar=dict(title="dBm"),
                                colorscale="Rainbow"),
                        text=value,
                        name=title))
        #add image
        floor_plan = Image.open(floor_plan_filename)
        fig.update_layout(images=[
                go.layout.Image(
                        source=floor_plan,
                        xref="x",
                        yref="y",
                        x=0,
                        y=height,
                        sizex=width,
                        sizey=height,
                        sizing="contain",
                        opacity=1,
                        layer="below",
                )
        ])
        #configure
        fig.update_xaxes(autorange=False, range=[0, width])
        fig.update_yaxes(autorange=False, range=[0, height], scaleanchor="x", scaleratio=1)
        fig.update_layout(
                title=go.layout.Title(
                        text=title or "No title.",
                        xref="paper",
                        x=0,
                ),
                autosize=True,
                width=900,
                height=200 + 900 * height / width,
                template="plotly_white",
        )
        fig.write_image(filename)
```

```
if __name__ == "__main__":
        floor_dirs = ["data/site1/B1" , "data/site1/F1" , "data/site1/F2" , "data/site1/F3" , "data/site1/F4",
                                "data/site2/B1" , "data/site2/F1" , "data/site2/F2" , "data/site2/F3" ,
"data/site2/F4" , "data/site2/F5",
                                "data/site2/F6" , "data/site2/F7" , "data/site2/F8"]
        for floor_dir in floor_dirs:
                floor_plan_filename = floor_dir + "/floor_image.png"
                floor_info_filename = floor_dir + "/floor_info.json"
                path_data_dir = floor_dir + "/path_data_files"
                output_dir = "output" + floor_dir.split("data")[1]+"/ibeacons"
                with open(floor_info_filename) as f:
                        floor_info = json.load(f)
                width_meter = floor_info["map_info"]["width"]
                height_meter = floor_info["map_info"]["height"]
                for data_file in os.listdir(path_data_dir):
                        ibeacon_datas , acce_datas , ahrs_datas , posi_datas =
read_file_ibeacon(os.path.join(os.path.dirname(os.path.realpath(__file__)) , path_data_dir , data_file))
                        ibeacon_extractions = getibeacon_to_position(ibeacon_datas , acce_datas , ahrs_datas ,
posi_datas)
                        ibeacon_rssi = getAxesValues(ibeacon_extractions)
                        print(f'This file has {len(ibeacon_rssi.keys())} ibeacons')
                        for target_ibeacon in ibeacon_rssi.keys():
                                heat_positions = np.array(list(ibeacon_rssi[target_ibeacon].keys()))
                                print(target_ibeacon)
                                print(heat_positions)
                                heat_values = np.array(list(ibeacon_rssi[target_ibeacon].values()))[:, 0]
                                output_filename = os.path.join(os.path.dirname(os.path.realpath(__file__)) ,
output_dir , f"./{target_ibeacon}.png")
                                Path(output_dir).mkdir(parents = True , exist_ok = True)
                                draw_heatmap(heat_positions, heat_values, floor_plan_filename, width_meter,
height_meter, f'iBeacon: {target_ibeacon} RSSI' , output_filename)
```

APPENDIX I – DEEP LEARNING MODEL BASED ON LINEAR REGRESSION (SOURCE CODE)

```
###### Linear  Regression ######

# -1) Get data
# 0) Prepare data
# 1) Design model (input, output size, forward pass)
# 2) Construct loss and optimizer
# 3) training loop
#     forward pass: compute prediction
#     backward pass: gradients
#     update weights

import numpy as np
import torch
import torch.nn as nn
import impyute as impy
import matplotlib.pyplot as plt
import Wifi_RSSI

## -1) Get data

# Choose site, and floor for model
site_dir = 'site1'
floor_dir = 'F1'
target_wifi = ['0a:74:9c:2d:06:cf', 'ec:56:23:f8:dc:74', '1e:74:9c:2b:3a:37']

# Get floor data directory, floor plan image and info, and create output save directory
floor_data_dir = './data/' + site_dir + '/' + floor_dir
path_data_dir = floor_data_dir + '/path_data_files'
floor_plan_filename = floor_data_dir + '/floor_image.png'
floor_info_filename = floor_data_dir + '/floor_info.json'
wifi_image_save_dir = './output/' + site_dir + '/' + floor_dir
Path(wifi_image_save_dir).mkdir(parents=True, exist_ok=True)

# Get floor info
with open(floor_info_filename) as f:
  floor_info = json.load(f)
width_meter = floor_info["map_info"]["width"]
height_meter = floor_info["map_info"]["height"]

# Get wifi data based on step positions
path_filenames = list(Path(path_data_dir).resolve().glob("*.txt"))
pos_wifi_datas = get_wifis_by_position(path_filenames)
step_positions = np.array(list(pos_wifi_datas.keys()))
wifi_rssi = extract_wifi_rssi(pos_wifi_datas)


## 0) prepare data

pos_wifi_rssi_dict = {}
# Dictionary keys: xy pos
# Dictionary values: [rssi_wifi1, rssi_wifi2, rssi_wifi3, x_pos | y_pos]
undetected = -100
# undetected = np.nan
n_features = len(wifi_rssi)

for i in range(len(target_wifi)):
  xy_pos_data = wifi_rssi[target_wifi[i]]
  for xy_pos in xy_pos_data:
    rssi_data = [undetected, undetected, undetected, xy_pos[0], xy_pos[1]]
    # rssi_wifi1 | rssi_wifi2 | rssi_wifi3 | x_pos | y_pos
    if xy_pos in pos_wifi_rssi_dict:
      pos_wifi_rssi_dict[xy_pos][i] = wifi_rssi[target_wifi[i]][xy_pos][0]
    else:
      rssi_data[i] = wifi_rssi[target_wifi[i]][xy_pos][0]
      pos_wifi_rssi_dict[xy_pos] = rssi_data

pos_wifi_rssi_array = np.array(list(pos_wifi_rssi_dict.values()))

# imp = SimpleImputer(missing_values=np.nan, strategy='mean')
# pos_wifi_rssi_array = imp.fit_transform(pos_wifi_rssi_array)

# Print some samples and examples
print(pos_wifi_rssi_array[0:15,:])
```

```
print('x position example: ', pos_wifi_rssi_array[0][3])
print('y position example: ', pos_wifi_rssi_array[0][4])
print('rssi example: ', pos_wifi_rssi_array[0][0])

# Get feature array and label array
features_np = pos_wifi_rssi_array[:,:3]
xy_pos_label_np = pos_wifi_rssi_array[:,3:5]

# Convert from numpy to torch
features = torch.from_numpy(features_np.astype(np.float32))
xy_pos_label = torch.from_numpy(xy_pos_label_np.astype(np.float32))

# Reshape labels
xy_pos_label = xy_pos_label.view(xy_pos_label.shape[0], 2)

n_samples = len(pos_wifi_rssi_array)
n_features = len(target_wifi)
print('Number of samples: ', n_samples)
print('Number of features: ', n_features)

## 1) model

# # Simplest model
# input_size = n_features
# output_size = 2
# model = nn.Linear(input_size, output_size)

# Testing Multilayer model
input_size = n_features
hidden_size1 = n_features * 8
hidden_size2 = hidden_size1 // 2
hidden_size3 = hidden_size2 // 2
output_size = 2

class DLModel(nn.Module):
    def __init__(self, n_input_features, hidden_sz1, hidden_sz2, hidden_sz3, output_sz):
        super(DLModel, self).__init__()
        self.n_input_features = n_input_features
        self.linear1 = nn.Linear(n_input_features, hidden_sz1)
        self.relu1 = nn.PReLU()
        self.linear2 = nn.Linear(hidden_sz1, hidden_sz2)
        self.relu2 = nn.PReLU()
        self.linear3 = nn.Linear(hidden_sz2, hidden_sz3)
        self.relu3 = nn.PReLU()
        self.linear4 = nn.Linear(hidden_sz3, output_sz)

    def forward(self, x):
        output = self.linear1(x)
        output = self.relu1(output)
        output = self.linear2(output)
        output = self.relu2(output)
        output = self.linear3(output)
        output = self.relu3(output)
        output = self.linear4(output)
        return output

model = DLModel(input_size, hidden_size1, hidden_size2, hidden_size3, output_size)

## 2) loss and optimizer
learning_rate = 0.000008
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
losses = []

## 3a) training loop
n_iterations = 100000
for epoch in range(n_iterations):
    # forward pass and loss
    xy_predicted = model(features)
    loss = criterion(xy_predicted, xy_pos_label)

    # backward pass
    loss.backward()
    losses = np.append(losses, loss.item())

    # update
```

```
    optimizer.step()
    optimizer.zero_grad()

    if (epoch+1) % 10000 == 0:
        print(f'epoch: {epoch+1}, loss = {loss.item():.4f}')

## 3b) Plot losses
plt.plot(losses)
plt.xlabel('Epoch count')
plt.ylabel('Loss')

## 4) Test model prediction accuracy using input data ##
with torch.no_grad():
    predicted_xy = model(features)
    acc = (((predicted_xy - xy_pos_label)**2).sum() / n_samples)**0.5
    print(f'Euclidean distance accuracy = {acc:.4f}m')

## 5) Get model predictions using random rssi values ##
# First sample of random rssi reading should have result similar to first few samples of training data
random_rssi_reading = np.array([[-82, undetected, undetected], [-85, -77, undetected]])
rssi_reading = torch.from_numpy(random_rssi_reading.astype(np.float32))

with torch.no_grad():
    predicted_xy = model(rssi_reading).detach().numpy()
    heat_position = predicted_xy
    for i in range(len(predicted_xy)):
        print(f'Predicted coordinates of random RSSI reading {random_rssi_reading[i]} = {heat_position[i]}')

    for i in range(n_features):
        heat_value = random_rssi_reading[:,i]
        fig = visualize_heatmap(heat_position, heat_value, floor_plan_filename, width_meter, height_meter,
title=f'{site_dir}, {floor_dir}, Wifi AP {i+1}: {target_wifi[i]} RSSI', show=True)
```

```
import os
from typing import Tuple

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

import torch
import torch.optim as optim
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader


torch.manual_seed(0)

# Dataset Related Functions
def read_data_file(data_filename: str, verbose: bool = False):
    """Function to read txt file provided and extract 4 key pieces of information."""
    with open(data_filename, 'r', encoding='utf-8') as file:
        lines = file.readlines()

    acce_data = []
    magn_data = []
    ahrs_data = []
    wayp_data = []

    for line_data in lines:
        line_data = line_data.strip()
        if not line_data or line_data[0] == '#':
            continue

        line_data = line_data.split('\t')

        if line_data[1] == 'TYPE_ACCELEROMETER':
            # timestamp | x | y | z
            acce_data.append(
                [int(line_data[0]), float(line_data[2]), float(line_data[3]), float(line_data[4])])
            continue

        if line_data[1] == 'TYPE_MAGNETIC_FIELD':
            # timestamp | x | y | z
            magn_data.append(
                [int(line_data[0]), float(line_data[2]), float(line_data[3]), float(line_data[4])])
            continue

        if line_data[1] == 'TYPE_ROTATION_VECTOR':
            # timestamp | x | y | z
            ahrs_data.append(
                [int(line_data[0]), float(line_data[2]), float(line_data[3]), float(line_data[4])])
            continue

        if line_data[1] == 'TYPE_WAYPOINT':
            # timestamp | x | y
            wayp_data.append(
                [int(line_data[0]), float(line_data[2]), float(line_data[3])])

    if verbose:
        print(f"# of accelerometer data: {len(acce_data)}")
        print(f"# of geomagnetic data  : {len(magn_data)}")
        print(f"# of rotation vect data: {len(ahrs_data)}")
        print(f"# of waypoints data    : {len(wayp_data)}")

    acce_data = np.array(acce_data)
    magn_data = np.array(magn_data)
    ahrs_data = np.array(ahrs_data)
    wayp_data = np.array(wayp_data)

    return acce_data, magn_data, ahrs_data, wayp_data

def get_nearest_position_to_magn_data(step_pos: np.array, magn_data: np.array) -> pd.DataFrame:
    """Maps magnetic strength data to its nearest position based on timestamp values."""
    # Convert to pandas dataframe
```

```
    magn_data_df = pd.DataFrame(data=magn_data, columns=['timestamp', 'x', 'y', 'z'])
    step_pos_df = pd.DataFrame(data=step_pos, columns=['timestamp', 'x', 'y'])

    # Ensure no duplicated timestamp
    magn_data_df = magn_data_df.sort_values(by=['timestamp']).drop_duplicates(keep='first')

    # Get join both dataframes based on nearest timestamp
    mag_pos_datas = pd.merge_asof(
        magn_data_df, step_pos_df, on="timestamp", direction='nearest', suffixes=('', '_pos'))

    return mag_pos_datas

def get_training_data(places_to_map: dict) -> np.arr:
    """Function to get all training data."""
    mag_pos_datas = []

    for site_id, floor_ids in places_to_map.items():
        for floor_id in floor_ids:
            path_dir = f'./data/site{site_id}/{floor_id}'
            # path_dir = f'./indoor-location-competition-20/data/site{site_id}/{floor_id}'
            data_files_path_dir = os.path.join(path_dir, 'path_data_files')
            data_filenames = [f for f in os.listdir(data_files_path_dir)
                              if os.path.isfile(os.path.join(data_files_path_dir, f))]

            for data_filename in data_filenames:
                data_filename = os.path.join(data_files_path_dir, data_filename)
                acce_data, magn_data, ahrs_data, wayp_data = read_data_file(data_filename)
                mag_pos_datas.append(get_nearest_position_to_magn_data(wayp_data, magn_data))

    mag_pos_data_df = pd.concat(mag_pos_datas, ignore_index=True)
    mag_pos_data_df = mag_pos_data_df[['x_pos', 'y_pos', 'x', 'y', 'z']]
    mag_pos_data_arr = mag_pos_data_df.to_numpy()

    return mag_pos_data_arr


# Set Up Data Loader and Model
class FingerprintDataset(Dataset):
    """Fingerprint dataset."""

    def __init__(self, fingerprint_data):
        """
        Args:
            fingerprint_data (np.array): array of arrays holding fingerprint data
                [[x1, y1, geom_x1, geom_y1, geom_z1],
                 [x2, y2, geom_x2, geom_y2, geom_z2],
                 ...]
        """
        self.labels = fingerprint_data[:, :, :, :2]
        self.features = fingerprint_data[:, :, :, 2:]

    def __len__(self):
        return self.features.shape[0]

    def __getitem__(self, idx):
        return self.features[idx], self.labels[idx]


class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()

        # First convolutional layer, outputting 16 convolutional features
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=1, stride=1, padding=0)
        # Second convolutional layer, outputting 32 convolutional features
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=1, stride=1, padding=0)
        # Third convolutional layer, outputting 1 convolutional features
        self.conv3 = nn.Conv2d(in_channels=32, out_channels=1, kernel_size=1, stride=1, padding=0)
        # First fully connected layer
        self.fc1 = nn.Linear(in_features=3, out_features=2)
        # Second fully connected layer
        self.fc2 = nn.Linear(in_features=2, out_features=2)

    def forward(self, x):
        out = F.relu(self.conv1(x))
        out = F.relu(self.conv2(out))
```

```
        out = F.relu(self.conv3(out))
        out = F.relu(self.fc1(out))
        out = F.relu(self.fc2(out))
        return out


# Set Up Training and Testing
def train(epoch: int, net: nn.Module, criterion: any,
          trainloader:torch.utils.data.DataLoader,
          optimizer: any, y_mean: torch.Tensor, y_std: torch.Tensor,
          scheduler: any = None, verbose: bool = False) -> Tuple[float, float]:
    """Function to train model for one epoch.s"""
    device = 'cuda'
    if verbose:
        print('Epoch: %d' % epoch)
    net.train()
    train_loss = 0
    train_error = 0

    for batch_idx, (inputs, targets) in enumerate(trainloader):
        inputs = inputs.to(device, dtype=torch.float)
        targets = targets.to(device, dtype=torch.float)
        optimizer.zero_grad()
        outputs = net(inputs)

        error = torch.sum(torch.sqrt(torch.sum((
            targets-(outputs*y_std + y_mean))**2, dim=1))) / targets.shape[0]

        loss = criterion(outputs, (targets-y_mean)/y_std)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        train_error += error.detach().item()

        if verbose:
            if ((batch_idx+1) % 50 == 0) or (batch_idx > 1650):
                print("iteration : %3d, loss : %0.4f, error : %2.2f" %
                    (batch_idx+1, train_loss/(batch_idx+1), train_error/(batch_idx+1)))

    if scheduler:
        scheduler.step()
    return train_loss/(batch_idx+1), train_error/(batch_idx+1)

def test(net: nn.Module, criterion: any, testloader:torch.utils.data.DataLoader,
         y_mean: torch.Tensor, y_std: torch.Tensor) -> Tuple[float, float]:
    """Function to predict model and compare results against true values."""
    device = 'cuda'
    net.eval()
    test_loss = 0
    test_error = 0
    with torch.inference_mode():
        for batch_idx, (inputs, targets) in enumerate(testloader):
            inputs = inputs.to(device, dtype=torch.float)
            targets = targets.to(device, dtype=torch.float)
            outputs = net(inputs)

            error = torch.sum(torch.sqrt(torch.sum((
                targets-(outputs*y_std + y_mean))**2, dim=1))) / targets.shape[0]
            loss = criterion(outputs, (targets-y_mean)/y_std)

            test_loss += loss.item()
            test_error += error.detach().item()

    return test_loss/(batch_idx+1), test_error/(batch_idx+1)

def train_x_epochs(num_epochs: int, net: nn.Module, criterion: any,
                   trainloader: torch.utils.data.DataLoader,
                   valloader: torch.utils.data.DataLoader,
                   optimizer: any, y_mean: torch.Tensor, y_std: torch.Tensor,
                   scheduler: any = None, early_stopping_rounds: int = 20,
                   verbose: bool = False) -> Tuple[list, list, list, list]:
    """Function to train model for x number of epochs."""
    train_losses = []
    train_errors = []
    val_losses = []
```

```
    val_errors = []

    for epoch in range(1, num_epochs+1):
        train_loss, train_error = train(epoch, net, criterion, trainloader,
                                        optimizer, y_mean, y_std, scheduler)
        val_loss, val_error = test(net, criterion, valloader, y_mean, y_std)

        if verbose:
            print(("Epoch : %3d, training loss : %0.4f, training error : " + \
            "%2.2f, validation loss : %0.4f, validation error : %2.2f") % (
                epoch, train_loss, train_error, val_loss, val_error))

        if early_stopping_rounds > 0:
            if epoch > 10 and all(past_error <= val_error for past_error
                                  in val_errors[early_stopping_rounds*-1:]):
                print(f'Early stopping at Epoch {epoch - 1}')
                break

        train_losses.append(train_loss)
        train_errors.append(train_error)
        val_losses.append(val_loss)
        val_errors.append(val_error)

    return train_losses, train_error, val_losses, val_error


if __name__ == "__main__":

    # Get training data
    places_to_map = {
        1: ['B1', 'F1', 'F2', 'F3', 'F4'],
        # 2: ['B1', 'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8'],
    }
    mag_pos_data_arr = get_training_data(places_to_map)

    # Do some data split
    trainset, valset = train_test_split(mag_pos_data_arr, test_size=0.2,
                                        random_state=0)

    scaler = StandardScaler()
    trainset[:,2:6] = scaler.fit_transform(trainset[:,2:6].copy())
    valset[:,2:6] = scaler.transform(valset[:,2:6].copy())

    trainset = np.expand_dims(trainset, axis=1)
    valset = np.expand_dims(valset, axis=1)
    trainset = np.expand_dims(trainset, axis=1)
    valset = np.expand_dims(valset, axis=1)

    # Get mean and std of labels (x and y coordinate)
    y = trainset[:, :, :, :2]
    y_mean = torch.Tensor(y.mean(axis=0)).to('cuda')
    y_std = torch.Tensor(y.std(axis=0)).to('cuda')

    # Get dataloader
    trainloader = torch.utils.data.DataLoader(
        FingerprintDataset(trainset), batch_size=128, shuffle=True, num_workers=2)
    valloader = torch.utils.data.DataLoader(
        FingerprintDataset(valset), batch_size=256, shuffle=False, num_workers=2)

    # Train Final Model
    learning_rate = 0.0001
    num_epochs = 300

    net = Model().to('cuda')
    criterion = nn.MSELoss().to('cuda')
    optimizer = optim.Adam(net.parameters(), lr=learning_rate)
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs)

    # With scheduler
    train_losses, train_accs, test_losses, test_accs = \
        train_x_epochs(num_epochs, net, criterion, trainloader, valloader,
                       optimizer, y_mean, y_std, scheduler=scheduler,
                       early_stopping_rounds=20, verbose=True)
```