T452-22B: The Engineering Project

Optimization of WiFi Access Point placement in floorplans using signal propagation loss models in Python

Henrique Aires Silva (J5734318)

Bachelor of Engineering (Honours) – Electronics Pathway

Contents

Introduction

WiFi has become a basic necessity in nearly all environments in the modern world, connecting government services to citizens, bringing streamed entertainment, and even enabling long-distance education. With the recent popularization of high-speed WiFi connections through cheaper service plans and infrastructure investments, more than 92% of households in the European Union now have access to internet (Eurostat, 2022).

Connection quality is becoming a very important characteristic of WiFi, given its new uses and dependence. One of the factors that can significantly impact this **term** is the Access Point (AP) placement inside the household since it will directly affect the received signal power on the devices under use. Correct placement of an AP is an often overlooked task by the average user without technical knowledge about networks, which leads to the search of unnecessary corrective measures, such as the installation of WiFi repeaters and boosters. These devices can improve signal strength, but at the cost of bandwidth.

This project aims to simulate WiFi signal propagation loss in a given floorplan using multiple established models, described in (C. B. Andrade and Hoefel, 2010) and (Lloret *et al.*, 2004), and optimize the access point placement. Using signal heatmaps, the AP location can be optimized to maximize coverage or signal quality in a specific area of the floorplan. It is important to note that the used signal simulation method in this project does not rely on electromagnetic wave interaction as basis, but rather realistic models of signal loss which are discussed in the theory review.

Image processing is used to load the floorplans and identify walls and rooms, which serve as basis for the propagation simulation and creation of the signal strength heatmaps.

The last step of the process includes an optimization routine for the AP placement considering location identified in the floorplan and uses the generated heatmaps as input.

All simulation and optimization run on Python in Jupyter Notebooks framework, which allows simple development of interactive tools and can be used in cloud-stored services by end-users, simplifying the setup on different machines.

Theory Review

Signal Loss Modelling

Considering indoor environments, wireless signals can be attenuated or obstructed by multiple agents, such as partitions, walls, passing people and other obstacles in general. Modelling these signals can be done in multiple ways. The two clearer distinctions between models are deterministic and empirical.

Deterministic models rely on physical models and can model the signal path or multiple interferences based on its properties. However, these models usually involve multiple matrix calculations and several interactions, which leads to long simulation times.

Empirical models differ from deterministic ones in the fact that they need pre-measured data as an input. In the case of path loss models, this input is usually the measurements of attenuation due multiple materials constituting the obstacles.

Even though deterministic models, namely ray tracing and ray casting, have been shown to be able to accurately model indoor propagation loss (Remley, Anderson and Weisshar, 2000), they are computationally heavy and require specialized hardware specialized hardware, such as processing clusters or powerful graphics cards, to be able to run simulations on a feasible time. Since in this project the aim is to run multiple simulations in order to get an optimal placement, they have to be able to be calculated rather quickly.

Multiple empirical models have been compared in previous works with site measurements, (C. B. Andrade and Hoefel, 2010) analyse five different models that are commonly used in WLAN planning: One-Slope, Dual-Slope, Partitioned, COST231 – Multi Wall, and Average Walls. From their results, the model that better predicts the path loss is the Average Walls, proposed by (Lloret *et al.*, 2004). The One-Slope model was also found to yield good predictions in small environments, where the receiver and transmitter are at most 20 m apart (C. Andrade and Hoefel, 2010).

Although both described models have shown good predictions, both need empirical measurements as a base input, which will not be available in this scope. Therefore, the COST231 Multi-Wall model shows a more promising approach, by using data from several measurement campaigns, it is defined as

$$L_{dB} = L_{0_{dB}} + 20\log_{10} d + \sum_{i=1}^{k_w} k_{wi} L_{wi}$$

Where $L_{0_{dB}}$ is the free-space pathloss in dB, d is the distance between transmitter and receiver, k_w is the number of wall types, k_{wi} is the number of the ith wall and L_{wi} the loss of the ith wall type. Note that since the layout considered in this project is a single-floor, the floor path loss can be excluded from the model.

The free-space pathloss can be calculated as

$$FSPL = L_{0_{dB}} = 20\log_{10}(d) + 20\log_{10}(f) + 20\log_{10}\left(\frac{4\pi}{c}\right) - G_t - G_r$$

Where d is the distance in meters between the antennas, f is the frequency of the RF signal, G_t and G_r are the gains of the transmitter and receiver, respectively. For simplicity, the gains here are considered to be inexistent.

The multi-wall model considers two types of walls, Light Walls, like plasterboards, particle boards or thin (<10 cm) concrete wall; and Heavy walls, load-bearing walls or other thick (>10 cm) walls. Their losses are defined as

Table 1: Loss for different wall types in COST231 MW model. Adapted from (C. Andrade and Hoefel, 2010)

Wall Type	Loss
Thin Wall	3.4 dB
Thick Wall	6.9 dB

A similar approach to this last model is to consider an average loss for all types of walls, aggregating the loss parameter into a single constant. This model is known as Average Walls Model (C. Andrade and Hoefel, 2010). It is defined similarly as the COST231 Multi-Wall model:

$$L_{dB} = L_{0_{dB}} + 20\log_{10} d + k_w L_w$$

Where k_w represents the number of penetrated walls and L_w the average attenuation. The L_w parameter can be obtained through measurement campaigns and surveys.

Image Processing

Image Thresholding

Image thresholding is a technique used in image processing to convert a raw image, colour or grey-scale, into a binary image, which is in its essence, black and white.

This process is used to highlight and select features present in the original image by applying a pixel intensity threshold. Upon setting a threshold value, any pixel with intensity value above it is converted to white (255), and the ones with a lower value are set to black (0). For coloured images, the threshold can be applied in a single channel to isolate a specific feature that has a distinct colour. In grey-scale images the threshold simply defines the level of brightness that will be cut-off.

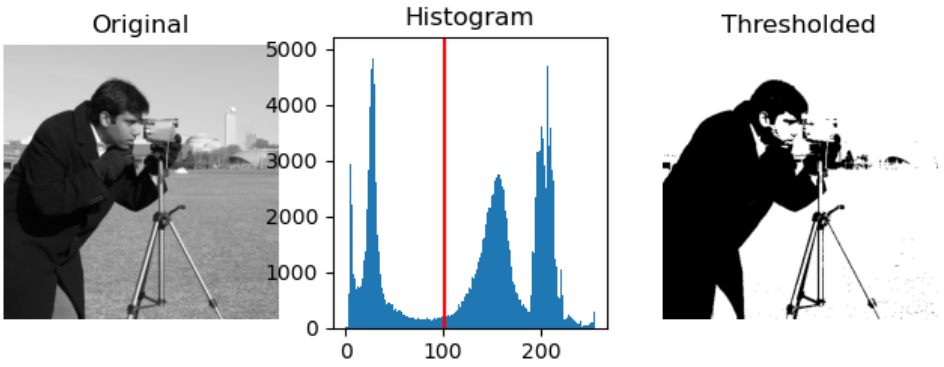


*Figure 1: Example of the image thresholding technique (*Thresholding — skimage v0.19.2 docs*, no date)*

Morphological Transformations

In image processing, the term morphological transformation refers to simple operations mostly applied in binary images, due to its nature of depending on the pixels' relative positions and ordering, not their relative value.

This technique consists in probing an image using a so called structuring element, or kernel, which is a small sample image or template that will be overlaid on all possible positions of the original image and a subsequent test will be made in order to transform or not the original pixel. If the kernel structure perfectly overlays all pixels, i.e., there are no differences then the test result is "fit". Conversely, if all pixels are different from the kernel, the test is a "miss". The last case is when just a few pixels differ between kernel and image, resulting in a "hit".

The action taken for each test result varies according to the selected morphological transformation and will be discussed further.

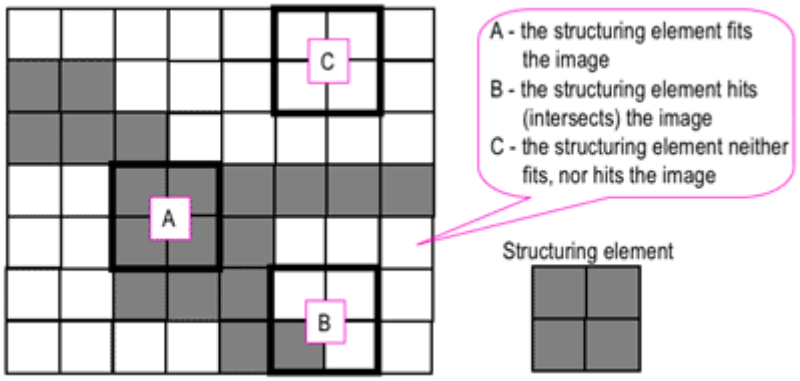


Figure 2: Probing of an image with a structuring element (white and grey pixels have zero and non-zero values, respectively) (Morphological Image Processing, *no date).*

Erosion

The erosion operator, as the name states, erodes the image by turning the probed pixel in 0 on any hit results. This essentially shrinks the figures in the image, and it is especially useful for removing small details and white noise from the image.

A drawback from this technique is that if the kernel size and shape must be optimized for each use case to not erode away wanted features.

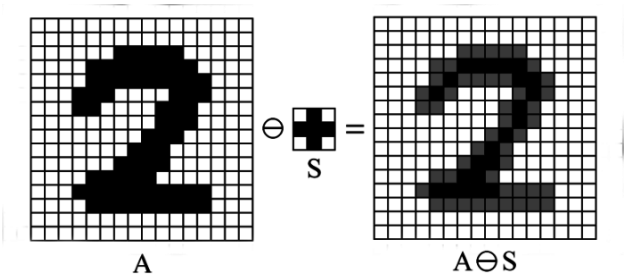


Figure 3: Example of erosion process. (Mardiris and Chatzis, 2016)

Dilation

Dilation is a complimentary function to erosion, converting the original pixel to 1 upon any hits. This effect causes the original image shape to expand, or “dilate”, and it is useful to fix small gaps or broken features caused by a previous erosion.

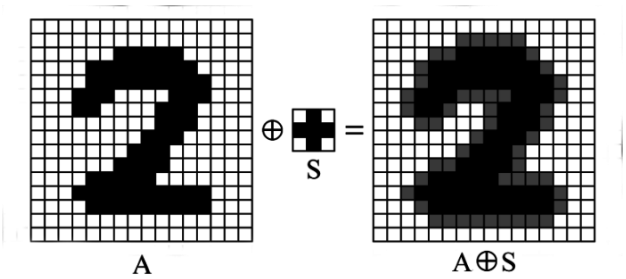


Figure 4: Example of dilation process. (Mardiris and Chatzis, 2016)

Methodology

Computational Environment Setup

Mathematical simulation is often a computational heavy task, including multiple calculations and optimizations, therefore, efficient scripting should be employed to make good use of available resources.

Python is one of the most famous scripting languages in the recent years due to its ease-of-use, simple language, fast execution and large community.

Despite being a solid scripting language, Python in its pure form lacks an easy way to handle user interaction. To tackle this problem, which is recurrent in many other scripting languages, the Jupyter framework was built. It features the so called “Notebooks”, which are embedded pages that can present formatted text, run code in isolated cells or depict images.

Using Jupyter as a base framework for Python scripts adds a lot of flexibility in regard to user interaction and data display.

Many add-on packages can be used to automate display functions and display data in a more logical and sensible way to the user. The selected package for data visualization in this project was Bokeh, mainly for its native feature of plotting categorical data, such as heatmaps, natively.

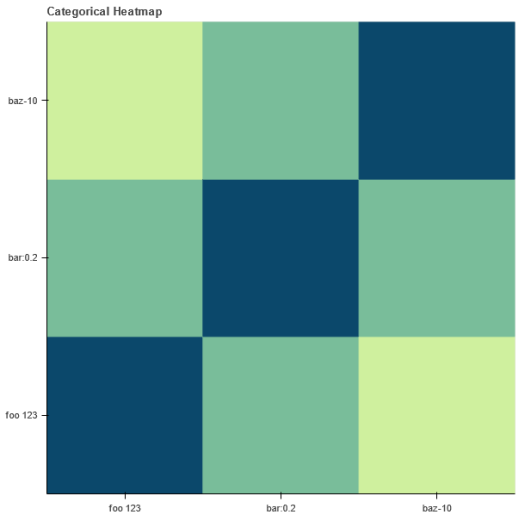


Figure 5: Example of categorial heatmap plot by Bokeh (Bokeh, no date)

The initial setup included running a local instance of JupyterLab, using the available hardware (AMD Ryzen 5 3600 6 Core processor and 16GB of RAM), but with the increased runtimes, the notebook was moved to the Google Colab platform, which provides Cloud-based Jupyter servers and several processor clusters to users. It also enables easy sharing of data, eliminating the need for a local setup for external users.

For the backend data handling and calculations, the Pandas package was used, given its easy access and handling of large amount of data in its DataFrames, offering also fast and efficient data transformation.

Given its heavy share of image processing, it was decided that this project would benefit from using a fast and simple to use library known as OpenCV. It provides efficient function and data handling for most image processing processes and can be used in conjunction with Pandas and Bokeh.

Floorplan Image Processing

The generation of a signal attenuation heatmap depends largely on the floorplan data available, given that the loss is proportional to the number of walls in the way between the receiver and the access point. Given that walls are the largest factor in signal attenuation at the studied frequency (2.4 GHz), it is important that the floorplan image depicts them in a clear way.

Small features, such as such room measurements, door and furniture drawings, and noise are frequent on this type of data, so an image processing step is needed before converting the raw data into location information.

The image processing algorithm can be summarized in the flow diagram shown in Fig 6.

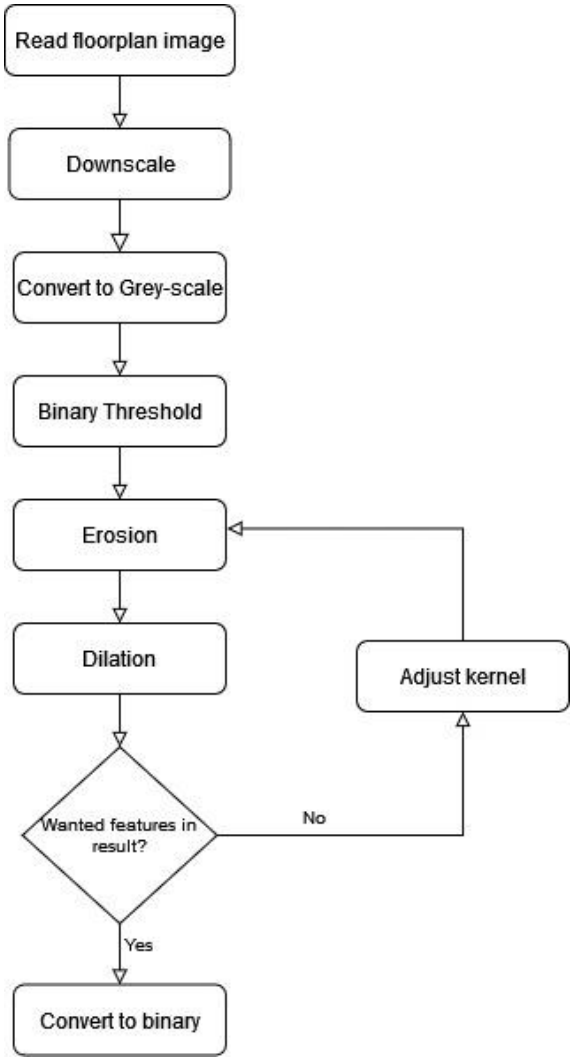


Figure 6: Flow diagram of floorplan image processing algorithm

Database

In this project, several floorplan images are needed to validate the capability of the image processing step to remove small features from the image and correctly identify walls. This process depends highly on image quality and type of extra annotations present on the file.

As the focus of this project is not computer vision in of itself, a database with good quality and clear floorplan images would be ideal to evaluate the heatmap generation script. The selected database [ref] is available with an open source license on Github and presents the qualities aforementioned, having only small furniture drawings to be removed, as shown in the following example.

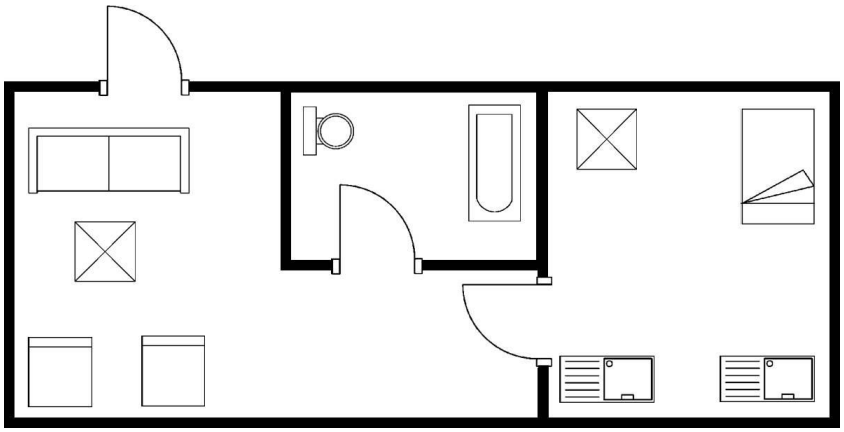


Figure 7: Example of floorplan image from the selected database

In preparation for the location optimization stage of the algorithm, it was necessary to add information about possible Access Point placement in the original floorplan. Adding coloured points to the original image was the easiest way to indicate the coordinates, since it could be easily parsed by the script. Using absolute coordinates would not be user friendly due to the relation between pixel and meters.

Therefore, for the optimization process, a few images were modified to contain possible AP placement, indicated by a blue rectangle, as shown in the following figure.

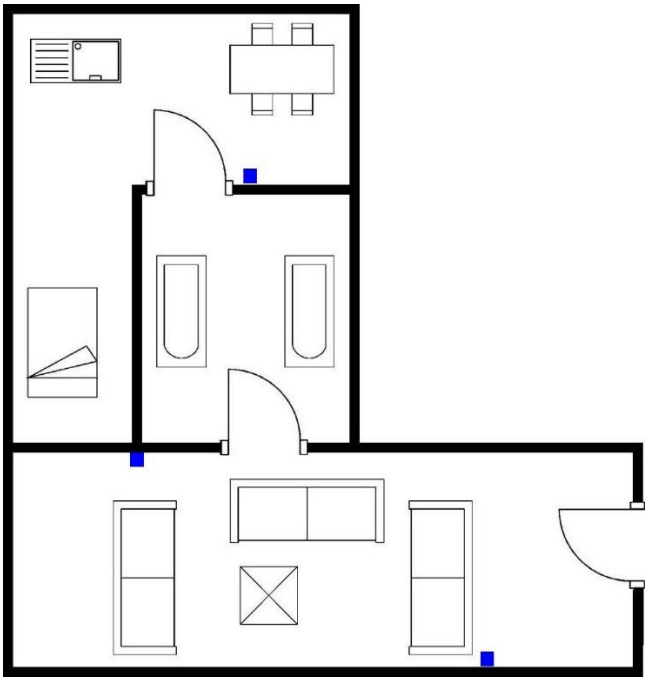


Figure 8: Floorplan with possible AP indications as blue rectangles

Given the high resolution of the images, processing them pixel by pixel would increase the algorithm runtime significantly, therefore, a downscaling process was added as an option during development, where many iterations are needed.

Morphological Transformations

The process of removing small features and identifying walls relies on morphological transformations implemented by the library OpenCV.

As the erosion and dilation depend directly on the shape and size of the kernel, these parameters have to be optimized manually. A small kernel will fail to erode the drawn furniture and noise, whilst a big kernel would simply erode away the walls themselves.

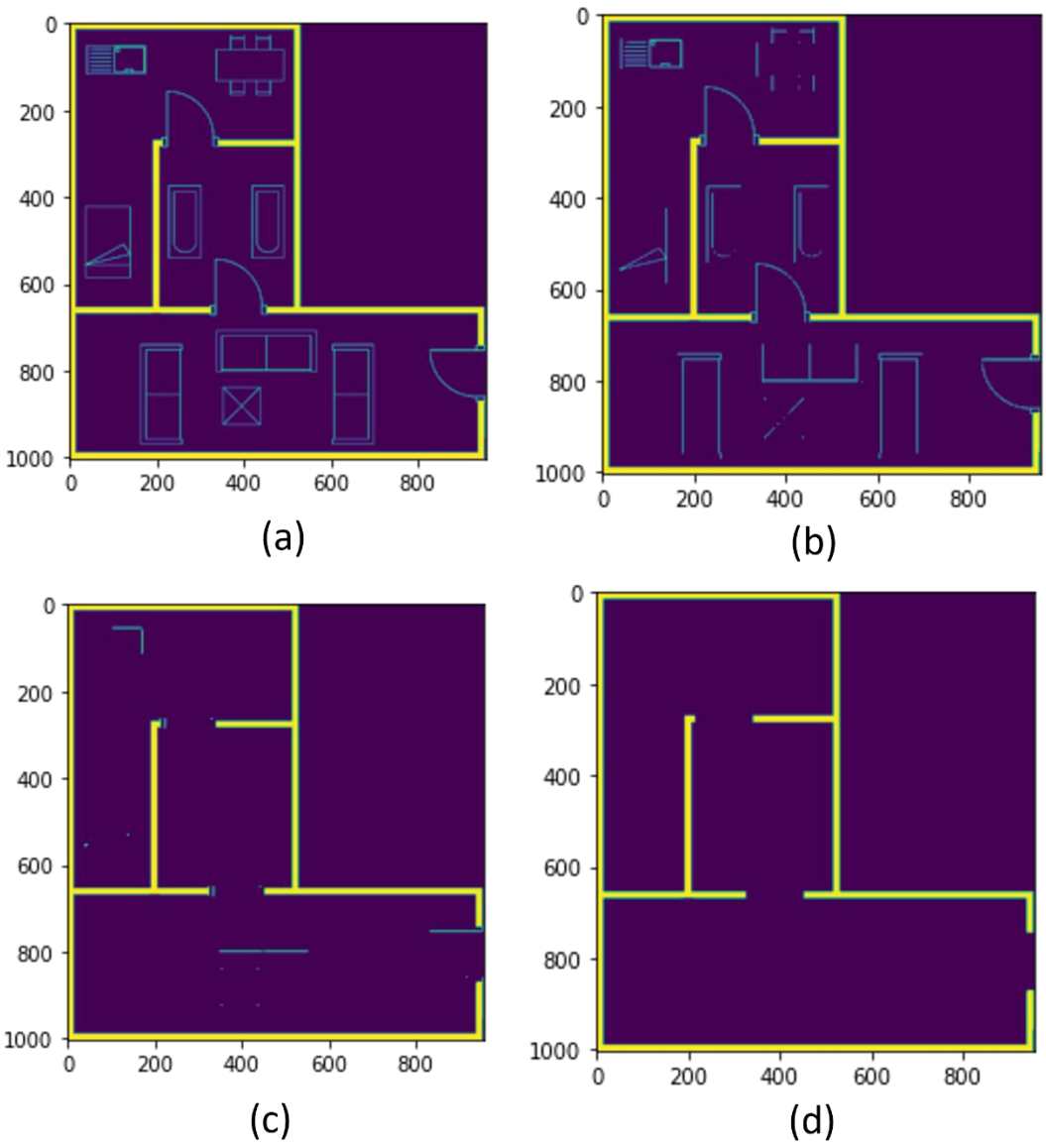


Figure 9: Difference of kernel size in morphological transformations. Kernel with dimensions (a) 1x1, (b) 2x2, (c) 3x3, (d) 4x4

It has been decided, then, to leave the kernel optimization as a user input, which can be adjusted during the notebook run.

Wifi Signal Loss Computational Modelling

All signal loss models used in this project have a common dependency on the distance between the access point and client, therefore this is the first operation to be implemented as base for the models.

Considering the floorplan image as a 2D cartesian map, the absolute distance between the coordinates of the transmitter and receiver can be calculated using the Pythagorean Theorem:

$$d = \sqrt{(x_{TX} - x_{RX})^2 + (y_{TX} - y_{RX})^2}$$

It is also important to take in consideration that floorplans are not always in a 1:1 scale, meaning that 1 pixel does not equal to 1 meter directly. Therefore, a scaling factor has to be added to the equation to transform pixels to meters.

Another scaling factor to be considered in of the image itself, as it can be up or downscaled in terms of its resolution. Therefore, the final distance calculation can be defined as

$$d = \sqrt{(x_{TX} - x_{RX})^2 + (y_{TX} - y_{RX})^2} \times \frac{S_{Floorplan}}{S_{Image}}$$

Where $S_{Floorplan}$ is the scale conversion in units of $m.pixel^{-1}$ and S_{Image} is the resolution scale in units of $pixel.pixel^{-1}$.

This equation is applied to all pixels in the map as a first step and the data is stored in a separate column of the model DataFrame. If the distance is plotted over the floorplan, then the following map is generated:

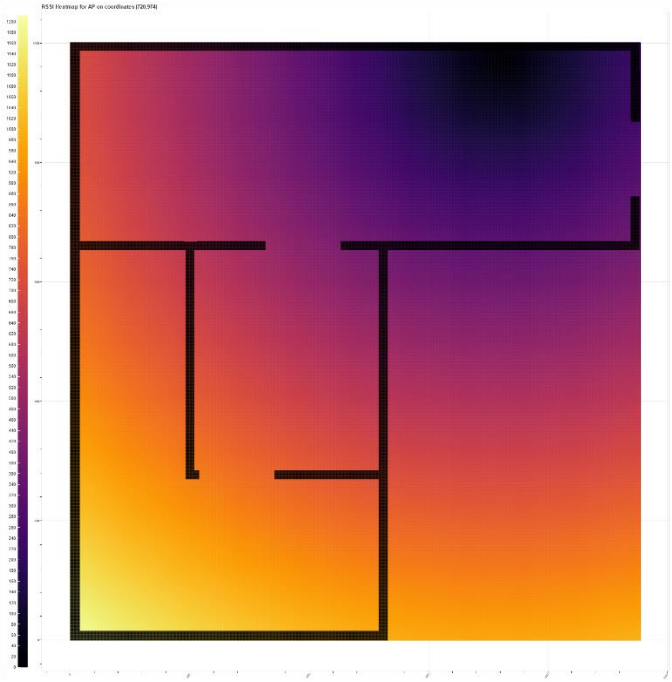


Figure 10: Distance of all pixels to the origin point plotted as a heatmap. Original walls plotted in black for reference.

The generic algorithm that describes the modelling, regardless of selected type, is shown in the following flow diagram.

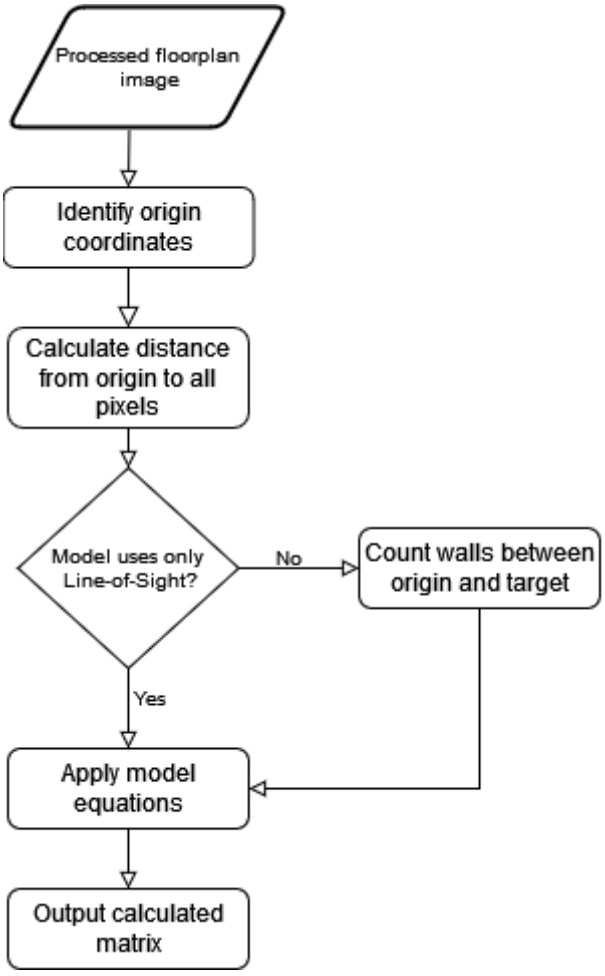


Figure 11: Flow diagram of loss modelling algorithm

Wall Identification and Counting

For the Multi-Wall and Average Wall Loss Models, the main parameter needed is the number of walls between the transmitter and receiver, and its type. Therefore, it was necessary to implement an algorithm that could count the number of walls in that path, described in Figure 12.

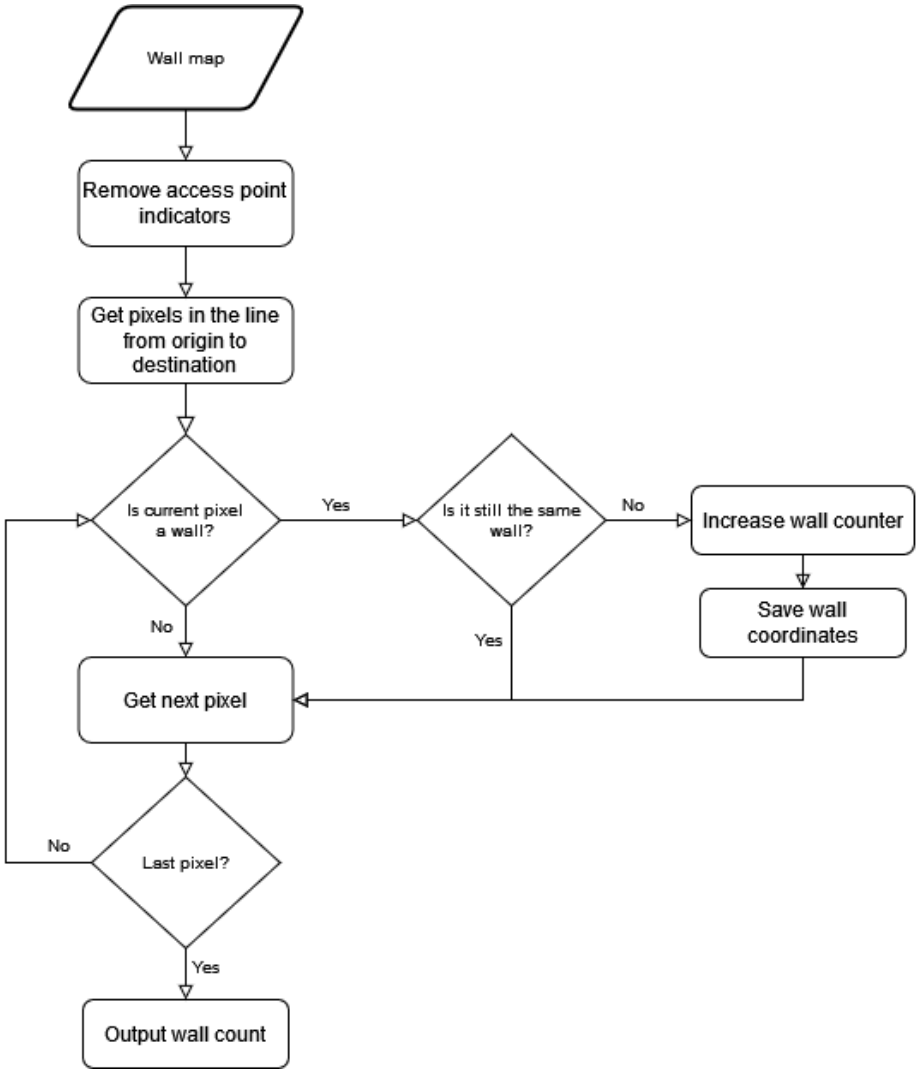


Figure 12: Flow diagram explaining the wall counting algorithm

It is important to notice that a pre-processing step is needed here to find and remove the blue pixel indicators for possible AP placement before the walls are identified, otherwise these blue rectangles would be considered as walls due to their thickness and interfere with the model.

This removal process is done by first converting the image into HSV (Hue, Saturation and Value) encoding and then filtering by the blue range using a mask, leaving only true blue pixels in the image. This step is important to discern between pixels that have blue components, such as white ones, and pure blue ones. After filtering, OpenCV can identify the rectangle contours automatically and from that, calculate their momentum, essentially returning the central point coordinates.

Using the found coordinates of the blue rectangles, the original image is painted, and the blue pixels become white as the background itself. The modified image is then the input for the wall identification algorithm.

A simple way to get a list of pixels that are in a line between two points is to apply the parametrization technique of a line, which involves calculating its slope and offset and create a function for any point.

$$y = mx + b$$
$$m = \frac{y_2 - y_1}{x_2 - x_1}$$
$$b = y - mx$$

The main issue with this approach is that the slope relies on the points not being aligned in the "x" coordinate, i.e., the line cannot be vertical. This imposes a serious restriction for the modelling since there are many points directly above or below the origin.

Therefore, another computational method for discreet image processing was used to generate the line pixel list, called Bresenham's line algorithm. This method is known as an incremental error algorithm, and it is commonly used to draw line primitives on bitmaps. Its implementation is out of the scope of this project and a pre-made function from the library SciKit Image was used.

User Interaction

During the development phase of the script, it was noticed that implementing user interaction would significantly increase the complexity of the project, since it would require extensive knowledge in JavaScript programming to interact with the Bokeh plots.

The main features that would require interaction would be selecting multiple points for the possible AP placement, customize the wall type, and selecting specific rooms for the optimization routine.

Therefore, the project scope was reduced, and a decision was taken to integrate the placement data in the floorplan image itself using coloured pixels. The wall type and room selection were also abandoned due to the added complexity.

With only one type of wall, the initially chosen model of Multi-Wall Loss essentially became the Average Wall Loss Model, only depending on the number of walls obstructing line-of-sight.

Access Point Placement Optimization

The initial placement optimization proposal would involve two different types of optimization goals: higher average signal in a room, or higher average signal in whole area. The later has a very straight-forward implementation and comes directly from the calculate heatmap data. It is even simpler considering that the Pandas DataFrame already computes this information automatically through the “mean” function applied to a column.

This simple algorithm can be visualized in the following flow diagram.

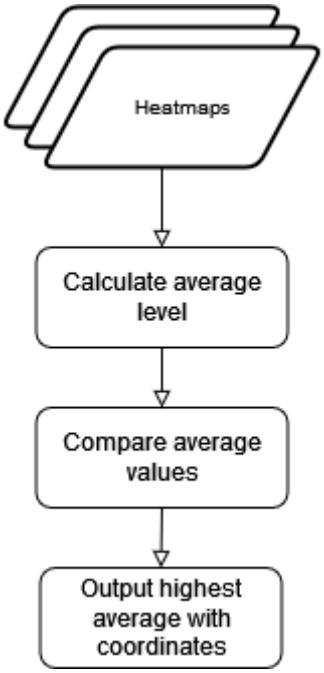


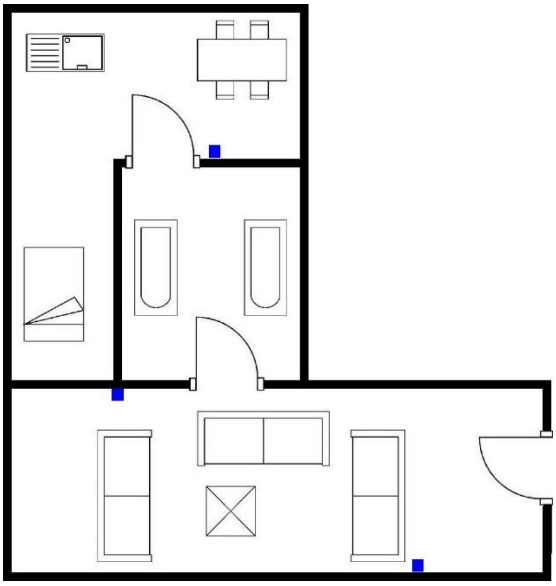
Figure 13: Flow diagram for signal optimization algorithm

The goal of optimizing for an area would require additional image processing to identify contours of walls and try to guess closed areas. Although possible, it would also increase processing time and complexity too much, so, again, the scope was reduced and only the total average signal goal was implemented.

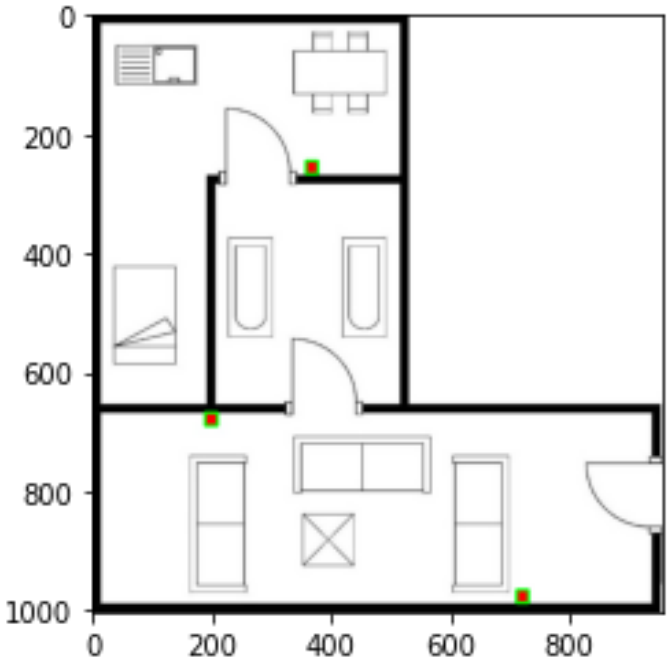
Although not implemented in the script, the information of the average signal in a room can be inferred from the heatmap data itself, leaving up the decision to the user.

Results

A few sample images from the floorplan database were modified to include blue squares representing the possible Access Points locations. The following figure contains the base floorplan with 3 AP placements, one in the Dining Room in the top part, one by the living room in the lower left quadrant and the last one in the lower right quadrant by the entrance door.



The first step is the identification and removal of the blue indicators. The image is processed as described earlier and the resulting image with the location's contours are painted in green.



The centre coordinates for each identified AP in the image are stored in the memory and the rectangles are then erased.

After applying the threshold and morphological transformations, the script outputs the processed floorplan map with the identified walls in a binary image, where pixels that represent walls have the value “1” and the rest are “0”.

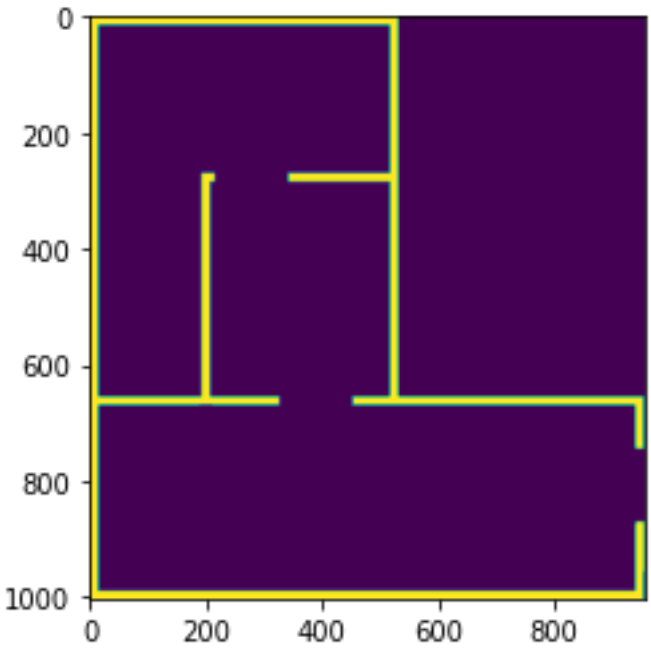


Figure 14: Processed floorplan showing the identified walls in yellow and the background in purple

Using the generated wall map, the script loops through all pixels to build a DataFrame consisting of X and Y coordinates, the distance to origin, calculated loss and RSSI (Received Signal Strength Indication). This DataFrame is then plotted in a heatmap by Bokeh for each Access Point location. The identified walls are also plotted over the heatmap in black to facilitate the visualization.

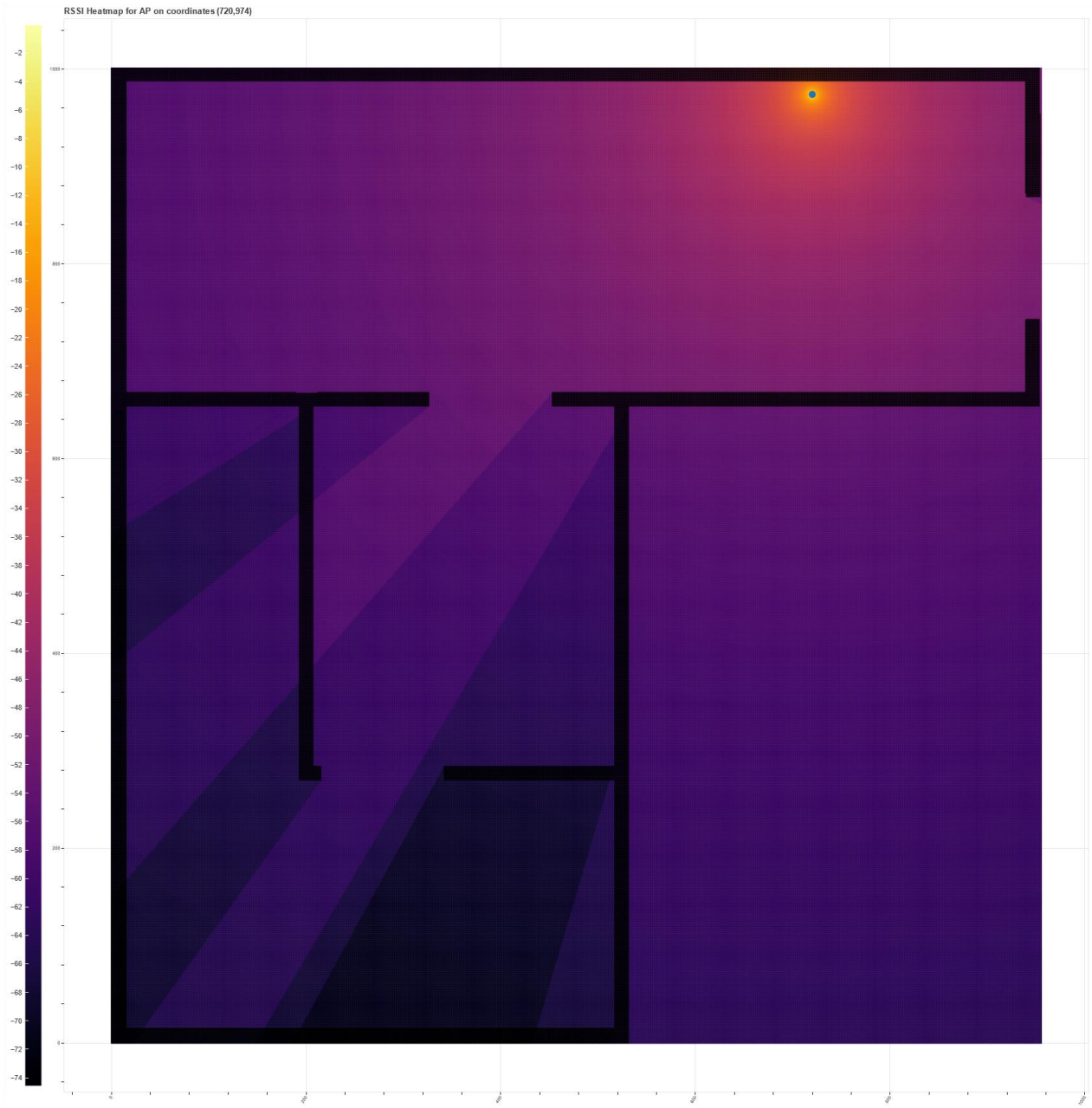


Figure 15: Example heatmap of RSSI for first access point location

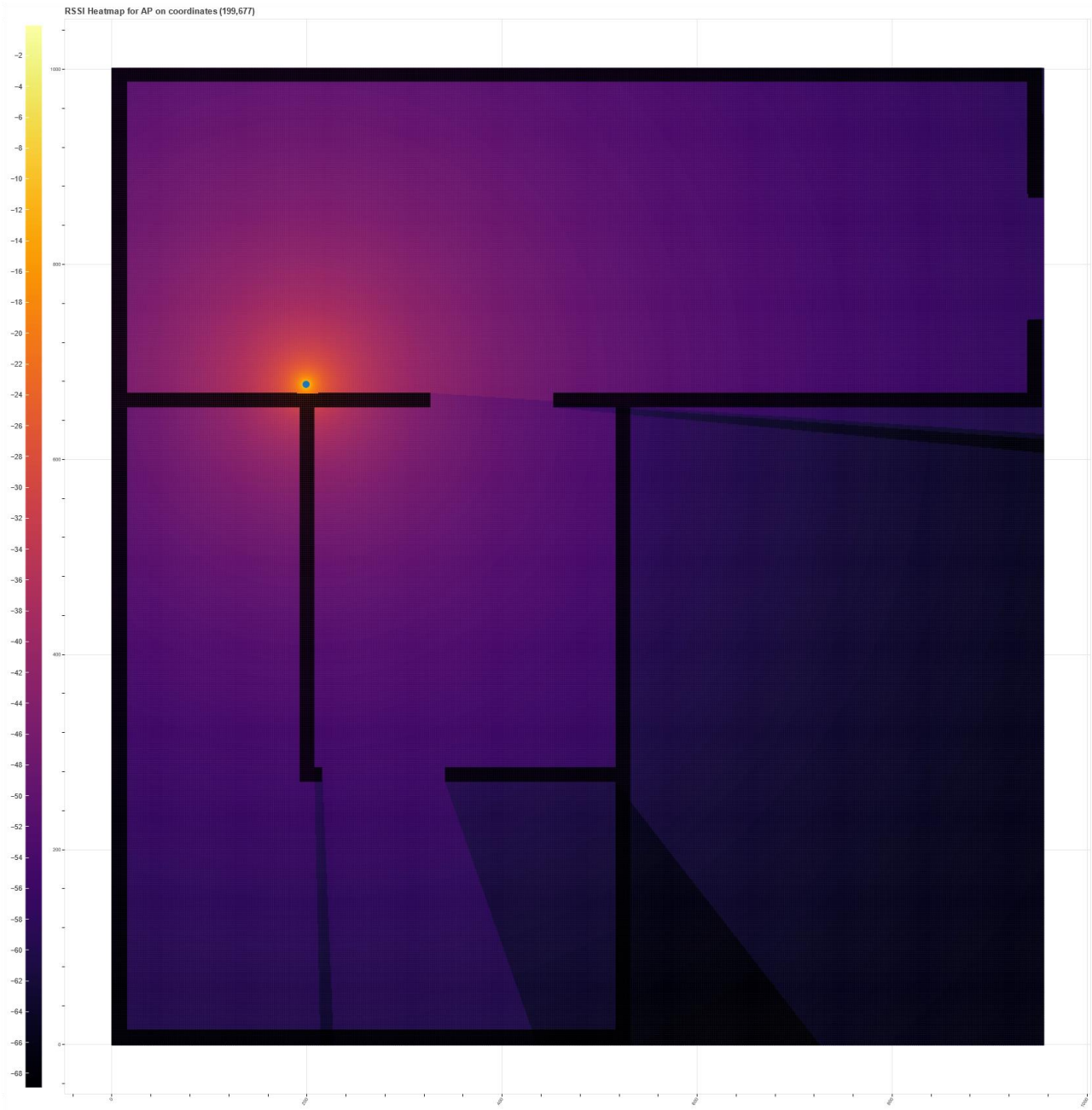


Figure 16: Example heatmap of RSSI for second access point location

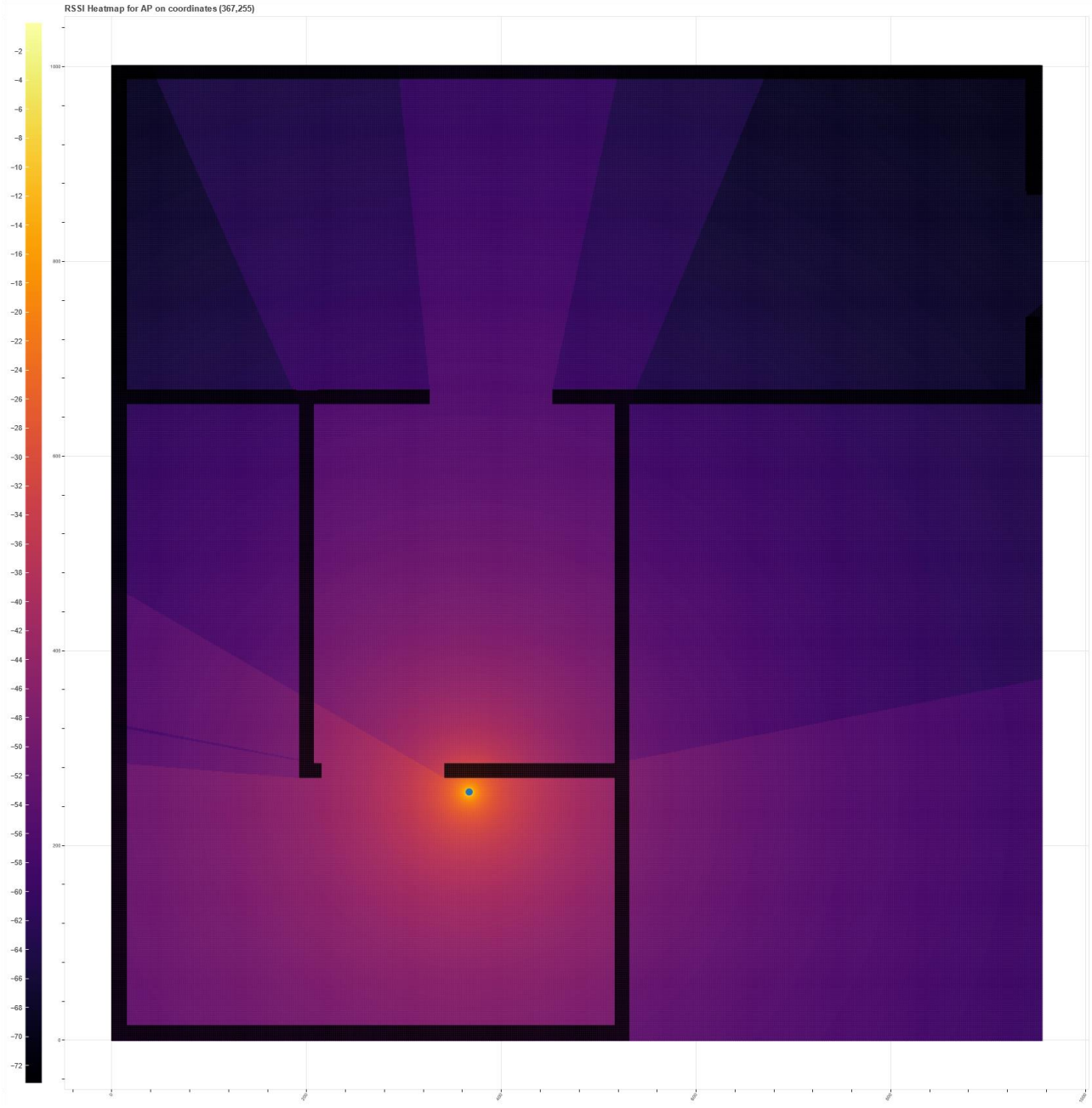


Figure 17: Example heatmap of RSSI for third access point location

The three generated heatmaps are then processed by the optimization algorithm and the one with highest average value in total is selected as optimal.

```
The AP placement with the highest average signal is on coordinates X:3.67 m, Y:2.55 m, with a value of -55.87 dBm
```

Figure 18: Result message from the script stating the optimal placement

Runtime

One of the biggest issues for this type of algorithm that iterates over all pixels in the image is runtime. As an example, the image used for demonstration above has original dimensions of 1200 by 1400 pixels. This means that the script has to run the model calculation approximately 1.68×10^6 times for each different Access Point origin. This also means that large memory arrays and buffers have to be used to store all the produced data, both internally as in the notebook output plot.

Without the image rescaling option, a usual run for 3 access points of the example image above takes about 5 minutes to compute and consumes about 2GB of RAM. This can be a limiting factor if the available hardware is not powerful enough.

When rescaling the image to about 30% of its size, the runtime decreases to about 40 seconds, at the cost of output resolution, as shown below.

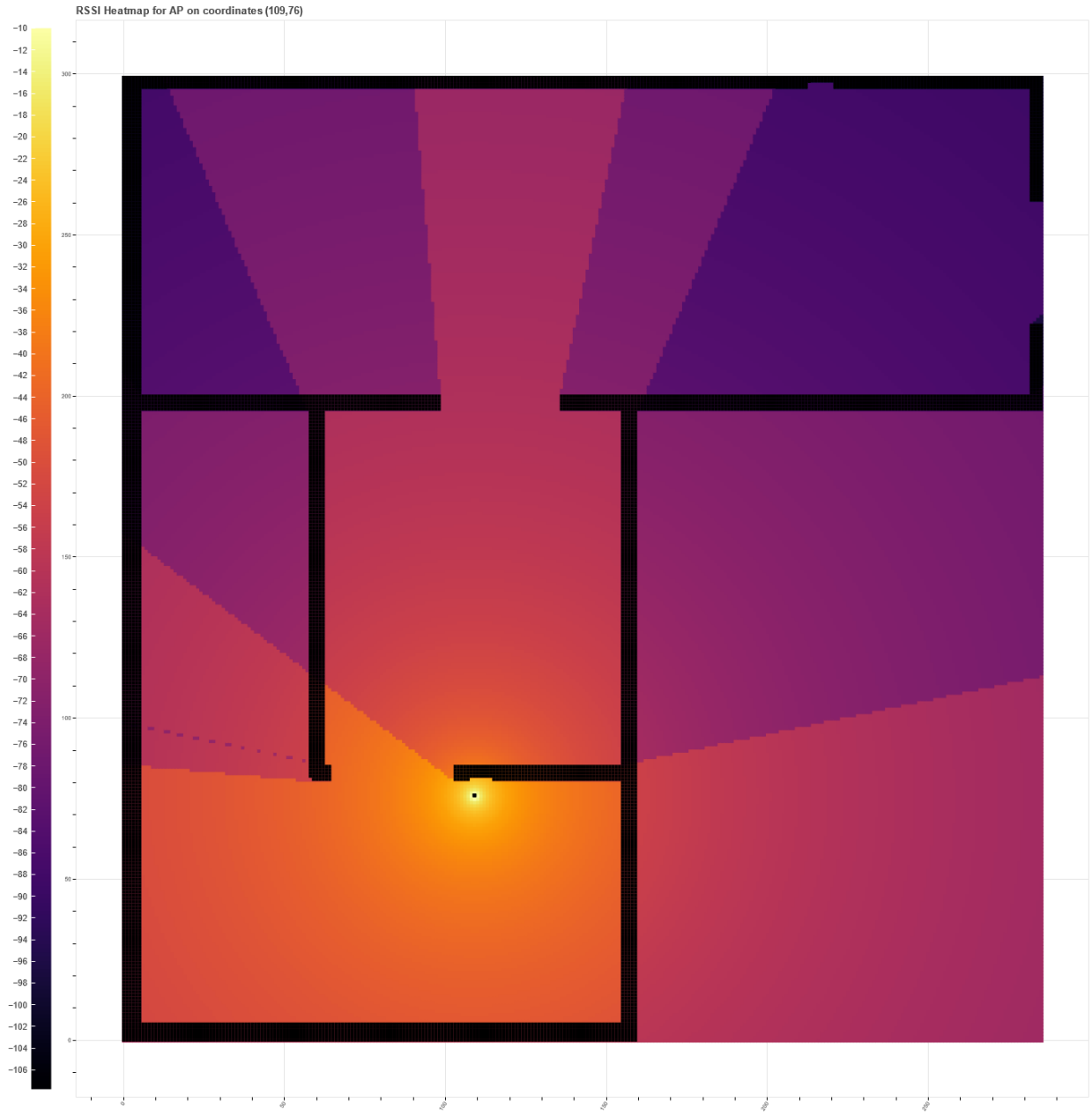


Figure 19: RSSI Heatmap with sample image downscale to 30% of its original size

Conclusions

Analysing the first proposed scope of the project, which included user interaction, advanced optimization routines and multiple model comparison, it is fair to say that the project itself diverted and had to be downscaled to fit the available time and resources. Having a smaller scope allowed the core functions of the project to be developed with certainty and with enough testing to be reliable and produce useful data. This larger scope was pointed out at the first review of the proposal and after it, it was agreed that if needed, the reduction would be favourable to the quality of the project.

The code was developed in such a way that with few modifications, the features that were dropped can be included in later releases without the need of a full rebase. Including user interaction can be done in a separate module that produces the images with the symbols expected by the current script, or an additional image processing step can be done to identify and select single walls, so that the user can change its properties.

As the goal of the project was not to be an optimized script, the algorithms and functions were not developed as the most efficient possible, which would require a longer time to develop. The solution to that was simply use less efficient code that would produce the same result but consuming more memory and taking longer. This is not a problem when computational are plenty, as in the Google Colab platform.

The heatmap data generated by this script is especially useful to identify shadowed areas in specific regions of a given floorplan. Despite not considering other interfering factors in the signal quality, such as reflections, furniture and dispersal, the estimative appears to be good enough, qualitatively, to select the best placement for the Access Point and mitigate losses.

The project code and database are hosted on Github (Silva, 2022) and the latest notebook version can be accessed at the Google Colab environment that is already prepared on the following link.

References

Andrade, C. and Hoefel, R.P.F. (2010) 'On Indoor Coverage Models for Industrial Facilities', in.

Andrade, C.B. and Hoefel, R.P.F. (2010) 'IEEE 802.11 WLANs: A comparison on indoor coverage models', in *CCECE 2010*. *CCECE 2010*, pp. 1–6. Available at: https://doi.org/10.1109/CCECE.2010.5575205.

Bokeh (no date) *categorical_heatmap.py*. Available at: https://docs.bokeh.org/en/latest/docs/gallery/categorical_heatmap.html (Accessed: 7 September 2022).

Eurostat (2022) *Level of internet access - households*, *Statistics | Eurostat*. Available at: https://ec.europa.eu/eurostat/databrowser/view/tin00134/default/table?lang=en (Accessed: 29 June 2022).

Lloret, J. *et al.* (2004) 'A fast design model for indoor radio coverage in the 2.4 GHz wireless LAN', in *1st International Symposium onWireless Communication Systems, 2004. 1st International Symposium onWireless Communication Systems, 2004.*, pp. 408–412. Available at: https://doi.org/10.1109/ISWCS.2004.1407279.

Mardiris, V. and Chatzis, V. (2016) 'A Configurable Design for Morphological Erosion and Dilation Operations in Image Processing using Quantum-dot Cellular Automata', *Journal of Engineering Science and Technology Review*, 9, pp. 25–30. Available at: https://doi.org/10.25103/jestr.092.05.

Morphological Image Processing (no date) *Additional Materials: Image Processing and Analysis*. Available at: https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm (Accessed: 7 September 2022).

Remley, K.A., Anderson, H.R. and Weisshar, A. (2000) 'Improving the accuracy of ray-tracing techniques for indoor propagation modeling', *IEEE Transactions on Vehicular Technology*, 49(6), pp. 2350–2358. Available at: https://doi.org/10.1109/25.901903.

Silva, H. (2022) 'Python notebook that simulates WiFi signals over a given floorplan'. Available at: https://github.com/henrique-silva/wifi-heatmap (Accessed: 27 April 2022).

Thresholding — skimage v0.19.2 docs (no date). Available at: https://scikit-image.org/docs/stable/auto_examples/applications/plot_thresholding.html (Accessed: 7 September 2022).