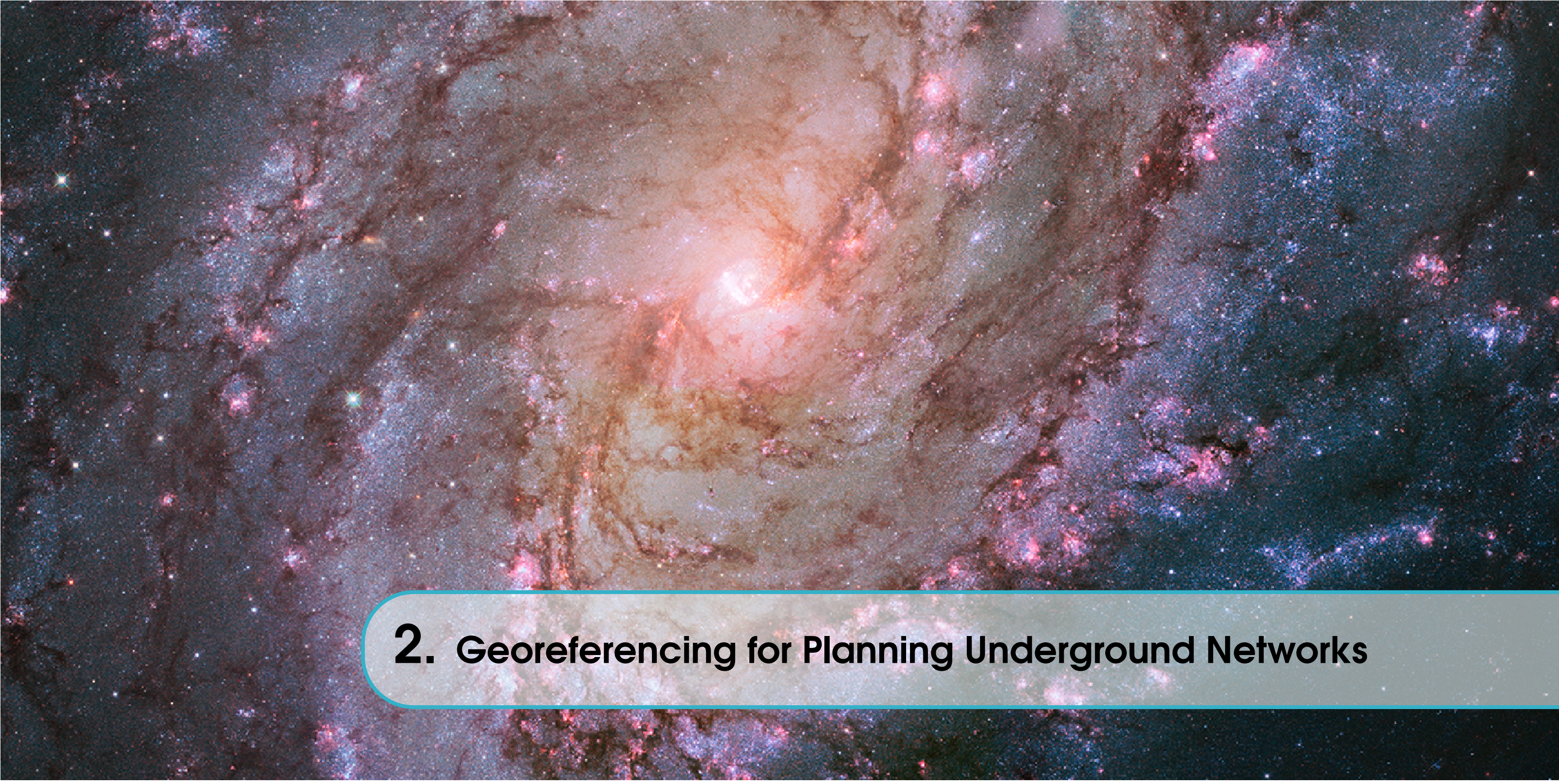


# 2. Georeferencing for Planning Underground Networks

Ph.D. Esteban Inga Ortega is Professor of the Master's Program in ICT for Education at Universidad Politécnica Salesiana (Ecuador).
Email: einga@ups.edu.ec
https://orcid.org/0000-0002-0837-0642
**DOI:** 10.17163/abyaups.44.347

## 2.1 Introduction

Often, problems related to planning electrical networks, drinking water (sewage), data, and transportation use ideal models that are not geo-referenced. However, the present work exposes a suitable and innovative methodology to achieve data analytics from OSM files. This geographic information file can be freely downloaded from $https://www.openstreetmap.org/$ (1).

Then, applying a geo-referenced model to network planning provides characteristics of the environment that are not present in a model of x and y coordinates. In this sense, using longitude and latitude allows a location with less error for distance and location calculations that are utilized for the optimal location of wireless sensors (smart meters), electrical transformers, determination of the route for burying power and data lines, as well as using the information to determine population growth in a specific area, evaluation of a rescue zone or planning geographical areas for vaccination campaigns, among other options, (2)–(4).

Over time, scientific publications seek to innovate concerning previous work by the scientific community. Thus, applications in planning electrical distribution networks based on the use of information from OSM perform modeling for the deployment of underground networks. Simulation tools such as Cymdist (5)–(7) generally evaluate the deployed model. In this way, the aim is to serve the deployment with the minimum cost for using the electric cable and considering variables such as voltage drops (8), (9).

In addition, the location of the electric vehicle charging center or the mass public transport route evaluation will require modeling in a geo-referenced scenario capable of identifying the optimal location and considering the microscopy of vehicular traffic (10). Public transportation presents management problems; therefore, works propose models for creating and using a specific database contemplating a dynamic spatial analysis of the public transportation network to optimize public transportation schedules during peak hours (1), (11).

The problem is not knowing the location of public green spaces required for family space activities. We have proposed articulating different tools to distinguish public and private green spaces considering a Bayesian hierarchical model and OSM data from OpenStreetMap (12). Work to reduce the risk of catastrophes includes applications based on OpenStreetMap that involve sketching methods and possible flood routes (13).

This paper highlights georeferencing as a technique that integrates geographic information and analytical data for decision-making in various areas of knowledge. Using georeferencing in conjunction with data analytics allows the identification of spatial patterns and the visualization of information in a more accurate and detailed way, which can contribute to process optimization and decision-making in various fields of application. In this sense, the OpenStreetMap OSM file is a source of geographic data that can be used to generate detailed and accurate maps of different world regions. The presented figure 2.1 shows the map of a part of Turkey with the selection of dwellings obtained from a data analytics process. In addition, a feasible wireless sensor connectivity mesh and the routing between sensors considering the maximum distance constraint are presented.

Henceforth, the present article is organized as follows: Section 2 briefly reviews related work. Section 3 presents the traditional problem formulation and the methodology to solve it. Section 4 contains the results analysis and the proposed model's validation. Finally, in section 5, conclusions are presented.

## 2.2 Related Works

Previous works have presented an electrical network from an OSM file articulated to simulation processes. Additionally, a heuristic technique using Matlab is evidenced (14). The synthesis of geospatial data morphologically intended for the transfer and quality control of urban form is presented as a contribution of georeferencing

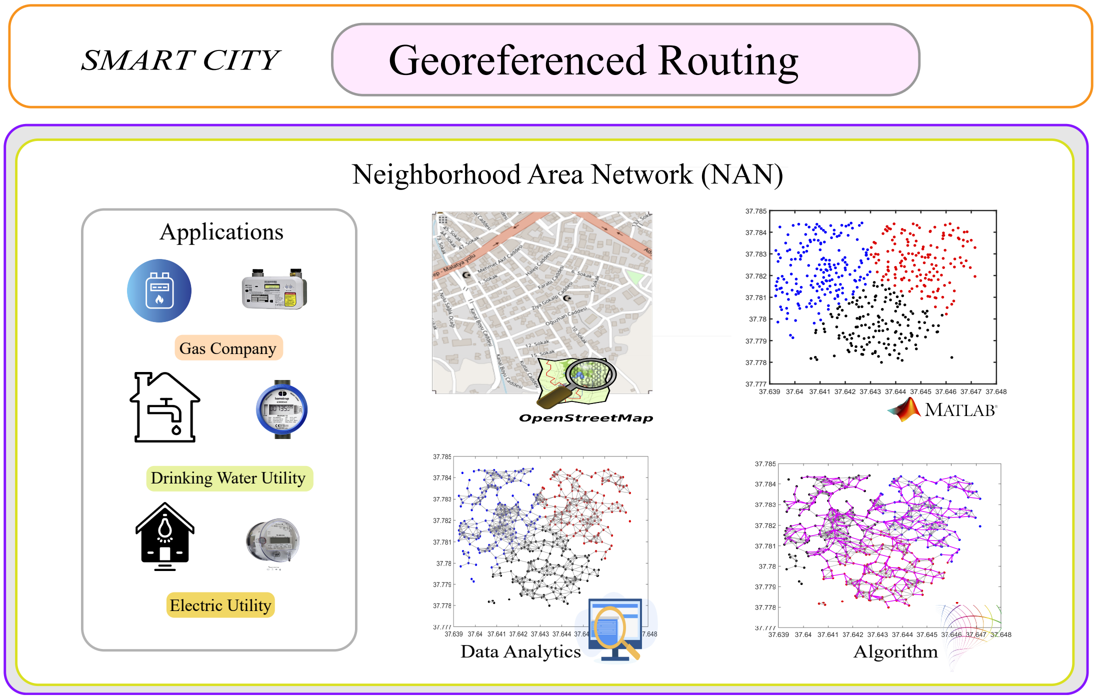



Figure 2.1: Georeferencing approach to solve problems on real scenarios

for development prediction and urban planning (15). A detailed schedule-based transit network is proposed to measure accessibility to hospitals based on transit in a growing city (16). Urban heritage management will require examination of the spatial distribution of buildings as a tangible aspect of the settlement places through spatial analysis and thus facilitate the processes and time reduction (17).

In the past, IEEE models were used to evaluate an electrical network; however, innovations involving geo-referenced scenarios are being introduced. These studies are accompanied by simulators such as Cymdist to verify that the expansion models meet scalability, reliability, resilience, and efficiency (6), (18). It has evidenced methods to detect traffic rules at intersections using GPS traces to assist location-based applications in the context of smart cities, such as accurately estimating travel time and fuel consumption from a starting point to a destination. Therefore, it proposes an automatic, fast, scalable, and inexpensive way to identify the type of intersection control, such as traffic lights and stop signs (19).

A combination of remote sensing data and statistical methods to estimate parking areas is proposed to solve the problem of adequate parking. Parking spaces and other traffic zones are detected by considering aerial images; furthermore, an obstacle model is estimated using parking zones detected from OpenStreetMap data. A relationship is found between length, street type, and parking zone obtained (20), (21). The relationship between cartography and urban management provides an option for the study of trends for decision-making by professionals in charge of drawing buildings with data from OpenStreetMap (OSM) (22). The management of geo-referenced information may include information that needs to be verified due to possible failures in the on-site visit; previous works evaluate labels and filters to evidence buildings in specific areas (23). Articulating raster maps into a single system could be a non-trivial task for institutions handling geo-referenced information and requiring integration into a geographic information system (GIS) because of scarce metadata (24).

Table 2.1 presents a summary of the contributions to the use of OpenStreetMap for different georeferencing applications. The applications are various, so it was necessary to articulate a bibliometric analysis from VosViewer.

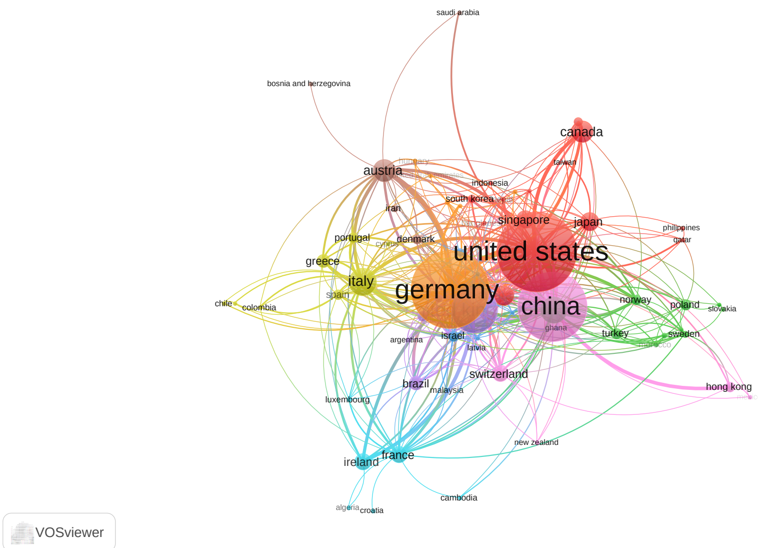
Table 2.1: Summary of related works.

| | | Applications | | | | | |
|---|---|---|---|---|---|---|---|
| Author, Year | Objectives | Electricity | Drinking Water | Data | Gas | Transport | Other |
| Garcia, 2023 [6] | Power Network Planning | ✓ | | ✓ | | | |
| Kim, 2023 [16] | Healthcare accessibility | | | ✓ | | ✓ | ✓ |
| Wu, 2023 [15] | Urban Development | | | ✓ | | ✓ | ✓ |
| Gaugl, 2023 [14] | Power Network Planning | ✓ | | ✓ | | | |
| Kersapati, 2023 [17] | Urban Management | | | ✓ | | ✓ | ✓ |
| Kim, 2022 [25] | Small unmanned aircraft | | | ✓ | | | ✓ |
| Song, 2022 [23] | Remote sensing - Urban planning | | | ✓ | | | ✓ |
| Hacar, 2022 [22] | Urban planning | | | ✓ | | | ✓ |
| Hellekes, 2022 [20] | Urban Planning | | | ✓ | | ✓ | ✓ |
| Zourlidou, 2022 [19] | Traffic Engineering | | ✓ | ✓ | | | ✓ |
| Milleville, 2022 [24] | Gerreferencing | | | ✓ | | | ✓ |
| Present Work | Wireless Sensor Network | ✓ | | ✓ | | | ✓ |

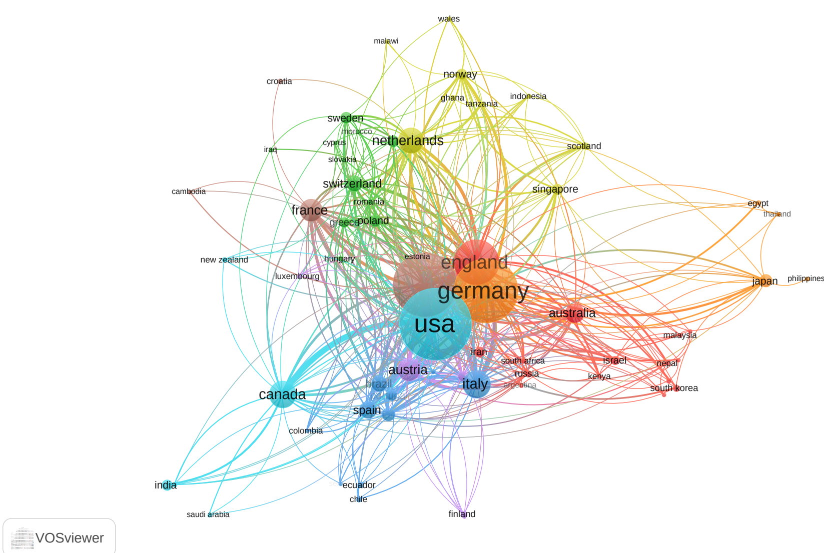
The sample used for the bibliometric analysis with VosViwer corresponds to 2000 scientific documents from Scopus and 1361 scientific documents from the Web of Science. Consequently, the bibliometric analysis corresponding to countries and authors allows us to identify the relevance concerning the amount of research using geo-referenced scenarios, accurately identifying the countries of origin and the authors that stand out. The information can be contrasted with various applications where experimentation is desired.

Figure 2.2 shows the list of countries for Scopus, among which the most crucial scientific impact are Germany, the U.S., China, the U.K., Italy, Netherlands, Austria, Canada, Ireland, and France. For Web of Science, countries such as the U.S., U.K., Germany, China, Canada, Netherlands, France, Italy, Austria, and Ireland are predominant.

Figure 2.3 shows the impact of the authors with relevance in the work of geo-referenced scenarios. Scopus authors are presented as Zipf Alexander, Boeing G, Neis P, Fan H, Jokar Arsanjani, and for Web of Science, relevant authors are Haklay, Mordechai; Zipf, Alexander; Neis, Pascal; Boeing, Geoff, Arsanjani, Jamal Jokar.

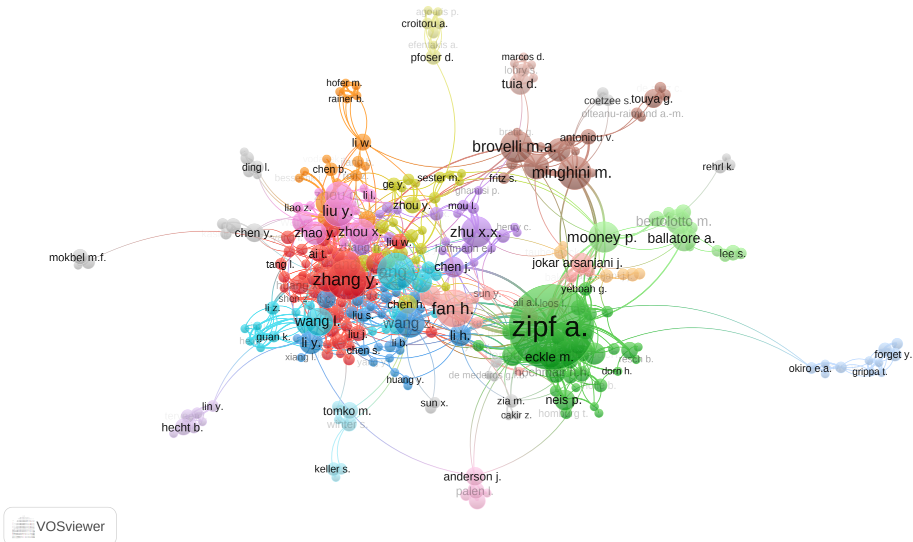



(**a**) Scopus - Countries

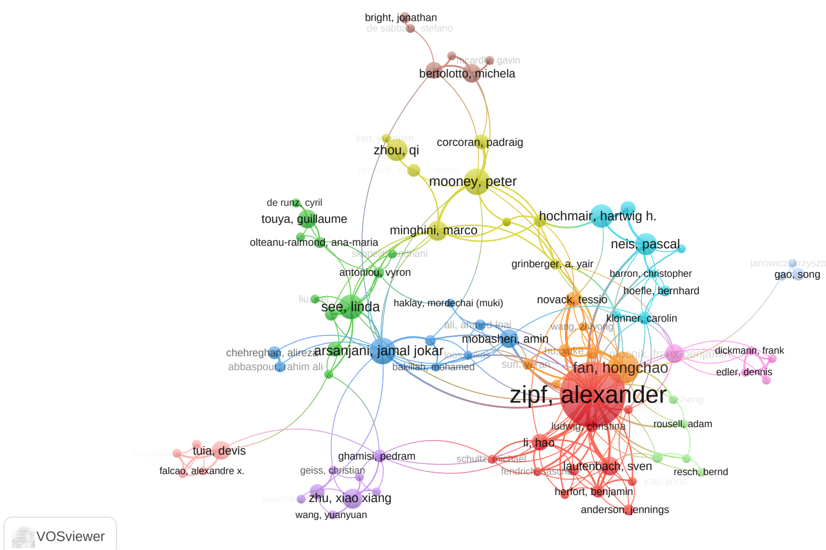



(**b**) Web of Science - Countries

Figure 2.2: Bibliometric Analysis of Countries. –(**a**) Scopus. (**b**) Web of Science. Source: Authors.




(**a**) Scopus - Authors




(**b**) Web of Science- Authors

Figure 2.3: Bibliometric Analysis of Authors. –(**a**) Scopus. (**b**) Web of Science. Source: Authors.

Using VosViewer in the bibliometric analysis of scientific documents from Web of Science and Scopus, specifically in applying OpenStreetMap in geo-referenced scenarios, allows the visualization of the scientific production of countries and authors rigorously and objectively. This tool is essential for analyzing and identifying the main trends, patterns, and research areas in this area, which translates into valuable information for strategic decision-making in developing and implementing geo-referenced projects. In summary, the graphs obtained through VosViewer provide an overview of the current state of research in this area, contributing significantly to the advancement of science and technology.

## 2.3 Problem Formulation and Methodology

Wireless network planning is a fundamental problem in deploying communication infrastructures, which involves making strategic decisions to reduce costs and ensure adequate coverage. In this work, a wireless network planning model is proposed based on geographic data from OpenStreetMap, which considers population growth in a scalable way and the exact location of each wireless sensor. The developed model uses an algorithm created with Matlab on a computer with an Intel Xeon 2.6GHz processor and 128GB RAM to process the data from an OSM extension file. The latitude and longitude location of the houses are used as the location of the wireless sensors.

The algorithm has as a constraint the maximum distance, dmax=45 meters, which can be modified according to the needs of the application. A feasible mesh of possible connectivity links is generated, and the internal heuristic employs Dijkstra's algorithm to find the minimum spanning tree with the distance constraint. It is important to note that there may be unconnected sensors because the maximum distance will place some sensors farther away than the allowed distance.

The proposed methodology will create an efficient and scalable wireless network planning model, reducing costs in deploying communications infrastructure and improving the quality of services offered to the population. Therefore, data analytics techniques will generate detailed maps of the region of interest to implement the proposed model. A data collection and cleaning process will be carried out to generate an OSM file containing accurate and updated information on the dwellings and existing infrastructure in the region. Subsequently, the algorithm developed for the location of wireless sensors will be used, considering the maximum distance restriction and the generation of a feasible mesh of possible connectivity links. The internal heuristics will employ Dijkstra's algorithm to find the minimum spanning tree with the distance constraint. Table 2.2 details the variables used in the equations.

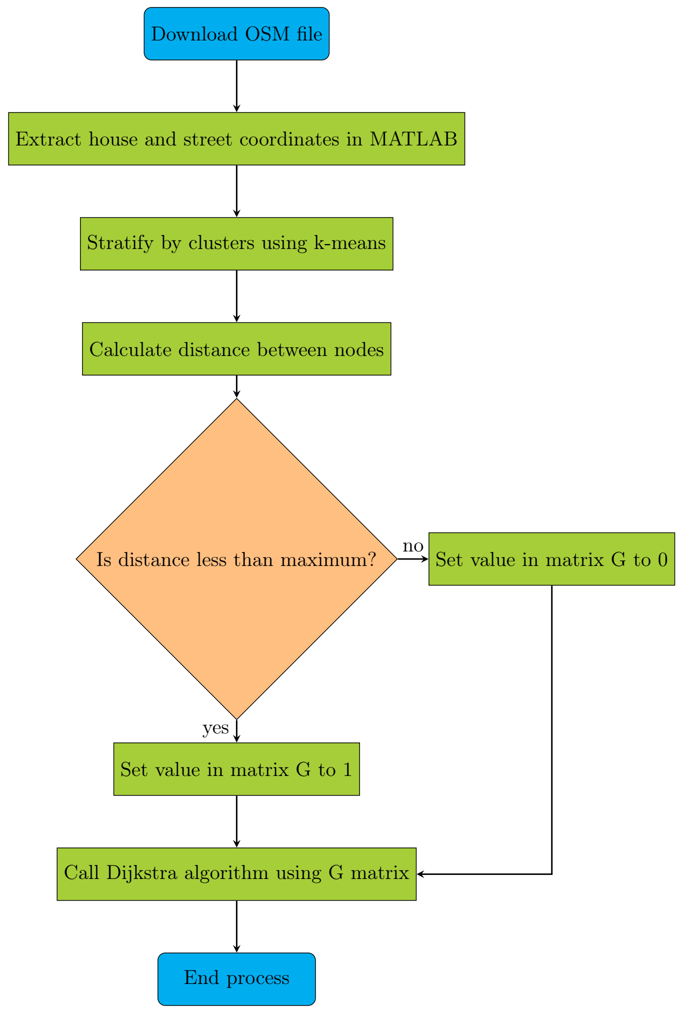



Figure 2.4: Methodology to perform data analytics on data from an OSM file.

The Earth's shape is irregular and approximately spherical, and its surface is curved. The Haversine formula is commonly used to calculate the distance between two points on the Earth's surface accurately. This equation considers the Earth's curvature and allows for accurate distance calculations in a geo-referenced system. The distance results are usually reported in kilometers. Equations 2.1 and 2.2 provide the mathematical expressions for calculating distances using the Haversine formula for two sets of coordinates.

Table 2.2: Variables related to Haversine equation

| Symbol | Description |
|---|---|
| $D_{ij}$ | Distance matrix nxn - km |
| $Ra$ | Earth curvature - km |
| $lat, lon$ | Latitude and longitude |
| $distH_{ij}$ | Haversine distance - km |
| $E$ | Haversine Equation |

$$D_{ij} = 2 * Ra * asin\sqrt{E} \tag{2.1}$$

$$E = sin^2\left(\frac{\Delta lat}{2}\right)^2 + cos\left(lat1\right) * cos\left(lat2\right) * sin^2\left(\frac{\Delta lon}{2}\right)^2 \tag{2.2}$$

This paper presents the possibility of performing data mining on an OSM file to determine the type of information available and how it could be used in basic, applied, and quantitative research. Mathworks presents a set of functions that serve for a process in Matlab, source: https://la.mathworks.com/matlabcentral/fileexchange/35819-openstreetmap-functions, and they are the following:

- assign _from _parsed.m
- debug_openstreetmap.m
- extract _connectivity.m
- get_unique_node _xy.m
- get_way_tag_key.m
- load_osm_xml.m
- main_mapping.m
- parse_openstreetmap.m
- parse_osm.m
- plot_nodes
- plot_road_network.m

- plot_route.m
- plot_way.m
- route_planner.m
- show_map.m
- usage_example.m

However, more than the exposed functions are required because they generate errors. After all, new functions are requested in Matlab that can be downloaded from the Internet by searching for them with the following names:

- xml2struct_fex28518.m
- lat_lon_proportions.m
- takehold.m
- plotmd.m
- textmd.m
- restorehold.m
- givehold.m

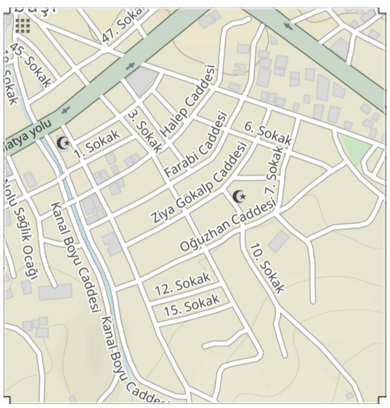
The features of each OpenStreetMap layer are detailed below:

- Standard layer: The Standard layer is the default base map layer of OpenStreetMap, which contains basic information such as roads, buildings, water bodies, and administrative boundaries. This layer is designed to provide a general view of a specific area and is suitable for navigation and available mapping applications.
- CyclOSM layer: The CyclOSM layer is specifically designed for cyclists and displays relevant information, such as bike routes, bike lanes, and bike parking facilities. It also includes information on points of interest for cyclists, such as bike shops and repair facilities.
- Cyclist Map layer: Similar to the CyclOSM layer, the Cyclist Map layer is designed for cyclists and displays information specific to them, such as bike routes, bike lanes, and bike parking facilities. It also focuses on safety and displays areas where cyclists should exercise caution.
- Transportation Map layer: The Transportation Map layer is designed to display information about public transportation, such as bus routes, train and metro stations, and bus stops. It may also include information about parking facilities and nearby points of interest.
- ÖPNVKarte layer: Similar to the Transportation Map layer, the ÖPNVKarte layer focuses explicitly on public transportation in Germany and displays information about bus routes, trams, metro, and trains. It may also include information about nearby points of interest.
- Humanitarian layer: The Humanitarian layer is designed for use in humanitarian crises such as natural disasters or armed conflicts. It contains relevant information for humanitarian aid, such as the location of shelters, hospitals, and water stations. It may also include information about roads and evacuation routes.

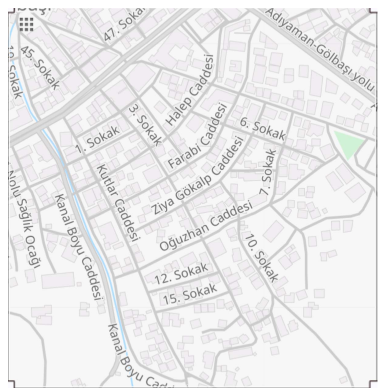
Table 2.3 presents the parameters used in the proposed georeferencing application scenario. Each variable will be modified according to the values retrieved from the OSM file. If the selected area is vast, the information on houses and streets will be increased. However, the computational time will also increase. Figure 2.5 shows the characteristic of the layers available in OpenStreetMap.


(a) Standard Layer

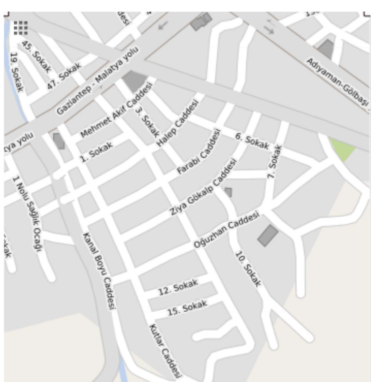

(b) CyclOSM Layer



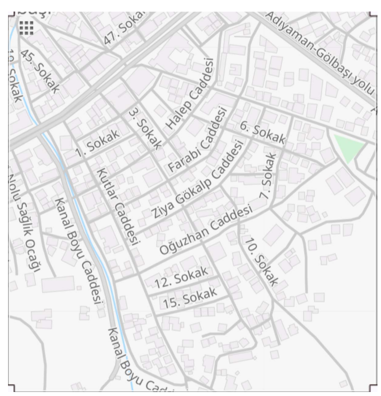

(c) Cyclist Map Layer




(d) Transportation Map Layer




(e) ÖPNVKarte Layer




(f) Humanitarian Layer

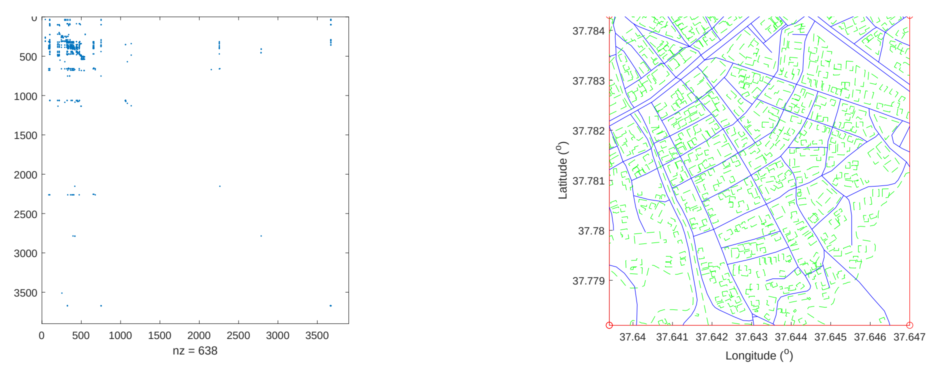
Figure 2.5: OpenStreetMap Layers Comparison: Standard, CyclOSM, Cyclist Map, Transportation Map, ÖPNVKarte, and Humanitarian

Table 2.3: Simulation parameters.

| Description | Details |
|---|---|
| Location map | Ziya Gökalp Caddesi, Gölbaşı, Adıyaman, Southeastern Anatolia Region, 02500, Turkey |
| Latitude | [37.7781 37.7844] - Bottom to top |
| Longitude | [37.6393 37.6469] - Left to right. |
| User household | 497 |
| Geographic area | 503.705,95 $m^2$ |
| Coverage distance | 0.045 km - constraint |
| Streets Sets | 67 |
| Topology evaluated | Tree |
| IEEE 802.15.4 Technology | Topology type Star, Tree |
| Candidate sites location | Location of the houses |

## 2.4 Matlab Coding and Results Analysis.

Based on the data available in an OpenStreetMap OSM file, it is necessary to exemplify a specific application considering a geo-referenced scenario. Therefore, once all the functions are in the same folder and $*.osm$ file, you can run the Matlab script to get the first approach to map retrieval in Matlab. Figure 2.6 a) shows a graph with the connectivity matrix, with the area's connections in blue and the non-connectivity of streets or avenues in white. The figure shows the initial behavior of the articulation between Matlab and OpenStreetMap. It can be seen that figure 2.6 b) has text that does not allow us to appreciate the map.




(a) No Zeros chart (b) Stage map

Figure 2.6: The original scenario articulated Matlab & OpenStreetMap considering the functions available in Mathworks and modified them with the required new functions

The script has been modified from the original Mathworks version authored by Ioannis Filippidis in 2010 and is presented below.

```
clc; clear all; close all;
warning('off','all');
```

```
3  %====================================
4  openstreetmap_filename = 'turkey.osm';%Scenario (Chosen City)
5  [parsed_osm, osm_xml] = parse_openstreetmap(openstreetmap_filename);
6  %Retrieve OSM information
7  %Connectivity Matrix and Intersections
8  [connectivity_matrix, intersection_node_indices] =
      extract_connectivity(parsed_osm);
9  %Clean duplicate data
10 intersection_nodes = get_unique_node_xy(parsed_osm,
      intersection_node_indices);%
11 start = 1; % node global index
12 target = 9;
13 dg = or(connectivity_matrix, connectivity_matrix.'); %sparse matrix
14 [route, dist] = route_planner(dg, start, target);
15 fig = figure;
16 ax = axes('Parent', fig);%Axis
17 hold(ax, 'on')%Hold
18 plot_way(ax, parsed_osm)
19 plot_route(ax, route, parsed_osm)
20 only_nodes = 1:10:10000; % Alert! not all nodes, to reduce graphics
      memory & clutter
21 plot_nodes(ax, parsed_osm, only_nodes)
22 %====================================
23 % Page setup before printing the figure in PDF format
24 figure(1);
25 hold(ax, 'off')
26 h=gcf;
27 set(h,'PaperPositionMode','auto');
28 set(h,'PaperType','A4');
29 set(h,'PaperOrientation','landscape');
30 set(h,'Position',[10 0 500 800]);
31 set(h, 'InvertHardcopy', 'off')
32 fig = gcf;
33 fig.Color = 'white';
34 print -dpdf -r800 figure1_12
35 %====================================
36 figure(2);
37 hold(ax, 'off')
38 h=gcf;
39 set(h,'PaperPositionMode','auto');
40 set(h,'PaperType','A4');
41 set(h,'PaperOrientation','landscape');
42 set(h,'Position',[10 0 500 800]);
43 set(h, 'InvertHardcopy', 'off')
44 fig = gcf;
45 fig.Color = 'white';
46 print -dpdf -r800 figure1_13
```

Next, the code is modified to include the background image in PNG format. It is important to note that the capture of the photo image must have been previously captured and saved from OpenStreetMap. After line 4, insert the next code:

- map_map img_filename = 'figure1_standard.png';

Afterward, line 19 should be modified by the following code

- plot_way(ax, parsed_osm,map_img_filename)

Once the above changes have been made, it can be seen in the figure 2.7 that the png figure is at the bottom of the road and housing map. The figure shows the houses in green color found in the OSM file. The blue color indicates the main streets of the selected area. It is important to note that not all regions are complete and should be observed in the standard layer if the map has houses.

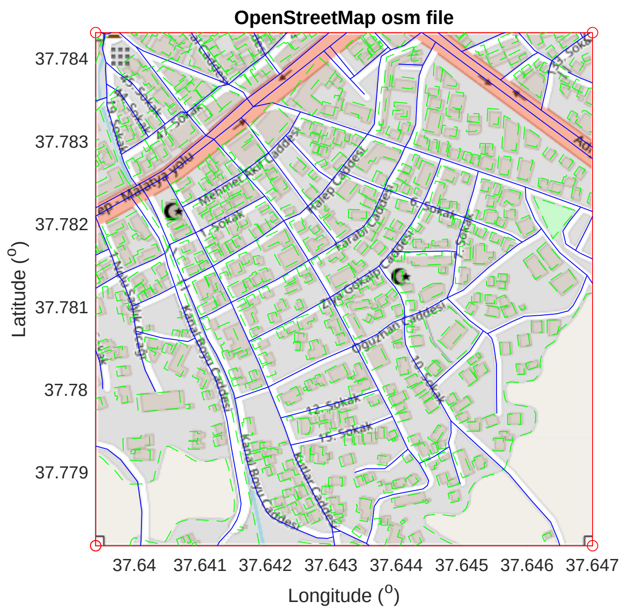



Figure 2.7: Scenario with the background image of the selected map.

The next step will be to modify the plot_way.m function to identify the information about the houses and roads. For this purpose, colors are chosen to identify each set of data. The new code for the plot_way.m function is detailed below.

```
%plot_way.m function
function [] = plot_way(ax, parsed_osm, map_img_filename)
if nargin < 3
    map_img_filename = [];
end
[bounds, node, way, ~] = assign_from_parsed(parsed_osm);
disp_info(bounds, size(node.id, 2), size(way.id, 2))
show_ways(ax, bounds, node, way, map_img_filename);

function [] = show_ways(hax, bounds, node, way, map_img_filename)
show_map(hax, bounds, map_img_filename)
house=[];zi=1;zj=1;zk=1;House=[];
key_catalog = {};
for i=1:size(way.id, 2)
    [key, val] = get_way_tag_key(way.tag{1,i} );
    % find unique way types
    if isempty(key)
    elseif isempty( find(ismember(key_catalog, key) == 1, 1) )
        key_catalog(1, end+1) = {key};
    end
    % way = highway or amenity ?
    flag = 0;
    switch key
        case 'highway'
            flag = 1;
            % bus stop ?
            if strcmp(val, 'bus_stop')
                disp('Bus stop found')
            end
        case 'amenity'
            % bus station ?
            if strcmp(val, 'bus_station')
                disp('Bus station found')
            end
            %===========================
        case 'building'
            % houses
            flag = 2;
            if strcmp(val, 'yes')
                disp('House')
            end
        case 'alt_name'
            % houses
            flag = 3;
            if strcmp(val, 'yes')
                disp('Extra Via')
            end
            %===========================
        otherwise
            disp('way without tag.')
    end
    % plot highway
```

```
    way_nd_ids = way.nd{1, i};
    num_nd = size(way_nd_ids, 2);
    nd_coor = zeros(2, num_nd);
    nd_ids = node.id;
    for j=1:num_nd
        cur_nd_id = way_nd_ids(1, j);
        if ~isempty(node.xy(:, cur_nd_id == nd_ids))
            nd_coor(:, j) = node.xy(:, cur_nd_id == nd_ids);
        end
    end
    % remove zeros
    nd_coor(any(nd_coor==0,2),:)=[];
    if ~isempty(nd_coor)
        % plot way (highway = blue, other = green)
        if flag == 1
            plot(hax, nd_coor(1,:), nd_coor(2,:), '-','LineWidth',1,'color',[0.74 0.33 0.18])% plot streets
            streets{1,zj}=[nd_coor(1,:); nd_coor(2,:)];
            zj=zj+1;
        else
        end
        if flag == 2
            plot(hax, nd_coor(1,:), nd_coor(2,:), '-','LineWidth',1,'color',[0.37 0.41 0.62]);
            plot(hax, nd_coor(1,end), nd_coor(2,end), '<','markersize',3,'color',[128/255 64/255 64/255],'markerfacecolor',[255/255 127/255 39/255]);
            house(zi,:)=[nd_coor(1,end) nd_coor(2,end)];
            House{1,zi}=[nd_coor(1,:); nd_coor(2,:)];
            zi=zi+1;
        end
        if flag == 3
            plot(hax, nd_coor(1,:), nd_coor(2,:), '-','LineWidth',1,'color',[0.74 0.33 0.18]);
            streets2{1,zk}=[nd_coor(1,:); nd_coor(2,:)];
            zk=zk+1;
        end
    end
    %waitforbuttonpress
end
disp(key_catalog.')

function [] = disp_info(bounds, Nnode, Nway)
disp( ['Bounds: xmin = ' num2str(bounds(1,1)),...
    ', xmax = ', num2str(bounds(1,2)),...
    ', ymin = ', num2str(bounds(2,1)),...
    ', ymax = ', num2str(bounds(2,2)) ] )
disp( ['Number of nodes: ' num2str(Nnode)] )
disp( ['Number of ways: ' num2str(Nway)] )
```

In addition, to remove the title of the figures, the function show_map.m must be entered, and the last line of the Matlab code must be disabled at the end of the line %title(ax, 'OpenStreetMap osm file').

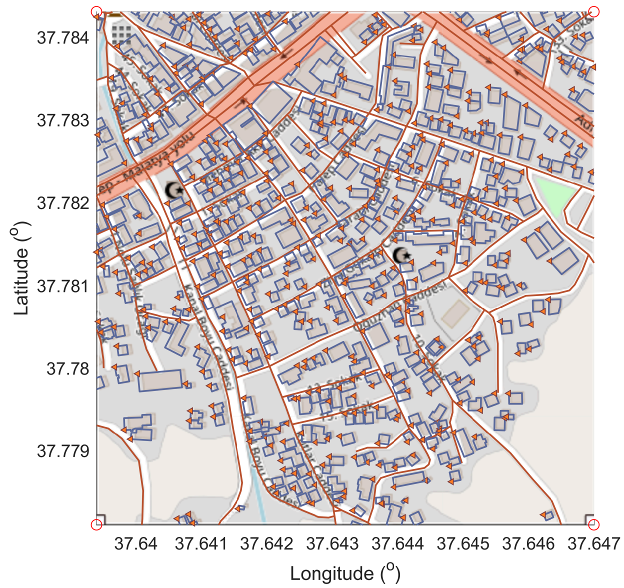



Figure 2.8: The scenario considers dwellings, roads, and the point of the residence closest to a street.

Generally, it is required to work with the information separately according to the application to be developed. The main algorithm retrieves information on houses, roads, and recreational areas. A triangle is placed at the point of each home. In this sense, the main algorithm and the plot_way function are modified and are now called new_plot_way.

The MATLAB code starts by clearing the console (clc), clearing all variables (clear all), and closing all open figures (close all). Then, you set the name of the map file to use (openstreetmap_filename) and the map image (map_img_filename). Next, the OSM (OpenStreetMap) information is extracted from the file, and the connectivity matrix and intersection node indices are obtained. Duplicate data is removed, and the start and end nodes are established.

The shortest path between these nodes is then found using Dijkstra's algorithm, and a figure is created to show the map and route. The working area is set, and data not in the specified region is removed. A legend is set for the figure, and the page is set up before printing the figure in PDF format. In summary, the code processes OSM data and displays the shortest route between two nodes on a map, which can be helpful for transportation network analysis and urban planning.

```
clc; clear all; close all;
warning('off','all');
```

```
%=====================================
openstreetmap_filename = 'turkey.osm';%Scenario (Chosen City)
%Image
map_img_filename = 'figure1_estandar.png';%Imagen PNG o EPS
[parsed_osm, osm_xml] = parse_openstreetmap(openstreetmap_filename); %Retrieve OSM information
%Connectivity Matrix and Intersections
[connectivity_matrix, intersection_node_indices] = extract_connectivity(parsed_osm);
%Clean duplicate data
intersection_nodes = get_unique_node_xy(parsed_osm, intersection_node_indices);%
start = 1; % node global index
target = 9;
dg = or(connectivity_matrix, connectivity_matrix.'); %sparse matrix
[route, dist] = route_planner(dg, start, target);
fig = figure;
ax = axes('Parent', fig);%Axis
hold(ax, 'on')%Hold
new_plot_way(ax, parsed_osm,map_img_filename)%Include Image
plot_route(ax, route, parsed_osm)
only_nodes = 1:10:10000; % Alert! not all nodes, to reduce graphics memory & clutter
plot_nodes(ax, parsed_osm, only_nodes)
%=====================================
% geo-referenced Scenario Work Area Grid
lonlim=[37.6393 37.6469]; % Left-Right X Limits
latlim=[37.7781 37.7844]; % Lower-Upper Y Limits
%=====================================
% Vector retrieved from OSM information
[House,house,streets,z1,z2,z3,z4]=new_plot_way(ax, parsed_osm, map_img_filename);
[House]=delete_data(lonlim,latlim, House');
House=House';
streets=[streets];
legend([z1,z2,z3,z4],'Street','Recreation','House','Reference','fontname','times new roman','fontsize',13,'location','SO','orientation','horizontal');
% Page setup before printing the figure in PDF format
figure(1);
hold(ax, 'off'),box('on');
h=gcf;
set(h,'PaperPositionMode','auto');
set(h,'PaperType','A4');
set(h,'PaperOrientation','landscape');
set(h,'Position',[10 0 500 800]);
set(h, 'InvertHardcopy', 'off')
fig = gcf;
fig.Color = 'white';
print -dpdf -r800 figure1_14
figure(2);
hold(ax, 'off'),box('on');
```

```
h=gcf;
set(h,'PaperPositionMode','auto');
set(h,'PaperType','A4');
set(h,'PaperOrientation','landscape');
set(h,'Position',[10 0 500 800]);
set(h, 'InvertHardcopy', 'off')
fig = gcf;
fig.Color = 'white';
print -dpdf -r800 figure1_17
```

The MATLAB algorithm "new_plot_way" is a function that receives three input arguments: the first argument is the "ax" object representing the coordinate system in which the map will be drawn, the second argument is the "parsed_osm" object representing the map data file in OSM format. The third argument is the filename of the map image. This function uses the "assign_from_parsed" function to extract node and path information from the OSM file and then calls the "show_ways" function to draw the nodes and paths in the "ax" object. The "show_ways" function uses the "show_map" function to display the map image in the "ax" object.

Then, the "show_ways" function traverses the paths in the "way" entity to determine their type and draws them in the "ax" object with a different color for each class. If a path is a house, it also draws a marker at the last position of the path and stores the place in a "house" array. Finally, the function "new_plot_way" returns the objects "house," "house," "streets," "z1", "z2", "z3," and "z4," containing information about the drawn nodes and paths.

The algorithm demonstrates a practical application of geospatial data processing and map visualization in MATLAB. The implementation of the algorithm uses structured programming techniques. The use comprises control structures such as cycles, case selection, and nested functions that perform specific tasks. Array indexing is also used, and attention is paid to code efficiency to avoid redundancy in data representation and improve performance. The algorithm is easily understandable and modular, facilitating its maintenance and extension. Table 2.4 presents the variables used in the algorithm's 1 pseudocode, and the Matlab code is also presented below.

Table 2.4: Variables related to the new_plot_way algorithm

| **Name** | **Description** |
|---|---|
| $ax$ | Axis object of the plot |
| $parsed\_osm$ | Parsed OpenStreetMap data |
| $map\_img\_filename$ | Filename of the map image |
| $bounds$ | Boundary coordinates of the plot |
| $node$ | Node coordinates |
| $way$ | Way information |
| $House$ | List of house coordinates |
| $house$ | Temporary list of house coordinates |
| $streets$ | List of street coordinates |
| $key\_catalog$ | Catalog of unique way tags |
| $i$ | Loop index variable |
| $key$ | Current way tag key |
| $val$ | Current way tag value |
| $flag$ | Flag variable used for differentiating between different types of ways |
| $nd\_coor$ | Coordinate of the current node |

**Algorithm 1** Function new_plot_way

```
function SHOW_WAYS(hax, bounds, node, way, map_img_filename)
    show_map(hax, bounds, map_img_filename)
    house ← []
    House ←
    streets ←
    key_catalog ←
    for i ← 1 to size(way.id, 2) do
        key, val ← get_way_tag_key(way.tag1, i)
        if ∼ isempty(key) & isempty(find(ismember(key_catalog, key) == 1, 1)) then
            key_catalogend + 1 ← key
        end if
        if key ==′ highway′ then
            if strcmp(val,′ bus_stop′) then
                continue
            else
                flag ← 1
                streetsend + 1 ← node.xy(:, way.nd1, i)
            end if
        end if
        if key ==′ amenity′ then
            if strcmp(val,′ bus_station′) then
                continue
            else
                flag ← 2
            end if
        end if
        if key ==′ building′ then
            if strcmp(val,′ yes′) then
                flag ← 3
                nd_coor ← node.xy(:, way.nd1, i)
                house(end + 1, :) ← [nd_coor(1, end), nd_coor(2, end)]
                Houseend + 1 ← nd_coor
            else
                continue
            end if
        end if
        if key ==′ alt_name′ then
            if strcmp(val,′ yes′) then
                flag ← 4
                streetsend + 1 ← node.xy(:, way.nd1, i)
            else
                continue
            end if
        end if
    end for
end function
```

```
function [House,house,streets,z1,z2,z3,z4] = new_plot_way(ax,
    parsed_osm, map_img_filename)
if nargin < 3
```

```
    map_img_filename = [];
    house=[];House=[];streets=[];streets2=[];z1=[];z2=[];z3=[];z4
       =[];
end
[bounds, node, way, ~] = assign_from_parsed(parsed_osm);
% disp_info(bounds, size(node.id, 2), size(way.id, 2))
[House,house,streets,z1,z2,z3,z4]=show_ways(ax, bounds, node, way,
    map_img_filename);
function [House,house,streets,z1,z2,z3,z4] = show_ways(hax, bounds,
    node, way, map_img_filename)
show_map(hax, bounds, map_img_filename)
house=[];zi=1;zj=1;zk=1;House=[];
% plot(node.xy(1,:), node.xy(2,:), '.r','markersize',10);
key_catalog = {};
for i=1:size(way.id, 2)
    [key, val] = get_way_tag_key(way.tag{1,i} );
    % find unique way types
    if isempty(key)
        %
    elseif isempty( find(ismember(key_catalog, key) == 1, 1) )
        key_catalog(1, end+1) = {key};
    end
    % way = highway or amenity ?
    flag = 0;
    switch key
        case 'highway'
            flag = 1;
            % bus stop ?
            if strcmp(val, 'bus_stop')
                disp('Bus stop found')
            end
        case 'amenity'
            % bus station ?
            flag = 2;
            if strcmp(val, 'bus_station')
                disp('Bus station found')
            end
        case 'building'
            % houses
            flag = 3;
            if strcmp(val, 'yes')
                disp('I found a house')
            end
        case 'alt_name'
            % houses
            flag = 4;
            if strcmp(val, 'yes')
                disp('Extra Via')
            end
        otherwise
            %              disp('Path.')
    end
```

```
    %plot highway
    way_nd_ids = way.nd{1, i};
    num_nd = size(way_nd_ids, 2);
    nd_coor = zeros(2, num_nd);
    nd_ids = node.id;
    for j=1:num_nd
        cur_nd_id = way_nd_ids(1, j);
        if ~isempty(node.xy(:, cur_nd_id == nd_ids))
            nd_coor(:, j) = node.xy(:, cur_nd_id == nd_ids);
        end
    end
    % remove zeros
    nd_coor(any(nd_coor==0,2),:)=[];
    if ~isempty(nd_coor)
        %plot way (highway = blue, other = green)
        if flag == 1
            z1=plot(hax, nd_coor(1,:), nd_coor(2,:), '-','LineWidth',1.25,'color',[185/255 122/255 87/255]);% Plot Street
            streets{1,zj}=[nd_coor(1,:); nd_coor(2,:)];
            zj=zj+1;
        else
                        z2=plot(hax, nd_coor(1,:), nd_coor(2,:), '-','LineWidth',1,'color',[0/255 162/255 232/255]);%Recreational areas
        end
        if flag == 3
            z3=plot(hax, nd_coor(1,:), nd_coor(2,:), '-','LineWidth',1,'color',[128/255 07/255 255/255]);%Plot Houses
                         z4=plot(hax, nd_coor(1,end), nd_coor(2,end), '<','markersize',4,'color',[1 0 0]);
            house(zi,:)=[nd_coor(1,end) nd_coor(2,end)];
            House{1,zi}=[nd_coor(1,:); nd_coor(2,:)];
            zi=zi+1;
        end
         if flag == 4
            plot(hax, nd_coor(1,:), nd_coor(2,:), '-','LineWidth',1,'color',[1 0 0]);
            streets2{1,zk}=[nd_coor(1,:); nd_coor(2,:)];
            zk=zk+1;
         end
    end
    %waitforbuttonpress
end
disp(key_catalog.')

function [] = disp_info(bounds, Nnode, Nway)
disp( ['Bounds: xmin = ' num2str(bounds(1,1)),...
    ', xmax = ', num2str(bounds(1,2)),...
    ', ymin = ', num2str(bounds(2,1)),...
    ', ymax = ', num2str(bounds(2,2)) ] )
disp( ['Number of nodes: ' num2str(Nnode)] )
disp( ['Number of ways: ' num2str(Nway)] )
```

The presented Matlab algorithm named delete_data uses an iterative approach to remove unwanted data from a geospatial dataset. First, boundaries are defined for longitude and latitude, which are assigned to two variables called "lonlim" and "latlim," respectively. Next, a list of intersection vectors "int" containing information about intersections between streets in a city or geographic area is traversed. At each iteration, the x and y coordinates of the street intersections are extracted and filtered to include only those within the previously defined longitude and latitude limits. The filtered data is stored in a "deleting" matrix to store the deleted data.

In summary, the algorithm uses an iterative filtering approach to remove unwanted data from a geospatial dataset based on the boundaries defined for longitude and latitude. The algorithm runs in a loop for each intersection vector "int," extracting and filtering street intersections' x and y coordinates. The filtered data is stored in a "deleting" array for further processing or deletion. This filtering approach can help remove noisy or irrelevant data in urban geography or visualize maps of specific geographic areas. Table 2.5 presents the variables used in the algorithm 2 that eliminates data outside the analysis area; the Matlab code is shown below.

Table 2.5: Variables related to the Function delete_data

| **Variable** | **Description** |
|---|---|
| *long* | A vector of longitudes |
| *lat* | A vector of latitudes |
| *int* | A cell array of street coordinates |
| *jg* | Index variable for iterating through *int* |
| *streetsx* | A vector of street longitudes |
| *streetsy* | A vector of street latitudes |
| *tot* | A matrix of selected street coordinates |
| *ui* | Index variable for iterating through *streetsx* or *tot* |
| *tot*2 | A matrix of selected street coordinates after both latitude and longitude filters |
| *deleting* | A cell array of deleted street coordinates |

**Algorithm 2** Function delete_data

```
function DELETE_DATA(long, lat, int)
    deleting ← []
    for jg ← 1 to length(int) do
        streetsx ← []; streetsy ← [];
        if length(cat(2, intjg, 1)) ≠ 0 then
            streetsx, streetsy ← cat(2, intjg, 1)(1, :), cat(2, intjg, 1)(2, :)
        end if
        tot ← []
        for ui ← 1 to length(streetsx) do
            if lonlim(1, 1) < streetsx(ui) < lonlim(1, 2) then
                tot ← [tot; streetsx(ui), streetsy(ui)]
            end if
        end for
        tot2 ← []
        for ui ← 1 to length(tot) do
            if latlim(1, 1) < tot(ui, 2) < latlim(1, 2) then
                tot2 ← [tot2; tot(ui, :)]
            end if
        end for
        deleting{jg, 1} ← [tot2(:, 1)′; tot2(:, 2)′]
    end for
    return deleting
end function
```

```
function [deleting]=delete_data(long,lat,int)
lonlim=[]; latlim=[];
lonlim=long; latlim=lat;
for jg=1:length(int)
    Xtr=[]; Ytr=[]; xeb=[]; yeb=[]; tx=[]; ty=[];
    streetsx=[]; streetsy=[]; delete=[];
    if length(cat(2,int{jg,1}))~=0
        streets_1=cat(2,int{jg,1});
        streetsx=streets_1(1,:);
        streetsy=streets_1(2,:);
    end
    Xtr=streetsx; Ytr=streetsy;
    xeb=Xtr'; yeb=Ytr';
    for ui=1:length(xeb)
        filtro1=xeb(ui);
        if filtro1>lonlim(1,1) && filtro1<lonlim(1,2)
            tx(ui)=filtro1;
        else
            tx(ui)=0;
        end
    end
    tot=[tx' yeb];
    if length(tot)~=0
        delete=find(tot(:,1)==0);
        tot(delete,:)=[];
        xeb=tot(:,1);
```

```
            yeb=tot(:,2);
        else
            xeb=[];
            yeb=[];
        end
        ty=zeros(1,length(yeb));
        for ui=1:length(yeb)
            filtro2=yeb(ui);
            if filtro2>latlim(1,1) && filtro2<latlim(1,2)
                ty(ui)=filtro2;
            else
                ty(ui)=0;
            end
        end
        tot2=[xeb ty'];
        if length(tot2)~=0
            delete2=find(tot2(:,2)==0);
            tot2(delete2,:)=[];
            xtr=tot2(:,1);
            ytr=tot2(:,2);
        else
            xtr=[];
            ytr=[];
        end
        xtr=xtr'; %stores the intersections within the scenario
        ytr=ytr'; %stores the intersections within the scenario
        deleting{jg,1}=[xtr;ytr];
    end
end
```

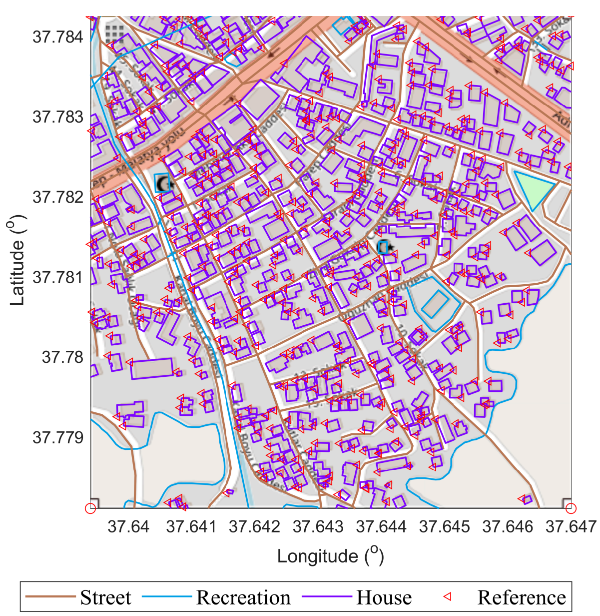



Figure 2.9: New scenario considering the location of housing, streets, and recreation areas.

Figure 2.9 represents the dwellings recovered from the OSM archive in purple. Each sector or stratum can be represented by color. The representation identifies a single color for didactic purposes. A correct planning process considers geo-referenced scenarios to determine the solution to a specific problem.

After evaluating the interface between OpenStreetMap and Matlab, we will load the information generated in the above file. For storing all the variables, you can type in the command window after obtaining the graphs the command: **save store.mat esc**. Then in a new script, we make an application to perform the routing of a set of sensors located in each house. Wireless communication is established with a maximum distance that must be met, in this case, 45 meters.

The algorithm in question uses MATLAB programming language to perform a series of operations on a data set stored in a file with a .mat extension. First, all variables in the current workspace are deleted, the screen is cleared, and all system warnings are disabled. Next, the store.mat file containing the data to be analyzed is loaded.

Subsequently, a figure is generated in which the data of the house variable is graphically represented, using the k-means algorithm to divide the data into three groups or clusters. Other symbols and colors represent the different clusters, and a reference point (centroids) is added for each cluster.

Then, a series of network analysis operations are performed using Dijkstra's technique to calculate the shortest path between all the network nodes. The distance between the different nodes is calculated using Haversine's formula, and a maximum length of 0.045 (45 meters) is set to establish the connections between the network nodes. Finally, the connections between the different nodes of the network are graphically represented, and the shortest path between each is calculated.

The presented MATLAB algorithm uses data and graph analysis techniques to graphically represent a data set and calculate the shortest path between its different nodes.

The generated between line 20 and line 40 of the code aims to calculate the Haversine distance between each pair of geographic coordinate points (latitude and longitude) provided in two different vectors. The result is stored in a distance matrix called distH, where each entry represents the distance between a pair of points. Then, a maximum distance limit (dmax) is set, and a line is drawn between two points only if their distance is less than or equal to dmax. It is done to visualize issues that are close enough to each other on a map, represented by points and lines so that they can be connected in a network graph.

In the Matlab algorithm, graph theory calculates the distance between nodes in a graph G=(V, E), where V is the set of vertices and E is the edges. The algorithm from line 42 uses a distance matrix distH to store the distances between each pair of nodes in the graph. The line of code distH(distH==0)=inf; states that if the distance between two nodes is zero, it should be considered infinite, which prevents it from being used in path planning. Then, the line G(distH<=dmax)=1; establishes a network connectivity matrix, where nodes at a distance less than or equal to dmax are connected. Finally, Dijkstra's algorithm is used by calling the function dijkstra_A, to find the shortest path between an initial node N and all other nodes in the network G, generating a distance matrix dp and a predecessor matrix that can be used to reconstruct the shortest path between any pair of nodes. Table 2.6 presents the variables used in the algorithm's 3 pseudocode for planning a wireless sensor network in a geo-referenced scenario; the Matlab code is presented below.

Table 2.6: Variables related to the Main algorithm

| **Variable** | **Description** |
|---|---|
| $X$ | Input data for k-means clustering |
| $k$ | Number of clusters |
| $idx2$ | Cluster indices for each data point |
| $C$ | Cluster centroids |
| $G$ | Adjacency matrix for graph |
| $n$ | Number of data points |
| $distH$ | Pairwise distances between data points |
| $lon$ | Longitudes for data points |
| $lat$ | Latitudes for data points |
| $dmax$ | Maximum distance threshold for connecting graph edges |
| $path$ | Array to store paths for each node |
| $dp$ | Array to store shortest distances for each node |
| $pred$ | Array to store predecessor nodes for each node in the shortest path |

**Algorithm 3** Main Algorithm of Network Planning

```
Load store.mat
X ← house
k ← 3
[idx2, C] ← kmeans(X, k)
G ← zeros(length(house))
n ← length(X)
distH ← zeros(n, n)
G ← zeros(n, n)
lon ← X(:, 1)
lat ← X(:, 2)
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        distH(i, j) ← haversine([lat(i), lon(i)], [lat(j), lon(j)])
        if ∼ isreal(distH(i, j)) then
            i, j, pause
        end if
        distH(j, i) ← distH(i, j)
    end for
end for
distH(distH == 0) ← ∞
G(distH ≤ dmax) ← 1
path ← []
[dp, pred] ← dijkstra_A(G, N)
for i ← 1 to N do
    node ← i
    pathnode ← [node]
    totalCost ← 0
    while pred(node) < N + 1 and pred(node) > 0 do
        pred(node)
        pathi ← [pathi, pred(node)]
        totalCost ← totalCost + distH(node, pred(node))
        node ← pred(node)
    end while
end for
```

```
clc; clear all; close all;
warning('off','all');
load store.mat
%=======================================
figure(2);hold on; grid on; box ('on');
X=house;
k=3;colores=lines(k);
[idx2,C] = kmeans(X,k);
z2=plot(X(idx2==1,1),X(idx2==1,2),'h','color',colores(1,:),'
    MarkerSize',12);
plot(X(idx2==1,1),X(idx2==1,2),'.r','MarkerSize',12);
z3=plot(X(idx2==2,1),X(idx2==2,2),'o','color',colores(2,:),'
    MarkerSize',12);
plot(X(idx2==2,1),X(idx2==2,2),'.k','MarkerSize',12);
z4=plot(X(idx2==3,1),X(idx2==3,2),'s','color',colores(3,:),'
    MarkerSize',12);
plot(X(idx2==3,1),X(idx2==3,2),'.b','MarkerSize',12);
z5=plot(C(:,1),C(:,2),'kx','MarkerSize',15,'LineWidth',2);
legend('Cluster 1','Cluster 2','Centroids','Location','NW');
G=zeros(length(house));
n=length(X); distH=zeros(n,n); G=0*distH;
lon=X(:,1);lat=X(:,2);
    for i=1:n-1
        for j=(i+1):n
            distH(i,j)=haversine([lat(i) lon(i)],[lat(j) lon(j)]);
            if ~isreal(distH(i,j))
                i,j,pause;
            end
            distH(j,i)=distH(i,j);
        end
    end
%=======================================
    dmax=0.045;
    M=length(lon);
    N=length(lat);
    for i=1:N
        for j=1:N
            if distH(i,j)<=dmax
                z6=plot([lon(j) lon(i)],[lat(j) lat(i)],'-','color'
                    ,[127/255 127/255 127/255]);hold on;
            end
        end
    end
%=======================================
% Graph Theory
distH(distH==0)=inf;
G(distH<=dmax)=1;
path=[];
[dp,pred]=dijkstra_A(G,N);
for i=1:N
    node=i;
    path{node}=[node];
```

```
    totalCost=0;
    while pred(node)<N+1 & pred(node)>0
        pred(node);
        path{i}=[path{i} pred(node)];
        totalCost=totalCost+distH(node, pred(node));
        if length(path{i})==2
            col=[1 0.4 0.2];
            ancho=1;
            if pred(node)>0 & pred(node)<=N & pred(node)>0 & pred(node)<=N,col=[1 0 1]; ancho=1.15; end % Connexion
            z7=plot([lon(node) lon(pred(node))],[lat(node) lat(pred(node))],'-','color',col,'linewidth',ancho);
            d = findobj('Color',[1 0 1]);
        end
        node=pred(node);hold on; grid on;
    end
end
%========================================
legend([z2,z3,z4,z5,z6,z7(1)],'Group1','Group2','Group3','Centroide','Mesh Feasible','Routing','fontname','times new roman','fontsize',13,'location','SO','orientation','horizontal');
%========================================
figure(2);
hold(ax, 'off'),box('on');
h=gcf;
set(h,'PaperPositionMode','auto');
set(h,'PaperType','A4');
set(h,'PaperOrientation','landscape');
set(h,'Position',[10 0 800 700]);
set(h, 'InvertHardcopy', 'off')
fig = gcf;
fig.Color = 'white';
print -dpdf -r800 figure1_18
figure(3);
imagesc(G);colormap(jet);colorbar;
xlabel('X-axis')
ylabel('Y-axis')
hold(ax, 'on'),grid on; box('on');
h=gcf;
set(h,'PaperPositionMode','auto');
set(h,'PaperType','A4');
set(h,'PaperOrientation','landscape');
set(h,'Position',[10 0 500 800]);
set(h, 'InvertHardcopy', 'off')
fig = gcf;
fig.Color = 'white';
print -dpdf -r800 figure1_19
```

The Dijkstra algorithm is a weighted, directed graph search algorithm that starts at an initial node and finds the shortest path to all other nodes. The algorithm uses a priority queue data structure to maintain a set of visited nodes and another set of unvisited nodes. The algorithm's pseudocode begins by initializing the priority queue with the starting node and its zero cost. Next, the algorithm selects the node with the lowest cost from the priority queue and marks it as visited. Then, the cost of the adjacent nodes to the visited node is updated if the current cost is greater than the sum of the cost of the visited node and the weight of the edge that connects them. This process is repeated until all nodes are visited or the final node is reached. At the end of the algorithm, the shortest path from the starting node to all other nodes in the graph is obtained. Table 2.7 presents the variables used in the algorithm's pseudocode 4; the Matlab code is presented below.

Table 2.7: Variables related to the Dijkstra's algorithm

| Variable | Description |
|---|---|
| $V$ | Set of vertices in the graph |
| $E$ | Set of edges in the graph |
| $s$ | Start vertex |
| $Q$ | Priority queue of vertices to be processed |
| $dist$ | Array of shortest distances from $s$ to each vertex |
| $prev$ | Array of previous vertices on the shortest path from $s$ to each vertex |
| $u$ | Current vertex being processed |
| $v$ | Neighbor of $u$ |
| $length(u, v)$ | Length of the edge between vertices $u$ and $v$ |
| $alt$ | Alternative distance from $s$ to $v$ via $u$ |

```
function [d pred]=dijkstra_A(A,u)
% David F. Gleich
% Copyright, Stanford University, 2008-2009
if isstruct(A),
    rp=A.rp; ci=A.ci; ai=A.ai;
    check=0;
else
    [rp ci ai]=sparse_to_csr(A); check=1;
end
if check && any(ai)<0, error('gaimc:dijkstra', ...
        'dijkstra''s algorithm cannot handle negative edge weights.'); end
n=length(rp)-1;
d=Inf*ones(n,1); T=zeros(n,1); L=zeros(n,1);
pred=zeros(1,length(rp)-1);
n=1; T(n)=u; L(u)=n; % oops, n is now the size of the heap
% enter the main dijkstra loop
d(u) = 0;
while n>0
    v=T(1); ntop=T(n); T(1)=ntop; L(ntop)=1; n=n-1; % pop the head off the heap
```

**Algorithm 4** Dijkstra's algorithm

```
Q ← priority queue of vertices, initialized with start vertex s
dist[s] ← 0
prev[s] ← undefined
for each vertex v ∈ V \ s do
    dist[v] ← ∞
    prev[v] ← undefined
    add v to Q
end for
while Q is not empty do
    u ← vertex in Q with minimum dist[u]
    remove u from Q
    for each neighbor v of u do
        alt ← dist[u] + length(u, v)
        if alt < dist[v] then
            dist[v] ← alt
            prev[v] ← u
            decrease-key v in Q to dist[v]
        end if
    end for
end while
```

```
    k=1; kt=ntop;                   % move element T(1) down the
       heap
    while 1,
        i=2*k;
        if i>n, break; end          % end of heap
        if i==n, it=T(i);           % only one child, so skip
        else                        % pick the smallest child
            lc=T(i); rc=T(i+1); it=lc;
            if d(rc)<d(lc), i=i+1; it=rc; end % right child is
               smaller
        end
        if d(kt)<d(it), break;     % at correct place, so end
        else T(k)=it; L(it)=k; T(i)=kt; L(kt)=i; k=i; % swap
        end
    end                             % end heap down
    % for each vertex adjacent to v, relax it
    for ei=rp(v):rp(v+1)-1           % ei is the edge index
        w=ci(ei); ew=ai(ei);         % w is the target, ew is the
           edge weight
        % relax edge (v,w,ew)
        if d(w)>d(v)+ew
            d(w)=d(v)+ew; pred(w)=v;
            % check if w is in the heap
            k=L(w); onlyup=0;
            if k==0
                % element not in heap, only move the element up the
                   heap
                n=n+1; T(n)=w; L(w)=n; k=n; kt=w; onlyup=1;
            else kt=T(k);
```

```
            end
            % update the heap, move the element down in the heap
            while 1 && ~onlyup,
                i=2*k;
                if i>n, break; end              % end of heap
                if i==n, it=T(i);               % only one child, so skip
                else                            % pick the smallest child
                    lc=T(i); rc=T(i+1); it=lc;
                    if d(rc)<d(lc), i=i+1; it=rc; end % right child is smaller
                end
                if d(kt)<d(it), break;          % at correct place, so end
                else T(k)=it; L(it)=k; T(i)=kt; L(kt)=i; k=i; % swap
                end
            end
            % move the element up the heap
            j=k; tj=T(j);
            while j>1,                          % j==1 => element at top of heap
                j2=floor(j/2); tj2=T(j2);       % parent element
                if d(tj2)<d(tj), break;         % parent is smaller, so done
                else                            % parent is larger, so swap
                    T(j2)=tj; L(tj)=j2; T(j)=tj2; L(tj2)=j; j=j2;
                end
            end
        end
    end
end
```

Josiah Renfree created the Haversine function in 2010. This function calculates the distance between two geographic locations using the Haversine formula. To use the function, the user must provide two locations in latitude and longitude, which can be in degrees, minutes, and seconds or decimal format. If the user provides the locations in degrees, minutes, and seconds format, the function converts them to decimals for calculation. The function also verifies that two sites are provided, and that the locations are valid before performing distance calculations. Once the inputs have been confirmed, the function uses the Haversine formula to calculate the distance between the two locations in kilometers and then converts this value to miles and nautical miles. In summary, a haversine function is a helpful tool for calculating the distance between two geographic locations in different units of measurement. The Matlab code for the Haversine distance is presented below.

```
function [km nmi mi] = haversine(loc1, loc2)
% Created by Josiah Renfree
```

```
% May 27, 2010
%% Check user inputs
% If two inputs are given, display error
if ~isequal(nargin, 2)
    error('User must supply two location inputs')
% If two inputs are given, handle data
else
    locs = {loc1 loc2};      % Combine inputs to make checking easier
    % Cycle through to check both inputs
    for i = 1:length(locs)
        % Check inputs and convert to decimal if needed
        if ischar(locs{i})
            % Parse lat and long info from current input
            temp = regexp(locs{i}, ',', 'split');
            lat = temp{1}; lon = temp{2};
            clear temp
            locs{i} = [];            % Remove string to make room for array
            % Obtain degrees, minutes, seconds, and hemisphere
            temp = regexp(lat, '(\d+)\D+(\d+)\D+(\d+)(\w?)', 'tokens');
            temp = temp{1};
            % Calculate latitude in decimal degrees
            locs{i}(1) = str2double(temp{1}) + str2double(temp{2})/60 + ...
                str2double(temp{3})/3600;
            % Make sure hemisphere was given
            if isempty(temp{4})
                error('No hemisphere given')
            % If latitude is south, make decimal negative
            elseif strcmpi(temp{4}, 'S')
                locs{i}(1) = -locs{i}(1);
            end
            clear temp
            % Obtain degrees, minutes, seconds, and hemisphere
            temp = regexp(lon, '(\d+)\D+(\d+)\D+(\d+)(\w?)', 'tokens');
            temp = temp{1};
            % Calculate longitude in decimal degrees
            locs{i}(2) = str2double(temp{1}) + str2double(temp{2})/60 + ...
                str2double(temp{3})/3600;
            % Make sure hemisphere was given
            if isempty(temp{4})
                error('No hemisphere given')
            % If longitude is west, make decimal negative
            elseif strcmpi(temp{4}, 'W')
                locs{i}(2) = -locs{i}(2);
            end
            clear temp lat lon
        end
    end
```

```
end
% Check that both cells are a 2-valued array
if any(cellfun(@(x) ~isequal(length(x),2), locs))
    error('Incorrect number of input coordinates')
end
% Convert all decimal degrees to radians
locs = cellfun(@(x) x .* pi./180, locs, 'UniformOutput', 0);
%% Begin calculation
R = 6371;                                   % Earth's radius in km
delta_lat = locs{2}(1) - locs{1}(1);        % difference in latitude
delta_lon = locs{2}(2) - locs{1}(2);        % difference in
   longitude
a = sin(delta_lat/2)^2 + cos(locs{1}(1)) * cos(locs{2}(1)) * ...
    sin(delta_lon/2)^2;
c = 2 * atan2(sqrt(a), sqrt(1-a));
km = R * c;                                 % distance in km
%% Convert result to nautical miles and miles
nmi = km * 0.539956803;                     % nautical miles
mi = km * 0.621371192;                      % miles
```

Figure 2.10 shows the feasible mesh obtained in gray due to the maximum distance restriction allowed, in this case, 45 meters. The pink color indicates the minimum spanning tree. There are unconnected nodes, which gives rise to the Steiner Tree Problem, and Steiner nodes could allow the connectivity of all nodes. The Steiner nodes will be a set of feasible points active only if they are required to connect to the wireless sensors proposed in this scenario. The scalability in the growing populations also facilitates the sensors to be interconnected due to their proximity.

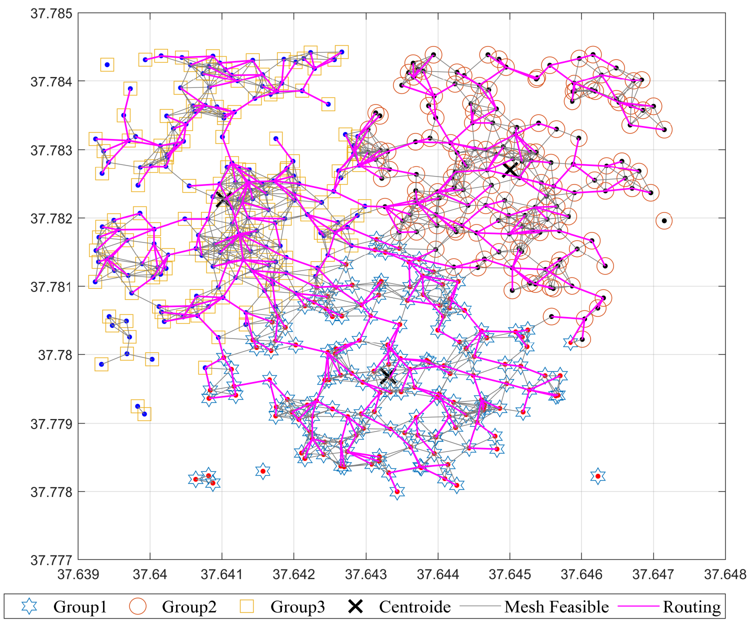



Figure 2.10: Node routing through the use of Dijkstra's Algorithm.

The connectivity matrix shown in figure 2.11 shows symmetry. Only the values in yellow have been considered possible connections in matrix G. Those represented in green are the unlabeled connections between the nodes.

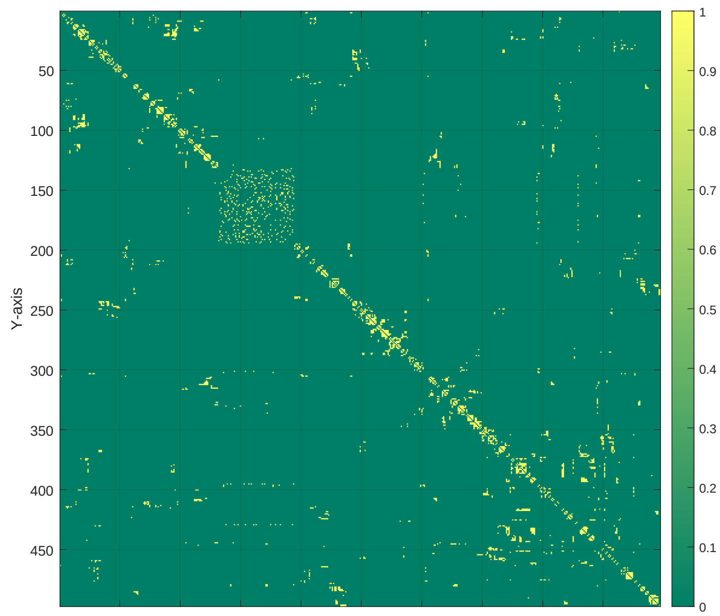



Figure 2.11: Connectivity matrix between 479 nodes.

## 2.5 Conclusions

This scientific article presented a method for OSM file reading in Matlab using a combination of existing tools. The limitation of direct OSM file reading by Matlab was overcome, and a structured file was obtained, allowing for efficient working with the data.

Furthermore, additional Matlab functions not found in MathWorks were explored. Several valuable functions were found that improved the efficiency of the data reading and filtering process, allowing for processing large datasets in a reasonable amount of time.

New Matlab scripts with the filtered house and dwelling data from OSM files have been generated. These scripts allow for extracting relevant information about the location and features of homes and dwellings, which is helpful in network planning.

It was demonstrated that applications for network planning in geo-referenced scenarios could be created using the data obtained from OSM files. It enables the identification of critical areas and planning for effective solutions in emergencies.

Finally, a didactic process was proposed that allows new research innovation from the information in OSM files and the generated Matlab codes. This process guides users in identifying new research opportunities using geo-referenced data obtained from OSM files and Matlab tools. Overall, the results demonstrate the usefulness of combining free software tools and Matlab for geospatial data processing.

Georeferencing has been a valuable tool in planning various activities and has allowed complex problems to be solved efficiently. The code presented in this paper has demonstrated how georeferencing can be combined with other techniques, such as graph theory, to solve problems in different areas. In particular, it has been presented how georeferencing and graph theory can be used to solve routing and route planning problems in geographic regions, which can have practical applications in logistics, transportation, and other activities.

Notably, the code presented in this paper demonstrates the innovation of using georeferencing and provides a practical and accessible tool for solving real-world problems. In addition, the code is presented clearly and concisely, which makes it easily reproducible and adaptable to solve other similar problems in different areas. In summary, the code shown in this article is a valuable contribution to the scientific community. It can be used as a basis for future georeferencing and geographic planning work.

# Bibliography


[1] L. Machado and E. Inga, "Optimal Placement of UDAP in Advanced Metering Infrastructure for Smart Metering of Electrical Energy Based on Graph Theory", *Electronics (Switzerland)*, vol. 11, no. 11, 2022, ISSN: 20799292. DOI: `10.3390/electronics11111767`.

[2] E. Inga, R. Hincapié, and S. Céspedes, "Capacitated Multicommodity Flow Problem for Heterogeneous Smart Electricity Metering Communications Using Column Generation", *Energies*, vol. 13, no. 1, p. 97, 2019. DOI: `10.3390/en13010097`.

[3] E. Inga, S. Céspedes, R. Hincapié, and A. Cárdenas, "Scalable Route Map for Advanced Metering Infrastructure Based on Optimal Routing of Wireless Heterogeneous Networks", *IEEE Wireless Communications*, vol. 24, no. April, pp. 1–8, 2017, ISSN: 1536-1284. DOI: `10.1109/MWC.2017.1600255`. [Online]. Available: `https://ieeexplore.ieee.org/document/7909154/`.

[4] E. Inga, J. Inga, and A. Ortega, "Novel approach sizing and routing of wireless sensor networks for applications in smart cities", *Sensors*, vol. 21, no. 14, pp. 1–17, 2021, ISSN: 14248220. DOI: `10.3390/s21144692`.

[5] E. Quintana and E. Inga, "Optimal Reconfiguration of Electrical Distribution System Using Heuristic Methods with Geopositioning Constraints", *Energies*, vol. 15, no. 15, pp. 1–20, 2022, ISSN: 19961073. DOI: `10.3390/en15155317`.

[6] J. García and E. Inga, "Georeferenced rural distribution network model considering scalable growth of users in rural areas", *Heliyon*, vol. 9, no. 1, e12724, 2023, ISSN: 24058440. DOI: `10.1016/j.heliyon.2022.e12724`. [Online]. Available: `https://doi.org/10.1016/j.heliyon.2022.e12724`.

[7] A. Valenzuela, E. Inga, and S. Simani, "Planning of a resilient underground distribution network using georeferenced data", *Energies*, vol. 12, no. 4, 2019, ISSN: 19961073. DOI: `10.3390/en12040644`.

[8] H. Lara and E. Inga, "Efficient Strategies for Scalable Electrical Distribution Network Planning Considering Geopositioning", *Electronics (Switzerland)*, vol. 11, no. 19, pp. 1–15, 2022, ISSN: 20799292. DOI: 10.3390/electronics11193096.

[9] F. Pabón, E. Inga, and M. Campaña, "Planning Underground Power Distribution Networks to Minimize Negative Visual Impact in Resilient Smart Cities", *Electricity*, vol. 3, no. 3, pp. 463–479, 2022. DOI: 10.3390/electricity3030024.

[10] M. Campaña, E. Inga, and J. Cárdenas, "Optimal sizing of electric vehicle charging stations considering urban traffic flow for smart cities", *Energies*, vol. 14, no. 16, pp. 1–16, 2021, ISSN: 19961073. DOI: 10.3390/en14164933.

[11] L. Andrei and O. Luca, "Open tools for analysis of elements related to public transport performance. Case study: Tram network in Bucharest", *Applied Sciences (Switzerland)*, vol. 11, no. 21, 2021, ISSN: 20763417. DOI: 10.3390/app112110346.

[12] C. Ludwig, R. Hecht, S. Lautenbach, M. Schorcht, and A. Zipf, "Mapping public urban green spaces based on openstreetmap and sentinel-2 imagery using belief functions", *ISPRS International Journal of Geo-Information*, vol. 10, no. 4, 2021, ISSN: 22209964. DOI: 10.3390/ijgi10040251.

[13] C. Klonner, M. Hartmann, R. Dischl, *et al.*, "The sketch map tool facilitates the assessment of openstreetmap data for participatory mapping", *ISPRS International Journal of Geo-Information*, vol. 10, no. 3, 2021, ISSN: 22209964. DOI: 10.3390/ijgi10030130.

[14] R. Gaugl, S. Wogrin, U. Bachhiesl, and L. Frauenlob, "GridTool: An open-source tool to convert electricity grid data", *SoftwareX*, vol. 21, p. 101 314, 2023, ISSN: 23527110. DOI: 10.1016/j.softx.2023.101314. [Online]. Available: https://doi.org/10.1016/j.softx.2023.101314.

[15] A. N. Wu and F. Biljecki, "InstantCITY: Synthesising morphologically accurate geospatial data for urban form analysis, transfer, and quality control", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 195, no. November 2022, pp. 90–104, 2023, ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2022.11.005. [Online]. Available: https://doi.org/10.1016/j.isprsjprs.2022.11.005.

[16] J. Kim, S. Rapuri, E. Chuluunbaatar, *et al.*, "Developing and evaluating transit-based healthcare accessibility in a low- and middle-income country: A case study in Ulaanbaatar, Mongolia", *Habitat International*, vol. 131, no. December 2022, 2023, ISSN: 01973975. DOI: 10.1016/j.habitatint.2022.102729.

[17] M. I. Kersapati, "Land use-based stakeholders mapping for natural heritage preservation – Case study: Hampstead Heath Ponds, London", *Urban Forestry and Urban Greening*, vol. 80, p. 127 821, 2023, ISSN: 16108167. DOI: 10.1016/j.ufug.2022.127821. [Online]. Available: https://doi.org/10.1016/j.ufug.2022.127821.

[18] L. Amaya, "Optimal Design of Electrical Distribution Networks Using Optimization Models . Diseño Óptimo de Redes Eléctricas de Distribución Mediante Modelos de Optimización .", *Ingeniería y Competitividad*, 2023. DOI: 10.25100/iyc.v25i1.11572.

[19] S. Zourlidou, M. Sester, and S. Hu, "Recognition of Intersection Traffic Regulations from Crowdsourced Data", *ISPRS International Journal of Geo-Information*, vol. 12, no. 1, p. 4, 2022, ISSN: 22209964. DOI: 10.3390/ijgi12010004.

[20] J. Hellekes, A. Kehlbacher, M. L. Díaz, *et al.*, "Parking space inventory from above: Detection on aerial images and estimation for unobserved regions", *IET Intelligent Transport Systems*, no. November, pp. 1–13, 2022, ISSN: 17519578. DOI: 10.1049/itr2.12322.

[21] C. M. Albrecht, R. Zhang, X. Cui, *et al.*, "Change Detection from Remote Sensing to Guide OpenStreetMap Labeling", *ISPRS International Journal of Geo-Information*, vol. 9, no. 7, 2020, ISSN: 22209964. DOI: 10.3390/ijgi9070427.

[22] M. Hacar, "Analyzing the Behaviors of OpenStreetMap Volunteers in Mapping Building Polygons Using a Machine Learning Approach", *ISPRS International Journal of Geo-Information*, vol. 11, no. 1, 2022, ISSN: 22209964. DOI: 10.3390/ijgi11010070.

[23] H. Song, L. Yang, and J. Jung, "Self-Filtered Learning for Semantic Segmentation of Buildings in Remote Sensing Imagery With Noisy Labels", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 16, pp. 1113–1129, 2022, ISSN: 21511535. DOI: 10.1109/JSTARS.2022.3230625.

[24] K. Milleville, S. Verstockt, and N. Van de Weghe, "Automatic Georeferencing of Topographic Raster Maps", *ISPRS International Journal of Geo-Information*, vol. 11, no. 7, pp. 1–17, 2022, ISSN: 22209964. DOI: 10.3390/ijgi11070387.

[25] J. Kim and E. Atkins, "Airspace Geofencing and Flight Planning for Low-Altitude, Urban, Small Unmanned Aircraft Systems", *Applied Sciences (Switzerland)*, vol. 12, no. 2, 2022, ISSN: 20763417. DOI: 10.3390/app12020576.