18 Mai 2018

UNIVERSITÉ LIBRE DE BRUXELLES

ECOLE POLYTECHNIQUE DE BRUXELLES

BA3 Irci Physicien

ELEC-H304 Physique des télécommunications
Réalisation d'un logiciel de ray-tracing

Coordinateur du cours : Philippe De Doncker

Christophe Lannoy

Aurélian Quinet

Année académique
2017 - 2018

Table des matières

1 Objectifs

L'objectif du projet est de développer un simulateur utilisant la méthode du *ray-tracing* pourprédire la puissance que capterait un récepteur connecté à une station de base placée en un point fixe d'une zone. Cette donnée permettra de déduire le débit binaire distribué aux différents endroits et d'établir la zone de couverture de la station de base considérée.

2 Fonctionnalités

Les communications étudiées sont celles de type IEEE 802.11g, soit des communications à une fréquence de 2.45 GHz. On suppose également que les antennes (émettrices ou réceptrices) sont de type $\frac{\lambda}{2}$, de gain et de résistance . Enfin, on ne considèrera que trois catégories différentes de murs, reprises dans le tableau ci-dessous.

Matière	**Permittivité relative** ϵ_r	**Conductivité** σ
Brique	4.6	0.02
Béton	5	0.014
Cloison	2.25	0.04

Le *ray-tracing* permet de imuler la propagation d'ondes dans milieux non-homogènes, comme l'étage d'un bâtiment par exemple. On suppose les ondes localement planes et les formules classiques peuvent être utilisées pour calculer les différentes interactions des ondes avec l'environnement. Cependant, nous ne prendrons en compte dans le cadre de ce projet que les interactions avec des murs, et nous les limiterons. Ainsi le logiciel propose calcule la puissance délivrée en un point via :

— les ondes directes, à savoir les ondes qui se propagent en ligne droite de l'émetteur au récepteur, indépendamment du nombre d'obstacles traversés

— les ondes réfléchies de 1 à 3 fois, à savoir les ondes qui atteignent le récepteur après une, deux, ou trois réflexions sur des murs, indépendamment du nombre d'obstacles traversés.

Le logiciel permet, à partir d'un certain nombre d'antennes émettrices placées sur un plan, de calculer la puissance reçue en tout point de la zone. EN effet, le logiciel place en chaque point de calcul une antenne réceptrice fictive et calcule la puissance qu'elle recevrait en cet endroit. Cette puissance est ensuite traduite en débit.

Le logiciel se charge ensuite de dresser la carte de l'étage en indiquant la qualité de la réception d'information à l'aide d'un code couleur. Une lecture rapide et qualitative est ainsi possible pour évaluer la zone de couverture de la station de base.

Plusieurs plans sont possibles :

— une pièce vide pour tester la propagation directe

— une pièce avec un seul mur visualiser la transmission et la réflexion

— un étage complet pour se rendre compte des résultats à grande échelle

3 Code

Le logiciel est codé en C, les codes sont disponibles dans les annexes. Le fichier *main* est le moteur du logiciel, il demande l'initialisation de la carte, puis lance les calculs de la puissance en chaque point de la carte avant de la traduire en une valeur facilement interprétable pour un moteur graphique et d'initier le tracé du graphe final. Le fichier *map* contient toutes les informations relatives aux conditions initiales du cas étudié. Ensuite, *powerCalculations* regroupe tous les calculs liés aux puissances : coefficients de réflexions ou de transmissions et puissances associées. Dans un rôle plus auxiliaire, *geometry* reprend les différentes opérations vectorielles nécessaires aux calculs de coefficients ou aux vérifications de conditions de réflexions. Enfin, le fichier *plot* s'occupe de tracer le graphe tant attendu.

3.1 Fichier main

Dans ce premier fichier, les variables globales (accessible partout ailleurs dans le code) sont définies comme la taille de la carte et les listes de murs ou émetteurs. Le but premier de cette section est d'appeler les fonctions principales qui vont tour à tour, créer une carte, calculer les puissance en chaque point par transmission directe puis réflexion et finalement de représenter graphiquement ces données.

3.2 Fichier map

Cette section s'occupe de générer une carte du lieu où l'on simulera la propagation des ondes. Les différents murs et émetteurs y sont crées en spécifiant leur position et leur propriétés. Une résolution est également définie afin de pouvoir jouer sur la précision du rendu et le temps de calcul.

La carte en tant que telle est représentée par une liste de vecteurs indiquant la position de chaque point de calcul. D'autres listes seront crées par la suite pour représenter les murs, antennes et puissance en chaque point.

3.3 Fichier powerCalculations

Les fichiers rangés sous l'égide *powerCalculations* constituent le nerf central du code dans lequel les calculs complexes sont effectués. Ils contiennent deux fonctions principales qui appellent toutes les autres : *directPathPower()* et *reflecRayPower()*, renvoyant respectivement la puissance d'un champ en propagation libre et celle d'un champ réfléchi. Le fichier est divisé en trois parties pour plus de clarté.

La première partie contient toutes les formules de bases qui seront nécessaires à tout calcul de puissance, peu importe que le champ ait subi quelques transmissions ou réflexions. Dans l'ordre du code, ces formules permettent de calculer :

— un champ complexe

— un coefficient de réflexion entre deux milieux pour un angle d'incidence donné

— un coefficient de réflexion d'un mur pour un angle d'incidence donné

— un coefficient de transmission d'un mur pour un angle d'incidence donné

— une puissance moyenne d'un champ

Les expression littérales de ces formules se trouvent dans la section suivante.

La seconde partie reprend le calcul du coefficient de transmission total d'un rayon en fonction du nombre de murs qu'il traverse pour aller de la station de base au récepteur, ainsi que le calcul de la puissance en chemin direct, c'est-à-dire sans réflexion.

La dernière partie regroupe les calculs liés à la réflexion. On y trouve une fonction qui détermine la puissance délivrée en un point par un rayon réfléchi et une fonction renvoyant la puissance totale délivrée en un point par tous les rayons réfléchis de une à trois fois qui y passent.

3.4 Fichier geometry

Le fichier geometry contient toutes les informations et opérations impliquant des vecteurs. Ces opérations servent principalement à déterminer si un rayon rencontre un mur et, le cas échéant, à calculer l'angle d'incidence du contact. Il sert également à implémenter la méthode des images via une fonction capable de calculer la position équivalente de l'antenne pour le calcul de puissance d'un rayon réfléchi.

3.5 Fichier plot

Dans cette dernière section un graphique va être généré afin de représenter graphiquement les données calculées par le logiciel. Pour de raison de lisibilité du graphique les valeurs de puissance supérieur a -20 dBi sont ramenées à -20 dBi et les valeurs inférieures à -93 dBi qui correspondent à un débit binaire nul, sont ramenées à -93 dBi.

Les données sont d'abord stockées dans un fichier 'data.txt' avant d'être portées sur un graphique par *gnuplot. Deux types de graphique peuvent être générée selon que l'on s'intéresse à la puissance réceptionnée ou au débit binaire.*

4 Validation de cas élémentaires

Commençons par calculer les différents paramètres des des antennes. Les antennes considérées sont des antennes dipôles $\lambda/2$. Leurs gain $G(\theta, \phi)$ est donné par le rapport de l'intensité rayonnée dans une direction sur l'intensité rayonnée qu'émettrait une antenne fictive sans perte et isotrope de même puissance P_{TX}. Cette antenne étant omnidirectionnelle, son gain dépendra uniquement de l'angle azimutal. Cependant, comme nous travaillons en deux dimensions pour cette simulation (en faisant l'hypothèse que l'émetteur et le récepteur sont à même hauteur du sol) nous considérerons uniquement l'angle $\theta = \pi/2$.

Nous obtenons donc

$$G_{TX}(\theta = \pi/2, \phi) = 0,52\pi \tag{1}$$

$$\vec{h_e}(\theta = \pi/2, \phi) = \frac{-3.10^8}{2,45.10^9.\pi}\vec{1_z}\ [m] \tag{2}$$

$$R_a = 75,86\ [\Omega] \tag{3}$$

$$P_{TX} = 0,1\ [W] \tag{4}$$

4.1 Rayonnement directs

La première situation utilisée pour vérifier le bon fonctionnement du logiciel est celle du rayonnement en espace libre. Dans ce cas, aucune réflexion ou transmission ne peut avoir lieu et le calcul se limite aux ondes directes sans transmission. Pour cette simulation nous allons placer un émetteur au centre d'une pièce fictive sans murs (pour éviter toute réflexion) et mesurer la puissance reçue en différents points. Prenons par exemple comme position pour l'émetteur $(20, 20)$ $[m]$ *et calculons les puissance reçue en* $(0, 0)$ $[m]$ *et* $(20, 30)$ $[m]$*. Ces deux positions correspondent respectivement à* $d \approx 28.28$ $[m]$ *et* $d = 10$ $[m]$*. En substituant ces valeurs dans les expressions suivantes, nous obtenons respectivement* $< P_{RX} > \approx 3,07.10^{-8}\ [W] = -45\ dBm$ *et* $< P_{RX} > \approx 2,45.10^{-7}\ [W] = -36\ dBm$*.*

$$\underline{\vec{E}} = \sqrt{60.G_{TX}.P_{TX}}.\frac{e^{-j\beta d}}{d}.\vec{1_z} \tag{5}$$

$$< P_{RX} >= \frac{1}{8R_a}\left|\vec{h_e}.\underline{\vec{E}}\right|^2 = \frac{1}{8R_a}.60.G_{TX}.P_{TX}.\left|\frac{h_e}{d}\right|^2 \tag{6}$$

Le graphique 1 est donc cohérent avec nos prédictions théorique. En demandant précisément au logiciel les valeurs de puissance de ces deux points, nous obtenons respectivement (arrondi à l'unité) -45 dBm et -36 dBm.

4.2 Rayons transmis

On se propose de simuler, comme seconde vérification du logiciel, la transmission d'une onde au travers d'un mur de brique. Pour ce faire, rajoutons à la situation précédente un mur s'étendant du point (0,10) [m] jusqu'au point (30,40) [m]. Considérons un mur de brique d'une épaisseur de 10 cm, de permittivité relative $\epsilon_r = 4,6$ *et de conductivité* $\sigma = 0.02$*.*

Calculons maintenant analytiquement la puissance reçue par un récepteur positionné en (20,35) [m].

Le champ électrique incident sur l'antenne, est donné par le champ de l'onde directe (que l'on calcule via l'équation 5) multiplié par le coefficient de transmission $T_m(\theta_i)$*. Ce coefficient dépend de l'angle d'incidence de l'onde sur le mur qui dans le cas de notre exemple vaut* $\theta_i = \pi/4$*.*

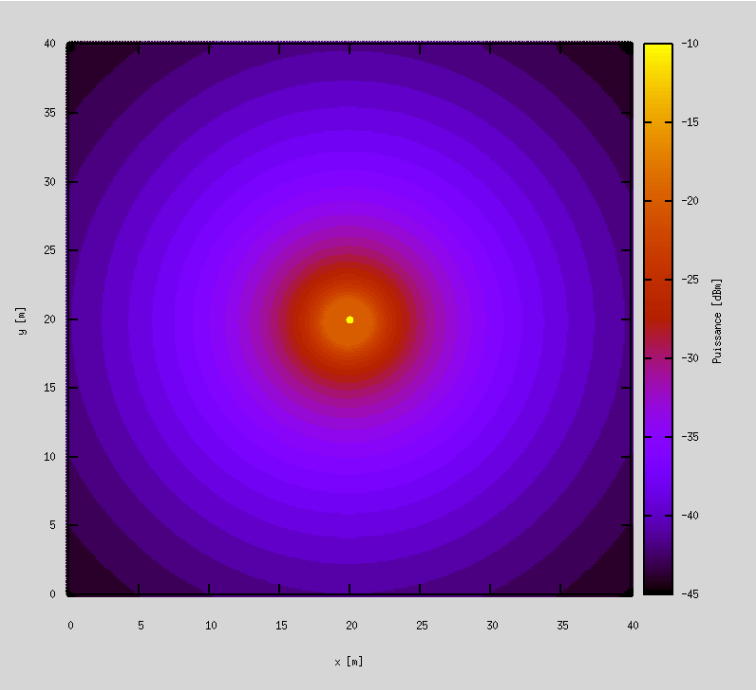

FIGURE 1 – Simulation ondes directes

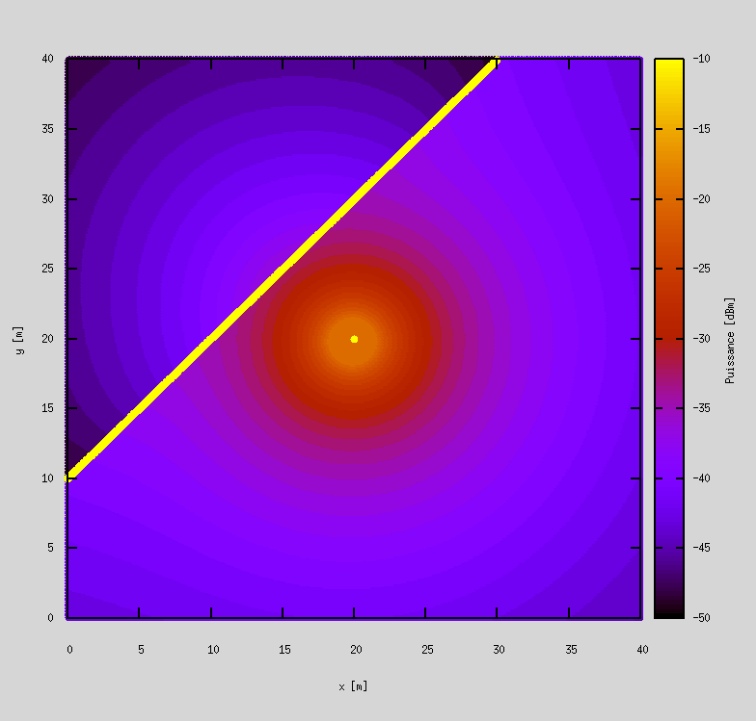

FIGURE 2 – Simulation de la transmission

$$T_m(\theta_i) = \frac{(1-\Gamma_\perp^2(\theta_i))e^{-\gamma_m s}}{1-\Gamma_\perp^2(\theta_i)e^{-2\gamma_m s}.e^{j\beta 2s\sin\theta_t\sin\theta_i}} \tag{7}$$

$$\Gamma_\perp(\theta_i) = \frac{Z_2\cos\theta_i - Z_1\cos\theta_t}{Z_2\cos\theta_i + Z_1\cos\theta_t} \quad avec \quad \cos\theta_t = \frac{Z_1}{Z_2}\cos\theta_i \tag{8}$$

Pour notre cas, nous trouvons un coefficient de transmission,

$$T_m(\theta_i = \pi/4) \approx 0,42 + i\ 0,42 \tag{9}$$

Le champ électrique incident sur l'antenne de réception et la puissance moyenne peuvent donc être calculés de manière similaire.

$$\underline{\vec{E}} = T_m(\theta_i)\sqrt{60.G_{TX}.P_{TX}}.\frac{e^{-j\beta d}}{d}.\vec{1_z} \tag{10}$$

$$< P_{RX} >= \frac{1}{8R_a}\left|\vec{h_e}.\underline{\vec{E}}\right|^2 = \frac{1}{8R_a}.60.G_{TX}.P_{TX}.\left|\frac{h_e}{d}\right|^2.\left|T_m(\theta_i)\right|^2 \quad \approx 3,84.10^{-8}\ [W] = -44dBm \tag{11}$$

A nouveau, le graphique 2 produit par le logiciel est cohérent avec cette prédiction théorique. Nous pouvons obtenir via le logiciel la valeur exacte de la puissance en ce point et nous trouvons également -44 dBm.

4.3 Rayons réfléchis

Une autre situation intéressante à analyser est celle de la réflexion de l'onde sur un mur. Pour ce faire, deux cloisons d'une épaisseur de 10 cm, de permittivité relative $\epsilon_r = 2,25$ et de conductivité $\sigma = 0,04$ ont été disposées comme indiqué sur la figure 3 . L'émetteur est centrée en (20,20) [m] et le récepteur en (30,20) [m]. Pour simplifier les calculs analytiques seule une réflexion sera prise en compte (ainsi que l'onde directe).

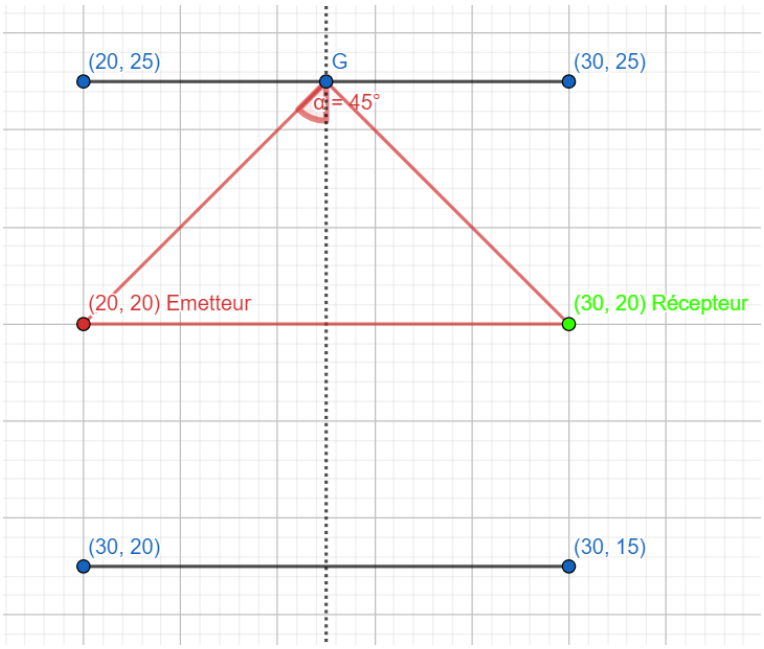

FIGURE 3

Premièrement, pour l'onde directe, nous avons une puissance reçue par le récepteur donnée par l'équation (6) où $d_{directe} = 10[m]$. *La puissance directe est donnée par l'équation (6).*

$$< P_{RX} >_{directe} = \frac{1}{8R_a}\left|\vec{h_e}.\underline{\vec{E}}_{direct}\right|^2 = \frac{1}{8R_a}.60.G_{TX}.P_{TX}.\left|\frac{h_e}{d_{directe}}\right|^2 \approx 2,48.10^{-7}[W] = -36dBm \quad (12)$$

En second lieu, pour l'onde réfléchie une seule fois, il vient selon l'équation (13), avec $\theta_i = \pi/4$ *(*α *sur la figure),* $\Gamma_m(\theta_i = \pi/4) = -0,342 + i0,112$

$$\Gamma_m(\theta_i) = \Gamma_\perp(\theta_i) + (1 - \Gamma_\perp^2(\theta_i))\frac{\Gamma_\perp(\theta_i)e^{-2\gamma_m s}.e^{j\beta 2s\sin\theta_t\sin\theta_i}}{1 - \Gamma_\perp^2(\theta_i)e^{-2\gamma_m s}.e^{j\beta 2s\sin\theta_t\sin\theta_i}} \quad (13)$$

Pour l'onde réfléchie (une seule des deux), $d_{reflechie} = 2\sqrt{50}[m]$ *et la puissance réfléchie reçue par le récepteur sera,*

$$\underline{\vec{E}}_{reflechi} = \Gamma_m(\theta_i)\sqrt{60.G_{TX}.P_{TX}}.\frac{e^{-j\beta d}}{d_{reflechie}}.\vec{1_z} \quad (14)$$

$$< P_{RX} >_{reflechie} = \frac{1}{8R_a}\left|\vec{h_e}.\underline{\vec{E}}\right|^2 = \frac{1}{8R_a}.60.G_{TX}.P_{TX}.\left|\frac{h_e}{d}\right|^2.\left|\Gamma_m(\theta_i)\right|^2 \approx 1,61.10^{-8} = -44dBm \quad (15)$$

Il suffit maintenait de sommer ces deux contributions en tenant compte qu'une deuxième onde est réfléchie symétriquement sur le mur du bas et donc il est nécessaire de multiplier par deux la puissance réfléchie pour trouver la puissance totale reçue par le récepteur. A nouveau cette puissance est identique à celle fournie par le logiciel (figure 5).

$$< P_{RX} >_{totale} = < P_{RX} >_{directe} + 2 < P_{RX} >_{reflechie} = 2,48.10^{-7} + 2.\ 1,61.10^{-8} \approx -36dBm \quad (16)$$

On remarque que la puissance réfléchie est négligeable par rapport à la puissance directe. Il est donc intéressant de simuler une situation fictive ou seulement la puissance réfléchie serait calculée afin de vérifier que le logiciel fonctionne correctement. Ceci est fait sur la figure et on constate que la puissance réfléchie vaut en effet -44 dBi (figure 4).

5 Application à un étage de bâtiment

Après la validation bloc par bloc entreprise au point précédent, on peut à présent appliquer le logiciel pour un cas plus complexe : l'étage complet d'un building par exemple. Le plan utilisé pour le test est visible ci-dessous.

Sur ce plan on peut voir l'émetteur dans l'enclave en haut à gauche (en $(x, y) = (9, 37)$*) et tous les murs. Les murs extérieurs, le grand mur horizontal (en* $y = 30$*), le grand mur vertical (en* $x = 24$*) et*

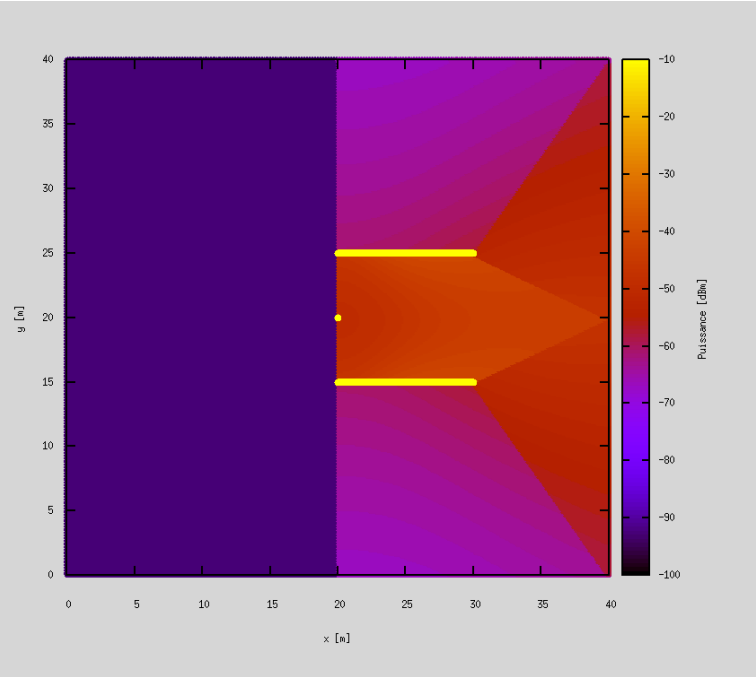

FIGURE 4 – Simulation de la réflexion uniquement (ondes directes non prise en compte)

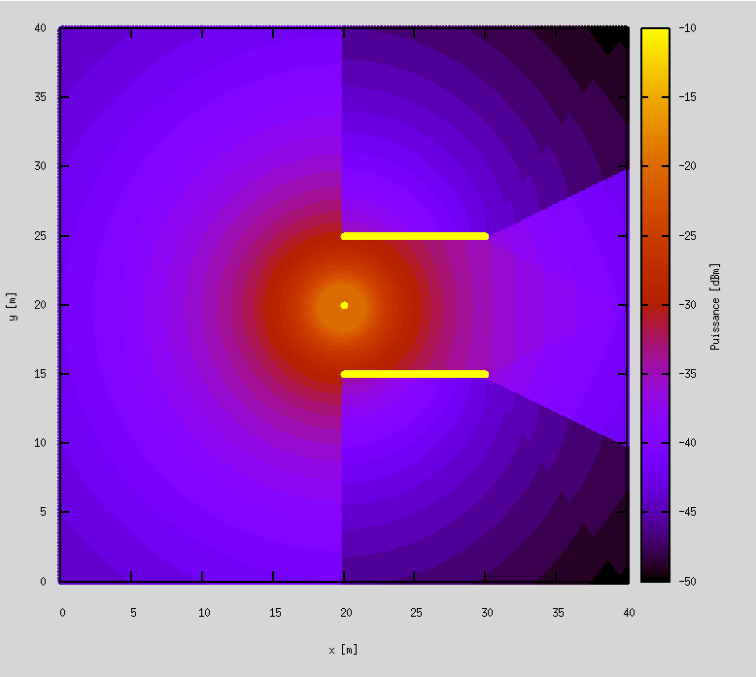

FIGURE 5 – Simulation ondes directes et réflexion unique

le mur en U retourné (de $x = 12$ à $x = 16$) sont des murs porteurs et son par conséquent en béton. Les murs formant l'enclave ainsi que la paroi diagonale sont eux considérés comme des cloisons. Tous les autres sont implémentés comme des murs en brique. Les cloisons et les murs de brique ont une épaisseur de 20cm tandis que les murs en béton sont larges de 40cm.

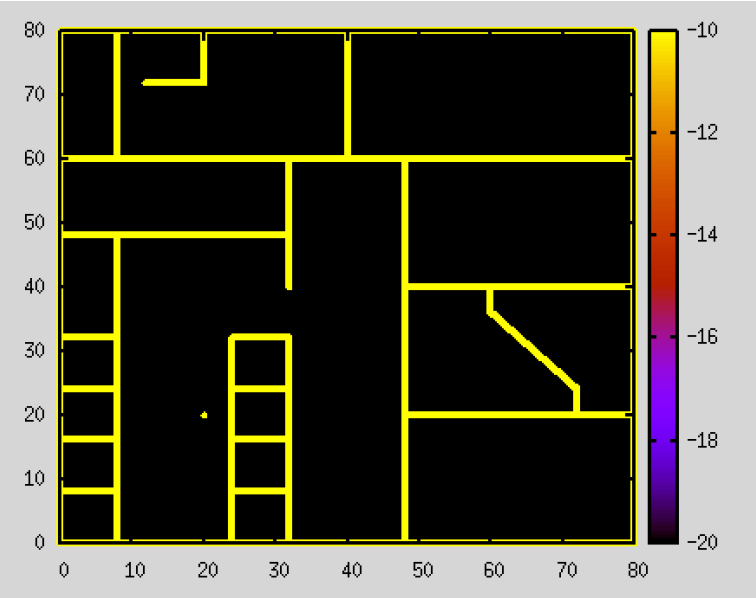

FIGURE 6 – Plan de l'étage considéré pour une application plus complexe

La cartographie de la puissance est donnée ci-après. La puissance a été préférée au débit pour les représentations graphique car elle sature moins vite, ce qui rend les données plus lisibles visuellement.

On constate très clairement l'effet atténuateur de la distance de propagation libre sur la puissance, notamment dans le rectangle central où le dégradé est bien visible. Le rôle des transmissions travers les murs est quant à lui net aussi : c'est à lui qu'est dû cette division en faisceau de la puissance radiée par l'émetteur. En ce qui concerne les réflexions multiples elles se remarquent grâce aux différents faisceaux de puissances rebondissant sur les parois.

La différence de puissance due à des murs différents est parfois ténue. Néanmoins, elle se remarque cumulativement : on peut constater sur le graphe qu'un faisceau passé à travers un certain nombre de murs en béton est plus faible qu'un faisceau passé à travers le même nombre de parois en brique ou cloisons.

A titre de comparaison, voici le relvé de puissances dans l'étage dans les mêmes conditions mais cette fois-ci en ignorant les réflexions. Le retrait des réflexions provoque un changement majeur. Il concerne la zone de couverture, qui est plus large que sur les relevés globaux. En effet, ici toute la puissance est transmise à travers les murs au lieu d'être scindée en deux, ce qui augmente fatalement partout la puissance moyenne reçue.

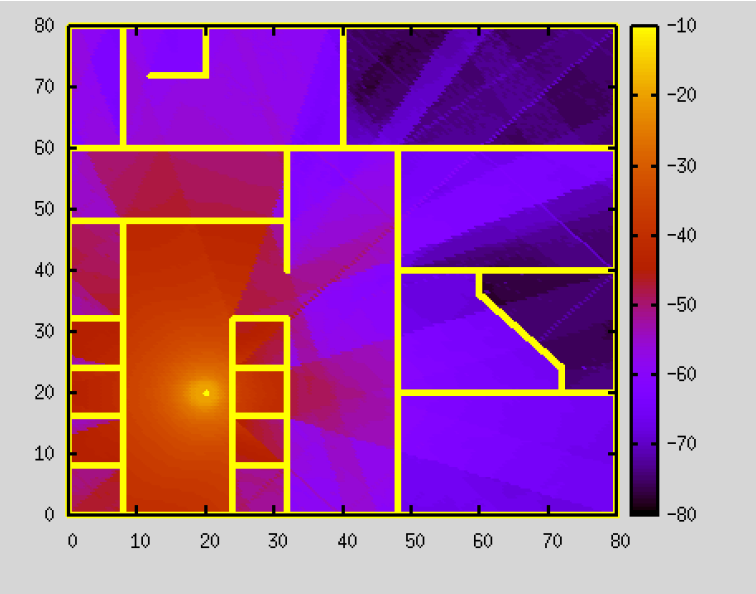

FIGURE 7 – Cartographie de la puissance

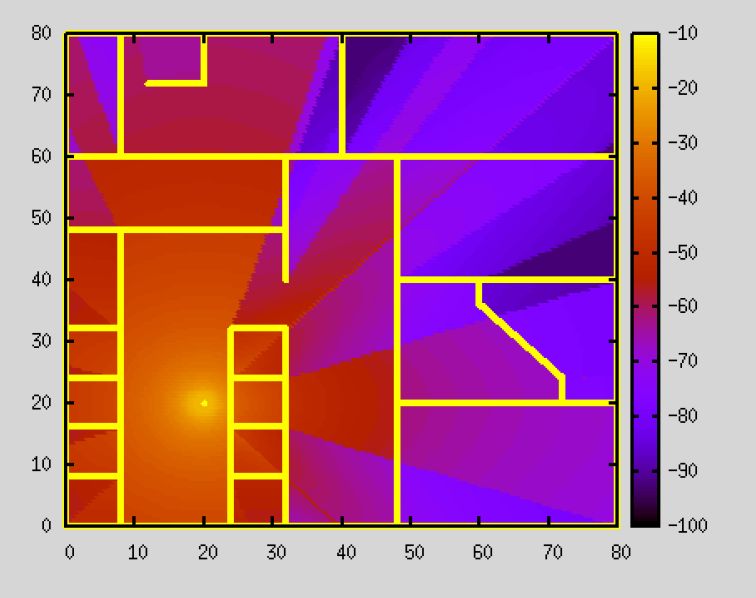

FIGURE 8 – Cartographie de la puissance sans réflexion des ondes

6 Possibles applications du logiciel

6.1 Placement d'une station de base

Lorsqu'une certaine topologie d'un étage est donnée, le logiciel peut être utilisé pour comparer les différents endroits possibles où placer une station de base pour que toute la zone profite au maximum d'un débit correct.

On compare ici la situation obtenue à la section précédente avec une adaptation : l'émetteur a été déplacé au centre de l'étage. Si certaines zones de la carte sont moins bien desservies, la zone de couverture a été améliorée et la puissance reçue varie beaucoup moins en fonction de la position à laquelle on se trouve

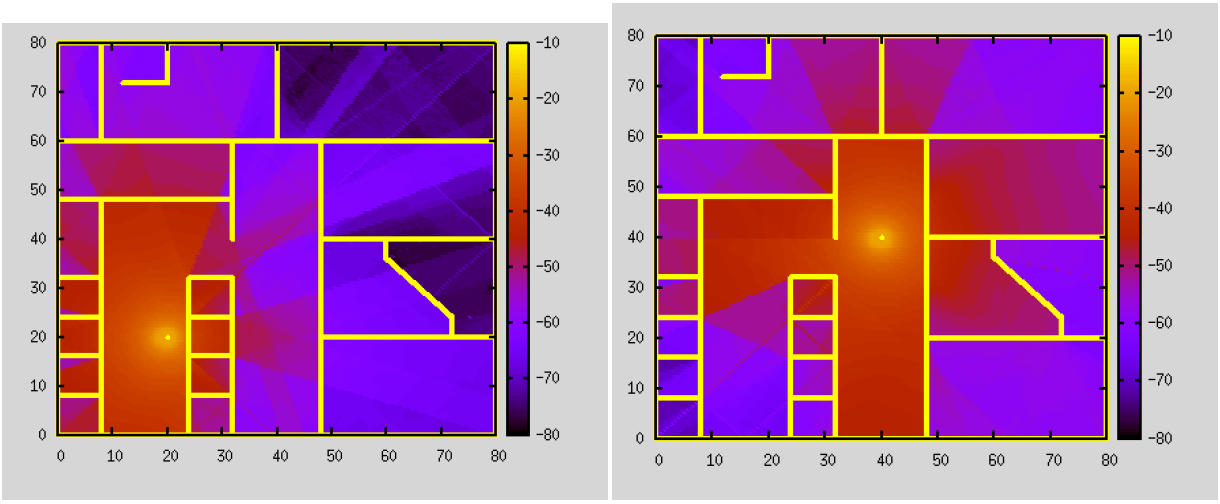

FIGURE 9 – Influence de l'emplacement de la station de base

dans l'étage.

6.2 Placement de relais pour une station fixe

Une autre application de ce code consiste en l'amélioration de la distribution du débit dans un étage via l'installation d'un ou plusieurs relais. Ceux-ci peuvent être modélisés par de nouvelles antennes de même puissance ou de puissance moindre pour simplifier l'approche. Ainsi, à partir du relevé graphique de la station de base, il est aisé de déduire des emplacements intéressants auxquels placer des relais pour augmenter efficacement la zone de couverture.

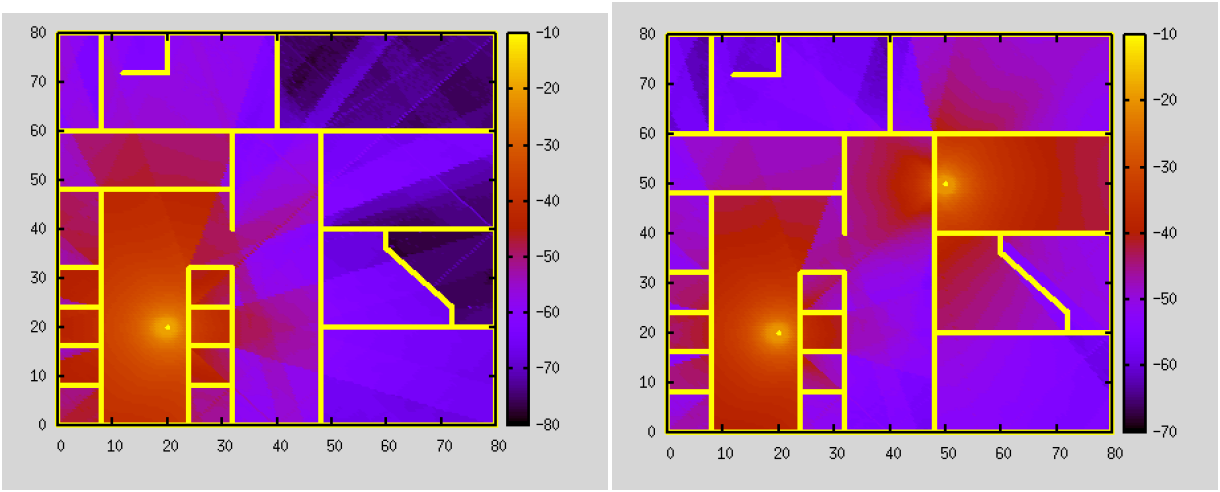

FIGURE 10 – Influence de l'introduction d'une station de base supplémentaire

L'introduction d'un relais, assimilé ici à une seconde station de base permet d'éliminer la zone d'ombre de la situation initiale. De plus, comme les deux antennes ont tendances à moyenner leurs contributions par réflexions dans les salles intermédiaires, la puissance dans chaque pièce est bien plus constante et subit moins les aléas directionnels des faisceaux.

6.3 Design de plans d'étages

Enfin, ce logiciel peut servir à résoudre le problème en sens inverse et à établir la pertinence de plans proposés : si le plan d'un étage à construire n'est pas satisfaisant d'un point de vue distribution du réseau, il peut être repensé. Il suffit parfois de déplacer un mur, de réduire son épaisseur, ou de changer le matériau qui le constitue pour améliorer la propagation des ondes.

L'introduction d'un nouveau mur en béton pour fermer la zone en bas à gauche de l'étage réduirait considérablement la fraction de puissance qui s'en échappe. Ceci a deux avantages : l'amélioration de la réception dans la zone fermée grâce aux réflexions engendrées, et l'atténuation relativement plus rapide de la puissance en dehors. Ces points sont par exemple souhaités lorsqu'un service souhaite installer son propre wi-fi sans interférer chez les voisins, tout en profitant d'une connexion intéressante.

7 Annexe : le code

7.1 main

main.h

```
#ifndef main_h
#define main_h

//Wall object
typedef struct Wall Wall;
struct Wall {
    double x1,y1;
    double x2,y2;
    double a;
    double b;
    double c;
    double epsilonr;
    double sigma;
    double l;
};

//Vector object
typedef struct Vector Vector;
struct Vector {
    double x;
    double y;
};

//Emitter object
typedef struct Emetteur Emetteur;
struct Emetteur {
    Vector position;
    double puissance;
};

Wall newWall(double x1, double y1, double x2, double y2, double epsilon, double
    sigma, double l);
void createMap();
void squareMap(double mapSize);

#endif
```

main.c

```
#include <stdio.h>
#include <stdbool.h>
#include <math.h>
#include "map.h"
#include "geometry.h"
```

```
6  #include "plot.h"
7  #include "powerCalculations.h"
8  #include "main.h"
9
10 //Submission of the parameters
11 extern int NumberOfPoints;
12 extern int NumberOfWalls;
13 extern int NumberOfEmetteurs;
14 extern Vector listPt[];
15 extern Wall listWalls[];
16 extern double powers[];
17 extern Emetteur listEmetteurs[];
18
19 double Power(Vector tx, double p_tx, Vector pt);
20 void powerType(int list[], double power, int i, int type);
21 void powerType2(int list[], double power, int i, int type);
22
23 int main(void){
24         createMap();
25         int powers_plot[NumberOfPoints];
26         for(int i = 0; i < NumberOfPoints; i++) {                    // Computes the
              power for each point of interest
27             for (int k = 0; k < NumberOfEmetteurs; k++) {     // Computes the
                   power for each transmitter
28                 Vector tx = listEmetteurs[k].position;
29                 powers[i] += Power(tx,P_TX,listPt[i]);
30             }
31
32             powerType2(powers_plot,powers[i],i,1);              // Convert the power
                    to a suitable form for plotting
33     }
34         plot(listEmetteurs,NumberOfEmetteurs,listWalls,NumberOfWalls,listPt,
               powers_plot, NumberOfPoints);     //plots the power
35 }
36
37 //First conversion of the power to a plotable value
38 void powerType(int list[], double power, int i, int type) {
39     list[i] = 2.4 *(10*log10(power/0.001)) + 229.2;
40 }
41
42 //Second conversion of the power to a plotable value
43 void powerType2(int list[], double power, int i, int type) {
44     list[i] = (10*log10(power/0.001));
45 }
46
47 //Calculate the power received at a specific point pt from the emitter tx
48 double Power(Vector tx, double p_tx, Vector pt) {
49     double power = 0;
50     power += directPathPower(pt,tx,p_tx,listWalls,NumberOfWalls);
```

```
51      power += reflecRayPower(pt, tx, p_tx, listWalls, NumberOfWalls, 3, -1);
52      return power;
53  }
```

7.2 map

map.h

```
1   #ifndef map_h
2   #define map_h
3
4   #include <math.h>
5   #include "geometry.h"
6   #include "main.h"
7
8   #define G_TX 0.13*4*M_PI          // Gain of the antenne
9   #define P_TX 0.1                  // Power of the antenna
10  #define H_E -3e8/(2.45e9*M_PI)    // Directivity of the antenna
11  #define R_A 75.86098878           // Resistance of the antenna
12
13
14
15  #endif
```

map.c

```
1   #include "map.h"
2
3   //Precision
4   double resolution = 0.5;
5
6   //Size of the map
7   double side = 80;
8
9   //Walls
10  const int NumberOfWalls = 29 ;
11  Wall listWalls[29];
12
13  //Points
14  int NumberOfPoints = 25600;         // (side/resolution) * (side/resolution)
15  Vector listPt[25600];
16  double powers[25600];
17
18  //Emitters
19  Emetteur newEmetteur(double x, double y, double puissance){
20          Emetteur emetteur;
21          emetteur.position = newVec(x,y);
22          emetteur.puissance = puissance;
23          return emetteur;
24  };
```

```
25  const int NumberOfEmetteurs = 2;
26  Emetteur listEmetteurs[2];
27
28  //Creates the map
29  void createMap(){
30          squareMap(side);
31  };
32
33  Wall newWall(double x1, double y1, double x2, double y2, double epsilonr, double
        sigma, double l) {
34      Wall w;
35      if (x2 >= x1) {
36          w.x1 = x1;  w.y1 = y1;
37          w.x2 = x2;  w.y2 = y2;
38      }
39      else {
40          w.x1 = x2;  w.y1 = y2;
41          w.x2 = x1;  w.y2 = y1;
42      }
43      // a.x + b.y + c = 0
44      if(x2-x1 != 0) {
45          w.a = (y2-y1)/(x2-x1);  w.b = -1;   w.c = y1 - w.a * x1;
46      }
47      else {
48          w.a = 1;    w.b = 0;    w.c = -x1;
49      }
50      w.epsilonr = epsilonr;    w.sigma = sigma;    w.l = l;
51      return w;
52  }
53
54  void squareMap(double mapSize) {
55      //Borders of the map
56      listWalls[0] = newWall(0,0,0,mapSize, 5,0.014,0.12);
57      listWalls[1] = newWall(0,mapSize,mapSize,mapSize, 5,0.014,0.12);
58      listWalls[2] = newWall(mapSize,mapSize,mapSize,0, 5,0.014,0.12);
59      listWalls[3] = newWall(mapSize,0,0,0, 5,0.014,0.12);
60
61      //Vertical walls
62      listWalls[4] = newWall(8,0,8,48,4.6,0.02,0.2);
63      listWalls[5] = newWall(8,60,8,80,4.6,0.02,0.2);
64      listWalls[6] = newWall(20,72,20,80,4.6,0.02,0.2);
65      listWalls[7] = newWall(24,0,24,32,5,0.014,0.4);
66      listWalls[8] = newWall(32,0,32,32,5,0.014,0.4);
67      listWalls[9] = newWall(32,40,32,60,4.6,0.02,0.2);
68      listWalls[10] = newWall(40,60,40,80,4.6,0.02,0.2);
69      listWalls[11] = newWall(48,0,48,60,5,0.014,0.4);
70      listWalls[12] = newWall(60,36,60,40,2.25,0.04,0.2);
71      listWalls[13] = newWall(72,20,72,24,2.25,0.04,0.2);
72
```

```
73      //Horizontal walls
74      listWalls[14] = newWall(0,8,8,8,4.6,0.02,0.2);
75      listWalls[15] = newWall(24,8,32,8,4.6,0.02,0.2);
76      listWalls[16] = newWall(0,16,8,16,4.6,0.02,0.2);
77      listWalls[17] = newWall(24,16,32,16,4.6,0.02,0.2);
78      listWalls[18] = newWall(0,24,8,24,4.6,0.02,0.2);
79      listWalls[19] = newWall(24,24,32,24,4.6,0.02,0.2);
80      listWalls[20] = newWall(48,20,80,20,4.6,0.02,0.2);
81      listWalls[21] = newWall(0,32,8,32,4.6,0.02,0.2);
82      listWalls[22] = newWall(24,32,32,32,5,0.014,0.4);
83      listWalls[23] = newWall(16,80,64,80,4.6,0.02,0.2);
84      listWalls[24] = newWall(48,40,80,40,4.6,0.02,0.2);
85      listWalls[25] = newWall(0,48,32,48,4.6,0.02,0.2);
86      listWalls[26] = newWall(0,60,80,60,5,0.014,0.4);
87      listWalls[27] = newWall(12,72,20,72,4.6,0.02,0.2);
88      listWalls[29] = newWall(8,32,48,32,5,0.014,0.4);
89
90      //Diagonal walls
91      listWalls[28] = newWall(60,36,72,24,2.25,0.04,0.2);
92
93      //add-ons
94      listWalls[30] = newWall(4,16,24,16,5,0.014,0.4);
95
96      int nbStep = ceil(mapSize/resolution);
97      resolution = mapSize/nbStep;
98      for(int i = 0; i < nbStep; i++) {
99          for(int j = 0; j < nbStep; j++) {
100             listPt[nbStep*i+j] = newVec(resolution*(0.5+j),resolution*(0.5+i));
101         }
102     }
103
104     //Emitters
105     listEmetteurs[0] = newEmetteur(20,20,P_TX);
106     listEmetteurs[1] = newEmetteur(50,50,P_TX);
107
108 };
```

7.3 powerCalculations

powerCalculations.h

```
1  #ifndef powerCalculations_h
2  #define powerCalculations_h
3
4  #include <complex.h>
5  #include "map.h"
6  #include "geometry.h"
7
8  double directPathPower(Vector rx, Vector tx, double p_tx, Wall walls[], int size)
       ;
```

```
9  double reflecRayPower(Vector rx, Vector tx, double p_tx, Wall walls[], int size,
       int iteration, int j);
10
11 #endif
```

powerCalculations.c

```
1  #include "powerCalculations.h"
2  #include <math.h>
3
4  #define EPS0 1/(36*M_PI)*1e-9    // Vacuum permittivity
5  #define MU0 4*M_PI*1e-7          // Vacuum permeability
6  #define FREQ 2.45e9              // Wi-Fi frequency
7  #define AIR_EPS EPS0             // Air permittivity
8  #define AIR_SIG 0                // Air conductivity
9  #define BETA 2*M_PI*FREQ/(3e8)   // Propagation constant in the air
10
11 const int NumberOfReflections = 3;  // Number of considered reflections
12
13 ////////////////////////////////////////
14 //             BASIC FORMULAS         //
15 ////////////////////////////////////////
16
17 //Returns the complex electric field for a direct path of length d
18 double complex e_field_direct(double d, double p_tx) {
19     double complex e = sqrt(60*G_TX*p_tx)* cexp(-I*BETA*d)/d;
20     return e;
21 }
22
23 //Returns the complex reflection coefficient between two mediums for a given
       angle theta_i
24 static double complex reflec_coeff(double complex epsilon1, double complex
       epsilon2, double theta_i) {
25     double complex z1 = csqrt(MU0/epsilon1);
26     double complex z2 = csqrt(MU0/epsilon2);
27     double theta_t = asin(sqrt(creal(epsilon1)/creal(epsilon2)) * sin(theta_i));
28     double complex coeff = (z2*cos(theta_i) - z1*cos(theta_t)) / (z2*cos(theta_i)
           + z1*cos(theta_t));
29     return coeff;
30 }
31
32 //Returns the complex reflection coefficient of the wall w for a given angle
       theta_i
33 double complex reflec_coeff_wall(Wall w, double theta_i) {
34     double complex epsilon1 = AIR_EPS - I*AIR_SIG/(2*M_PI*FREQ);
35     double complex epsilon2 = w.epsilonr*EPS0 - I*w.sigma/(2*M_PI*FREQ);
36     double theta_t = asin(sqrt(creal(epsilon1)/creal(epsilon2)) * sin(theta_i));
37     double complex ref_coeff = reflec_coeff(epsilon1,epsilon2,theta_i);
38     double complex gamma_m = I*2*M_PI*FREQ*csqrt(MU0*epsilon2);
39     double l = w.l;
```

```
40      double s = l/cos(theta_t);
41
42      double complex var = cexp(-2*gamma_m*s + I*BETA*2*s*sin(theta_t)*sin(theta_i)
            );
43      double complex coeff = ref_coeff + (1-cpow(ref_coeff,2)) * ref_coeff*var/(1 -
             cpow(ref_coeff,2)*var);
44      return coeff;
45  }
46
47  //Returns the complex transmission coefficient of the wall w for a given angle
        theta_i
48  double complex trans_coeff_wall(Wall w, double theta_i) {
49      double complex epsilon1 = AIR_EPS - I*AIR_SIG/(2*M_PI*FREQ);
50      double complex epsilon2 = w.epsilonr*EPS0 - I*w.sigma/(2*M_PI*FREQ);
51      double theta_t = asin(sqrt(creal(epsilon1)/creal(epsilon2)) * sin(theta_i));
52      double complex ref_coeff = reflec_coeff(epsilon1,epsilon2,theta_i);
53      double complex gamma_m = I*2*M_PI*FREQ*csqrt(MU0*epsilon2);
54      double l = w.l;
55      double s = l/cos(theta_t);
56
57      double complex var = cexp(-2*gamma_m*s + I*BETA*2*s*sin(theta_t)*sin(theta_i)
            );
58      double complex coeff = (1-cpow(ref_coeff,2)) * cexp(-gamma_m*s) / (1 - cpow(
            ref_coeff,2)*var);
59      return coeff;
60  }
61
62  //Returns the average power given the electric field e_field
63  double average_pow(double complex e_field) {
64      return 1/(8*R_A)*pow(cabs(H_E*e_field),2);
65  }
66
67  ////////////////////////////////////
68  //            TRANSMISSION        //
69  ////////////////////////////////////
70
71  /*
72   *  Returns the transmission coefficient of the ray joining the receiver rx and
         the transmitter tx. This
73   *  coefficient is equal to 1 when it is a direct ray. It takes also into account
          multiple transmissions.
74   *  The arguments j and k can be used to exclude 2 walls (of index j and k), j/k
         = -1 when not used
75   */
76  static double complex transCoeff(Vector rx, Vector tx, Wall walls[], int size,
        int j, int k) {
77      double complex coeff = 1;
78      for(int i = 0; i < size; i++) {
79          if(i != j && i != k && wallInTheWay(rx, tx, walls[i])) {
```

```
80              coeff *= trans_coeff_wall(walls[i], incidenceAngle(rx, tx, walls[i]))
                    ;
81          }
82      }
83      return coeff;
84  }
85
86  /*
87   *  Returns the average power of the direct ray joining the receiver rx and the
         transmitter tx
88   *  with or without transmission(s)
89   */
90  double directPathPower(Vector rx, Vector tx, double p_tx, Wall walls[], int size)
        {
91      double complex coeff = transCoeff(rx,tx,walls,size,-1,-1);
92      Vector v = {rx.x - tx.x, rx.y - tx.y};
93      double complex e_direct = coeff * e_field_direct(norm(v),p_tx);
94      return average_pow(e_direct);
95  }
96
97  ////////////////////////////////////
98  //            REFLECTION          //
99  ////////////////////////////////////
100
101 /*
102  *  Returns the reflection power for a multi-reflected ray. pts[] contains the
         receiver point and
103  *  all the (reflected) antennas: {rx,tx,tx',tx",...}. reflecWalls[] contains the
         index of the walls which
104  *  reflect. reflections is the number of reflections incurred
105  */
106 static double reflectionPower(double p_tx, Wall walls[], int size, Vector pts[],
        int reflecWalls[], int reflections) {
107     double complex coeff = 1;
108     Vector pt1 = pts[0];
109     Vector pt2 = pts[reflections+1];
110     int i = reflections+1;
111     while(i > 0) {
112         Wall w = walls[reflecWalls[i]];
113         // If the intersection point is not on the wall -> no reflection
114         if(i > 1 && !wallInTheWay(pt2, pt1, w)) {
115             coeff = 0;
116             break;
117         }
118         else if(i > 1) {
119             coeff *= reflec_coeff_wall(w, incidenceAngle(pt1, pt2, w));
120             Vector inter = intersec(pt1, pt2, w);
121             if(i == reflections+1) {
122                 // Compute the transmission coefficient from the receiver to the
```

```
                    last
123                 // reflection point (excluding the wall on which it is)
124                 coeff *= transCoeff(inter,pt1, walls, size,reflecWalls[i],-1);
125             }
126             else {
127                 // Compute the transmission coefficient from a reflection point
                        to another
128                 // (excluding the 2 walls on which they are)
129                 coeff *= transCoeff(inter,pt1, walls, size,reflecWalls[i],
                        reflecWalls[i+1]);
130             }
131             pt1 = inter;
132             pt2 = pts[i-1];
133         }
134         else if(i == 1) { // Compute the transmission coefficient from the first
                reflection point to tx
135             coeff *= transCoeff(pt1, pt2, walls, size,reflecWalls[2],-1);
136         }
137         i--;
138     }
139     if(creal(coeff) != 0 && cimag(coeff) != 0) {
140         Vector v = {pts[0].x - pts[reflections+1].x, pts[0].y - pts[reflections
                +1].y};
141         double complex e_field = coeff * e_field_direct(norm(v),p_tx);
142         return average_pow(e_field);
143     }
144     else {
145         return 0;
146     }
147
148 }
149
150 // Global variables used for computing the reflection power
151 static Vector pts[2+3];
152 static int reflecWalls[2+3];
153
154 /*
155  *  Recursive function which returns the reflection power for a given receiver rx
        , transmitter tx
156  *  and number of reflections iteration. j indicates the index of the last
        reflected wall in the
157  *  recursivity and is used to compute the reflections below the max number of
        reflections iteration
158  */
159 double reflecRayPower(Vector rx, Vector tx, double p_tx, Wall walls[], int size,
        int iteration, int j) {
160     double power = 0;
161     if(iteration == NumberOfReflections) { // Save rx and tx in pts[] which
            contains the receiver point and
```

```
162             pts[0] = rx;                // all the (reflected) antennas: {rx,tx,tx',tx
                    '',...}.
163             pts[1] = tx;
164         }
165         for(int i = 0; i < size; i++) {
166             if(i == j) { // Reflections below the max number of reflections (
                    iteration)
167                 // Example : iteration = 3, this condition computes the power of the
                        double and single
168                 // reflections
169                 power += reflectionPower(p_tx, walls, size, pts, reflecWalls,
                        NumberOfReflections-iteration);
170             }
171             else if(iteration > 1) { //Recursivity
172                 Vector reflecPt = reflecLine(tx, walls[i]);
173                 pts[2+NumberOfReflections-iteration] = reflecPt;
174                 reflecWalls[2+NumberOfReflections-iteration] = i;
175                 power += reflecRayPower(rx,reflecPt,p_tx,walls,size,iteration-1,i);
176             }
177             else { // iteration = 1, compute the power of the potential reflection
178                 Vector reflecPt = reflecLine(tx, walls[i]);
179                 pts[2+NumberOfReflections-iteration] = reflecPt;
180                 reflecWalls[2+NumberOfReflections-iteration] = i;
181                 power += reflectionPower(p_tx, walls, size, pts, reflecWalls,
                        NumberOfReflections);
182             }
183         }
184         return power;
185     }
```

7.4 geometry

geometry.h

```
1  #ifndef geometry_h
2  #define geometry_h
3
4  #include "main.h"
5  #include "map.h"
6  #include <stdbool.h>
7
8  #define M_PI 3.1415
9
10
11
12  Vector newVec(double x, double y);
13  void vecSub(Vector *v1, Vector v2);
14  double norm(Vector v);
15  double angle(Vector v1, Vector v2);
16
```

```
17 double incidenceAngle(Vector p1, Vector p2, Wall w);
18 Vector intersec(Vector p1, Vector p2, Wall w);
19 Vector reflecLine(Vector v, Wall w);
20 bool wallInTheWay(Vector v, Vector w, Wall m);
21 int diffractionPcts(Vector diffrac[], Wall diffracWalls[], Wall listWalls[], int
       nbWalls);
22 Vector diffracAngle(Vector diffracPos, Wall w, Vector rx, Vector tx);
23
24 #endif
```

geometry.c

```
1  #include "geometry.h"
2  #include <math.h>
3  #include <stdio.h>
4
5  #define MAX(x, y) (((x) > (y)) ? (x) : (y)) // Returns the max between x and y
6  #define MIN(x, y) (((x) < (y)) ? (x) : (y)) // Returns the min between x and y
7
8  Vector newVec(double x, double y) {
9      Vector vec = {x,y};
10     return vec;
11 };
12
13 //Adds the Vector v2 to v1 (v1 + v2)
14 void vecAdd(Vector *v1, Vector v2) {
15     v1->x += v2.x;
16     v1->y += v2.y;
17 }
18
19 //Substracts the Vector v2 to v1 (v1 - v2)
20 void vecSub(Vector *v1, Vector v2) {
21     v1->x -= v2.x;
22     v1->y -= v2.y;
23 }
24
25 //Returns a new Vector which is the substraction of v1 and v2 (v1 - v2)
26 Vector newVecSub(Vector v1, Vector v2) {
27     Vector sub = {v1.x - v2.x, v1.y - v2.y};
28     return sub;
29 }
30
31 //Returns the norm of the Vector v
32 double norm(Vector v) {
33     return sqrt(pow(v.x,2)+pow(v.y,2));
34 }
35
36 //Returns the dot product of v1 and v2
37 double dotProd(Vector v1, Vector v2) {
38     return v1.x*v2.x + v1.y*v2.y;
```

```
39 }
40
41 //Returns the angle in radians between v1 and v2
42 double angle(Vector v1, Vector v2) {
43     return acos(dotProd(v1,v2)/(norm(v1)*norm(v2)));
44 }
45
46 //Returns the incident angle (in radians) between the ray, defined by p1 and p2,
       and the normal of the wall w
47 double incidenceAngle(Vector p1, Vector p2, Wall w) {
48     Vector ray = newVecSub(p2,p1);
49     Vector wall = {w.x2 - w.x1, w.y2 - w.y1};
50     return M_PI/2 - acos(fabs(dotProd(ray,wall)/(norm(ray)*norm(wall))));
51 }
52
53 //Returns the incident angle (in radians) between the two walls m1 and m2
54 double incidenceAngle2(Wall m1, Wall m2) {
55     Vector p1 = newVec(m1.x1, m1.y1);
56     Vector p2 = newVec(m1.x2, m1.y2);
57     return incidenceAngle(p1, p2, m2);
58 }
59
60 //Returns the intersection point of the line defined by the 2 points p1,p2 with
       the wall w
61 Vector intersec(Vector p1, Vector p2, Wall w) {
62     Wall w2 = newWall(p1.x,p1.y,p2.x,p2.y,0,0,0);
63     Vector res = {0,0};
64     if(w.b * w2.b != 0 && w.a != w2.a) {
65         res.x = (w.c - w2.c)/(w2.a - w.a);
66         res.y = w.a * res.x + w.c;
67     }
68     else if(w.b == 0 && w2.b != 0) {
69         res.x = -w.c;
70         res.y = w2.a * res.x + w2.c;
71     }
72     else if(w.b != 0 && w2.b == 0) {
73         res.x = -w2.c;
74         res.y = w.a * res.x + w.c;
75     }
76     return res;
77 }
78
79 //Returns the intersection point of two walls
80 Vector intersec2(Wall m1, Wall m2){
81     Vector p1 = newVec(m1.x1, m1.y1);
82     Vector p2 = newVec(m1.x2, m1.y2);
83     return intersec(p1,p2,m2);
84 }
85
```

```
86  /*
87   *  Returns the reflection of a point over a line
88   *  v the coordinates (x,y) of the point being reflected and w the line (wall)
         over which the point
89   *  v is reflected
90   */
91  Vector reflecLine(Vector v, Wall w) {
92      Vector reflecPt = newVec(w.x2-w.x1,w.y2-w.y1);
93      Vector temp = newVec(v.x-w.x1,v.y-w.y1);
94      double var = 2*dotProd(temp,reflecPt)/dotProd(reflecPt,reflecPt);
95      reflecPt.x *= var;      reflecPt.y *= var;
96      reflecPt.x -= temp.x;   reflecPt.y -= temp.y;
97      reflecPt.x += w.x1;     reflecPt.y += w.y1;
98      return reflecPt;
99  }
100
101 //Returns a boolean to determine whether the wall is in the way or not
102 static bool isBetween(Vector v, Vector w, Vector inter){
103     return inter.x >= MIN(v.x,w.x) && inter.x <= MAX(v.x,w.x) && inter.y >= MIN(v
            .y,w.y) && inter.y <= MAX(v.y,w.y)
104         && !(inter.x == v.x && inter.y == v.y) && !(inter.x == w.x && inter.y ==
                w.y);
105 }
106
107 //Returns a boolean to determine wether the intersection point is located in the
        square ]x1;x2[x]y1;y2[ or not
108 static bool isOnTheWall(Vector inter, Wall m){
109     return inter.x >= m.x1 && inter.x <= m.x2 && inter.y >= MIN(m.y1,m.y2) &&
            inter.y <= MAX(m.y1,m.y2)
110         && !(inter.x == m.x1 && inter.y == m.y1) && !(inter.x == m.x2 && inter.y
                == m.y2);
111 }
112
113 //Returns a boolean to determine wether the point is located on the wall or not
114 static bool isACorner(Vector v, Wall m){
115     return (v.x == m.x1 && v.y == m.y1) || (v.x == m.x2 && v.y == m.y2);
116 }
117
118 /*
119  *  Returns a boolean to determine wether the wall is located between the points
        v and w or not
120  *
121  *  v = absolute position of antenna
122  *  w = point of calculation
123  *  m = wall considered for transmission
124  */
125 bool wallInTheWay(Vector v, Vector w, Wall m) {
126     Vector inter = intersec(v, w, m);
127     //printf("inter: inter_x = %2f, inter_y = %2f\n", inter.x,inter.y);
```

```
128         if(isBetween(v, w, inter)) {
129             return isOnTheWall(inter, m);
130         }
131         else {
132             return false;
133         }
134     }
135 
136     /*  Returns a boolean to determine wether the potentially reflecting wall is
            located between v and w or not
137      *
138      *  v = absolute position of antenna
139      *  w = point of calculation
140      *  m = wall considered for reflection
141      */
142     bool wallInReflexion(Vector v, Vector w, Wall m){
143         Vector ReflecPt = reflecLine(v, m);
144         return wallInTheWay(ReflecPt, w, m);
145     }
```

7.5 plot

plot.h

```
1  #ifndef plot_h
2  #define plot_h
3  
4  #include "geometry.h"
5  
6  /*
7   *  Use this function to plot the points defined in listPts and the lines defined
        in listWalls.
8   *  sizeList1 is the size of listPts and sizeList2 is the size of listWalls
9   */
10 void plot(Emetteur listEmetteurs[], int numberOfEmetteurs, Wall listWalls[], int
        numberOfWalls, Vector listPts[], int power[], int numberOfPoints);
11 
12 #endif
```

plot.c

```
1  #include "plot.h"
2  
3  #include <stdio.h>
4  #include <math.h>
5  #include <stdlib.h>
6  
7  
8  
9  void plot(Emetteur listEmetteurs[], int numberOfEmetteurs, Wall listWalls[], int
       numberOfWalls, Vector listPts[],  int power[], int numberOfPoints) {
```

```
10
11      FILE *data = fopen("data.txt", "w"); //ecriture de fichier
12      int debit;
13      for (int i = 0; i < numberOfPoints; i++) {
14
15                  //if (power[i] >= 54) {debit = 54;}
16                  //else if (power[i] < 6) {debit = 0;}
17                  //else {debit = power[i];}
18
19                  if (power[i] >= -20) {power[i] = -20;}
20                  if (power[i] < -93) {power[i] = -93;}
21
22                  fprintf(data, "%f %f %d \n",listPts[i].x, listPts[i].y , power[i
                        ]);
23
24          }
25
26
27          double dx;
28          double dy;
29          double resolutionWall = 1000;
30          for (int n = 0; n < numberOfWalls; n++ ) {
31
32                  dx = (listWalls[n].x2 - listWalls[n].x1) / resolutionWall;
33                  dy = (listWalls[n].y2 - listWalls[n].y1) / resolutionWall;
34
35                  for (int jj = 0; jj < resolutionWall; jj++){
36                  fprintf(data, "%f %f %d \n",listWalls[n].x1 + jj*dx, listWalls[n
                        ].y1 + jj*dy , -10);
37                  }
38
39          }
40                  for (int k = 0; k < numberOfEmetteurs; k++) {
41
42                  fprintf(data, "%f %f %d \n",listEmetteurs[k].position.x,
                        listEmetteurs[k].position.y , -10);
43
44          }
45
46          fclose(data);
47
48  FILE *cmd = fopen("cmd.txt", "w");
49          fprintf(cmd,
50                  "set view map\n"
51                  "set size ratio .9\n"
52                  "set object 1 rect from graph 0, graph 0 to graph 1, graph 1 back
                        \n "
53                  "set object 1 rect fc rgb 'black' fillstyle solid 1.0 \n "
54
```

```
55                      //"set xrange [0:40] \n"
56                      //"set yrange [0:40] \n"
57
58                      //"set cbrange [0:54] \n"
59                      //"set zrange [0:54] \n"
60
61                      //"set palette defined (-94 'black',-93 'navy', -83 'dark-blue
                            ', -73 'royalblue',-63 'dark-green',"
62                      //"-53 'green',-43 'salmon', -33 'orange', -20 'yellow', -19'
                            white') \n" // , 45'dark-red',"
63                      //"50 'purple', 55'dark-violet', 64'white') \n"    //,200'light-
                            magenta',220'magenta',240'dark-magenta',260'white')"
64
65                      "splot 'data.txt' using 1:2:3 with points pointtype 7 pointsize 1
                             palette linewidth 30 \n"
66
67                       );
68              fclose(cmd);
69
70              //execution
71              system("gnuplot -persistent cmd.txt");
72      }
73
74      //http://gnuplot.sourceforge.net/docs_4.2/node217.html
```