MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI DE TIZI OUZOU
FACULTE DE GENIE ELECTRIQUE ET DE L'INFORMATIQUE
DEPARTEMENT D'INFORMATIQUE

MEMOIRE

Présenté par

Yazid Takfarinas

Pour l'obtention du grade de

Master académique de l'université d'UMMTO

Spécialité : Systèmes d'informatiques

Thème

Localisation dans les réseaux de capteurs sans fil en utilisant les réseaux de neurones

Soutenance prévue le 04 octobre 2011 devant le jury composé de :

Daoui Mehammed	MC Classe B, UMMTO	Président de jury
Aoudjit Rachida	MA Classe B, UMMTO	Membre de jury
Belkadi Malika	MA Classe B, UMMTO	Membre de jury
LALAM Mustapha	Professeur, UMMTO	Pro>moteur

Laboratoire de Recherche en Informatique LARI

REMERCIEMENTS

Gloire et Louange à Allah Seigneur des mondes le miséricordieux, je LE remercie de m'avoir donné la force et la volonté pour mener à bien mon travail.

Je tiens à exprimer mon vifs remerciements pour Monsieur LALAM, pour m'avoir donné l'opportunité d'effectuer ce travail, pour son aide, conseils et son soutient constant

je remercie également les membres de jury qui nous ferons l'honneur d'évaluer ce modeste travail.

Que tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail, l'expression de notre profonde gratitude.

Résumé

La localisation des nœuds dans les réseaux de senseurs est une question d’actualité de la recherche récente, puisqu'elle peut augmenter l'efficacité dans le calcul et réduire au minimum la puissance d'énergie des nœuds.

Une classe des protocoles de calcul nécessite quelques nœuds qui ont leurs informations de localisation, appelés les nœuds d'ancres. Cependant, beaucoup de facteurs influencent les erreurs de position des nœuds. Dans cette classe de protocoles, le placement significatif des nœuds d'ancres réduit considérablement l’erreur sur la localisation des nœuds.

En utilisant l’analyse neuronale, les cartes auto-organisatrices de Kohonen, on montre que l'impact du placement des nœuds d'ancres et un ensemble de règles assurent des meilleurs résultats.

Abstract

Localization of the nodes in sensor networks is an important research issue, since it can enhance the efficiency in computation and minimizes the power consumption of the nodes.

A class of the protocols of calculates require the nodes which know their absolute coordinates, called the nodes of anchor. While many factors influence the errors of position of the nodes, in this class of the protocols, the significant placement of the anchors nodes reduces considerably the errors on the localization of the nodes.

Using the neuronal analysis, the self-organizing charts of Kohonen, we are shown that the impact of the placement of the anchors nodes and a whole of rules ensured the best results.

Dédicaces

Je dédie cet humble travail à ma très chère mère qui n'a jamais manquée de me soutenir tout au long de mes études.

Localisations dans les réseaux de senseurs

Localisation par l'application des réseaux de neurones

Conception & Validation

Figures

Notations

WSN	Wireless Sensors Network : Réseau de senseurs sans fil
N	Le nombre de Capteurs dans le réseau de senseurs
m	Le nombre d'ancres dans le réseau
n	Le nombre de capteurs sans coordonnées (capteur normal)
p	Dimension de l'espace d'entré
s	Dimension de l'espace de sortie
i, j, k	Indices
$\hat{x}$	Valeur de la position estimée
$\hat{d}$	Valeur de la distance estimée
x_{ij}	Le i^{ème} nœud et j^{éme} coordonnée
d_{ij}	La distance entre le nœud i et j
GPS	Global position System
UWB	L'Ultra Wide Band
$RSSI$	Received Signal Strength Indicator
dBm	décibels au-dessus d'un milliwatt
ACC	Analyse en Composante Curviligne : Component Curvilinear Analysis
MDS	MultiDimensional Scaling : l'étalonnage multidimensionnel
RN	Réseau de neurones
D	L'ensemble des observations
A	Sous-ensemble de l'ensemble D
R^p	Ensemble des nombres réels de p dimensions
z_i	L'observation i
W	L'ensemble des référents
w_c	Le référent c
$\mathcal{X}$	Fonction d'affectation
I	Fonction de coût
$x(z)$	représente l'indice de référent auquel est associée l'observation z

P_c	Ensemble des référents
$A \cap P_c$	L'ensemble des observations communes entre l'ensemble A et P_c
t	L'itération actuelle
N_{iter}	Nombre d'itération
$W^{t-1(resp.t)}$	l'ensemble des référents à l'étape précédente (resp. actuelle)
μ^t	représente le pas de la correction pour l'itération t
$\mathcal{C}$	ensemble des neurones interconnectés
$\delta(c,r)$	La longueur du plus court chemin entre le neurone c et r
T	Taille de la carte
SOM	Self-Organizing Map : carte auto-organisatrice
NLM	Non Linear Mapping : carte non linéaire
X_{ij}	La distance entre le nœud i et le nœud j dans l'espace d'entrée (R^p)
Y_{ij}	La distance entre le nœud i et le nœud j dans l'espace de sortie(R^s)
∇E	Gradient de la fonction de coût
$F(.)$	La fonction de pondération
$\alpha(t), \lambda(t)$	Fonctions décroissantes suivant le nombre d'itérations
$O\ (N)$	Complexité linéaire
$O\ (N^2)$	Complexité quadratique
D_{N*N}	Matrice de distances entre les nœuds de réseau
2D,3D	Deux dimensions, trois dimensions
$Noeud$	Capteur
$CCA-MAP$	Carte d'analyse en composante curviligne
σ	Ecart type
$V_c(d)$	voisinage d'ordre d de neurone c
nb.	Nombre
dist.	Distance
Pos.	Position
r	Rayon de portée radio
%	Pourcentage
Xr	Multiplie par r

Introduction Générale

Introduction Générale

Dans les réseaux de capteurs sans fil, la localisation des senseurs joue un rôle important dans la plupart des applications. Quand les senseurs sont déployés dans un réseau, ils ont seulement l'information de connectivité, Cependant, l'information de la localisation reste inconnue.

Les résultats de la recherche récente prouvent que les nœuds avec leurs informations de localisation augmentent l'exécution des applications et réduisent la puissance d'énergie. En outre, un endroit plus précis mène à un résultat d'application plus précis. En résumé, la localisation est une part essentielle de WSN.

Toutefois, les senseurs sont équipés d'un matériel peu couteux, et les solutions actuellement développées telles que le Global Position System (GPS) sont insatisfaisantes concernant les contraintes sur le coût et la puissance d'énergie pour chaque nœud liée au réseau. Cependant, les protocoles de la localisation nécessitent l'information des coordonnées sur un certain nombre de nœuds, ces derniers appelés les nœuds d'ancres. En effet, la problématique liée à la localisation est celui du placement de ces nœuds dans un réseau de capteur, c'est à dire comment positionner ces nœuds les uns par rapport aux autres afin d'obtenir un meilleur résultat.

Dans ce travail on a proposé une démarche de localisation par l'application des réseaux de neurones qui détermine les positions automatiques de nœuds d'ancres afin de conduire le système de réseau de capteurs à un résultat plus précis quelque soit sa topologie.

Pour mieux cerner les enjeux du sujet, nous présenterons dans le premier chapitre des généralités sur la localisation dans les réseaux de capteurs, tel que leur architecture, leurs usages, la manière dont ils sont constitués, ainsi que la question de la localisation d'un nœud appartenant au réseau de capteurs, on présentera aussi les méthodes et les algorithmes de localisation.

Introduction Générale

Dans le second chapitre nous décrivons le principe de la localisation appliquons les réseaux de neurones notamment les cartes auto-organisatrices, ainsi que deux exemples de localisations qui emploies les méthodes neuronales MDS, ACC.

Dans le dernier chapitre, nous trouverons une description du simulateur d'analyse neuronale. L'efficacité du simulateur tourne autour de la possibilité d'effectuer un apprentissage supervisé de données qui permettent d'obtenir des résultats très intéressants.
Finalement, dans le même chapitre, on validera notre démarche de localisation par la simulation de deux exemples de réseau de senseurs, puis on testera les résultats obtenus sous Matlab.

Chapitre I :

Localisation dans les réseaux de senseurs

Chapitre 1 : Localisation dans les réseaux de senseurs

I. Introduction

Les réseaux de senseurs sans fil sont nés dans le contexte de brassage et de globalisation technologique. Ils sont constitués de petites unités déployées dans un certain environnement et vouées à accomplir une tâche donnée. Chaque unité est un micro-ordinateur autonome accompagnée d'une radio sans fil à courte portée, ainsi que d'une série de senseurs.

Ce qui est intéressant et nouveau par rapport aux ordinateurs classiques : le senseur est intégré dans le monde réel et a une position qui peut changer avec le temps. Il est équipé de senseurs qui lui permettent de mesurer des grandeurs physiques liées à l'environnement et au temps et il peut communiquer avec les nœuds voisins. En général, la localisation consiste à trouver la localité (ou position) d'un ou plusieurs objets.

Les systèmes de localisation automatique existants sont nombreux et utilisent plusieurs voies technologiques, ils sont très différents les uns des autres et répondent en général à des besoins différents (Localisation dans une pièce ou à l'échelle de la planète, localisation précise ou imprécise, localisation d'un seul ou plusieurs objets, localisation statique ou suivi de cible,…).

II. Technologies et algorithmes de localisation

II.1. Architecture d’un WSN

Un réseau de senseurs est composé de trois couches : la première coche : le réseau sans fil qui est composé de l’ensemble des éléments du réseau, la deuxième coche : le réseau de senseurs qui est composé des éléments qui peuvent recevoir une information du monde extérieur, et la troisième couche : les grappes (clusters) de capteurs qui réalisent des tâches complexes de traitement du signal. La figure 1 montre l’organisation hiérarchique des réseaux de capteurs.

Dans la plupart des cas, un traitement local de l'information est effectuée par les capteurs, et l'information utile est transmise à un nœud central (au niveau de la grappe), qui réunit et analyse ces informations. [12]

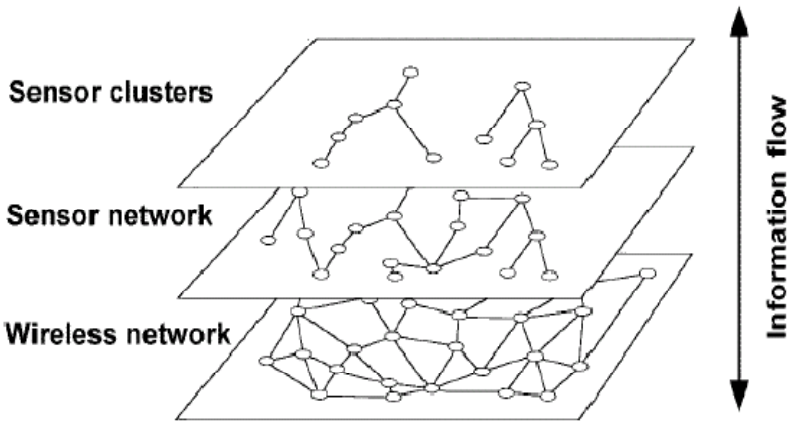


Figure I.1 *: Architecture d'un réseau de capteurs*

II.2. Le Nœud : Composition d'un Capteur

Un nœud d'un réseau de capteur est composé de quatre sous-systèmes principaux.
Il s'agit des sous-systèmes de calcul, de communication, de capteur, et de génération de courant. [12]

a. **Le sous-système de Calcul :** Il est aussi appelé MCU, MicroController Unit, a la charge du contrôle des capteurs, et des protocoles de communication. Il peut fonctionner selon des modes différents, en fonction de la consommation souhaitée ou de la quantité d'énergie restante dans la batterie.

b. **Le sous-système de communication :** C'est un système radio à courte portée. Il peut fonctionner en quatre modes : Transmit, Receive, Idle, Sleep.

c. **Le sous-système de capteurs :** C'est l'ensemble de capteurs et d'actionneurs, qui relie le noeud au monde extérieur. La consommation peut être minimisée par l'usage de composants à basse consommation et par la réduction des performances globales.

d. **Le sous-système de génération de courant :** La batterie. L'enjeu est d'augmenter au maximum sa durée de vie, ce qui peut conduire à l'éteindre complètement si l'on ne s'en sert pas.

II.3. Les contraintes et challenges imposés par les WSN

Nous pouvons dénombrer plusieurs contraintes :

En premier lieu, la faible longévité des batteries nous impose d'être économes, autant que possible, sur toutes les ressources qui consomment de l'énergie. La ressource la plus demandeuse est la radio qui a une consommation supérieure de un à plusieurs ordres de grandeurs sur toutes les autres ressources.

La puissance de calcul dérisoire de chaque nœud empêche également les traitements trop complexes, à moins de les implémenter de façon distribuée, avec les difficultés qui en résultent.

Les taux de transmission radio entre les nœuds sont très bas.

Les nœuds sont dans un réseau ad hoc. Un réseau ad hoc est un réseau dans lequel chaque nœud est disposé à transmettre des données pour d'autres nœuds, et ainsi la détermination des nœuds par lesquels passeront des données forwardées (le relais des données de réseau à l'autre par des nœuds dans un réseau informatique) est faite automatiquement en fonction de la connectivité. En contraste avec les technologies réseau plus anciennes, dans lesquelles certains nœuds - appelés routeurs, switches ou hubs - ayant généralement un hardware spécialisé accomplissent les tâches du routage des paquets.[17]

II.4. Localisation dans WSN

La connaissance des positions des senseurs dans l'environnement est souvent souhaitable, afin de pouvoir déterminer l'origine des flux de mesures collectées. La méthode de localisation étudiée dans ce mémoire a pour but d'estimer ces positions de manière automatique.

Une méthode de localisation dans les réseaux de senseurs est composée de deux parties :

- **Estimation des distances :**

Dans cette phase, les nœuds communiquent entre eux et collectent différents indicateurs de qualité des communications radios. Le hardware radio peut rapporter diverses informations sur le signal radio entre deux nœuds. En effet le simple fait qu'ils communiquent entre eux nous indique qu'ils sont à portée radio l'un de l'autre.

De plus le hardware radio de nos nœuds peut nous rapporter diverses caractéristiques à propos du signal radio entre les deux nœuds, à partir desquelles les distances séparant les nœuds peuvent être estimées.

- **Dérivation des positions :**

Le but de cette phase est de trouver les positions des nœuds qui respectent au mieux les distances inter-nœuds estimées. Si nous connaissons la position de quelques nœuds du réseau dans un certain système de coordonnées, les positions des autres nœuds dans ce système de coordonnées peuvent être trouvées.

Notons que divers raffinements peuvent être appliqués, par exemple : Les distances estimées entre les nœuds peuvent être raffinées en refaisant communiquer les nœuds. De la même façon, on peut affiner la position des nœuds et appliquer diverses vérifications sur leurs positions. Une dernière transformation des positions de nœuds peut être réalisée afin de les intégrer dans une carte existante.

On peut voir un schéma de la localisation dans les WSN dans la Figure 3.

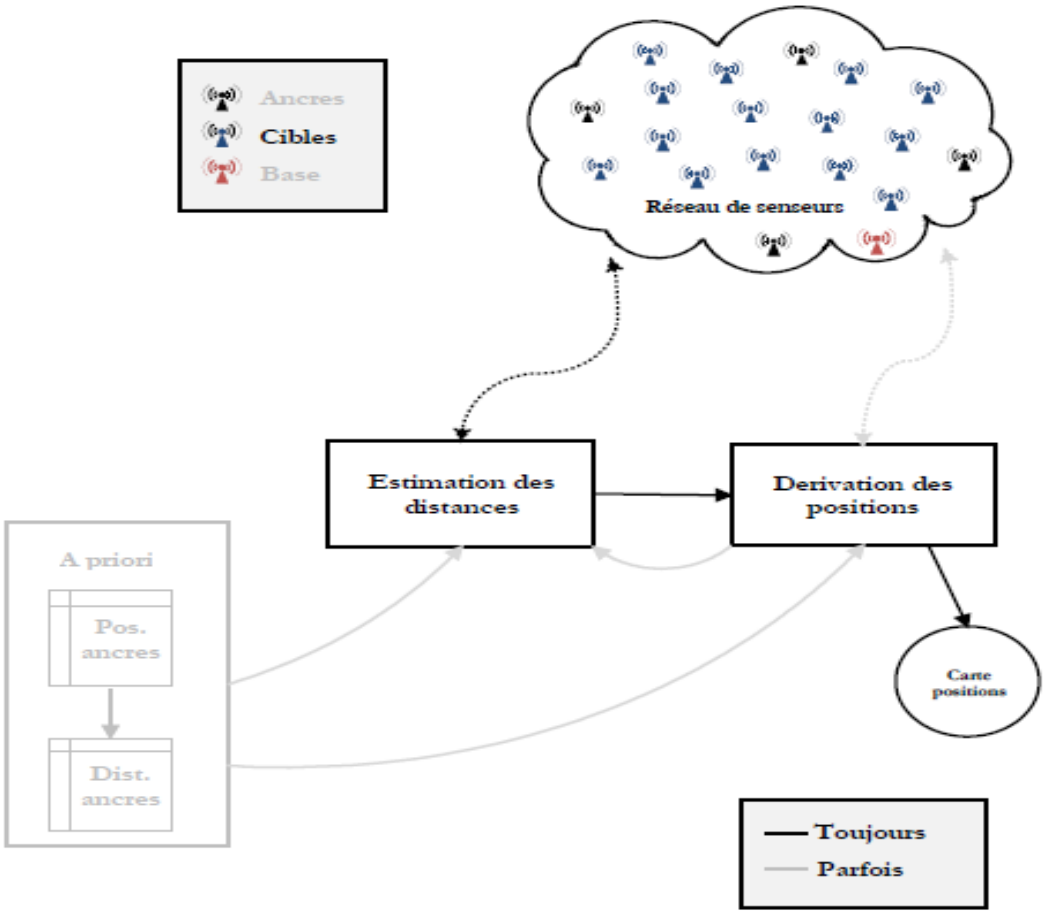


Figure I.2 : Cette figure représente le processus de localisation dans les réseaux de senseurs de la façon la plus générale possible. On peut remarquer qu'il y a des parties noires et des parties grises. Les parties noires sont communes à toutes les méthodes de localisation. Les parties grises dépendent des différentes méthodes de localisation.

II.5. Caractérisations des méthodes :

Il existe de nombreuses approches pour résoudre le problème de la localisation et cha que méthode a ses avantages et ses inconvénients.

II.5.1. Utilisation d'estimations de distances

- **Les méthodes range-free** : Ces méthodes ne calculent jamais de distances entre voisins. Elles utilisent d'autres informations telles que la connectivité pour identifier la position des nœuds. Elles semblent donner de bons résultats dans les réseaux denses et réguliers.

- **Les méthodes range-based :** Ces méthodes estiment les distances entre les nœuds et ensuite dérivent de ces distances les positions des nœuds.

II.5.2. Nécessité de connaître la position d'ancres

Si une méthode requiert l'encodage au préalable de la position d'un certain nombre d'ancres, cela signifie qu'il faudra qu'une personne intervienne avant un déploiement pour mesurer la position d'un certain nombre de nœuds. Cela est parfois difficile, voire impossible dans certaines situations.

Et donc, le fait que la méthode de localisation requiert ou non de connaître la position d'un certain nombre d'ancres est une caractéristique importante de la méthode.

- **Les méthodes anchor-based** : Sont celles qui ne fonctionnent pas sans connaître la position d'un certain nombre d'ancres à priori.

- **Les méthodes anchor-free** : Sont celles qui n'ont besoin de la position d'aucun nœud pour fonctionner ; elles créent donc une carte relative du réseau. Par relative, nous entendons une carte qui est à une translation, une rotation orthogonale, une réflexion et une dilatation près de la 'vraie' carte. Autrement dit, c'est une carte qui conserve les rapports entre les distances entre tous les points.

II.5.3. Forme d'implémentation

Nous distinguons plusieurs façons d'implémenter le processus de localisation :

- **Les méthodes centralisées** : Tous les nœuds communiquent avec leurs voisins et renvoient à l'ordinateur central soit des informations sur le signal, soit directement les distances. L'ordinateur central s'occupe si nécessaire d'estimer les distances à partir des informations sur le signal et ensuite de localiser les nœuds. [18], [19]

- **Les méthodes distribuées :** Ici tous les nœuds communiquent avec leurs voisins pour estimer les distances et échangent leurs informations de voisinage. Ils dérivent ensuite de

façon distribuée la position de tous les nœuds dans le réseau. C'est-à-dire qu'à la fin du processus de localisation, chaque nœud doit connaître sa position ainsi que celles de ses voisins et ce sans l'aide d'un ordinateur central qui effectuerait les calculs. Pour les grands réseaux, on considère qu'une méthode distribuée est nécessaire car les méthodes centralisées demanderaient trop de communication pour l'acheminement des informations vers l'unité centrale et consommeraient donc trop d'énergie. [20],[21]

II.6. Autres critères des algorithmes de localisation

Nous distinguons beaucoup d'autres critères à propos des méthodes de localisation. Nous en donnons ici une liste non exhaustive. Il est très intéressant de les découvrir même si dans la pratique, il est impossible de tenir compte de tous ces critères lors du développement d'un algorithme de localisation. Néanmoins, il peut être intéressant de les garder à l'esprit affin de pouvoir rendre notre méthode meilleure selon tel ou tel critère.

Précision de la localisation : Nous parlons de l'erreur qu'il y a entre les vraies positions des nœuds et les positions calculées par la localisation.

Coût énergétique de la localisation : Dans les WSN, une gestion de l'énergie très économique est nécessaire et comme le facteur dominant de la consommation d'énergie est la communication radio, il faut trouver un algorithme qui communique le moins possible via la radio.

Robustesse au bruit : Il faut analyser comment un algorithme se comporte face au bruit rencontré dans les mesures de distances avec les voisins.

Passage à l'échelle : Est-ce qu'un algorithme fonctionne sur un réseau de plusieurs milliers de nœuds ? Et si oui, est-il toujours aussi efficace. Ce critère est en rapport avec le fait qu'un algorithme soit implémentable de façon distribuée ou non.

Tolérance à la faible connectivité : Est-ce qu'un algorithme fonctionne dans un réseau à faible connectivité (un réseau où chaque nœud ne sait communiquer qu'avec un petit nombre de ses voisins) ? Comment sont affectées les performances d'un algorithme face à cette situation ?

Réactivité du système : Avec quelle rapidité le système de localisation nous renvoie-t-il les positions des nœuds ? Ceci est particulièrement important lorsque l'on veut s'occuper de nœuds mobiles et de suivi de cibles.

II.7. Localisation : technologies et algorithmes

II.7.1. Notations

Cette section a pour but d'introduire la notation utilisée pour le problème de la localisation automatique dans un réseau de senseurs communiquant entre eux.

Soit N le nombre de nœuds déployés dans un certain environnement et m ≥ 0 le nombre de nœuds appelés ancres et étiquetés : 1, . . . ,m dont la position est connue à priori.

L'objectif de la procédure de localisation est de trouver la position de n > 0 noeuds cibles étiquetés : $m + 1, \ldots, N$ et donc $N = n + m$.

On représente la position réelle du nœud i dans un espace euclidien de dimension p par un vecteur dans R^p : $x_i = (x_{i1}, \ldots, x_{ip})$, $1 \leq i \leq N$. p est en général égal à 2 ou 3.

Nous définissons $Xancres = [x_i]$, 1 ≤i ≤m, pouvant être représenté par une matrice de taille $p \times m$ qui est connue au début.

Le but de l'algorithme est de trouver $Xcibles = [x_j], m + 1 \leq j \leq N$, pouvant être représentés par une matrice de taille $p \times n$. Soit $\hat{x}_j$ notre estimation de x_j .

Le problème revient à minimiser la fonction d'optimisation suivante :

$$J_1(\hat{x}) = \sum_{m+1 \leq i \leq N} \sqrt{(x - \hat{x})^2}$$

Pour estimer ces positions nous utilisons la communication dans le réseau affin d'estimer les distances entre les nœuds.

Soit $d_{ij} = ||x_i - x_j|| = \sqrt{\sum_{k=1}^{p} (x_{ik} - x_{jk})^2}$ la distance euclidienne entre 2 nœuds i et j. Et soit $\hat{d}_{ij}$ notre estimation de d_{ij} .

On peut maintenant reformuler cela comme un autre problème d'optimisation où il faut minimiser la fonction suivante :

$$J_2(\hat{x}) = \sum_{m+1 \le i,j \le N} (\hat{d}_{ij} - ||\hat{x}_i - \hat{x}_j||)^2$$

Nous ne connaissons au début que l'ensemble des distances ancres-ancres qui peuvent être calculées à partir de leurs positions (connues)

$$D_{aa} = \left\{ d_{ij} = \left|\left| x_i - x_j \right|\right| = \sqrt{\sum_{k=1}^{p} (x_{ik} - x_{jk})^2} \right\}, 1 \le i,j \le m$$

Dans certains cas, le nombre d'ancres m est égal à zéro et donc $D_{aa} = \emptyset$. Comme nous nous plaçons dans le cas d'une localisation statique, on considère que deux nœuds i, j peuvent communiquer ensemble s'ils ont pu communiquer suffisamment longtemps pour permettre d'estimer la distance les séparant.

II.7.2. Technologies

II.7.2.1. Infrarouges

On distingue plusieurs types d'utilisation de l'infrarouge pour la localisation. Premièrement on peut placer une série d'émetteurs infrarouges autour d'une zone et alors un dispositif muni de détecteurs infrarouges peut, en utilisant les caractéristiques des divers signaux infrarouges émis par les différentes sources, se localiser et trouver son orientation dans la zone couverte par les émetteurs. [22]

On utilise également les infrarouges pour permettre à des robots de se localiser dans un environnement (chaque robot a des émetteurs et des récepteurs infrarouges placés dans plusieurs directions et peut émettre des infrarouges. L'intensité de lumière reçue en retour permet d'estimer la distance entre eux.

Technologie	Remarques
Infrarouge	Doit être en ligne de vue Inutilisable en lumière du soleil directe Précision 5-10m
Ultrasons	Pas d'obstacle Précision 1-10m
Radion Fréquance	Tiré utilisé GPS précision : 5-10m RFID précision : 5cm-5m Cellular précision : 50 100m Wifi précision : 2-50m Bluetooth, Zigbee : 2-10m
Optique	Doit être en ligne de vue Bonne précision
UWB radion	Expérimental Précision 6-10cm

Figure I.3 *: les technologies de localisation* [23]

II.7.2.2. Sons et Ultrasons

Il y a eu beaucoup de recherches sur la localisation à base de sons ou ultrasons. Le principal avantage de la technologie est que la vitesse de propagation du son est assez lente, en comparaison à celle des ondes.

Cela permet de mesurer les temps de propagation précisément et ainsi d'obtenir des estimations de distances fiables. Le système cricket [24], par exemple, combine des communications par radio et des émetteurs/récepteurs de sons pour se localiser.

Pour calculer les distances jusqu'à ses voisins, un 'cricket' envoie simultanément une onde radio et une onde sonore ayant une certaine forme caractéristique. Les nœuds qui reçoivent l'onde radio démarrent un minuteur et attendent l'arrivée de l'onde sonore. Lorsque celle-ci arrive ils consultent le minuteur et peuvent ainsi déduire le temps de propagation du

son. A partir de la ils calculent les distances de façon assez précise et en déduisent les positions.

Pour comprendre cela on peut se souvenir que lorsqu'il y a des orages, on compte parfois le nombre de secondes entre l'éclair lumineux et le coup de tonnerre pour en déduire la distance de l'éclair. Ici l'éclair lumineux est l'onde radio émise et le coup de tonnerre est l'onde sonore caractéristique.

La seule restriction de ce système est qu'il ne peut être utilisé sur de trop longues distances et qu'il est parfois difficile à utiliser dans des environnements trop bruyants ou ayant des obstacles ou des murs. [25, 26]

II.7.2.3. Radio et fréquences

II.7.2.3.1. GPS : Global Position System

La technologie de localisation la plus répandue est sans doute le GPS. Il remplit très bien son rôle de système de localisation à l'échelle planétaire. Le GPS appartient au département de la défense aux Etats-Unis. Son nom officiel est NAVSTAR system (Navigation Satellite Timing and Ranging).

Il est composé de 3 parties ou segments : Le segment spatial est composé d'au moins 24 satellites qui tournent autour de la terre à une altitude d'approximativement 20 km. Chaque satellite fait 2 tours de la terre en 24h. Ils sont agencés de telle façon que chaque endroit (ou presque) de la terre ait à tout moment au moins 6 satellites en ligne de vue (en plein air).

Ces trajectoires sont optimisées pour fournir la meilleure couverture dans la zone comprise entre la latitude 75° Nord et la latitude 75° Sud. Le segment de contrôle est composé de plusieurs sites dispersés tout autour de la terre.

Ils enregistrent les mouvements des satellites et envoient leurs données à une station centrale. Une ou deux fois par jour la station centrale transmet divers informations aux satellites (mise à jour d'horloge, correction de trajectoire, ...) Les satellites émettent en continu des signaux radios. Ces signaux radios contiennent une description des trajectoires de chaque satellite. Les récepteurs GPS reçoivent ces informations et connaissent ainsi les positions des

satellites. Ils calculent également les temps que prends l'onde radio pour arriver de chaque satellite et en déduisent ainsi les distances jusqu'à chacun d'eux. Ils déduisent ensuite, en utilisant la multilatération, leurs positions.[27,28,29]

II.7.2.3.2. RF-UWB : Ultra Wide Band

L'Ultra Wide Band est un nouveau type de radio sans fil. Il est caractérisé par une grande largeur de bande par rapport à la fréquence centrale des ondes émises. Il y a deux facteurs derrière le concept UWB : (1) Bande passante relativement large et (2) une fréquence centrale relativement petite.

La grande largeur de bande permet des résolutions temporelles précises et dans des systèmes bien architecturés une meilleure confidentialité. La basse fréquence centrale, quant à elle devrait permettre un meilleur passage des ondes à travers les différents matériaux. Divers systèmes de localisation ont été testés sur les radios UWB et semblent donner des résultats très prometteurs. Néanmoins nous n'avons pas trouvé de dispositif UWB commercialisé. Il semble que cette technologie en soit encore à stade expérimental. [30]

II.7.2.3.3. RF-WIFI, RF-Bluetooth

Les méthodes utilisées pour la localisation dans les réseaux WIFI et Bluetooth sont assez comparables à celles utilisées dans les réseaux de senseurs. Ces 3 types de réseaux utilisent des technologies radio assez comparables (le standard radio des WSN est un sous-standard de Bluetooth).[17]

II.7.2.4. Image

Les méthodes de localisation par la vision sont nombreuses, basées sur du traitement d'images et sont très différentes des méthodes de localisation habituelles.

La difficulté consiste à extraire des informations pertinentes des pixels formant l'image. L'extraction d'information depuis des images forme un domaine de recherche à part entière et qui sort du cadre de ce mémoire. Divers travaux de recherche sont en cours dans ce domaine.[17]

II.7.3. Méthodes de localisation.

Nous donnons dans cette section un aperçu non exhaustif des méthodes de localisation pour les réseaux de senseurs.

II.7.3.1. Bounding Box

L'algorithme Bounding Box est un algorithme assez simple, il utilise les distances $\hat{d}_{ij}$ pour contraindre les coordonnées inconnues de la cible i par rapport aux coordonnées connues de l'ancre j. Ces contraintes prennent la forme suivante :

$$x_{j1} - \hat{d}_{ij} < x_{i1} < x_{j1} + \hat{d}_{ij}$$
$$x_{j2} - \hat{d}_{ij} < x_{i2} < x_{j2} + \hat{d}_{ij}$$

L'algorithme calcule la position d'une cible comme étant au centre du rectangle formé par l'intersection des rectangles formés par toutes les contraintes (on travaille à 2 dimensions). [32]

II.7.3.2. GPS-Less

Un nombre fixé de nœuds spéciaux, appelés nœuds 'beacon', sont placés à l'avance. Ils ont des régions de couverture superposées et servent de points de référence. Chaque nœud beacon transmet un message 'beacon' tous les temps T, qui contient sa position.

Un nœud voulant se localiser écoute pendant un certain temps les beacons. Il choisit un certain nombre de nœuds beacons et calcule sa position en prenant la centroïde des positions des beacons choisis.

Pour choisir les beacons, il calcule pour chaque beacon i entendu :

$$CMi = \frac{nombre\ messages\ envoyes\ par\ i}{i\ nombre\ message\ reçus} * 100$$

Il calcule cela pendant un certain laps de temps (en sachant que tous les temps T chaque beacon est censé avoir émis un message) et choisit tous les nœuds qui sont au dessus d'un certain seuil (par exemple 90%).

Le désavantage principal de cette méthode est le fait d'avoir à placer ces nœuds beacons partout où l'on veut être localisable, De plus les nœuds 'beacons' consomment une grande quantité d'énergie. [18]

II.7.3.3. APIT

La méthode APIT est spécialement optimisée pour le critère de la consommation d'énergie. Elle nécessite un certain nombre de nœuds 'beacon' comme pour le GPS-Less. Il est spécifié que ces nœuds ont des émetteurs longue portée. Ces beacons émettent des messages contenant leur position en continu.

Un noeud qui veut se localiser écoute les beacons passants. Pour chaque triplet de beacons différent découvert, il crée un triangle (il connait les positions des beacons). Il rejette certains d'entre eux via un test appelé PIT test. Il trouve ensuite l'intersection des triangles restants et obtient ainsi une région dans laquelle se trouve le nœud. Le centroïde de cette région est pris comme estimation de position. On peut voir cela de façon intuitive dans la Figure 5.[32]

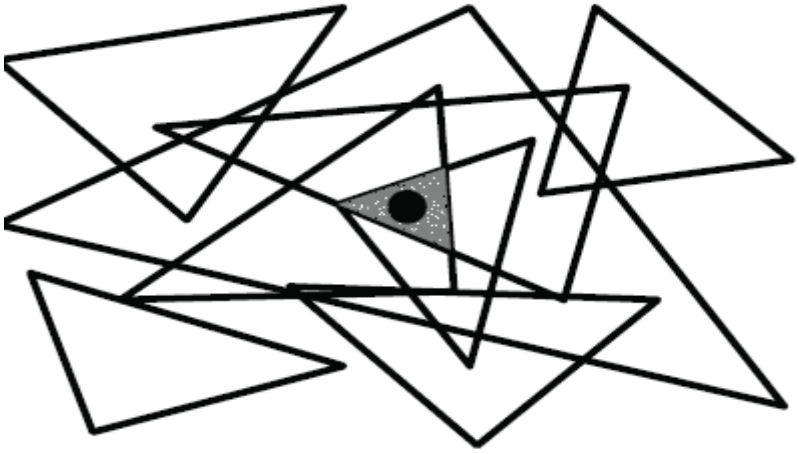


Figure I.4 *: La méthode APIT*

II.7.3.4. GPS-Free

Le GPS-Free est une méthode distribuée qui se déroule en 2 phases. Tout d'abord chaque nœud estime les distances à tous ses voisins. L'ensemble formé du nœud et de ses voisins est appelé un patch. Chaque patch se localise dans son propre système de coordonnées. Dans la

deuxième phase on procède au 'mélange' de tous les patchs pour les placer dans le même système de coordonnées.

Pour localiser un patch : le noeud i central de chaque patch se place en (0,0). Un autre nœud j au hasard sera positionné en $(\hat{d}_{ij}, 0)$ et le 3ème noeud k sera ainsi en $(\hat{d}_{ij}cos(a), \hat{d}_{ij}sin(a))$,

$$a = arcos\left(\frac{\hat{d}_{ij}^{\,2} + \hat{d}_{ik}^{\,2} - \hat{d}_{jk}^{\,2}}{2(\hat{d}_{ij} + \hat{d}_{ik})}\right)$$

Pour localiser les nœuds restants de chaque patch on utilise la trilatération. Deux patchs peuvent être 'mélangés' en utilisant les nœuds communs à ces deux patchs. On applique une transformation à un des patchs pour placer les nœuds communs sur les mêmes positions. On peut répéter ce processus pour plusieurs patchs et ainsi localiser tous les nœuds les uns par rapport aux autres dans le réseau. [33]

III. Estimation des distances

Nous distinguons deux parties pour l'estimation des distances :

- Estimation des distances à partir des liens de communications
- Calcul des distances manquantes

Un exemple intuitif d'estimation des distances est donné Figure 6

III.1. Estimation des distances à partir des liens de communication

L'estimation de distance peut se faire sur a base de différents indicateurs :

- Le temps de propagation d'une onde.
- La puissance du signal à la réception (RSSI : Received Signal Strength Indicator).

De manière générale : Si c_{ij} est la valeur de l'indicateur entre les noeuds i et j et si la fonction f nous permet de déduire une estimation de la distance entre i et j (que nous notons $\hat{d}_{ij}$), nous avons donc :

$$\hat{d}_{ij} = f(c_{ij})$$

Selon la nature de l'indicateur, cette fonction f peut ne pas être connue à priori.

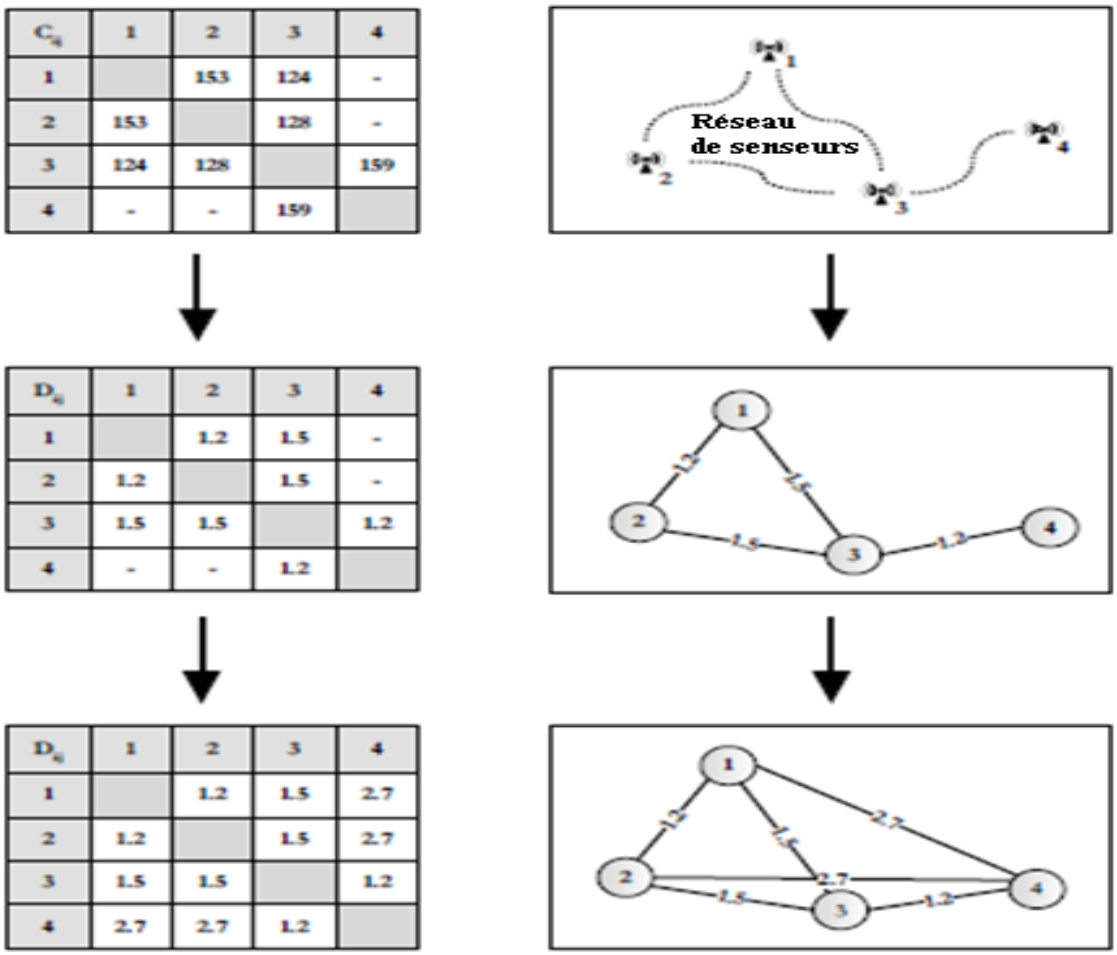


Figure I.5 *: Exemple de collecte d'informations de voisinage.*
Dans cet exemple l'algorithme de localisation requiert toutes les distances.

1. *Les valeurs de l'indicateur pour les liens de communication sont collectées.*
2. *Ces valeurs sont transformées en distances.*
3. *Les distances manquantes sont complétées.*

III.1.1. Le temps de propagation

En général, f est connue à priori et traduit le rapport entre distance et temps de propagation de l'onde dans le milieu de propagation. Les méthodes existantes mesurent le temps de propagation d'une onde sonore plutôt que celui d'une onde radio.

Un nœud émet simultanément une onde radio et un signal sonore prédéfini. Les nœuds aux alentours reçoivent d'abord le message radio, lancent un minuteur et attendent le signal

sonore. Lorsqu'il arrive, le temps de propagation de l'onde sonore est calculé et la distance estimée.

Ces méthodes semblent donner de très bonnes précisions pour l'estimation des distances (de l'ordre de la cinquantaine de centimètres) mais elles nécessitent du matériel adapté et coûteux.

III.1.2. Le RSSI (Received Signal Strength Indicator)

C'est un indicateur de la puissance de signal au moment de la réception d'un message.
La puissance du signal radio peut être calculée par la formule de calibration :

$$P = RSSI_{value} + RSSI_{offset}[dBm]$$

où l'offset est une correction trouvée empiriquement lors de la conception du hardware.
Un modèle très répandu existe pour représenter le RSSI en fonction de la distance : le modèle log-normal shadowing

$$RSSI(d) = P_T - PL(d_0) - 10\eta log_{10}\left(\frac{d}{d_0}\right) + X_\sigma$$

où P_T est la puissance de transmission, $PL(d_0)$ est la perte de signal pour une distance de référence d_0 et η est l'exposant de la perte de signal. Les variations aléatoires dans les RSSI sont exprimées comme une variable aléatoire gaussienne de moyenne 0 et de variance σ^2 : $X_\sigma - N(0, \sigma^2)$. [35]

III.2. Calcul des distances manquantes pour la dérivation des positions

En général les distances déduites des liens de communication ne sont pas suffisantes pour la deuxième partie qui consiste à dériver les positions. Pour calculer les distances manquantes, il existe plusieurs possibilités, dont :

a. La méthode Sum-dist : Si 2 nœuds a et b ne peuvent pas communiquer entre eux mais peuvent tous les deux communiquer avec un nœud c intermédiaire, nous pouvons déduire la distance entre a et b par $d_{ab} = d_{ac} + d_{cb}$. Cette méthode ne donne néanmoins pas toujours de bons résultats, surtout dans des réseaux ou la topologie est très irrégulière. Elle a le mérite d'être simple et peu coûteuse.

b. La méthode Euclidienne : une autre méthode appelée Euclidienne qui est basée sur la géométrie locale des nœuds autour d'une ancre. C'est une méthode géométrique qui utilise les propriétés des angles dans les triangles pour estimer les distances manquantes. [17]

IV. Dérivation des positions

IV.1. Distances inter-nœuds requises

Les méthodes nécessitent des données différentes si elles sont anchor-based ou anchor-free, nous détaillons ici ces différents prés requis :

- Les méthodes anchor-based : Dans ce cas nous devons connaître la position à priori d'au moins 3 nœuds ($m \geq 3$) Pour localiser les $N - m$ nœuds, nous utilisons les distances ancres noeuds : $\hat{d}_{ij}, 1 \leq i \leq m, m+1 \leq j \leq N$, que l'on peut représenter sous la forme d'une matrice $DA_{m,n}$ qui doit être complète.

- Les méthodes anchor-free Dans ce cas nous n'avons besoin d'aucune position à priori $(m = 0)$. Ces méthodes localisent les N noeuds sur la base de toutes les distances internoeuds $\widehat{d}_{ij}, 1 \leq i \leq m, m+1 \leq j \leq N$. Ces distances forment une matrice $DN_{N,N}$ qui doit être complète, symétrique et à diagonale nulle.

IV.2. Multilatération

L'algorithme de multilatération. C'est l'algorithme utilisé par le GPS. Plusieurs autres systèmes de localisation utilisent également des variantes de multilatération. La Multilatération est une méthode relativement simple et intuitive.

IV.2.1. Théorie

Le fonctionnement de la multilatération va être décrit en détail. Une description détaillée de cet algorithme peut être trouvée dans [35].

La multilatération nous permet de retrouver la position d'un nœud à partir des positions d'un certain nombre d'ancres et des distances entre ces ancres et le nœud à localiser.

Soit une cible a dont on veut trouver la position $\hat{x}_a$, et soit m ancres i dont nous connaissons les positions $x_i \in R^p, 1 \leq i \leq m$.

Nous supposons que nous connaissons aussi une estimation des distances $\hat{d}_{ia}, 1 \leq i \leq m$ entre chaque ancre i et le nœud a.

$$(x_{11} - x_{a1})^2 + (x_{12} - x_{a2})^2 + \ldots + (x_{1p} - x_{ap})^2 = \hat{d}^2{}_{1a}$$
$$\ldots$$
$$(x_{m1} - x_{a1})^2 + (x_{m2} - x_{a2})^2 + \ldots + (x_{mp} - x_{ap})^2 = \hat{d}^2{}_{ma}$$

Le système peut être linéarisé en soustrayant la dernière équation des $m - 1$ équations précédentes.

$$\begin{aligned}
&x_{11}^2 - x_{m1}^2 - 2(x_{11} - x_{m1})x_{a1} \\
&+x_{12}^2 - x_{m2}^2 - 2(x_{12} - x_{m2})x_{a2} \\
&\qquad + \ldots \\
&+x_{1p}^2 - x_{mp}^2 - 2(x_{1p} - x_{mp})x_{ap} && = \hat{d}_{1a}^2 - \hat{d}_{ma}^2 \\
&\qquad \vdots && \qquad \vdots \\
&x_{(m-1)1}^2 - x_{m1}^2 - 2(x_{(m-1)1} - x_{m1})x_{a1} \\
&+x_{(m-1)2}^2 - x_{m2}^2 - 2(x_{(m-1)2} - x_{m2})x_{a2} \\
&\qquad + \ldots \\
&+x_{(m-1)p}^2 - x_{mp}^2 - 2(x_{(m-1)p} - x_{mp})x_{ap} && = \hat{d}_{(m-1)a}^2 - \hat{d}_{ma}^2
\end{aligned}$$

En réordonnant les termes, nous obtenons un système d'équations linéaires de la forme $Ax = b$ où :

$$\mathbf{A} = \begin{bmatrix} 2(x_{11} - x_{m1}) & \ldots & 2(x_{1p} - x_{mp}) \\ \vdots & & \vdots \\ 2(x_{(m-1)1} - x_{m1}) & \ldots & 2(x_{(m-1)p} - x_{mp}) \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} x_{11}^2 - x_{m1}^2 + \ldots + x_{1p}^2 - x_{mp}^2 + \hat{d}_{ma}^2 - \hat{d}_{1a}^2 \\ \vdots \\ x_{(m-1)1}^2 - x_{m1}^2 + \ldots + x_{(m-1)p}^2 - x_{mp}^2 + \hat{d}_{ma}^2 - \hat{d}_{(m-1)a}^2 \end{bmatrix}$$

Comme nous avons des erreurs dans les estimations de distances, nous ne pouvons pas trouver de solution exacte à ce système d'équations. La solution la plus proche (au sens des moindres

carrés) de la solution exacte est obtenue en minimisant $(Ax - b)^T (Ax - b)$. Et donc : $\hat{x}_a = argmin(Ax - b)^T (Ax - b) = (ATA)^{-1}A^T b$.

Où $\hat{\mathbf{x}}_a = \begin{bmatrix} \hat{x}_{a1} \\ \hat{x}_{a2} \\ \vdots \\ \hat{x}_{ap} \end{bmatrix}$, c'est-à-dire notre estimation de la position du nœud a.

Le processus peut être recommencé avec tous les nœuds inconnus du réseau. Nous obtenons ainsi les positions de tous les nœuds dans le réseau.

IV.2.2. Illustration

Nous avons dans l'exemple :

$d_{14} = 1.41, d_{24} = 2, d_{34} = 1.41$ *et* $x_1 = (1,2), x_2 = (0,1), x_3 = (1,0)$.

Ce que nous cherchons est : $x_4 = (x_{41}, x_{42})$. Nous remplaçons les valeurs que nous avons :

$$(1 - x_{41})^2 + (2 - x_{42})^2 = \sqrt{2}^2 = 2$$
$$(0 - x_{41})^2 + (1 - x_{42})^2 = 2^2 = 4$$
$$(1 - x_{41})^2 + (0 - x_{42})^2 = \sqrt{2}^2 = 2$$

Ceci donne pour A et b :

$$\begin{pmatrix} 2(1-1) & 2(2-0) \\ 2(0-1) & 2(1-0) \end{pmatrix} x = \begin{pmatrix} 4 \\ -2 \end{pmatrix}$$

Le système est $\begin{bmatrix} 0 & 4 \\ -2 & 2 \end{bmatrix} \begin{bmatrix} x_{41} \\ x_{42} \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ qui donne La solution trouvée est $\begin{pmatrix} x_{41} \\ x_{4,2} \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$.

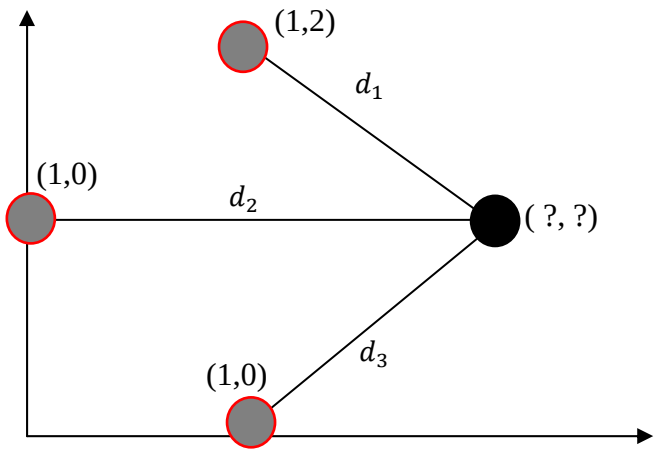


Figure I.6 *: illustration d'une étape de multilatération*

IV.3. L'étalonnage multidimensionnel

Le multidimensional scaling (l'étalonnage multidimensionnel) ou MDS est une méthode utilisée pour l'analyse de données de similarité (La similarité est une mesure qualitative de la distance ou proximité entre deux objets) ou de dissimilarité sur un ensemble d'objets.

Le MDS a été inventé dans le milieu de la psychologie dans les années cinquante. Il est né d'un besoin de visualisation de données par les psychologues.

Le MDS est une méthode pour déduire les positions des objets depuis les distances interobjets. Supposons que l'on considère un ensemble de N objets, et que pour chaque paire d'objets $(r, s), 1 \leq r, s \leq N$, on ait une mesure de "similarité" d_{rs} entre les deux objets.

Le MDS permet d'associer à chaque objet des positions dans un espace de dimension donnée, de telle sorte que les similarités $\hat{d}_{rs}$ estimées entre les objets respectent au mieux les similarités originales d_{rs}. Les techniques utilisées pour la recherche de cet espace et des configurations de points associées forment le MDS. Le MDS classique a été proposé par Torgerson [8].

Soit N points dont la position doit être trouvée. Les seules informations dont nous disposons sont, pour chaque couple de nœuds (i, j), la distance dij (ou une estimation $\hat{d}_{ij}$).

Ces distances sont représentées par une matrice carrée symétrique et à diagonale nulle $D_{N,N} = [d_{ij}]$. Nous cherchons les positions des points dans un espace euclidien à dimension p : R^p. Soit le vecteur $xi = (x_{i1}, \ldots, x_{ip})^T$ représentant la position du point i. Nous pouvons représenter toutes les positions par une matrice $X_{N,p} = (x_1, \ldots, x_n)^T$.

- **Matrice des produits scalaires**

Considérons la matrice des produits scalaires entre les vecteurs d'observations : $T = XX^T$. Nous allons montrer que les distances peuvent s'exprimer en fonction des produits scalaires et inversement.

Soit $T = [t_{ij}] = x_i^T * x_j$.

Si $\hat{T}$ est notre estimation de la matrice des produits scalaires, calculée à partir de $\hat{D}$, alors $T \neq \hat{T}$ et on peut montrer que le MDS classique minimise la fonction d'optimisation suivante, appelée Strain [8] :

$$L(\hat{X}) = ||T - \hat{T}||^2 = ||XX^T - \hat{X}\hat{X}^T||^2$$

- **Relations de Torgerson**

Si les distances sont euclidiennes nous trouvons :

$$\begin{aligned} d_{ij}^2 &= ||x_i - x_j||^2 \\ &= ||x_i||^2 + ||x_j||^2 - 2x_i^T x_j \\ &= t_{ii} + t_{jj} - 2t_{ij} \end{aligned}$$

Et nous inversons cette relation :

$$t_{ij} = \frac{1}{2}(t_{ii} + t_{jj} - d_{ij}^2)$$

Il nous reste donc à déterminer t_{ii} et t_{jj} .

Remarquons que les distances sont invariantes : par translation, par rotation, par réflexion.

Nous imposons que le centre de gravité ou centroïde des points correspondants aux individus se situe à l'origine : $\sum_i^n x_i = 0$ (0 représente le vecteur rempli de 0).

Comme nous imposons que les individus soient centrés :

$$\sum_i^n t_{ij} = \sum_i^n x_i^T x_j = \sum_i^n x_j^T x_i = x_j^T \left(\sum_i^n x_i \right) = 0$$

Nous avons également :

$$\sum_{ij}^{n} t_{ij} = \sum_{ij}^{n} x_i^T x_j = \left(\sum_{i}^{n} x_i\right)^T \left(\sum_{j}^{n} x_j\right) = 0$$

Soit : $tr(T)$ la trace de la matrice T, et ${d_{.j}}^2$, $d_{..}$ sont définit comme suit :

$$tr(T) = \sum_{i}^{n} t_{ii} \, , {d_{.j}}^2 = \sum_{i}^{n} d_{ij}^2 \, , d_{..} = \sum_{ij}^{n} d_{ij}^2$$

Calculons la première somme par rapport à i, j :

$$\sum_{ij}^{n} t_{ij} = \sum_{ij}^{n} \frac{1}{2}\left(t_{ii} + t_{jj} - d_{ij}\right)$$

$$= \frac{1}{2}(Ntr(T) + Ntr(T) - d_{..})$$

Donc comme nous savons que cette somme est nulle : $tr(T) = \frac{{d_{..}}^2}{2N}$

Calculons la deuxième somme par rapport à i :

$$\sum_{i}^{n} t_{ij} = \sum_{i}^{n} \frac{1}{2}(t_{ii} + t_{jj} - {d_{ij}}^2)$$

$$= \frac{1}{2}\left(tr(T) + Nt_{jj} - d_{.j}\right)$$

Et donc :

$$t_{jj} = \frac{1}{N}\left(d_{.j} - tr(T)\right)$$

$$= \frac{1}{N}\left({d_{.j}}^2 - \frac{{d_{..}}^2}{2N}\right)$$

De même :

$$t_{ii} = \frac{1}{N}\left({d_{.i}}^2 - \frac{{d_{..}}^2}{2N}\right)$$

Si on remplace t_{ii} et t_{jj} dans l'équation précédente, nous trouvons :

$$t_{ij} = -\frac{1}{2}\left({d_{ij}}^2 - \frac{{d_{.j}}^2 + {d_{i.}}^2}{N} + \frac{{d_{..}}^2}{N^2}\right)$$

Les t_{ij} peuvent donc être calculés à partir des d_{ij} , ce qui permet de calculer T à partir de D. Il nous faut maintenant de trouver un moyen pour passer de T à X. Rappelons que $T = XX^T$.

IV.4. Localisation par l'information d'angle

Dans cet algorithme de localisation, les nœuds sont classifiés en tant que normaux et nœuds d'ancres, ces nœuds sont déployés aléatoirement dans la région de surveillance, les nœuds normaux n'ont aucune information de localisation, les nœuds d'ancres sont dotés d'une puissance de calcul et de ressource énergétique et de l'information de localisation.

Dans cet algorithme on suppose que les nœuds d'ancres disposent d'une métrique (marge de communication) et ils peuvent détecter l'information d'angle de leurs voisins.

Le réseau de capteurs est divisé en un ensemble de clusters dans chaque cluster on doit avoir un seul nœud d'ancre.

Lors de mesure de la distance entre le nœud d'ancre et les nœuds normaux basés sur RSSI, nous proposons un algorithme pour estimer l'information de localisation des nœuds normaux [11].

Lors de réception d'un paquet par le nœud d'ancre, celui-ci doit identifier la distance et la direction d'arrivée de paquet, puis il calcul les positions de chaque nœud correspondant au paquet envoyé.

Le formalisme de calcul des positions des nœuds est décrit ci-après : Considérons le système présenté dans la figure suivante

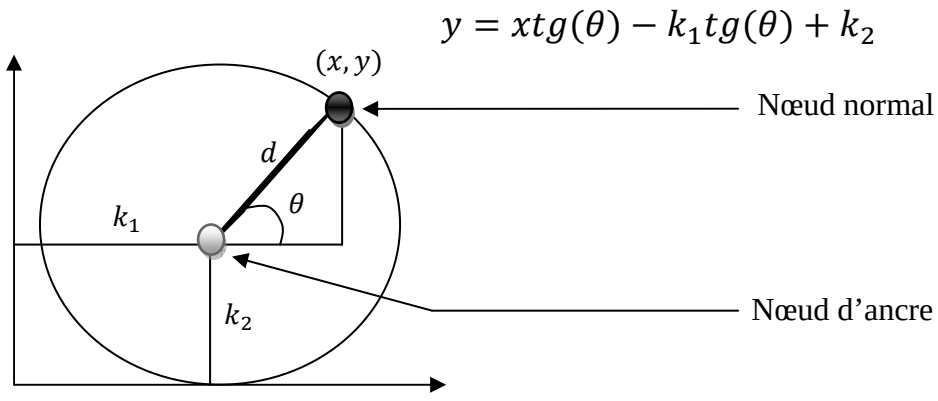


Figure I.7 *: Le système de localisation par l'information d'angle*

Nous considérons l'équation linéaire qui traverse les deux nœuds présenté dans la figure 8 d'une part et l'équation de cercle d'autre part (le nœud d'ancre est le centre de ce cercle $(x-k_1)^2+(y-k_2)^2=d^2$). Substituant l'équation de la droite dans l'équation de cercle et après simplification nous obtenons l'équation suivante :

$$(tg(\theta)^2+1)x^2+2\big(-k_1+Rtg(\theta)\big)x+R^2+{k_1}^2-d^2=0, tq\ R=-k_1tg(\theta)$$

Cette équation possède deux solutions, les points de l'intersection de la droite avec le cercle, on les distingue grâce à la direction d'arrivée (θ).

Une autre méthode plus simple est d'utiliser les coordonnées polaires

$$\begin{cases} x = d * cos(\theta) + k_1 \\ y = d * \sin(\theta) + k_2 \end{cases}$$

V. Conclusion

L'objectif de ce chapitre est d'introduire quelques généralités sur la localisation dans les réseaux de capteurs, On a vu que chaque méthode de localisation présente des avantages et des inconvénients, et surtout, chaque méthode a un champ d'applications bien précis. Ensuite on a présenté les méthodes classiques de calcul des distances.

Le chapitre suivant présente quelques algorithmes de réseau de neurones, les cartes auto-organisatrices ou cartes topologiques qui sont introduites par Kohonen, en effet notre but est d'améliorer la prédiction des positions des senseurs.

Chapitre II :

Localisation par l'application des réseaux de neurones

Chapitre 2 : Localisation par l'application des réseaux de neurones

I. Introduction

L'information de localisation des nœuds est essentielle pour divers applications dans le calcul automatique.

La vision derrière les calculs automatique est de réduire la gestion des systèmes complexes, les données observables caractérisant ce processus sont, dans de nombreux cas, de très grande dimensionnalité. Par voie de conséquence, elles sont très difficiles à interpréter et c'est pour cette raison que les techniques d'analyse de données ont été créées.

Leur but est de retrouver la structure des données dans le sous espace intrinsèque à celui des observations. Les méthodes les plus anciennes, telle que Classical Multidimentional Scaling présentée dans le chapitre précédent, sont concluantes uniquement sur des données corrélées linéairement.

Quelques méthodes plus récentes, telles que les cartes de Kohonen ou le Sammon Non Linear Mapping permettent d'extraire des relations non linéaires entre les variables observées.

Ce chapitre présente en premier lieu un ensemble de définitions de base de réseaux de neurones notamment les cartes auto-organisatrices, par la suite on présentera deux méthodes utilisant le principe de réseau de neurones MDS, ACC.

II. Réseaux de neurones

II.1. Définition

Un neurone est une fonction non linéaire, paramétrée, à valeurs bornées. La sortie du neurone est une fonction non linéaire d'une combinaison des entrées $\{x_i\}$ pondérées par les paramètres $\{w_i\}$, qui sont alors souvent désignés sous le nom de «poids»

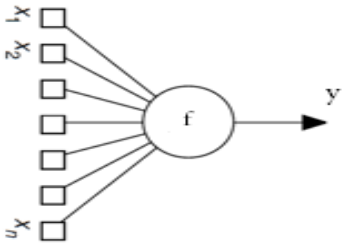


Figure II.1 : *Un neurone réalise une fonction non linéaire bornée* $y = f(x_1, \dots x_n\,; w_1 \dots w_n)$ *où les* $\{x_i\}$ *sont les variables et les* $\{w_i\}$ *sont des paramètres.*

Cette combinaison linéaire est appelée «potentiel». Le potentiel *v* le plus fréquemment utilisé est la somme pondérée, à laquelle s'ajoute un terme constant ou «biais» :

$$v = w_0 + \sum_{i=1}^{n-1} x_i * w_i$$

La fonction *f*(*v*) est appelée fonction d'activation. On utilise généralement des fonctions à valeurs dans l'intervalle réel [0,1]. Quand le réel est proche de 1, on dit que l'unité (le neurone) est active alors que quand le réel est proche de 0, on dit que l'unité est inactive.

L'intérêt des neurones réside dans les propriétés qui résultent de leur association en réseaux, c'est-à-dire de la *composition* des fonctions non linéaires réalisées par chacun des neurones.

On distingue deux types de réseaux de neurones : les réseaux non bouclés et les réseaux bouclés.

Le réseau non bouclé (feed-forward network) réalise une (ou plusieurs) fonctions de ses entrées, par composition des fonctions réalisées par chacun de ses neurones. Si l'on se déplace dans le réseau à partir d'un neurone quelconque, en suivant les connexions, on ne peut pas revenir au neurone de départ.

Le réseau neurones bouclé (recurrent network with feedback connections) Lorsque on se déplace dans le réseau en suivant le sens des connexions, il est possible de trouver au moins

un chemin qui revient à son point de départ (cycle). La sortie du réseau peut donc être une fonction d'elle-même. Il faut par contre faire attention aux retards, c'est pourquoi, tous cycles du graphe des connexions d'un réseau de neurones bouclés doit comprendre au moins une connexion de retard non nul.

Tout réseau bouclé, aussi complexe soit-il, peut être mis sous forme canonique comportant un RN non bouclé dont certaines sorties sont ramenées aux entrées par des bouclages de retard. [2], [1], [3].

II.2. Apprentissage des réseaux de neurones

On appelle « apprentissage » des réseaux de neurones la procédure qui consiste à estimer les paramètres des neurones du réseau, afin que celui-ci remplisse au mieux la tâche qui lui est affectée.

On peut distinguer deux types d'apprentissages : « apprentissage supervisé » et «apprentissage non supervisé ».

L'apprentissage supervisé, on connaît les valeurs que doit avoir la sortie du RN en fonction des entrées correspondantes. On va donc fournir au réseau des exemples de ce qu'il doit faire.

L'apprentissage non supervisé, Il n'y a pas d'exemples donnés. C'est au réseau de découvrir les ressemblances entre les éléments, c'est le type de réseau qu'on va exploiter dans notre étude (les cartes auto-organisatrices ou carte Kohonen) [2], [3].

II.3. Cartes auto-organisatrices

Cette section est consacrée à la seconde grande famille de réseaux de neurones : les cartes auto-organisatrices. Ces dernières font partie dans la famille de modèles à apprentissage non supervisé. Les données à analyser sont maintenant constituées d'observation dont on cherche à comprendre la structure. [2],[1]

II.3.1. Notation et définitions

Ce paragraphe introduit les notations utilisées dans l'ensemble de cette section. L'ensemble D représente l'espace des observations ; les observations sont supposées réelles et de dimension multiple ; on suppose que l'espace des observations est de dimension p et que $D \subset R^p$ .chaque vecteur de D correspond à un codage particulier des individus issus d'une population donnée. On suppose, par la suite, que l'on dispose d'observations correspondant à N individus, représentées par le sous-ensemble $A=\{z_i ; i=1,\ldots..N\}$ de D. on fait, bien entendu, l'hypothèse que A est représentatif de la population en cours d'étude, et qu'il constituera l'ensemble d'apprentissage permettant d'estimer les paramètres des différents modèles.

L'ensemble de toutes les méthodes présentées cherche, dans un premier temps, à réduire l'information contenue dans D ; elles le font :

- En la résumant sous forme d'un ensemble $W = \{w_c : c = 1 \ldots k\}$ $de\ k$ vecteurs de $\mathfrak{D}$ sont appelés les référents.
- En définissant une fonction $\mathcal{X}$ application de $\mathfrak{D}$ dans l'ensemble des indices $\{1 \ldots k\}$ qui permet de réaliser une partition $P = \{P_1 \ldots P_c \ldots P_k\}$ de $\mathfrak{D}$ en P sous ensembles $P_c = \{z \in \mathfrak{D} / \mathcal{X}(z) = c\}$. Le principe est résumé dans la figure 2.

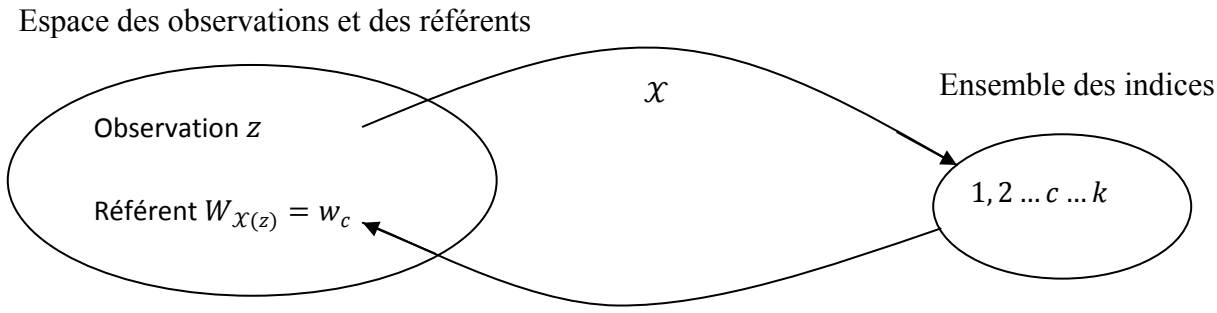


Figure II.2 *: principe générale de la modélisation : une observation z est associée à un indice choisi parmi p à l'aide de la fonction X ; cet indice permet de définir le référent* w_c

La connaissance de l'ensemble des vecteurs référents W et de la fonction d'affectation $\mathcal{X}$ détermine ce que l'on appelle la **quantification vectorielle**.

La détermination de X et de W se fait par minimisation de la fonction de coût, la quantification vectorielle est utilisée pour affecter une observation $z \in \mathfrak{D}$ à son référent $w_{x(z)}$; $x(z)$ représente l'indice de référent auquel est associée l'observation z. la quantification vectorielle permet aussi de définir la partition P de $\mathfrak{D}$ en k sous-ensembles. Dans la section suivante en présentera quelques méthodes de classification automatique. [4][2].

II.3.2. Algorithme K-moyennes

La méthode des K-moyenne qui est une méthode de quantification vectorielle, détermine l'ensemble des vecteurs référents W ; et la fonction de d'affectation X, en minimisant la fonction de coût $I(W,X)$ que l'en cherche à minimiser qui représente la somme des inerties locales I_c , pour procéder à la minimisation de $\mathrm{I}(W,X)$, il faut apparaitre la fonction d'affectation X ; la quantité que l'en cherche à minimiser s'écrit alors :

$$I(W,X) = \sum_{\mathrm{c}} I_c \text{ , tel que } I_c = \sum_{z_i \in P_c \cap A} ||z_i - w_c||^2 \qquad (1)$$

Cette méthode permet de d'obtenir un minimum local d'un critère à optimiser, cette méthode repose sur l'utilisation de deux entités : l'ensemble des partitions, en k sous ensembles, de l'espace de données, et l'espace W des représentations.

L'algorithme présenté procède d'une manière itérative, chaque itération comportant deux phases. La première phase minimise $I(W,X)$: en supposant les valeurs des référents fixées aux valeurs calculées précédemment, elle calcule une valeur de la fonction X, la second phase suppose que la fonction d'affectation est fixée à la valeur qui vient d'étire calculée ; elle minimise la fonction de coût par apport aux paramètres W . On procède ainsi en deux phases, on fait décroitre la valeur de la fonction de coût à chaque itération.

- *phase d'affectation* : il s'agit, dans cette phase, de minimiser la fonction $I(W,X)$ par rapport à la fonction d'affectation X ; à cette étape, l'ensemble W des référents est fixé : il est égal à la valeur calculée précédemment. La minimisation s'obtient en effectuant chaque observation xi au référant Wc à l'aide de la fonction d'affectation

$$X(z) = \arg\min_{\mathrm{r}} | \, |z - w_r| \, |^2 \qquad (2)$$

Dans cette expression, r varie de 1 à k (nombre de référents). En effectuant chaque observation z, au référent le plus proche w_c ,on réduit le terme correspondant à z_i dans la fonction de coût $I(W, X)$. la nouvelle fonction d'affectation définit une nouvelle partition P de l'ensemble D, chaque observation *z* étant effectuée au référent le plus proche au sens de la distance euclidienne. Par la suite, n_c représente le nombre d'éléments de $A \cap P_c$

- *phase de minimisation* : la seconde phase de l'itération fait décroitre à nouveau $I(W, X)$ en fonction de l'ensemble des référents W ; la fonction d'affectation X utilisée à la phase précédente est fixée. La fonction $I(W, X)$ est alors une fonction quadratique convexe par rapport à W, dont le minimum global est atteint pour :

$$w_c = \frac{\sum_{z_i \in A \cap P_c} z_i}{n_c} \qquad (3)$$

L'algorithme des K-moyenne se résume de la manière suivante :

1. phase d'initialisation : t=0, choisir les k référents initiaux (en général d'une manière aléatoire), fixer le nombre maximal d'itérations N_{iter}.

2. étape itérative : à l'itération t, l'ensemble des référents W^{t-1} de l'étape précédente sont connus :

Phase d'affectation : mise à jour de la fonction d'affectation X^t associé à W^{t-1} ; on affecte chaque observation z au référent définit à partir de l'expression (2).

Phase de minimisation : calcul des nouveaux référents W^t en appliquant l'équation (3).

3. répéter l'étape itérative jusqu'à ce que l'on atteigne l<N_{iter} itération ou une stabilisation de *I*.

Figure II.3 *: Algorithme des K-Moyennes*

Un sous ensemble P_l sera représenté par l'élément w_l qui sera son représentant dans W. l'adéquation d'une données x à un représentant donné w_l sera quantifiée par une fonction positive d, ainsi plus $d\ (x, w_l)$ est petite, plus x est en adéquation avec le présentant w_l.

Il s'agit donc de définir une partition en P sous-ensembles $P = \{P_l\ , l = 1 \dots k\}$ de l'espace de données et un ensemble de k représentant w_l $w = \{w_l\ = 1 \dots k\}$ qui minimisent un critère donné. Ce dernier sera défini par l'intermédiaire d'un ensemble d'apprentissage A de la manière suivante :

$$H(P, W) = \sum_{l=1}^{k} \sum_{x_{i \in P_l \cap A}} d(x, w_k)$$

II.3.3. Algorithme stochastique des k-Moyennes

Pour éviter de mener la fonction de coût vers un minimum local, dans le cas de l'apprentissage non supervisé, on choisit le plus souvent, parmi tous les solutions trouvées, celle qui réduise en minimum I(W,X), en peut utiliser une simple méthode de gradient pour chaque itération, le calcule vecteurs référents effectué à chaque itération :

$$w_c{}^t = w_c^{\,t-1} - 2\,\mu^t \sum_{\substack{z_i \in A \\ X^t(z_i)=c}} (w_c^{t-1} - z_i) \qquad (4)$$

On reconnaît ici la minimisation par la méthode de gradient simple, la fonction d'affectation X^t ($X(z) = \arg\min_r | \, |z - w_r^t| \, |^2$) qui apparait dans la fonction de gradient est celle qui définie dans la phase d'affectation de l'itération t, la quantité μ^t représente le pas de la correction pour l'itération t, le référent W_c^{t-1} est celui qui a été calculé à l'itération précédente.

Cette méthode de minimisation n'est pas adaptative, car il fait intervenir la fonction I(W,X) et donc la globalité de la base d'apprentissage A. La minimisation I(W,X) s'effectue maintenant d'une manière stochastique : on envisage séparément les différents termes de la somme qui a apparaissent dans l'expression I(W,X).

A chaque itération, une seul observation z_i de la base d'exemples présentée ; il entraine la correction de vecteur référent $w_{X(z_i)}$ le plus proche. Cela revient à faire décroitre le seul terme $|| z_i - w_{X^t(z_i)} ||$ de la fonction I(W,X), par une méthode de gradient ; la sommation disparaît de l'expression de la drivé partielle de gradient :

$$w^t{}_{x(z_i)} = w^{\,t-1}_{x(z_i)} - 2\,\mu^t\,(w^{t-1}_{x(z_i)} - z_i) \qquad (5)$$

L'algorithme stochastique des k-moyennes se résume de la façon suivante :

1. **Phase d'initialisation : t=0** choisir les k référents initiaux (en général, d'une manière aléatoire) fixer le N_{iter} (nombre d'itérations), la valeur initiale et la loi de décroissance de pas de correction μ^t (ex : $1/\sqrt{t}$)
2. **Etat itérative :** l'ensemble des référents de l'état précédente étant connus, choisir une observation z_i (séquentiellement), calculer le pas de gradient μ^t.
   - En affecte z_i au référent le plus proche parmi ceux de W^{t-1}, en calculant la fonction suivante $x(z) = \arg\min_r | |z - w_r| |^2$ (cette fonction minimise la fonction $I(X,Y)$ par apport à la fonction d'affectation X)
   - Calcule du nouveau référent de $W^t{}_{X(zi)}$.
3. **Répéter :** l'état itérative jusqu'à atteindre $l > N_{iter}$, ou I est stable.

Figure II.4 *:* Algorithme stochastique des k-moyenne

II.3.4. Carte topologique auto-organisatrice

La carte topologique auto-organisatrice est un algorithme d'auto-organisation qui projet l'espace de données en une grande dimension sur un espace discret de faible dimension (cet espace qu'on appelle carte sera noté $\mathcal{C}$ qui est un ensemble des neurones interconnectés). δ est une distance discrète sur $\mathcal{C}$ ($\delta(c,r)$ est la longueur du plus court chemin entre le neurone c et

r sur le graphe .), cette distance permet de définir la notion de voisinage d'ordre d de c : $V_c(d) = \{r \in \mathcal{C},\ \delta(c,r) \leq d\}$.(figure 4)

Figure II .5 : *représentation de la topologie discrète d'une carte topologique à deux démentions constituée 5*5 neurones c. La distance δ est définie sur le maillage* [9].

$V_{N_{13}(1)}(8,14,18,12), V_{N_{13}(2)}(9,8,14,18,12,19,17,7,3,15,23,11)$
$\delta(13,14) = 1, \delta(13,15) = 2, \delta(13,9) = 2$

On veut associer à chaque neurone de $\mathcal{C}$ un vecteur référent w_c de l'espace de données $\mathcal{D}$. L'apprentissage s'effectue de sorte que les vecteurs référents captent au mieux la densité de probabilité (figure 5) aux observations.

Il impose que deux neurones c et r, voisins par apport à la topologie discrète de la carte soit associés à deux vecteurs référents proches par apport à la distance euclidienne sur $\mathcal{D}$. l'objectif est de minimiser la fonction de coût, cette fonction de coût doit tenir compte, d'une part, de l'inertie interne de la partition dans l'espace $\mathcal{D}$, et chercher, d'autre part, à assurer la conservation de la topologie.

Une manière de réaliser cet objectif consiste à généraliser la fonction d'inertie utilisée dans l'algorithme des k-moyennes en introduisant dans l'expression de cette fonction des termes spécifiques qui sont définit à partir de la carte. Cela est réaliser par l'intermédiaire de la distance définie sur la carte et de la notion de voisinage qui lui est attaché.

La notion de voisinage est une fonction positive est symétrique K($\lim_{|x|\to\infty} k(x) = 0$) qui permettent d'introduire des zones d'influence autour de chaque neurone. les distances $\delta(c,r)$

permettent de varier l'influence relative des différents neurones : cette importance est quantifier par la quantité K($\delta(c,r)$). Pour gérer la taille de voisinage, en utilise une fonction K^T paramétrée T : $K^T(\delta) = K(\frac{\delta}{T})$.

$$K(\delta) = \begin{cases} 1 \; si \; \delta < 1 \\ 0 \quad sinon \end{cases} \quad Ansi \quad K^T = \begin{cases} 1 \; si \; \delta < T \\ 0 \quad sinon \end{cases} ;$$

$$K(\delta) = \mathrm{e}^{(-|\delta|)} \; dou \; K^T(\delta) = \mathrm{e}^{(-\frac{|\delta|}{T})}$$

Plus le nombre T est petit plus le nombre de neurones inclus dans le voisinage V_c^T est réduit ([4], page 362) et le nombre d'observations de A qui interviennes pour calculer w_c diminue. Pour des valeurs suffisamment petites, V_c^T se restreint au seul neurone c, et la fonction de coût représente exactement la fonction de coûts de l'algorithme des K-moyennes.

L'algorithme de cartes auto-organisatrices proposé par Kohonen fait décroître le paramètre T dans l'intervalle $[Tmin, Tmax]$ la convergence vers la solution peut se décomposés en deux étapes. La première étape correspond aux grandes valeurs de T pour assurer la conservation de l'ordre topologique, la seconde étape à lieu pour les petites valeurs de T, l'algorithme commence à se rapprocher à l'algorithme de K-moyennes ([4], pages 364-367), la fonction de décroissance utilisées dans la pratique : $T = Tmax * (\frac{Tmin}{Tmax})^{\frac{t}{Niter-1}}$.

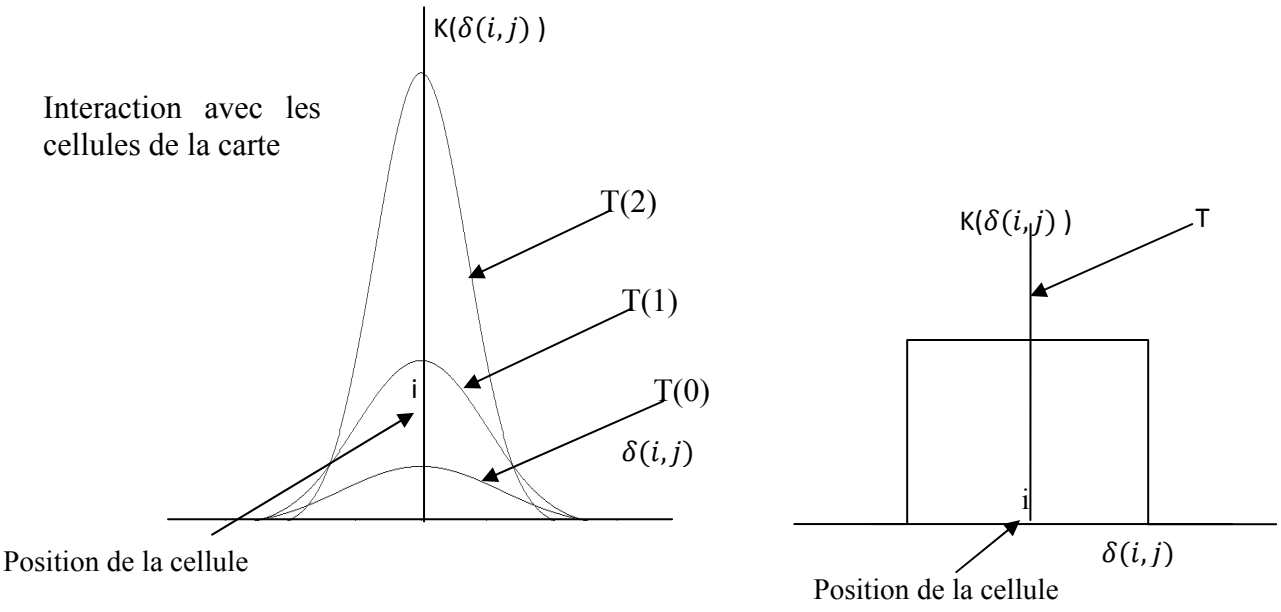


Figure II .6 : *Dans la fonction de voisinage à seuil, les neurones du voisinage ont la même influence. Dans la fonction de voisinage de type gaussien, l'influence entre deux neurones dépend de la distance entre ces neurones.(T :La Taille de la zone d'influence)*

La nouvelle fonction de coût :

$$J^{T}_{som}(X,W) = \sum_{z_i \in A} d^T\left(z_i, w_{X(z_i)}\right); \qquad d^T\left(z_i, w_{X(z_i)}\right) = \sum_{c \in C} K^T(\ \delta(c, X(z_i))||z_i - w_c||^2$$

L'algorithme des cartes auto-organisatrices minimisent la fonction J^{T}_{som} dont le minimum fournit une partition formée de sous-ensembles qui sont suffisamment compacts, mais pour lesquels on est capable de définir un ordre induit.

Dans cette expression X représente une fonction d'affectation, et W l'ensemble des P vecteurs référents qui forme la carte, l'expression $X(z_i)$ est le neurone particulier qui est affecté à l'observation z_i, et $\delta(c, X(z_i))$ représente la distance sur la carte $\mathcal{C}$ entre un neurone c quelconque et le neurone $X(z_i)$ affecté à l'observation z_i. d^T est une distance euclidienne entre z_i et $w_{X(z_i)}$ qui est une somme pondérée de la distance euclidienne de z à tous les vecteurs référents W_c de voisinage de neurone $X(z)$.

La version stochastique de l'algorithme des cartes auto-organisatrices est proposée au-dessous, à l'itération t et pour un neurone c, on a :

$$w_c^t = w_c^{t-1} - \mu^t \frac{\partial J^t_{som}}{\partial w_c^{t-1}}, \qquad \frac{\partial J^t_{som}}{\partial w_c} = 2\sum_{z \in A} K^T(\delta(c, X(z_i))(z_i - w_c)$$

Où μ^t est le pas de gradient de l'itération t. de même que pour l'algorithme des k-moyennes, on peut utiliser la méthode de gradient stochastique, qui recalcule les référents chaque fois l'observation z_i est présentée. La fonction X dans cet algorithme est celle qui est utilisée pour l'algorithme des k-moyennes.

A chaque présentation d'une observation z_i les nouveaux référents sont alors calculés pour tous les neurones de la carte C en fonction e neurone sélectionné :

$$w_c^t = w_c^{t-1} - \mu^t K^T(\delta(c, X(z_i))(z_i - w_c^{t-1}).$$

L'algorithme des carte auto-organisatrices (algorithme Kohonen) ce résume donc de la manière suivante :

1. **Phase d'initialisation : t=0**
   - Choisir la structure et la taille de la carte et les k référents initiaux (en général, d'une manière aléatoire)
   - Fixé les valeurs Tmax, Tmin et le nombre d'itération Niter.
2. **Etat itérative :** l'ensemble des référents de l'état précédente étant connus,
   - choisir une observation z_i (séquentiellement), calculer le pas de gradient μ^t.
   - Calculer la nouvelle valeur de T :

$$T = Tmax * \left(\frac{Tmin}{Tmax}\right)^{\frac{t}{Niter-1}}$$

   Pour cette valeur de paramètre T effectuer les deux phases suivantes :

   Phase d'affectation : En affecte z_i au neurone $X^t(z_i)$ défini à partir de la fonction d'affectation suivante $x(z) = \arg\min_r | |z - w_r| |^2$

   Phase de minimisation : calcul de l'ensemble de nouveaux référents W^t. les vecteurs référents sont modifiés selon la dernière formule en fonction de leur distance au neurone sélectionné à l'étape d'affectation.
3. **Répéter :** l'étape itérative en faisant décroitre la valeur de paramètre T jusqu'à ce que l'on atteigne $t = N_{iter}$

Figure II .7 *: algorithme de Kohonen*

III. Méthodes de réduction de dimension

Avant de mettre en œuvre un réseau de neurones, il peut s'avérer nécessaire de construire de nouvelles variables d'entrée afin de réduire leur nombre, tout en perdant le mois d'information possible sur leur réparation.

La réduction de dimension ne vise pas seulement à diminuer le nombre de variable décrivent chaque exemple, elle permet également de construire des représentations plus synthétiques des données, en facilitant l'analyse.

On a évoqué dans la section suivante quelques méthodes de réduction de dimension utilisant les réseaux de neurones partant de la méthode d'étalonnage multidimensionnel (Multidimensional Scaling) pour finir avec la méthode d'Analyse en Composantes Curvilignes.

III.1. positionnement multidimensionnel (MDS)

Le positionnement multidimensionnel (MultiDimensional Scaling) a été introduit par Shepard dès 1962. Le MDS se propose de représenter des mesures de dissimilitude (ou similarité) entre des paires d'objets sous la forme de distances entre des points dans un espace multidimensionnel de faible dimension.

Le but du MDS est d'obtenir une représentation graphique où chaque point est un des objets des données. Les points sont disposés de manière à ce que la distance les séparant respecte au mieux leur similarité ou leur corrélation.

Sammon a proposé une variante, le «NLM». Cette méthode est basée sur la minimisation d'un critère de distorsion à l'aide d'une descente de gradient. Le critère est une erreur quadratique pondérée entre les distances dans l'espace d'entrée et celui de sortie.
On cherche :

$$y_i = ArgMin\{E\}, \qquad y_i : p \qquad E = \frac{1}{2} \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} \frac{(X_{ij} - Y_{ij})^2}{X_{ij}}}{\sum_{i=1}^{N}\sum_{j=1}^{N} X_{ij}}$$

La fonction d'énergie E peut se réécrire sous la forme :

$$E = \frac{1}{k}\sum_{i=1}^{N}\sum_{j=1}^{N} (X_{ij} - Y_{ij})^2 . F(X_{ij}) \quad avec \quad F(X_{ij}) = \frac{1}{X_{ij}} \quad et \quad k = 2\sum_{i=1}^{N}\sum_{j=1}^{N} X_{ij}$$

X_{ij} est **L**a distance entre les points *i* et *j* de l'espace d'origine est :

$$X_{ij} = d(x_i, x_j) = \sqrt{\sum_{k=1}^{p} (x_{ik} - x_{jk})^2}$$

Y_{ij} est La distance entre les points *i* et *j* de l'espace réduit :

$$Y_{ij} = d(y_i, y_j) = \sqrt{\sum_{k=1}^{s} (y_{ik} - y_{jk})^2}$$

Au cours de l'évolution de l'algorithme, l'énergie E est minimisée, lorsque les Y_{ij} (distances de sorties) correspondent le mieux possible au X_{ij}. L'adaptation se fait par :

$$y_i(t+1) = y_i(t) + \mu(t)\nabla E_{y_i(t)}$$

$$\nabla E_{y_i(t)} = \frac{\partial E(t)}{\partial y_i(t)} = -\frac{4}{k}\sum_{j=1}^{N} (y_i - y_j).(X_{ij} - Y_{ij}).F(X_{ij})$$

La fonction de pondération $F(.)$ qui sert à favoriser le respect de proximité (courtes distances) dépend uniquement des distances d'entrée ; ce qui limite considérablement les facultés de dépliage. En effet, il est très difficile dans ce cas de « couper » des données refermées pour les projeter dans l'espace de sortie.

Sammon note deux limitations à cette méthode : la première est que celle-ci offre une faible fiabilité de projection sur des structures de grandes dimensions. La deuxième est plutôt d'ordre calculatoire ; il est nécessaire de stocker en mémoire toutes les distances X_{ij} et Y_{ij}, de recalculer toutes les distances de sorties Y_{ij}, et d'ajuster tous les vecteurs de sortie dans le but de minimiser E à chaque itération. De plus, les algorithmes de type descente de gradient souffrent d'un effet de moyennage et favorisent le blocage dans un minima local. [05]

III.2 Analyse en composantes curvilignes

La technique non linéaire MDS, déplie des données complexes, mais de façon limitée. Comme il a déjà été souligné, elles sont très coûteuses en calculs et en ressources mémoires.

Pierre Demartines dans sa thèse [10]. propose une nouvelle approche pour traiter des données multidimensionnelles : Analyse en Composantes Curvilignes, initialement baptisée Vector Quantization and Projection (cf II.3.1).

L'ACC réalise une projection non-linéaire des données au moyen d'un réseau de neurones à deux couches. La topologie de l'espace de sortie n'est pas fixée *a priori*. Les poids des neurones de la couche de sortie y_i sont initialisés aléatoirement. Ensuite, un neurone de sortie, dit « neurone gagnant », est choisi aléatoirement et son poids est modifié de façon à minimiser la fonction de coût.

III.3.1. Fonction de coût

Posons N vecteurs d'entrées, $\{x_i; i = 1 \dots N\}$ dont tout x_i sont des vecteurs à p dimension. ACC cherche les N vecteurs de sortie telle que $\{y_i \; ; i = 1 \dots N\}$ dont tout les y_i sont des vecteurs à s dimension ($s < p$), aussi la distance entre deux vecteurs d'entrés x_i et x_j est préservée par la distance de vecteurs de sorties y_i, y_j. ;

La fonction de coût donnée au-dessous indique la distorsion sur la variété qui est calculée en comparants les distances *Xij* aux distance *Yij*

$$E = \frac{1}{2} \sum_{i}^{N} \sum_{j \neq i}^{N} (X_{ij} - Y_{ij})^2 F(Y_{ij}, \lambda_y). \qquad (5)$$

$F(Y_{ij}, \lambda)$ est un terme de pondération, fonction positive monotone décroissante en fonction de Y_{ij} , La fonction F (.) est choisi de façon à favoriser la conservation locale de topologie, semblable à l'idée dans les cartes auto-organisatrices (SOM) [Kohonen 2001]. Les fonctions exponentiels décroissants, sigmoïde, Lorentz, ou même fonctions d'étape simples, sont tous des choix appropriés pour F [10].

Pour des calcules exactes de ACC, il est nécessaire que la fonction de coût soit minimum possible, en utilisant la méthode de l'algorithme de gradient stochastique qui permet la convergence ainsi la rapidité de calcul, mais cette démarche augmente le risque de tomber dans un espace local minima. La nouvelle règle consiste à adapter tous les autres points en fonction d'un point retenu *i*, au lieu d'adapter le point en fonction des autres points :

$$\Delta y_j = \alpha(t) F(Y_{ij}, \lambda_y)(X_{ij} - Y_{ij}) \frac{y_j - y_i}{Y_{ij}} \quad \forall j \neq i$$

Ici α(t) diminue avec le temps pareillement aux méthodes de gradient stochastiques habituelles. Pour un cycle d'adaptation de tous les nœuds (excepté i), la complexité est donc seulement O (N) au lieu d'O (N^2).

Le calcul appliquant l'algorithme ACC converge non seulement beaucoup plus rapidement, mais est également pour s'échapper par la suite des minimums locaux pour atteindre un minimum beaucoup plus profond.

III.3.2. La carte des coordonnées des nœuds

ACC préserve la distance entre l'espace de donnée en entrée (input data) puis il génère l'espace de données en sortie (output data) dont chaque point de données a une dimension réduite. On présente par la suite la formalisation de problème de localisation.

Posons une matrice de distance $D_{(N*N)}$ de N nœuds, cherchons les coordonnées de tous les points qui respectent la contrainte suivante :

$$\min\left(\sum_{i,j}\left(d_{ij} - p_{ij}\right)^2\right) \quad \text{for } i,j = 1,2,\dots N \qquad (6)$$

d_{ij} est la distance mesurée connue entre le nœud *i* et le nœud *j*

p_{ij} est la distance entre le nœud *i* et *j* calculés utilisant les coordonnées de position calculées de *i* et de *j*.

Si le d_{ij} est pris comme matrice de distance de l'ensemble de données d'entrée et p_{ij} la matrice de distance de l'ensemble de données de sortie, ACC alors pousse (6) à un minimum pendant qu'il réduit au minimum la fonction de coût en (5)

Puis que la matrice de distance est la seule donnée connue de ces N nœuds $D_{(N*N)} = d_{ij}$, nous prenons cette matrice en tant que deux ensembles de données d'entrée, $x_{(N*N)} = D_{(N*N)}$, et la matrice d'inter-nœuds, $X_{ij} = D_{(N*N)}$.

L'algorithme d'ACC comprend deux étapes simples suivantes pour projeter des coordonnées d'un nœud, la fonction $F\left(Y_{ij}, \lambda\right) = e^{-\frac{Yij}{\lambda(t)}}$, ainsi $\lambda(t)$ est une fonction décroissante suivant le temps.

1. Initialisation la sortie $y_{(N*2)}$ estimée, en utilisant les valeurs moyennes des deux premières colonnes d'ensemble de données entrées $x_{(N*N)}$.

2. Dans chaque cycle, un nœud i sera choisi et calculer pour chaque nœud j (j $\neq$ i) la nouvelle valeur y(t+1) en utilisant la valeur courante y(t)

$$y_j(t+1) = y_j(t) + \alpha(t) e^{-\frac{Y_{ij}}{\lambda(t)}} \left(\frac{X_{ij}}{Y_{ij}} - 1\right)(y_j - y_i)$$

3. Dans les expériences, la fonction suivante (équation (4)) est employée pour implémenter $\lambda(a), et\ \alpha(t)$ (c-a-d utilisant α =ν and λ =ν) bien que d'autres fonctions semblables puissent être choisis

$$\nu(t) = \nu(0) \times \left(\frac{\nu(c)}{\nu(0)}\right)^{\frac{t}{c-1}} \qquad \ldots \quad 4$$

c est le nombre total de cycles calculés, souvent appelé longueur d'apprentissage « training length » dans ACC , Par exemple nous peuvent choisir les valeurs suivantes : $a(0) = 0{,}5, a(c) = \frac{a(0)}{100}$, $\lambda(0) = \max\{std_1, std_2, std_3, \ldots std_N\} * 3\ et\ \lambda(c) = 0{,}01$

std_i est l'écart type pour la colonne i^{th} de l'ensemble de données d'entrées $D_{(N*N)}$

III.3.3. Algorithme de cartographies distribué utilisant ACC

Dans cette section on présente l'algorithme de mappage ACC-MAP semblable à MDS-MAP, pour chaque nœud il construit les cartes locales dans le réseau puis les fusionnent l'ensemble pour former la carte globale, cependant ACC-MAP calcul les coordonnés d'un nœud dans la carte locale.

Chaque nœud calcul sa carte locale utilise seulement l'information locale, si la possibilité de rangement sont disponibles dans le réseau, la distance locale entre chaque paire de nœuds voisins est mesurée et connue.

L'algorithme est :

- Pour chaque nœud, construire la carte locale dans R sauts. Calculons la matrice de distance la plus courte de la carte locale et la prendre comme matrice de distance approximative LD.
- Chaque nœud applique l'algorithme de CCA, utilisant la matrice de distance locale LD comme ensemble de données d'entrées décrite dans la section précédente. Ceci produit des coordonnées relatives pour chaque nœud dans la carte locale du nœud X de son voisinage R sauts.
- Fusionner les cartes locales.
- Posons suffisamment de nœuds d'ancres (3 ou plus, pour le l'espace 2D et 4 ou plus, pour l'espace 3D), transformant la carte fusionnée au une carte absolue basée sur les positions absolues des nœuds d'ancres

Dans le schéma CCA-MAP, aucune amélioration n'est appliquée parce que les résultats sont souvent satisfaisants sans optimisation. [13][14][15][16].

IV. Conclusion

Comme nous l'avons constaté, les méthodes d'analyse neuronale permettent d'augmenter la précision de calcul des positions des nœuds et d'avoir une marge d'erreur très faible, de plus l'analyse neuronale est une méthodes récent en reconnaissance des formes, pour cela on a choisit à implémenter cette démarche de localisation qui sera l'objet de chapitre suivant.

Chapitre III :

Conception & Validation

Chapitre 3 : Conception & Validation

I. Introduction

Le but de ce chapitre est de montrer l'efficacité de notre démarche de localisation par l'application des réseaux de neurones.

En premier lieu, nous présentons la conception et l'utilisation de notre simulateur de localisation par l'analyse neuronale ensuite nous validons notre démarche avec deux exemples de déploiement de 200 nœuds enfin nous allons comparer des résultats obtenus avec et sans analyse neuronale sous Matlab.

Nous montrons d'abord l'efficacité de l'analyse neuronale par rapport à l'analyse normale au même titre que le comportement de l'analyse neuronale en fonction de la connectivité, nous finirons par la comparaison des algorithmes neuronaux.

II. Architecture générale de la plateforme de simulation

Notre outil de simulation peut être vu comme un ensemble de couches superposées.

La figure 1 présente un schéma plus détaillé de l'architecture de notre plateforme de simulation, et spécifie toutes les étapes que l'utilisateur suivra durant la simulation. Notre application est décomposée en quatre parties principales :

II.1. Création des Nœuds

Cette partie prend en charge le langage manipulé par l'utilisateur tel que la définition des nœuds, configuration de la simulation (changement de repaire, choix de la moyenne et de l'écart type, définition des variables ainsi que le type d'analyse,) et récupération des résultats.

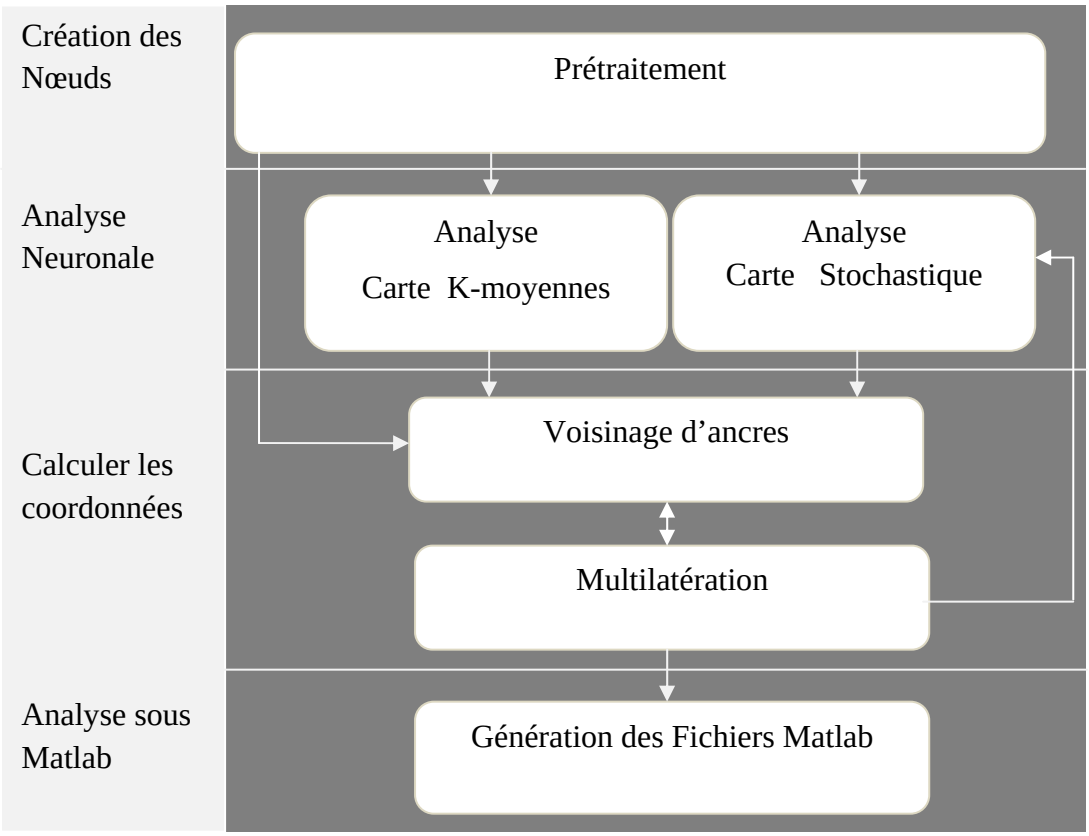


Figure III.1 *: Architecture en couche de l'application*

II.2. Analyse Neuronale

Elle permet à l'utilisateur d'implémenter les algorithmes qui sont définis dans le chapitre précédent, les cartes auto-organisatrices, pour définir la localisation optimale des nœuds d'ancres. La procédure de calcul des coordonnées utilisant l'analyse neuronale est la suivante :

1. Initialiser les nœuds d'ancres
2. Prédire les coordonnées sans analyse neuronale
3. Réaliser l'analyse neuronale pour les coordonnées estimées
4. Répéter 2 et 3 tant que le système prédit des nouveaux nœuds

La fiabilité de l'analyse neuronale repose sur deux contraintes suivantes :

- Améliorer la prédiction des coordonnées
- Localiser un nombre maximum des nœuds dans un réseau de capteurs.

II.3. Calcul des coordonnées

Le système de réseau de capteurs est composé de deux types de nœuds, les nœuds d'ancres (les coordonnées sont connues) et les nœuds normaux (les coordonnées sont des inconnues). Nous avons appliqué la méthode distribuée, où chaque nœud a une certaine puissance de calcul interne. On peut distinguer deux types de calcul au sein de notre application.

- Calcul au voisinage d'ancres : Puisque les nœuds d'ancres sont très puissants en terme de calcul, ces nœuds peuvent estimer l'information de la distance ainsi que sa direction d'arrivée, d'où le système de localisation par l'information de l'angle peut être appliqué et obtenir des résultats plus précis des positions des nœuds qui sont au voisinage de celui-ci

- Calcul par multilatération: Ces nœuds peuvent estimer uniquement l'information de la distance auprès de leurs nœuds voisins, ce calcul ne converge pas toujours vers un résultat précis car, si les coordonnées sont très proches, celui-ci converge vers une solution fausse en terme de la position, de plus chaque nœud nécessite trois autres nœuds possédant leurs positions pour calculer ses coordonnées.

La procédure de calcule des coordonnées est réalisée comme suit :

- ✓ Appliquer la méthode de localisation par l'information d'angle pour chaque nœud d'ancre.
- ✓ **Pour** chaque nœud normal **faire :**
  - o **Si** les coordonnées de ce nœud sont inconnues **alors**
  - o **S'**ils existent au moins trois nœuds possédants leurs coordonnées au voisinage de ce nœud **alors** Appliquer la multilatération.

Le but de notre application est de réduire les calculs en utilisant la multilatération, donc augmenter au maximum les nœuds au voisinage d'une ancre, est cela est effectué par l'analyse neuronale qui va déterminer la localisation automatique des nœuds d'ancres dans un système de réseau de capteurs, quelque soit sa structure.

II.4. Analyse sous Matlab

Le but de cette rubrique est de tester puis comparer les résultats, suivant plusieurs critères d'analyse (sous Matlab 7.5.0), entre l'analyse neuronale et l'analyse normale, aussi la différence entre l'analyse Stochastique et K-moyennes suivant les deux contraintes importantes dans chaque réseau de senseurs : **Erreur médiane de positionnement des nœuds** et **pourcentage des nœuds positionnés,** notre objectif est de positionner tous les nœuds en utilisant un minimum de nœuds d'ancres toute en minimisant l'erreur médiane.

III. Analyse des composants de la plateforme

Dans cette section, nous allons décrire les différentes composantes de notre plateforme.

L'utilisateur a l'accès aux différentes possibilités de configuration qui lui sont offertes : description d'un réseau, Analyse et implémentation des algorithmes neuronaux, calcul des coordonnées, analyse sous Matlab. La figure 2 montre un aperçu sur notre application.

III.1. Définition de l'architecture du réseau

Le réseau est défini sur une zone de capture. Puis il est constitué de nœuds qui sont décris et déployés sur la zone choisie.

Figure III.2 : Interface de l'application

III.1.1. Zone de capture *(Entourée par la couleur jaune, Figure 2)*

La zone de capture représente la surface sur laquelle les nœuds sont déployés. Il est utile de donner à l'utilisateur la possibilité de changement de repaire, d'écart type et de personnaliser les nœuds. Cela lui permet de bien positionner les nœuds. Ces options sont définies dans la zone entourée par la couleur bleu.

III.1.2. Définition des nœuds *(Les points sur l'interface de la zone de capture)*

La deuxième étape consiste à définir les nœuds à déployer dans le réseau. Cela revient à saisir les caractéristiques de nœud. Pour avoir une meilleure visibilité, nous attribuons à chaque type de nœud un label, une échelle, des coordonnées, une erreur d'estimation sur l'échelle.

Après leurs définitions, les nœuds sont déployés dans la zone de capture de manière aléatoire ou à des positions choisies par l'utilisateur (par un clic de la souris), les points en couleur rouge sont les nœuds d'ancres, les points noires sont les nœuds normaux, ainsi à l'aide d'un passage sur un nœud les caractéristiques de celui-ci seront affichées. Le cercle décrit l'échelle de captage de nœud, l'info-bulle (format d'un rectangle jaune) contient toutes les informations de ce nœud, exemple : le nœud « AutoNoeud79 » sur la figure 2. La zone entourée par la couleur bleu montre les informations de prétraitement de données

III.2. Définition de la zone d'analyse (Entourée par la couleur rouge, Figure 2)

Cette zone est conçue pour implémenter les algorithmes neuronaux ainsi que le calcul des coordonnées de chaque nœud en utilisant la démarche précédente (II.2, II.3), après avoir déployé les nœuds. L'utilisateur a la possibilité de choisir le type d'algorithme neuronal à implémenter ainsi que la prédiction des coordonnées. Après la prédiction des nœuds qui sont définis en effectuant la localisation par l'information d'angle (couleur bleu). Les autres nœuds sont prédis par la multilatération représentés en couleur verte.

III.3. Définition de la zone d'analyse sous Matlab

III.3.1. Description (Entourée par la couleur verte, Figure 2)

Cette zone est réalisée en utilisant plusieurs tests d'analyse de calcul des coordonnées. La zone d'analyse sous Matlab est constituée de :

- Graphe 1 (resp. Graphe 2) : Le test de calcul des coordonnées actuel est enregistré dans un point de Graphe 1 (resp. Graphe 2) dans la figure Matlab, la zone de texte qui se trouve à gauche est la légende correspondante à ce graphe.

- La zone liste : Contient le type de graphe tel que :
  - ✓ Erreur médiane moyenne : Les positions des nœuds calculées seront comparées aux positions réelles, puis on calcule l'erreur médiane de l'ensemble.
  - ✓ Pourcentage des nœuds positionnés : le nombre des nœuds qui sont localisés par le système sur le nombre total des nœuds ; le rendement de réseau de capteurs.
  - ✓ Erreur moyenne / Pourcentage : ce type de graphe examine l'évolution de l'erreur moyenne avec le pourcentage des nœuds localisés suivant le nombre d'ancres utilisés
- Cycle, Nombre d'ancres, échelle : indiquent la variable sur l'axe des abscisses suivant le type de graphe à utiliser suivant l'objectif voulu.
- La zone de texte qui suit, indique le nom de fichier Matlab à générer.

Le bouton qui suit, génère un fichier Matlab.

III.3.2. Procédure d'utilisation

Chaque point d'un graphe est enregistré manuellement, après avoir défini le réseau de senseurs, suivant la procédure suivante :

1. Remplir le contenu de l'interface d'analyse sous Matlab décrite ci-dessus.
2. Incrémenter la variable utilisée sur l'axe des abscisses (Nombre d'ancre, Cycles et l'échelle), puis sauvegarder les nœuds d'ancre pour une éventuelle utilisation.
3. Calcul et l'analyse neuronale des coordonnées.
4. Le point calculé par l'étape 3, sera ajouté au graphe correspondant en cliquant sur le bouton 'Graphe 1' ou 'Graphe 2'.
5. Initialiser les nœuds d'ancres pour utiliser le même test pour le graphe 2.
6. Répéter cette procédure pour un autre point.

7. Cliquer sur le bouton ‘SAUVER’ pour générer le fichier Matlab.

IV. Application et Validation

Dans cette partie on démontrera l’efficacité de l’analyse neuronale, par rapport à l’analyse normale, la démarche à suivre consiste à déployer deux types de réseau qui contiennent 200 nœuds, l’un contient 4 distributions symétriques par rapport au centre, l’autre contient une distribution centrée (Figure 3).

IV.1. Initialisation des nœuds d’ancres

L’initialisation des nœuds d’ancres se fait dans chaque réseau, suivant le type de graphe. On défini deux types d’initialisation, une initialisation dite libre (déployée aléatoirement dans le réseau) et une initialisation absolue (déployée dans des endroits caractéristiques de système étudié).

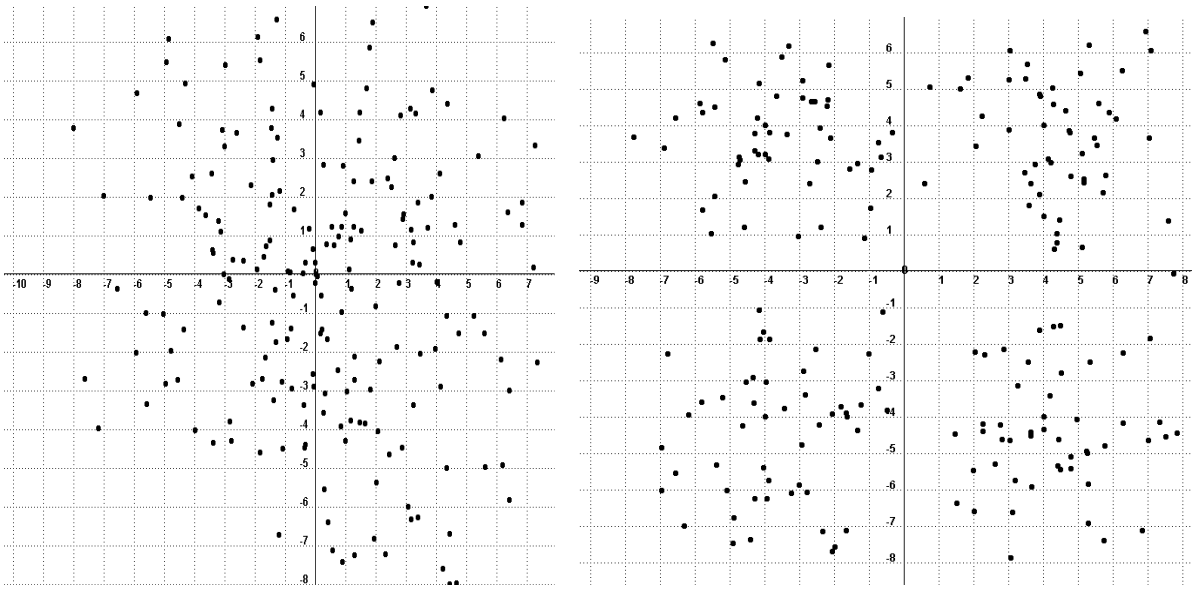


Figure III.3 : *Distribution aléatoire de 200 nœuds, déployés en utilisant deux types de réseaux*

IV.2. Autres caractéristiques des nœuds normaux

Les autres caractéristiques des nœuds normaux sont :

- ✓ L’échelle est choisie aléatoirement dans l’intervalle [0.5Xr -1.8Xr]
- ✓ L’erreur sur l’échelle est constante et égale à 2% du rayon (r)
- ✓ Le nombre de cycles d’analyse neuronale est de 30 cycles

- ✓ L'écart type constant et égale à 2

V. Résultats

V.1. Réseau avec 4 distributions aléatoires et symétrique

Les figures III 4 (a.1), III 4 (a.2) (initialisation libre), Le réseau de capteurs avec un seul nœud d'ancre donne des résultats presque similaires pour les deux types d'analyses car il est capté par tous les distributions. Dans ce cas le nœud reste au centre de réseau (l'effet de la moyenne, Figure III. 6).

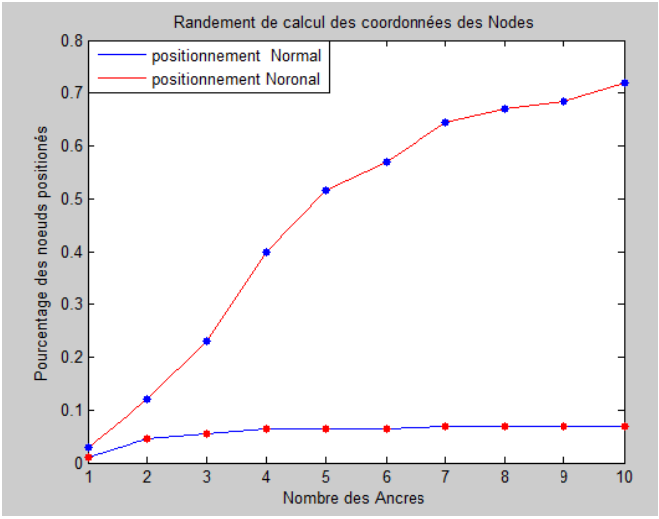
Le nœud d'ancre 2, dans le cas d'analyse neuronale, augmente le nombre des nœuds localisés, mais aussi l'erreur médiane puisque le système utilise la multilatération pour calculer les nœuds qui ne sont pas au voisinage des nœuds d'ancres. En revanche, dans le cas d'analyse normale, le système est constant avec une faible augmentation, à cause de leur initialisation libre.

Les noeuds d'ancres 3, 4, 5 : diminuent le calcul par la multilatération dans le cas d'analyse neuronale, mais aussi le nombre de nœuds localisés reste sensiblement croissant en fonction des noeuds d'ancres. Contrairement à l'analyse normale, l'erreur et le nombre de nœuds localisés restent presque constants (moins 10% pourcent des nœuds localisés en utilisant 10 ancres !).

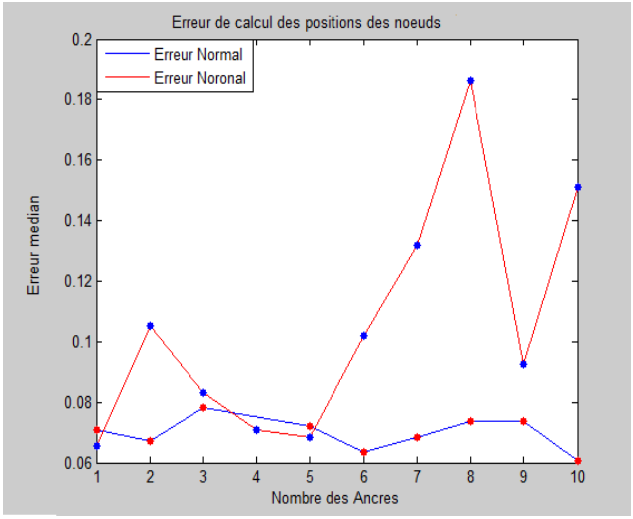
Les nœuds d'ancres (6, 7, 8, 9 et 10), l'erreur médiane et les nœuds localisés augmentent plus vite dans le cas d'analyse neuronale, parce que le système est actif. En effet le système est précis car l'erreur ne dépasse pas 0.2Xr (r est le rayon).

Dans les figures III 4.b.1, III, 4.b.2, l'initialisation est absolue, l'analyse neuronale suivant l'erreur médiane et le nombre des nœuds localisés converge vers la solution d'analyse normale.

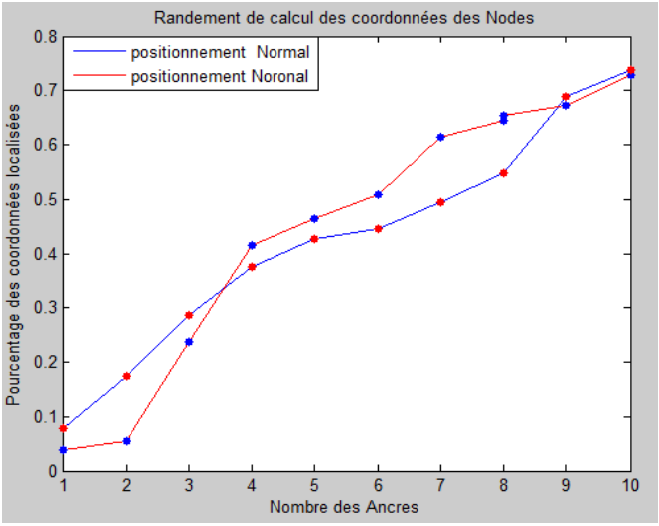
On constate dans ces quatre figures citées ci-dessus, que l'analyse neuronale ne dépend pas de l'initialisation des ancres, en effet le système converge vers un résultat précis dans les deux cas d'initialisations (libre et absolue).



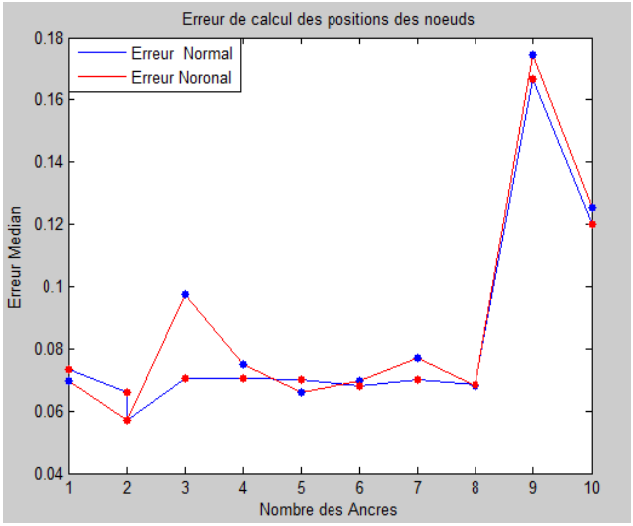
(a.1)



(a.2)



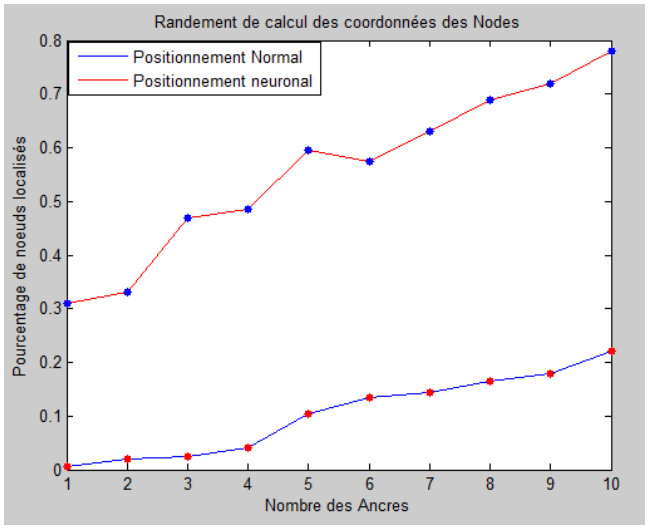
(b.1)



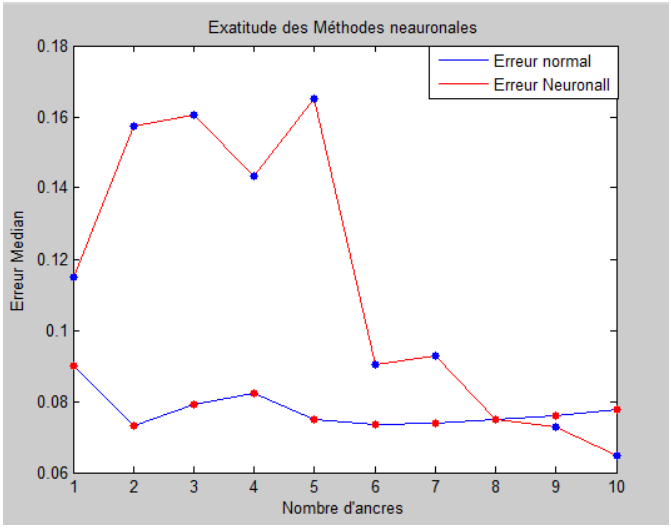
(b.2)

Figure III.4 : Résultat de simulation de réseau symétrique, a.1, b1 : Les nœuds localisés suivant le nombre d'ancres, a.2,b2 : l'erreur médiane suivant le nombre d'ancres, a) initialisation libre, b) initialisation absolue.

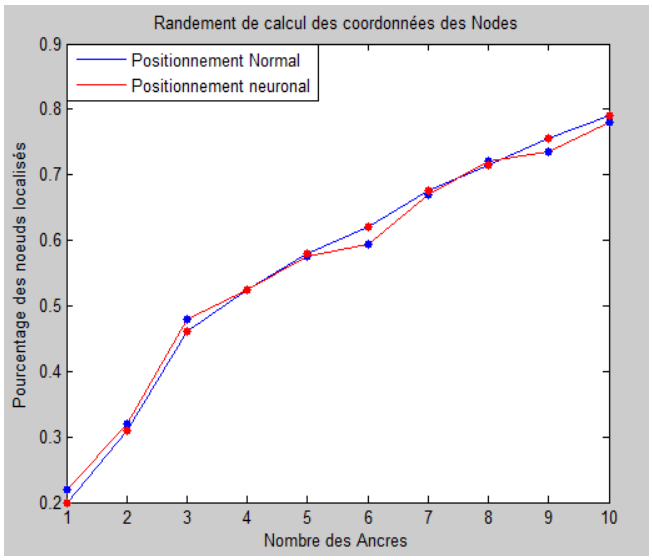
V.2. Une distribution centrée de 200 nœuds



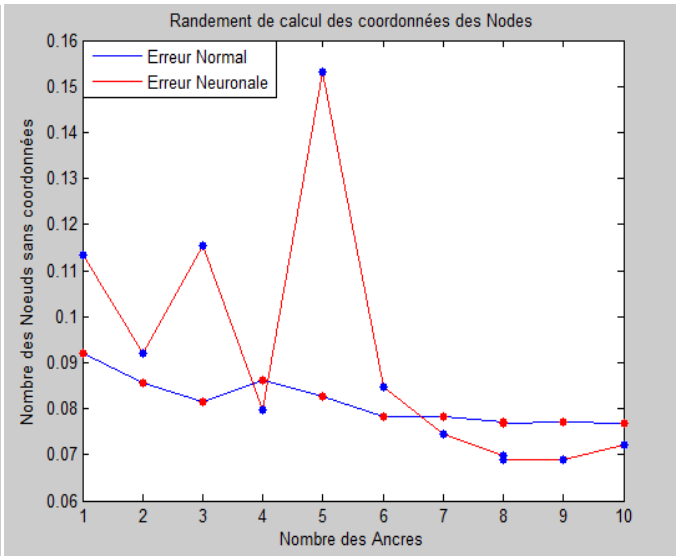
(a.1)



(a.2)



(b.1)



(b.2)

Figure III.5 : Résultat de la simulation de réseau avec une distribution centrée, a.1, b1 : Les nœuds localisés en fonction des nombres d'ancres, a.2,b2 : l'erreur médiane en fonction de nombre d'ancres, a) initialisation libre, b) initialisation absolue.

Les figures III 5(a.1), III. 5 (a.2) (initialisation libre) ; montrent une différence remarquable entre l'analyse neuronale et l'analyse normale. Nous constatons que plus de 30% des nœuds sont localisés avec un seul nœud d'ancre puis le système augmente sensiblement en fonction des nœuds d'ancres.

Avec moins de 7 nœuds d'ancre, l'erreur médiane des nœuds localisés par l'analyse neuronale est supérieur à c'elle de l'analyse normale car le système cherche un maximum des nœuds à localiser, ce qui augmente le calcul par la multilatération.

Plus de 7 nœuds d'ancres, le système possède une précision très fiable (<0.08) avec un maximum de nombre des nœuds (plus 75%), on comparant avec l'analyse normale moins de 25% de nœuds localisés avec une erreur médiane inferieur à celle de l'analyse neuronale.

Ce résultat , traduit le fait que l'analyse neuronale essaie d'identifier tous le réseau de capteurs, l'utilisation d'au plus six (6) nœuds d'ancres diminue le calcul par la multilatération.

Dans figures III (b.1), III. (b.2) ; plus le nombre des nœuds d'ancres augmente plus le nombre des nœuds localisés par l'analyse neuronale tend vers le nombre des nœuds localisés par l'analyse normale avec initialisation absolue. Cependant, l'erreur n'est pas stable dans le cas d'au moins à 6 nœuds d'ancres par rapport à l'analyse normale avec initialisation absolue, ce qui explique l'effet de la moyenne (un ancre capté par deux clusters de nœuds éloignés est positionné à la moyenne de ces deux ensembles, Figure III. 6), d'où l'augmentation de calcul par la multilatération.

Après le sixième nœud d'ancre, plus de 60% des nœuds positionnés, le système cherche à minimiser les calculs effectués par la multilatération, ce qui rend le système converge vers l'analyse normale avec initialisation absolue.

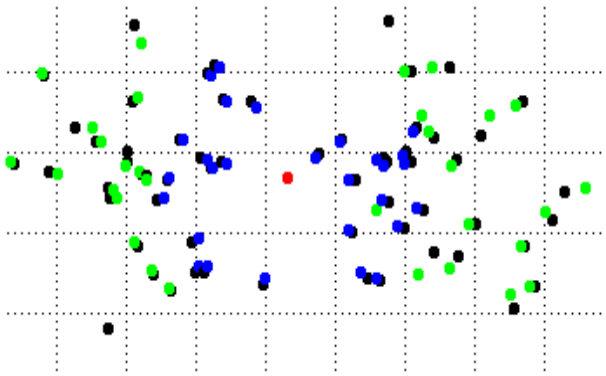


Figure III.6 : l'effet de la moyenne

V.3. Résultat suivant l'échelle

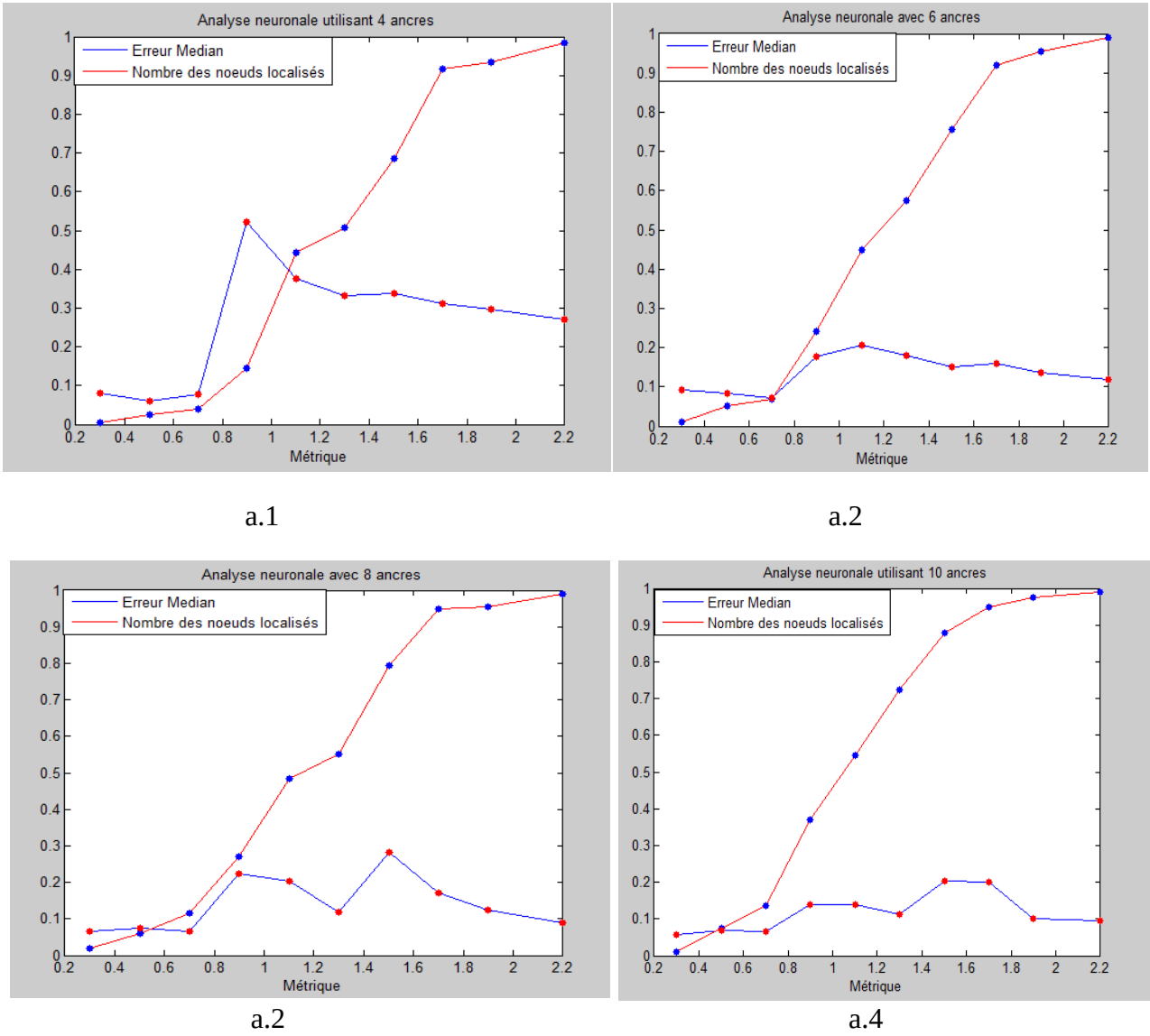
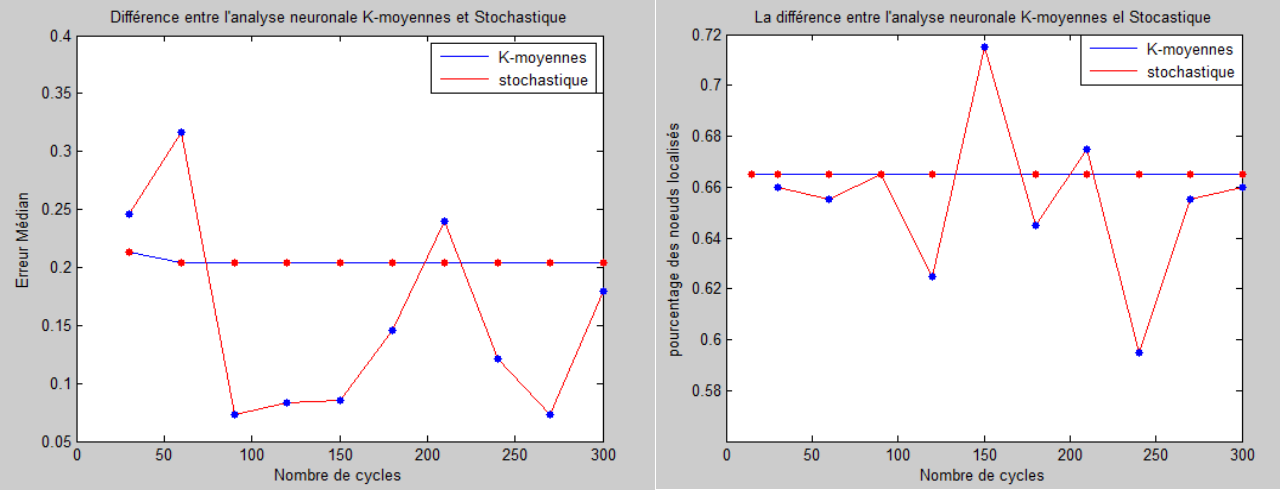


Figure III.7 : les quatre figures décrivent le comportement d'analyse neuronale en fonction de la échelle, le trait rouge indique le pourcentage des nœuds en fonction de la échelle, l'autre décrit l'erreur médiane en fonction de la échelle

Dans chaque graphe dans la figure 7 on constate que le résultat est fiable, et il dépend de nombre d'ancres utilisés ainsi que l'échelle de chaque nœud.

La connectivité inferieure à 0.3Xr est nulle (la seuil de ce réseau), moins de 0.6Xr le système très faible, puisque la distribution ne permet pas la multilatération, uniquement les nœuds qui se trouvent au voisinage des nœuds d'ancres qui sont prédis. Supérieure à 0.6Xr, on remarque l'augmentation des nœuds localisés ainsi que l'erreur dans chaque figure, la différence se trouve dans le nombre d'ancres utilisés. Avec 4 nœuds d'ancres l'erreur atteint une valeur maximale, dans le cas de 10 nœuds d'ancres on constate une petite augmentation de l'erreur, cela traduit que le nombre d'ancres utilisés diminue les calculs par la multilatération, à partir de l'utilisation de 6 nœuds d'ancres et une échelle égale à 1 le système est pseudo-équivalent.

V.4. Comparaison entres les algorithmes neuronaux



(a.1) (a.2)

Figure III.8 *: La différence entre les algorithmes d'analyses neuronales, a.1) Erreur médiane suivant le nombre de cycles a.2) le nombre des nœuds localisés suivant le nombre de cycles,*

Les deux figures ci-dessus décrivent le comportement de deux algorithmes neuronaux en utilisant 6 nœuds d'ancres, en effet, l'algorithme K-moyennes est stable, la valeur d'erreur médiane et le nombre de nœuds localisés sont constants, contrairement à l'algorithme stochastique, on constate que celui-ci suit un chemin instable dans les deux figures, cela

traduit le fait que l'algorithme K-moyennes converge vers un seul minimum local durant tous le nombre d'itérations, par contre, l'algorithme stochastique transite d'un minimum local vers un autre minimum local.

Dans les deux figures au-dessus le minimum global est (150Xcycles, 0.08Xr) dans et la figure III (a.1), et (150Xcycles, 71.9%N) dans la figure III (a.2).

VI. Conclusion

Nous avons conçu un algorithme de localisation utilisant une technique efficace de réduction de complexité et d'augmenter la précision des nœuds localisés, qu'on appelle l'analyse neuronale, pour obtenir les coordonnées des nœuds dans un réseau de senseurs.

Nous avons montré que l'analyse neuronale donne des résultats précis dans un réseau avec initialisation libre, et il converge vers les résultats de l'analyse normale avec initialisation absolue, d'où la fiabilité de la méthode neuronale.

L'application conçue rend l'apprentissage des cartes auto-organisatrices supervisé. En effet, dans la figure de comparaison des méthodes neuronales, l'utilisateur peut facilement avoir une configuration optimale de placement des nœuds d'ancres dans le réseau de senseurs.

Conclusion Générale

Conclusion Générale

L'auto-localisation des nœuds est un problème crucial dans les réseaux de capteurs. Pour le résoudre, nous avons exploité des développements récents des méthodes de reconnaissance des formes.

La méthode de localisation utilisant l'analyse neuronale nous permet de prédire les positions inconnues des capteurs avec des précisions fortement améliorés, et ceci à partir des mesures de portée inter-capteurs et de la position connue d'une fraction de capteurs. Nous avons proposé de résoudre ce problème selon un mode distribué mieux adaptée aux réseaux de capteurs sans fil.

Puisque l'intuition humaine est irremplaçable, à l'aide de notre simulateur, Les résultats obtenus, en contrôlant manuellement les paramètres sous l'influence de la représentation visuelle, sont bien meilleurs que dans le cas d'une évolution classique des paramètres.

Bien que des prédictions des positions inconnues des nœuds par l'analyse neuronale aient été réalisées dans ce travail, il reste encore beaucoup de perspectives de recherche à envisager pour améliorer la qualité de prédiction des résultats. Les difficultés constatées concernent l'effet de la moyenne, et les calculs par la Multilatération, de plus, les cartes Kohonen possèdent également des limitations notamment dans le fait d'avoir une connaissance à priori de la forme de la structure ; pour des raisons numériques ci-dessus. pour contourner le problème rencontré, il serait nécessaire d'introduire à l'algorithme de localisation neuronal d'autres facteurs d'au plus de la densité d'inter-nœuds.

Bibliographie

[1] : Alp Mestan , « *Introduction aux Réseaux de Neurones Artificiels Feed Forward* », alp.developpez.com, 2008.

[2] : Gérard Dreyfus, « *Réseaux de neurones : Méthodologie et applications* », EYROLLES, 2004, ISBN : 2-212-11464-8.

[3] Angélique Byrde, « *Auto-calibration d'un réseau de capteurs de pollution* », haute école spécialisée de suisse occidentale, 2006

[4] : Gérard Dreyfus ,« *Apprentissage Statistique : réseaux de neurones carte topologiques machines à vecteurs supports* » EYROLLES,2008,ISBN : 978-2-212-12229-9

[5] : CEDRIC DUCHENE, « *traitement de données multidimensionnelles par analyse en composantes curvilignes* », D.E.A. Traitement des Images et du Signal de Cergy Pontoise, 2003.

[6] : Nathalie Guyader, « *Catégorisation basée sur des modèles de perception : Approche (neuro) computationnelle et psychophysique* ».

[7] : W.S. Torgerson, «*Multidimensional scaling : I. Theory and method. Psychometrika* », 401-419, 1952.

[8] : J.C. Gower, « *Some Distance Properties of Latent Root and Vector Methods Used in Multivariate Analysis* », Biometrika, 325-338, 1966.

[9] : Manuel de Matlab , "*Self-Organizing Feature Maps*",2007.

[10] [Thèse]: Demartines P., « *Analyse de données par réseau de neurones auto-organisées* », thèse de l'institut national polytechnique de Grenoble.

[11] : Prasan Kumar Sahoo, I-Shyan Hwang, Shi-Yao Lin, « *A Distributed Localization Scheme for Wireless Sensor Networks* »,2008

[12] : Pierre Parrend INSA Lyon, *« Localisation dans les Réseaux de Capteurs* », Janvier 2005.

[13] : LI LI, THOMAS KUNZ, « *Cooperative Node Localization Using Nonlinear Data Projection* », Communications Research Centre Canada, Carleton University, 2009.

[14] : Li Li, « *Localization in self-healing autonomous sensor networks (SASNet) : Studies on cooperative localization of sensor nodes using distributed maps »*, Communications Research Centre,2008.

[15] : Pierre Demartines and Jeanny H´erault," *Curvilinear Component Analysis: A Self-Organizing Neural Network for Nonlinear Mapping of Data Sets*", IEEE TRANSACTIONS ON NEURAL NETWORKS,1997.

[16] :[Thèse] Benjamin Tatham, "*Anchor Node Placement for Localization in Wireless Sensor Networks*", Faculty of Graduate Studies and Research, Ottawa-Carleton Institute for Electrical and Computer Engineering, Department of Systems and Computer Engineering Carleton University Ottawa, Ontario, Canada,2010.

[17] : Van der Haegen Mathieu , « *Réseaux de senseurs sans fil :problèmes de localisation* », Université Libre de Bruxelles Faculté des, 2007

[18] : N. Bulusu, J. Heidemann, and D. Estrin, "*GPS-less low-cost outdoor localization for very small devices*", Personal Communications, IEEE, 7(5) :28-34, 2000.

[19] : N. Bulusu, "*Self-Configuring Location Systems*", PhD thesis, University of California, 2002.

[20] : K. Langendoen and N. Reijers. "*Distributed localization in wireless sensor networks : a quantitative comparison"*. Computer Networks, 43(4) :499-518, 2003.

[21] : N.B. Priyantha, H. Balakrishnan, E. Demaine, and S, "*Teller. Anchorfree distributed localization in sensor networks*". Proc. 1st Inter. Conf. on Embedded Networked Sensor Systems (SenSys 2003), pages 340-341, 2003.

[22] : J. Pugh, "*Local Range and Bearing Sensing Using Infrared Transceivers in Mobile Robotics*", 2004.

[23] : M. Kushwaha, K. Molnar, J. Sallai, P. Volgyesi, M. Maroti, and A. Ledeczi, "*Sensor Node Localization Using Mobile Acoustic Beacons*", PhD thesis, Vanderbilt University, 2005.

[24] : Seattle Robotics Society, "*Implementing infrared object detection"*

[25] : K. Whitehouse, C. Karlof, A. Woo, F. Jiang, and D. Culler, "*The effects of ranging noise on multihop localisation : an empirical study. Proceedings of the fifth international conference onInformation processing in sensor networks*", 2005.

[26] : H. Piontek, M. Seyffer, and J. Kaiser, "*Improving the accuracy of ultrasound-based localisation systems*", Lecture notes in computer science, pages 132-143.

[27] : Aerospace Corporation, "*GPS Primer : A student guide to the Global Positionin System*".

[28] : Peter H. Dana. "*Global Positioning System Overview*". 1999.

[29] : F.G. Snider, "*GPS theory, practice and applications*"

[30] : R. Scholtz, D. Pozar, and W. Namgoong, "*Ultrawideband radio : a tutorial*", EURASIP J. on Applied Sig. Proc, 2005.

[31] : A. Savvides, H. Park, and M.B. Srivastava, "*The bits and flops of the nhop multilatération primitive for node localization problems*", Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications, pages 112-121, 2002.
[32] : T. He, C. Huang, B.M. Blum, J.A. Stankovic, and T. Abdelzaher, "*Rangefree localization schemes for large scale sensor networks*", ACM Press New York, NY, USA, 2003.
[33] : S. Capkun, M. Hamdi, and J.P. Hubaux, "*GPS-free Positioning in Mobile Ad Hoc Networks*", Cluster Computing, 5(2) :157-167, 2002.
[34] : D. Lymberopoulos, Q. Lindsey, and A. Savvides. "*An Empirical Analysis of Radio Signal Strength Variability in IEEE 802.15. 4 Networks using Monopole Antennas*", Proceedings of IEEE Conference on Mobile Ad-Hoc and Sensor Systems, 2005.
[35] : K. Langendoen and N. Reijers, "*Distributed localization in wireless sensor networks : a quantitative comparison*", Computer Networks, 43(4) :499-518, 2003.

Annexe

Listing de Code

```
/*****************************************************************************
 *                                                                           *
 *       Ce projet est proposé par  : Mr Pr. Lalam                           *
 *     Réalisé par : Yazid Takfarinas                                        *
 *     promotion : 2011                                                      *
 *****************************************************************************/

//*****************************************************************************

/*****************************************************************************
 *                                                                           *
 *  Cette classe implémente toutes les caractéristiques d'un nœud se trouvant  *
 * dans un réseau de capteurs                                                *
 *                                                                           *
 *                                                                           *
 *****************************************************************************/

package NeuronalLocalzation;
import java.awt.Dimension;
import java.awt.Point;
import java.util.Arrays ;

public class Node {

     private static int NumNode=1;       // Numéro de nœud
                                         // Statique car cette variable utilisée
                                         // par l'ensemble des nœuds
     private String NomeNode="Noeud";    // Le label de nœud
     private String NodeType="Normal";   // Le type de nœud
     private float PosX ,PosY ;          // Coordonnées de nœud réelles
     private Double metrique=1.5;        // L'échelle de nœud
     private Double Erreur=5.;           // L'erreur effectuée sur l'échelle
                                           //  [0%,+Erreur%]
     public float[] VectDist=new float[50];// Distances des voisins de ce nœud
     private int[] Angles=new int[50];   // angles de voisins dans
                                         // le cas de nœud d'ancre
     private float PosEstX ,PosEstY ;  // coordonnées estimées de ce nœud


            /*************************************************************
             * déclaration de deux constructeurs :                       *
             * un pour les variables par défaut de nœud.                 *
             * l'autre, pour les variables de nœud qui sont personnalisés   *
             * ***********************************************************/

    public Node(){};
      public Node(int NumNode,String Nom,String NodeType,float PosX,float PosY,Double
    metrique,Double Erreur,float[] VectDist){
            Node.NumNode=NumNode;
            NomeNode=Nom;
```

```
        this.NodeType=NodeType;
        this.PosX=PosX;this.PosY=PosY;
        this.metrique=metrique;
        this.Erreur=Erreur;
        this.VectDist=VectDist;
  }


    /*************************************************************************
     * déclaration des méthodes getters et setters, pour l'utilisation externe *
     * de toutes les variables privées                                         *
     *************************************************************************/

public String getNomeNode() {return NomeNode;}
public void setNomeNode(String nomeNode) {NomeNode = nomeNode;}
public String getNodeType() {return NodeType;}
public void setNodeType(String NodeType) {this.NodeType = NodeType;}
public float getPosX() {return PosX;}
public void setPosX(float posX) {PosX = posX;}
public float getPosY() {return PosY;}
public void setPosY(float posY) {PosY = posY;}
public Double getMetrique() {return metrique;}
public void setMetrique(Double metrique) {this.metrique = metrique;}
public Double getErreur() {return Erreur;}
public void setErreur(Double erreur) {Erreur = erreur;}
public float getPosEstY() {return PosEstY;}
public void setPosEstY(float posEstY) {PosEstY = posEstY;}
public void setPosEstX(float posEstX) {PosEstX = posEstX;}
public float getPosEstX() {return PosEstX;}
public int[] getAngles() {return Angles;}
public void setAngles(int[] angles) {Angles = angles;}

   // La fonction toString pour afficher les caractéristique de ce noeud

public String toString() {
      return "Node [ NomeNode=" + NomeNode +  ",Type de Noeud= "+this.NodeType+",
metrique=" + metrique+ ", Erreur=" + Erreur+" PosX=" + PosX + ", PosY=" + PosY+""
+
" , VectDist="+  Arrays.toString(VectDist) + "]";

  }
}
```

Figure Annexe.1 : *cette fenêtre permet d'ajouter un noud dans le réseau de capteur ainsi tous ces caractéristiques*

```
/**************************************************************************
 * Ce programme réalise le calcul des coordonnées d'un nœud               *
 **************************************************************************/


package NeuronalLocalzation;
public class CalculeCoord {
   private float xi,yi,di,xi1,yi1,di1;
   private float xi2,yi2,di2;
   private float tita;
   private float x,y;

/**************************************************************************
     *    l'équation de cercle est : (x₂ -x₁)²+(y₂ -y₁)²=d                  *
     *                                                                      *
     *    Xi, xi1, xi2 : les coordonnées des cercles 1,2, 3 dans l'axe des Xs  *
     *    yi, yi1, yi2 : les coordonnées des cercles 1, 2, 3 dans l'axe des Ys *
     *    x, y : sont des variables qui contiennent les coordonnées         *
     *           de la position calculée.                                   *
     *    di,di1,di2 : les distances entre les centres des cercle di,di1, di2  *
     *                 et les points calculés, celles-ci sont multipliées par  *
     *                 40 car le calcul est réalisé dans le repère absolu.    *
     *                                                                      *
     *                                                                      *
     *   Rappel sur les lois  des méthodes :                                *
     *                                                                      *
     *  L'information d'angle:                                              *
     *                 1 : (tg(θ)² + 1)x² + 2(−k₁ + Rtg(θ))x + R² + k₁² − d² = 0, tq R = −k₁tg(θ)  *
     *                 2 : {x = d * cos(θ) + k₁                             *
     *                     {y = d * sin(θ) + k₂                             *
     *   La multilatération :                                               *
     *                                                                      *
     *                                                                      *
     *                                  [a b][x] = [h]                      *
     *                                  [c d][y]   [g]                      *
     *                                a = 2 * (x₁ − x₃) , b = 2 * (y₁ − y₃),  *
     *                                c = 2 * (x₂ − x₃), d = 2 * (y₂ − y₃),   *
     *                                                                      *
     *                                h = x₁² − x₃² + y₁² − y₃² + d₃² + d₁²,  *
     *                                g = x₂² − x₃² + y₂² − y₃² + d₃² + d₂²   *
     *                                                                      *
     *    voir chapitre 1                                                   *
      **************************************************************************

// Chaque méthode utilise un constructeur différent :

// Ce constructeur est utilisé dans le cas de la méthode
// Multilatération

   public   CalculeCoord(float xi,float yi,float di,float xi1,float yi1,float
di1,float xi2,float yi2,float di2){
         this.xi=xi;
         this.yi=yi;
         this.di=di*40;
         this.xi1=xi1;
         this.yi1=yi1;
         this.di1=di1*40;
         this.xi2=xi2;
         this.yi2=yi2;
         this.di2=di2*40;
```

```
   }

// Ce constructeur est utilisé dans le cas de la méthode
// de localisation par l'information d'angle

   public   CalculeCoord(float xi,float yi,float di,float tita){
       //  ce consttructure utilisé dans la méthode des angles
         this.xi=xi;
         this.yi=yi;
         this.di=di*40;
         this.xi1=xi1;
         this.yi1=yi1;
         this.tita=tita;
   }
public void Multilateration(){

float alfa=2*(xi-xi2),beta=2*(yi-yi2),sigma=2*(xi1-xi2),psi=2*(yi1-yi2);
float a=carre(xi)-carre(xi2)+carre(yi)-carre(yi2)+carre(di2)-carre(di);
float b=carre(xi1)-carre(xi2)+carre(yi1)-carre(yi2)+carre(di2)-carre(di1);

x=(b*beta-a*psi)/(sigma*beta-psi*alfa);
y=(a-alfa*x)/beta;

   }

 public void infAngle(int i){
 float alpha=(float) (45*tita*4/Math.PI);// convertir tita de radial vers le degré

        // si i=1 ; la méthode des coordonnées cartésiennes
        // si i !=1 ; la méthode des coordonnées polaires

        if(i==1){
          float R1=-xi*tg(tita);
            float a=(carre(tg(tita))+1);
            float b=2*(-xi+R1*tg(tita));
            float c=carre(R1)+carre(xi)-carre(di);
              this.x=calculeX(a,b,c,1);
         // choisir selon tita la valeur x
              if(alpha<90 || (alpha<360 && alpha>245))
                if(x<xi)this.x=calculeX(a,b,c,2);
              if((alpha>90 && alpha<(180+90)))
                if(x>xi)this.x=calculeX(a,b,c,2);
              this.y=(-(x*tg(tita)+R1)+yi);

               // le repère absolu exige la multiplication par moins(-)
               // de la première partie de l'équation qui calcul y
               // voir figure anexe.2

       }else{
             this.x=(int)(di*Math.cos(tita))+xi;
             this.y=-(int)(di*Math.sin(tita))+yi;
             // même chose pour cette méthode
             //voir figure annex.2
         }
       }
```

```
//Méthodes internes utiles pour améliorer la librairie Math

   private float tg(float tita) return (float)(Math.tan(tita));}
   private float carre(float x){ return x*x; }
   private float calculeX(float a,float b, float c,int sol){

    float delta=carre(b)-4*a*c,res;
          switch(sol){
          case 1 :res=(float) ((-b+Math.sqrt(delta))/(2*a));break;
          case 2 : res=(float) ((-b-Math.sqrt(delta))/(2*a));break;
          }
          return res;
   }


  // Calculer l'angle de la droite qui relie ces deux points (x1,y1), (x2, y2)
   public void Angle(float x1,float y1,float x2,float y2){
         if((x1-x2)!=0){
         tita=(float) Math.atan(Math.abs((y2-y1)/(x2-x1)));
         tita=(float) (45*tita*4/Math.PI); // conversion : radial→ degré

         }else tita=90;
         if(x2<x1 && y2<y1){
              tita=(90-tita)+90;
         }
         if(x2<x1 && y2>y1){
                    tita=tita+180;
               }
         if(x2>x1 && y2>y1){
                    tita=(90-tita)+180+90;
               }
         if(x2<x1 && y2==y1){
                    tita=180;
               }
         if(x2==x1 && y2>y1){
                    tita=90+180;
               }

   }
public float getTita() {
      return tita;
    }
public float getX() {
      return x;
}
public float getY() {
      return y;
}
}
```

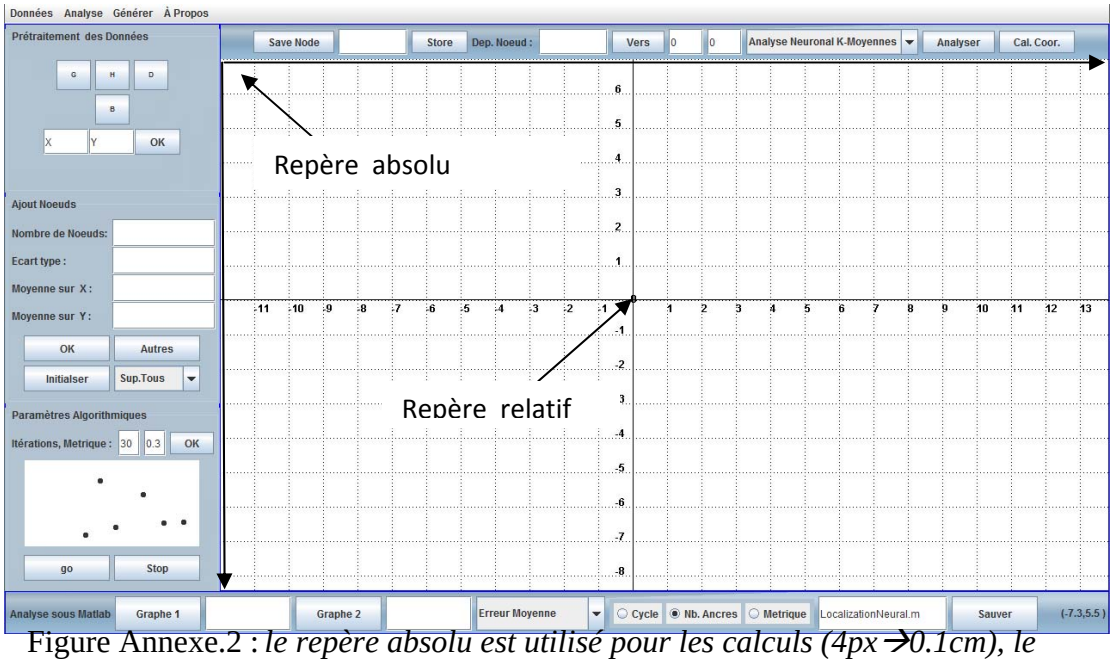


Figure Annexe.2 : *le repère absolu est utilisé pour les calculs (4px→0.1cm), le repère relatif est utilisé pour la représentation des résultats*

```
/*****************************************************************
 *   Les Méthodes suivantes réalisent la conversion de           *
 *   repère relatif vers le repère absolu et  de repaire         *
 *   absolu vers le repère relatif                               *
 *****************************************************************/

public Point ConvertCoorAbs(point point) {
        // On multiplie par 10 pour avoir une précision de 0.1
        //abs→ rel
        //posX,Y : coordonnées de repère relatif dans le repère absolu

      X= ((point.x-PosX)/40.f-1)*10;
      Y=(-(point.y-PosY)/40.f)*10;
}
Point p=new Point((int)X,(int)Y);
return p;

}

public Point ConvertCoorRel(float Xrel,float Yrel){

Point p=new Point();
Xabs=(Xrel+1)*40+PosX;
Yabs=(-Yrel*40+PosY);
p.x=(int)X;
p.y=(int)Y;
return p;
}
```

```
/*****************************************************************************
 *    Cette méthode calcule les distances entre les nœuds voisins,           *
 * ainsi l'information de l'angle pour les voisins d'un nœud d'ancre,         *
 * en supposant que la valeur retournée par RSSI est l'échelle de nœud        *
 * multipliée par une valeur (100-[0 :: Err])%                                *
 *                                                                            *
 *  • NodeX,Y[] :   vecteur comportant les coordonnées réelles des nœuds      *
 *  • Dist() : est une fonction qui calcule la distance euclidien entre deux  *
 *             points                                                         *
 *  • Nœud[] : est un vecteur de nœuds (capteurs)                             *
 *  • vectDist[i] : est un vecteur des voisinages d'un nœud i                 *
 *  • vectAngle[i] : est un vecteur comportant les angles des voisins         *
 *                   d'un nœud d'ancre i.                                     *
 *                                                                            *
   La complexité de ce programme N(N-1)/2, O(N²)                             *
*****************************************************************************/


public void NodesDist(){
    for(int i=1;i<NodeX.length;i++){
      float[] vectDist=new float[n];int[] vectAngle=new int[n];
         if(NodeX[i]!=0 && NodeY[i]!=0)
            for(int k=1;k<NodeX.length;k++){
               if(k==i)continue;
                if(NodeX[k]!=0 && NodeY[k]!=0){
                  float d=Dist(NodeX[i],NodeY[i],NodeX[k],NodeY[k]);
            float err=(float) (Math.random()*Noeud[i].getNoeud().getErreur());
                   if(Noeud[i].getNoeud().getMetrique()*(100-err)/100>d/40){
                     vectDist[k]=((int)d/4)/10.f;
                    if(getFenNode()[j].getNoeud().getNodeType().equals("Ancre")){
                     // ces inst. Transforme angle(radial)→ angle(degré)
                       CalculeCoord pt=new CalculeCoord();
                        pt.Angle(NodeX[i], NodeY[i],NodeX[k],NodeY[k]);
                        vectAngle[k]=(int)pt.getTita();
                               }
                             }
                          }
                            Noeud[i].getNoeud().setVectDist(vectDist);
                    if(getFenNode()[j].getNoeud().getNodeType().equals("Ancre"))
                        Noeud[i].getNoeud().setAngles(vectAngle);
              }
       } // fin boucle for 2
 } // fin boucle for 1
```

```
/*************************************************************************
*     Procédure de calcul effectuant la méthode de localisation par      *
*   l'information d'angle.                                               *
*     • n : le nombre des capteurs                                       *
*     • NodeEstX,Y[] : sont des vecteurs des positions estimées          *
*     • NodeX,Y,[] : vecteurs de coordonnées des nœuds d'ancres          *
*     • ColorMethode : est un vecteur qui contient la couleur de nœud    *
*          Correspondante; Blue pour les calculs par l'inf. d'angle      *
*                                                                        *
*  1. Pour tous les nœuds d'ancres faire                                 *
*  2. Pour tous les nœuds normaux qui se trouvent au voisinage de nœud   *
*     d'ancre actuel faire                                               *
*  3. Récupérer la distance et l'angle de nœud normal par rapport        *
*     au nœud d'ancre puis calculer les coordonnées.                     *
*     Dans ce programme, nous avons utilisé la méthode des coordonnées   *
*     cartésiennes                                                       *
*  4. Fin pour 2, puis fin pour 1                                        *
*                                                                        *
*  La complexité O(k*l) d'inter nœuds,                                   *
*  k= nombre de nœuds d'ancres, l= nombre des  nœuds voisins de nœud d'ancre *
*                                                                        *
*************************************************************************/

public void infoAngle(){
      int[] Ancres=new int[n];int l=0;float[] vectDist=new float[n];
      for(int i=1;i<n;i++){
            if(getFenNode()[i].getNoeud().getNodeType().equals("Ancre")){
                  vectDist= getFenNode()[i].getNoeud().getVectDist();
                  Ancres=getFenNode()[i].getNoeud().getAngles();
                 for(int j=1;j<n;j++)
                   if(vectDist!=null && vectDist[j]!=0){

if(!getFenNode()[j].getNoeud().getNodeType().equals("Ancre")){
CalculeCoord infAngle=new CalculeCoord(NodeX[i],NodeY[i],vectDist[j],Ancres[j]);
                        infAngle.infAngle(1); // cal. Des coordonnées
// 1 : pour la méthode de coordonnées cartésiennes
// 2 : pour la méthode de coordonnées polaires
                       this.NodeEstX[j]=(int) infAngle.getX();
                       this.NodeEstY[j]=(int)(infAngle.getY());
                       this.ColorMethode[j]=Color.BLUE;
                          }
                  }
            }
      }
}
```

```
/*****************************************************************************
 *   NodeKnowPos[] : est un vecteur des nœuds qui possèdent leurs positions  *
 *   Procédure de calcul par la multilatération :                            *
 *    •  Pour chaque nœud normal faire :                                     *
 *    •  Si les coordonnées de ce nœud sont inconnues alors                  *
 *          o  Si  plus de trois nœuds possédants leurs coordonnées          *
 *             au voisinage de ce nœud alors                                 *
 *                   ▪  Appliquer la multilatération.                        *
 *     La complexité inter-nœud est O(k*j)                                   *
 *     k : est la valeur des nœuds qui ne possèdent pas leurs coordonnées    *
 *     j : est le nombre des nœuds au voisinage de nœud inconnu.             *
 *****************************************************************************/
public void multilateration(){
      float[] vectDist=new float[n];
      for(int i=1;i<n;i++){
            if(!getFenNode()[i].getNoeud().getNodeType().equals("Ancre")){
            if((NodeEstX[i]==0 && NodeEstY[i]==0)&&((NodeX[i]!=0 ||
NodeY[i]!=0))){
      vectDist= getFenNode()[i].getNoeud().getVectDist();
      int l=0;int[] NodeKnowPos=new int[4]; float[] DistNodes =new float[4];

       for(int j=1;j<n;j++)
 if((NodeEstX[j]!=0 ||
NodeEstY[j]!=0)||(getFenNode()[j].getNoeud().getNodeType().equals("Ancre"))){
            if(vectDist[j]!=0){
              l=l+1;
                  NodeKnowPos[l]=j;
                 DistNodes[l]=vectDist[j];
                 if(l==3)break;
                  }
             }
    if(l>=3){
CalculeCoord cal=new
CalculeCoord(NodeX[NodeKnowPos[1]],NodeY[NodeKnowPos[1]],DistNodes[1],NodeX[NodeKn
owPos[2]],NodeY[NodeKnowPos[2]],DistNodes[2],NodeX[NodeKnowPos[3]],NodeY[NodeKnowP
os[3]],DistNodes[3]);
            cal.Multituration();
             if(cal.getX()<999)// [-999 ::999] : intervalle de calcul par multila.
                  if(cal.getX()>-999){
                     NodeEstX[i]=(int)cal.getX();
                     NodeEstY[i]=(int)cal.getY(); }
                      this.ColorMethode[i]=Color.green;
     // couleur verte pour les pos. de nœuds calculées par multilatération
                  }else
{System.out.println("ERREUR : multilatération invalide pour Le Noeud : "+i+" qui
Possède moins de trois nœud voisins qui ont des positions connus ! ");}

           }
         }
      }
   }
```

```
}
/*************************************************************************
 *  Cette classe réalise l'analyse neuronale :                           *
 *                                                                       *
 *    Les observations (z_i) :sont les coordonnées de nœuds estimés      *
 *    Les référents (w_i) : sont les nœuds d'ancres                      *
 *    K : est le nombre d'itérations (t)                                 *
 *   Voir  chapitre 2                                                    *
 *************************************************************************/

package NeuronalLocalzation;
 // les imports
public class NeurnalAnalysis {
  private  int NbC;         //Nombre des nœuds d'ancres
  private float[][][] NodesEstimate;// NodesEstimate [i][j][k]
                            // i est l'indice pour nb. de noeuds d'ancres
                            // j : est le numéro de nœud
                            // k =0 ou 1, les coordonnées estimées (x,y) de nœud j
  private float[][] Dist;  // matrice de distances entre les nœuds
  private float[][] vect;  // vect[i][j]: position estimée de nœud i et la
                            // coordonnée j
  private float[][] posAncre;//est la position de nœud calculée
  private int[] init;      // vecteur contient les indices des nœuds d'ancres
  private int Nbit;        // Nombre d'itérations
  private int n=500;       // le nombre maximal des nœuds dans le réseau
                 // Constructeur
  NeurnalAnalysis(int Nbit,int NbC,float[][] vect,int[] init){
       this.NbC=NbC;
       this.vect=vect;
       this.DistN=new float[NbC][NbC];
      // modifier n, selon le nombre des nœuds estimés
    for(int i=1;i<vect.length;i++)
        if(vect[i][0]==0 && vect[i][1]==0){n=i;break;}
       this.Nbit=Nbit;
       NodesEstimate=new float[NbC][n][2];
       Dist=new float[NbC][n];
       posAncre=new float[NbC][2];

  }
public void AnalyseStochastque(){
       int iteration=0;
  // Début de la procédure  de calcul
```

```
for(int k=1;k<=Nbit;k++){
        initialiser();
        for(int z=1;z<NbC;z++){
              if(iteration==0){
                    for(int i=1;i<NbC;i++){
                  // affecter les nœuds d'ancres au vect. posAncre
                         posAncre[i][1]=NodesEstimate[i][1][1];
                         posAncre[i][0]=NodesEstimate[i][1][0];
                    }
              }
              iteration++;
    // appliquer l'équation :      w_c^{t-1} - 2 μ^t Σ_{z_i∈A, X^t(z_i)=c} (w_c^{t-1} - z_i)

for(int i=1;i<NodesEstimate[z].length;i++){
 if(NodesEstimate[z][i][1]!=0 || NodesEstimate[z][i][0]!=0){
PosAncre[z][1]=(float) (PosAncre[z][1]-
2*(1/(Math.sqrt(iteration)))*(PosAncre[z][1]-NodesEstimate[z][i][1]));
PosAncre[z][0]=(float) (PosAncre[z][0]-
2*(1/(Math.sqrt(iteration)))*(PosAncre[z][0]-NodesEstimate[z][i][0]));
                  }

                  }
            Distance(); // recalculer les distances inter-nœuds
            Affcter();  // affecter les nœuds normaux aux nœuds d'ancres
         }
        }
      }
    }

  public void AnalyseKmean(){
        initialiser();
         // appliquer la relation 3 de chapitre 2,   w_c = (Σ_{z_i∈A∩P_c} z_i) / n_c
          for(int k=1;k<=Nbit;k++){
            for(int i=1;i<NbC;i++){
                PosAncre[i][1]=Moyenne(NodesEstimate[i],1);
               PosAncre[i][0]=Moyenne(NodesEstimate[i],0);
      }
        Distance();
        Affcter();
       }
```

```
  }
// Affect() est la procédure de calcul de la fonction d'affectation définie dans
// le chapitre 2      X(z) = arg min_r | |z − w_r| |^2

private void Affcter() {
     // ajouter chaque observation au référent correspondant
     // l'indice i correspond au numéro de référent
        for(int j=1;j<n;j++){
               float m=Dist[1][j];int l=1;
             for(int i=2;i<NbC;i++){
                if(m>Dist[i][j]){
                  {m=Dist[i][j];
                   l=i;//référent cherché
                  }
                }
         }
       // ajouter la valeur vect[j][l] dans NodesEstimate
         Ajouter(vect[j],l);
       }

}

// chaque référent est initialisé en lui affectant les coordonnées de noeuds
//d'ancres correspondant, mais aussi le vecteur des positions des nœuds d'ancres.

public void initialiser(){
        for(int i=1;i<NbC;i++){
           this.NodesEstimate[i][1][0]=vect[init[i]][0];
              this.NodesEstimate[i][1][1]=vect[init[i]][1];
              this.PosAncre[i][1]=vect[init[i]][1];
             this.PosAncre[i][0]=vect[init[i]][0];

        }
  }
// Distance() : est une fonction qui remplie la matrice de distance
  public void Distance(){
         for(int k=1;k<NbC;k++){
              for(int i=1;i<n;i++){
     Dist[k][i]=Dist(PosAncre[k][0],PosAncre[k][1],vect[i][0],vect[i][1]);
              }
```

```
        }
  }
// Ajout() : ajouter la valeur dans NodesEstimate[j] (j : le numéro de référent)
// La procédure consiste à extraire la valeur à ajouter de référent précédent
// puis ajouter cette valeur au nouveau référent
private void Ajouter(float[] m, int l) {
// extraire la valeur de référent précédent
for(int i=1;i<NbC;i++){
              for(int j=1;j<NodesEstimate[i].length;j++){
if(NodesEstimate[i][j][0]==m[0]&&NodesEstimate[i][j][1]==m[1])
              {NodesEstimate[i][j][0]=0;NodesEstimate[i][j][1]=0;}
             }
       }
  // ajouter la valeur au référent indiqué par la variable l
for(int i=1;i<NodesEstimate[l].length;i++){
 if(NodesEstimate[l][i][1]==0 && NodesEstimate[l][i][0]==0)
      {NodesEstimate[l][i][1]=m[1];NodesEstimate[l][i][0]=m[0];break;}
       }
}
// Dist : calcul de la distance euclidien entre deux points.
  public float Dist(float x1,float y1,float x2,float y2){
        float  Dist=(float) (int)(Point2D.distance(x1, y1, x2, y2)*100)/100.f;
           return Dist;
       }
 // Moyenne() : est une fonction qui implémente l'équation 3 de chapitre 2.
  public float Moyenne(float[][] NodesEstimate,int k){
           float som=0,l=0;
           for(int i=1;i<NodesEstimate.length;i++){
                 if(!(NodesEstimate[i][0]==0 && NodesEstimate[i][1]==0))
                   {som=som+NodesEstimate[i][k];l++;}
           }
           if(l!=0)return som/l;
           else return 0;
      }
}
```

```
/**************************************************************************
 *    Cette class génère automatiquement un fichier Matlab (nomDeFichier.m) *
 * Qui représente une simulation graphique de réseau de capteurs           *
 **************************************************************************/

package NeuronalLocalzation;
// les imports;

public class OutFile {
       private String Nom ;             // Nom de fichier
       private String TitreX;           // titre sur l'axe des Xs
       private String TitreY;           // titre sur l'axe des Ys
       private String Label;            // titre de la figure
       private String Value1;           // Légende de graphe 1
       private String Value2;           // légende de graphe 2
       private int n=500;               // le nb. Max. des nœuds
       private float[] X=new float[n];  // Les points sur les Xs de graphe 1
       private float[] Y=new float[n];  // Les points sur les Ys de Graphe 2
       private float[] Xx=new float[n]; // Les points sur les Xxs de graphe 1
       private float[] Yy=new float[n]; // Les points sur les Ys de graphe 2

  OutFile( String Nom,String Value1,String Value2,float[] X,float[] Y,float[]
  Xx,float[] Yy,String TitreX,String TitreY,String Label){
             this.TitreY=TitreY;
             this.TitreX=TitreX;
             this.Label=Label;
             this.Nom=Nom;
             this.Value1=Value1;
             this.Value2=Value2;
             this.X=X;
             this.Y=Y;
             this.Xx=Xx;
             this.Yy=Yy;
       }

public  void Generer() {
                   File file = new File(Nom)
                   FileWriter fw;
                   FileReader fr;
```

```
String
str="%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n" +
"   %\tCe fichier est généré automatiquement pour une analyse sous Matlab \t%\n
%\t\t\t\t\t de logigiel de simulation\n" +
"   % Localisation dans les réseaux de soneur par application des réseaux de
neurones \n" +
"   \n% Projet de fin d'étude Master 2 Recherche " +
"   \n% Auteur : Yazid Takfarinas " +
"    \n% Promoteur : pr. Lalam" +
"    \n% Promotion : 2011" +
"\n%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n\n";
str += "X="+Arrays.toString(X)+";\nY="+Arrays.toString(Y);
str+="\n\nplot(X,Y);\nxlabel('"+TitreX+"');\nylabel('"+TitreY+"');\ntitle('"+Label
+"');\n";
str+="hold
on;\nXx="+Arrays.toString(Xx)+";\nYy="+Arrays.toString(Yy)+";\nplot(Xx,Yy,'r')\nho
ld on\nplot(Xx,Yy,'.b','markersize',17);" +
"\nlegend({'"+Value1+"','"+Value2+"'})\nholdon;plot(X,Y,'.r','markersize',15);\n";
                  try {
                        //Création de l'objet
                        fw = new FileWriter(file);
                        //On écrit la chaîne
                        fw.write(str);
                        //On ferme le flux
                        fw.close();
                        //création de l'objet de lecture
                        fr = new FileReader(file);
                        str = "";
                        int i = 0;
                        //Lecture des données
                        while((i = fr.read()) != -1)
                              str += (char)i;
                  } catch (FileNotFoundException e) {
                        e.printStackTrace();
                  } catch (IOException e) {
                        e.printStackTrace();
            }
      }
}
```

```
/*************************************************************************
 *  Cette méthode permet de réaliser la procédure de l'analyse neuronale  *
 *                                                                        *
 *      1.   Prédire les coordonnées sans analyse neuronale               *
 *      2.   Réaliser l'analyse neuronale pour les coordonnées  estimées  *
 *      3.   Répéter 1 et 2 tant que le système prédit des nouveaux nœuds *
 *    Alg : choix des algorithmes neuronaux 1 : K-mean, 2 : Stochastique  *
 *   NbFois : nombre d'itérations                                         *
 *                                                                        *
*************************************************************************/

public void ProcedureAnalyseNeuronale(int Alg){

// les coordonnées des nœuds d'ancres sont inchangeables
for(int i=1;i<centre.getFenNode().length;i++){
if(centre.getFenNode()[i].getNoeud().getNodeType().equals("Ancre")){
        centre.NodeEstX[i]=centre.NodeX[i];centre.NodeEstY[i]=centre.NodeY[i];
      }
int[] init=null ;
int[] initD=null ;
NeurnalAnalysis m=null;//variable neuronale
for(int h=1 ;h<NbFois ;h++){

// 1. estimer les coordonnées des capteurs
         centre.infoAngle() ;
         centre.multilateration() ;

// 2. résaliser l'analyse neuronale

      init=new int[centre.n]; // indexer  le vect. v
      initD=new int[centre.n];// indexer le vect. des positions estimées
      int l=1,inc=1;
      float[][] v=new float[centre.n][2];  // valeurs des capteurs

   // le vecteur v reçoit les données depuis le vecteur des  positions estimées

      for(int i=1;i<v.length;i++){
      if(centre.NodeEstX[i]!=0 || centre.NodeEstY[i]!=0){
         v[inc][0]=centre.NodeEstX[i];v[inc][1]=centre.NodeEstY[i];
         if(centre.getFenNode()[i].getNoeud().getNodeType().equals("Ancre"))
            {init[l]=inc;initD[l]=i;l++;}
                        inc++;
                       }
                  }
       // appel de l'analyse neuronale  selon Alg

         m=new NeurnalAnalysis(Nbit,l,v,init);
                  switch(Alg){
                        case 1 : m.AnalyseKmean(); break;
                        case 2 : m.AnalyseStochastque();break;
                  }
   } //3. répéter, l'estimation et l'analyse neuronale
```

```
// vider les vecteurs de positions estimées pour une prochaine utilisation

              for(int p=1;p<centre.NodeEstX.length;p++){
                    centre.NodeEstX[p]=0;centre.NodeEstY[p]=0;
                          }

float[][] moyen=m.getPosAncres();

// sauver les positions estimées qui sont générées par l'analyse neuronale

for(int i=1;i<moyen.length;i++){
Thread t=new Thread(new thread(centre.NodeX[initD[i]],centre.NodeY[initD[i]],
(int)moyen[i][0],(int)moyen[i][1],initD[i]));
 t.start();
    }

}


/**************************************************************************

                        ******************************

    Contacter l'e-mail suivant pour télécharger le simulateur ainsi que
    son code source entier : takfarinas_yazid@yahoo.fr

                        ******************************

**************************************************************************/
```