UNIVERSITÉ DE
SHERBROOKE
Faculté de génie
Génie électrique et génie informatique

Détermination de la vitesse relative d'une sonde par réseaux de Kohonen lors d'une phase d'atterrissage autonome sur un corps planétaire

Mémoire de maîtrise ès sciences appliquées
Spécialité : génie électrique

Sébastien OLMICIA

Sherbrooke (Québec) Canada Septembre 2005

RÉSUMÉ

L'atterrissage autonome d'une sonde sur une planète ou tout autre corps céleste est une tâche difficile. Il est en effet nécessaire d'avoir une bonne connaissance de l'environnement qui entoure la sonde au moment de cette phase critique. Pour cela, différentes technologies rivalisent tant sur le plan technique que sur le plan des coûts engendrés. Une des technologies qui se démarque est le LIDAR (LIght Detection And Ranging). Cet outil permet de générer des cartes en trois dimensions du relief scanné.

Ce mémoire porte plus particulièrement sur la façon de traiter les données du LIDAR efficacement pour déterminer la vitesse relative de la sonde par rapport au sol. Cette technologie est directement en concurrence avec les systèmes de cameras avec altimètre ou le radar Doppler qui permettent eux aussi de déterminer la vitesse relative d'une sonde. Les avantages du LIDAR, grâce à l'apport de technologies dites d'intelligence artificielle adaptée à l'atterrissage autonome d'une sonde sur une planète, seront démontrés tout au long de ce mémoire. L'intelligence artificielle encore peu utilisée en aérospatiale semble posséder tous les atouts pour être déterminante dans les missions spatiales de demain.

La méthode proposée à l'avantage d'être adaptée aux conditions extrêmes que représente un atterrissage. Les différents tests ont montré que cette méthode était robuste au bruit de capteur et à la descente de la sonde entre deux balayages de la surface par le LIDAR. De plus, le type de terrain ne semble pas avoir d'influence sur le résultat de l'algorithme. En effet, que la surface sous la sonde soit extrêmement plane ou accidentée, les résultats sont toujours excellents. L'utilisation de deux réseaux de Kohonen permet en effet d'être très réactif aux types de surfaces et conditions rencontrées lors d'une phase d'atterrissage. Cela est dû à l'adaptation permanente de ceux-ci dans la méthode décrite dans ce mémoire.

REMERCIEMENTS

Je tiens à remercier mes deux directeurs de maîtrise, Charles-Antoine Brunet et Jean de Lafontaine pour leur support tous au long de ces travaux ainsi que mes parents pour leur soutien moral et financier. Finalement, je tiens à remercier tous mes collègues de travail et amis qui ont rendu ces deux années très agréables, en particulier Mimi.

TABLE DES MATIÈRES

LISTE DES FIGURES

LEXIQUE

LIDAR : Capteur actif utilisant comme source émettrice un laser. Le lidar est utilisé pour la mesure des distances, la détection et, éventuellement, la localisation de constituants des milieux rencontrés. Le terme LIDAR est l'acronyme de l'expression anglo-américaine *LIght Detection And Ranging*, qui se traduit en français par détection et télémétrie par la lumière.

Point d'élévation : Dans ce document, le point d'élévation caractérise la distance entre le LIDAR et la surface qu'il scanne.

Carte : Dans ce document, une carte représente la surface 3D, située sous la sonde, générée par le LIDAR. Chaque point de cette carte représente la distance entre la sonde et le sol dans un plan cartésien.

Unité de mesure : Grandeur déterminée, adoptée par convention, utilisée pour exprimer quantitativement des grandeurs de même dimension. Dans ce mémoire, cela se rapporte au pourcentage de la taille d'une carte.

Intelligence artificielle : Discipline scientifique relative au traitement des connaissances et du raisonnement humain dans le but de les reproduire artificiellement et ainsi de permettre à un appareil d'exécuter des fonctions normalement associées à l'intelligence humaine comme le raisonnement, la compréhension et l'adaptation. L'intelligence artificielle regroupe un ensemble de techniques dont les plus connues sont les réseaux de neurones artificiels, la logique floue, les algorithmes génétiques et les systèmes experts.

Réseau de Kohonen : Réseaux de neurones basés sur un apprentissage non-supervisé, c'est à dire que le réseau converge vers une solution dépendante des exemples fournis en entrée. Ils sont aussi parfois appelés SOM pour *Self Organizing Map*.

Tache : Une tache représente un minimum ou un maximum local et les points d'élévation qui l'entourent agrandi par l'algorithme de détection des marqueurs.

Marqueur : Un marqueur représente les coordonnées d'un centre de gravité d'une tache.

Plan : Dans ce document, chaque matrice est définie dans un plan d'origine (1,1). Les coordonnées (x,y) d'un point sont toujours exprimées dans le plan qui les contient.

Tracking **:** Termes anglais définissant l'action de suivre quelque chose, dans ce mémoire, ce sont les marqueurs.

CHAPITRE 1

INTRODUCTION

1.1 Problématique

L'aérospatiale lance de grands défis sur le plan de l'autonomie des systèmes embarqués, plus particulièrement l'envoi de sondes autonomes sur des planètes du système solaire. En effet, parmi les problèmes qui se posent lorsqu'un engin est envoyé sur une autre planète, l'atterrissage à un endroit désiré est déterminant. Bien que la maîtrise de l'énergie et des trajectoires dans l'espace soient primordiales, ce ne sont pas les difficultés majeures. De nos jours, dans ce type d'expédition, l'interaction de la sonde avec son environnement est le problème essentiel par sa complexité et le peu de connaissance des conditions rencontrées. L'inconnu constitue un frein majeur pour l'exploration spatiale autonome, car il est difficile de concevoir des engins sans connaître toutes les situations auxquelles ils vont être confrontés. Il faut donc penser à l'imprévisible et concevoir ces machines pour qu'elles soient le plus robustes possible.

De nombreux projets parlent d'expéditions sur une planète telle que Mars ; des missions où des engins autonomes seraient capables de se poser, de prendre des échantillons sur la planète et de repartir en direction de la Terre, tout cela sans aucune interaction, ou presque, avec l'homme. En effet, il est impossible d'interagir en temps réel avec les engins envoyés dans l'espace, car les temps de réponse sont trop longs. En reprenant l'exemple de Mars, la durée aller-retour du signal varient entre 20 à 40 minutes. Il est donc impossible de les contrôler à distance. Cela est d'autant plus vrai lors d'une phase d'atterrissage qui se déroule en quelques minutes. L'autonomie des engins spatiaux est donc un des principaux défis dans l'exploration spatiale de demain.

Ce projet traite particulièrement d'une phase délicate : l'atterrissage. C'est le moment critique dans une mission spatiale sur une planète. La sonde doit connaître sa vitesse relative

et sa position par rapport au sol ainsi que sa vitesse angulaire à tout moment pour réussir à contrôler son approche finale. Elle doit pouvoir corriger sa trajectoire et déclencher à l'instant idéal les différentes manoeuvres. Pour cela, de nombreux outils de mesure sont nécessaires ce qui augmente considérablement le coût d'une mission et accroît la masse de la sonde. Dans ce projet, il est question d'estimer la vitesse relative de la sonde par rapport au sol ce qui est une des variables importantes lors d'un atterrissage. La solution décrite dans ce mémoire doit donc être robuste. En effet, une erreur trop grande sur la vitesse relative pourrait avoir des conséquences catastrophiques pour la mission. La connaissance de la vitesse relative permet ainsi au système embarqué de la sonde de contrôler la direction et la vitesse lors d'un atterrissage autonome pour atteindre une vitesse nulle au contact avec le sol.

Dans ce projet, un LIDAR (LIght Detection And Ranging) est utilisé pour balayer la surface de la planète sur laquelle la sonde atterrit. C'est un radar à base de rayon laser balayé sur deux axes. Le LIDAR est une technologie relativement récente. La technique développée dans ce mémoire est donc novatrice. En effet, le LIDAR joue le rôle d'organe de visée optique. Vu qu'il n'est pas conçu à la base pour estimer une vitesse de déplacement, il faut donc lui adjoindre un algorithme performant pour réussir à traiter les données mesurées par celui-ci. Aucune autre méthode n'a pour l'instant été développée pour calculer la vitesse relative de façon fiable à l'aide d'un LIDAR. C'est pour cela qu'il n'y a pas de comparaison avec d'autres méthodes tant sur le plan théorique que pratique. Cependant, les bases de l'algorithme développé dans ce document se trouvent dans les recherches existantes.

De nombreux chercheurs se sont penchés sur l'autonomie de systèmes ou engins tels que des sondes ou des robots. Deux des difficultés sont la perception de l'environnement et la prise de décision. Dans ce mémoire, l'accent est porté sur la perception de l'environnement. Ce projet de recherche s'inscrit dans un scénario d'atterrissage autonome LAPS pour *Lidar-based Autonomous Planetary Landing System* [de Lafontaine et Gueye, 2004]. Dans ce projet, le LIDAR est utilisé pour déterminer un site sécuritaire dans la surface balayée par le LIDAR. Le LIDAR étant un outil coûteux, il est intéressant de voir s'il peut remplacer d'autres capteurs pour limiter le coût global d'une telle mission. De là est venue l'idée de l'utiliser pour déterminer la vitesse relative à la place d'un radar Doppler. La réduction du nombre

de capteurs permettrait de justifier l'utilisation d'un LIDAR qui est un outil très efficace pour modéliser la surface se situant sous la sonde pendant son atterrissage. Cette efficacité permet déjà de déterminer parfaitement les sites sécuritaires d'atterrissage qui se trouvent à la portée de la sonde. De plus, la réduction du nombre d'appareils de mesure allégera la sonde tout en minimisant les risques de pannes.

1.2 Objectifs

Le principal objectif de ce mémoire est de compléter les capacités du LIDAR en le transformant en véritable compteur de vitesse. La méthode suggérée pour y parvenir est de chercher des points marquants dans deux cartes consécutives, d'en déterminer ensuite les points marquants communs pour calculer la distance parcourue et finalement la vitesse de la sonde.

Le but du projet est donc, à partir de techniques d'intelligence artificielle et des données du LIDAR, de déduire la vitesse relative de la sonde par rapport au sol. Seule la vitesse horizontale est recherchée, car l'orientation relative et la vitesse verticale sont connues de façon indépendante, permettant ainsi des corrections.

Du fait du domaine d'application de ce projet, certaines exigences techniques sont requises : les principales sont la robustesse et la stabilité de l'algorithme puisque l'atterrissage est une phase critique. Dans une moindre mesure, la méthode proposée doit aussi tenir compte des capacités de calcul limitées des systèmes embarqués. L'algorithme doit aussi fonctionner sans aucune connaissance préalable du terrain. Finalement, l'erreur sur l'estimation de la vitesse horizontale doit être inférieure à 10 % et l'algorithme doit fonctionner avec une translation maximale entre deux cartes égale à 50 % de la taille d'une carte. Cette translation correspond à la vitesse horizontale maximale que peut atteindre la sonde lors d'une phase d'atterrissage.

1.3 Contributions

Plusieurs méthodes pour déterminer la vitesse relative d'une sonde, sans usage du LIDAR, pendant une phase d'atterrissage ont été proposées et décrites dans la littérature. Les prin-

cipales sont basées sur un système de cameras et d'altimètre. Ces technologies sont basées sur la méthode de Shi-Tomasi [Shi et Tomasi, 1994] ou de corrélation [Misu et coll., 1999] qui permettent de suivre des cibles d'une carte à une autre. Dans ce projet de maîtrise, la détection et le suivi des cibles sont effectués par deux réseaux de Kohonen [Kohonen, 1982] avec comme seul outil de mesure le LIDAR. Cette approche est totalement nouvelle dans la littérature et n'a jamais encore été proposée. Les réseaux de Kohonen (ou Self Organizing Map (SOM)) sont des réseaux de neurones basés sur un apprentissage non supervisé. Au contraire des autres réseaux de neurones, il ne nécessite pas de données d'entraînement dans le cas précis de ce projet. La convergence du réseau se fait par compétition. L'utilisation de ces réseaux a donc permis de se dégager du problème des données d'entraînement. En effet, aucune donnée précise sur les surfaces de planètes ou d'autre corps du système solaire ne sont accessibles.

Les principales contributions de la méthode proposée dans ce mémoire sont donc :

1. De déduire seulement à l'aide des données du LIDAR la vitesse relative d'une sonde lors d'une phase d'atterrissage.
2. D'être robuste aux différents reliefs recontrés lors d'une mission d'atterrissage.
3. D'être robuste à la différence de hauteur de prise de vue entre deux balayages de surface consécutifs ou cartes 3D.
4. D'avoir une erreur de précision sur la vitesse inférieure à 10 %.
5. D'être capable de pouvoir estimer la vitesse avec un déplacement entre deux cartes égales à 50 % de la taille d'une carte.

Les différents résultats exposés tout au long de ce mémoire valideront ces contributions.

1.4 Plan

La suite du mémoire est organisée de la façon suivante :

- Le chapitre 2 donne une explication des différents concepts d'intelligence artificielle abordés dans ce mémoire.

- Le chapitre 3 présente une revue de la littérature sur les différentes techniques permettant de faire du suivi de cible ou d'améliorer le prétraitement des données du LIDAR. Cette partie recense principalement les méthodes utilisées en traitement d'images. L'accent sera mis sur les méthodes dites d'intelligence artificielle. Les avantages de ces techniques seront détaillés et détermineront l'orientation prise par ce projet de maîtrise.
- Le chapitre 4 décrit la conception de l'algorithme. La conception sera d'abord abordée de façon générale, puis détaillera les différents modules de l'algorithme. Enfin, l'accent sera mis sur la méthode de prétraitement et la méthode de *tracking* utilisées. Ces deux méthodes utilisent des réseaux de Kohonen pour effectuer leur tâche.
- Le chapitre 5 montre les différents résultats des deux modules principaux de l'algorithme puis de la globalité du système. L'influence des différents paramètres de l'algorithme sera discutée ainsi que la dépendance de l'algorithme vis-à-vis du type de terrain, du bruit du LIDAR ou de la différence de hauteur de prise de vue entre deux cartes consécutives. La fin du chapitre 5 montre les différentes possibilités de l'algorithme décrit dans ce mémoire dépassant les exigences de ce projet. En particulier, la réaction de l'algorithme face à une rotation entre deux cartes puis à une rotation plus une translation. Cette partie montrera des résultats qualitatifs de ces deux conditions. Les résultats ainsi obtenus traduiront la fiabilité du couple LIDAR + SOM dans des conditions plus complexes.
- Le chapitre 6 revient sur les contributions de ce memoire, décrit les améliorations futures possibles et conclut ce document.

PREMIÈRE PARTIE

REVUE DE LITTÉRATURE

CHAPITRE 2

PRINCIPES D'INTELLIGENCE ARTIFICIELLE

Ce chapitre permet au lecteur d'avoir une idée générale des différentes techniques d'intelligence artificielle abordées dans la revue de littérature.

2.1 Logique floue

La logique floue [Zadeh, 1965] est née de la constatation suivante : la plupart des phénomènes ne peuvent pas être représentés à l'aide de variables booléennes (0 ou 1). La logique floue, en introduisant une infinité de valeurs entre vrai et faux, permet de déterminer un degré d'appartenance à l'un ou l'autre des états. La logique floue comble donc les lacunes de la logique booléenne à l'aide de la notion de continuité entre les états booléens.

Par exemple, si la température extérieure est de 22°C, il fait plus chaud que froid, mais il ne fait pas vraiment trop chaud. Alors que les ordinateurs ne traitent que des nombres, les individus utilisent essentiellement des concepts liés entre eux par des règles logiques. Ces concepts sont matérialisés par des mots, plus ou moins vagues. La logique floue se propose de formaliser l'usage des termes vagues, dans le but de les rendre manipulables par les ordinateurs. Ainsi, l'intérêt de la logique floue réside dans sa capacité à simplifier la réalisation et l'utilisation d'applications informatiques. Elle permet de remplacer des modèles mathématiques par des modèles basés sur des descriptions verbales simples.

La logique floue s'utilise pour des applications nécessitant une expertise humaine pour résoudre un problème contrairement aux réseaux de neurones qui reposent essentiellement sur des données. Tout au long de cette explication, l'exemple d'une piscine extérieure chauffée sera pris. Les deux paramètres en entrée seront la température de l'eau et le temps extérieur. Il est utile pour la compréhension de la logique floue de définir plusieurs termes qui seront utilisés :

Ensemble flou : ensemble de valeur dont les limites sont imprécises (ex : l'ensemble des températures considérée comme chaudes).

Fonction d'appartenance : fonction caractérisant un ensemble flou (ex : fonction triangulaire ou gaussienne).

Terme linguistique (ou étiquette) : terme représentant une valeur (ex : chaude et froide).

Variable linguistique : terme représentant une catégorie et auquel des termes linguistiques peuvent s'appliquer (ex : la température).

La logique floue utilise les mêmes opérateurs que la logique booléenne. Les règles standards afin d'évaluer le degré d'appartenance, habituellement notées μ sont les suivantes :

- la conjonction avec $\mu(A \wedge B) = min(\mu_A, \mu_B)$
- la disjonction avec $\mu(A \vee B) = max(\mu_A, \mu_B)$
- et la négation avec $\mu(\neg A) = 1 - \mu_A$,

À l'aide de ces opérateurs, des règles floues peuvent être créées. La forme générale d'une règle est la suivante :

$$Si\ x\ est\ A\ alors\ y\ est\ B$$

Une règle régissant le problème du chauffage d'une piscine extérieure pourrait être :

$$Si\ eau\ est\ froide\ et\ temps\ est\ nuageux\ alors\ chauffer\ est\ fort.$$

Le traitement flou se décompose en plusieurs étapes distinctes (figure 2.1) : la fuzzification, l'inférence et la défuzzification. La fuzzification consiste à déterminer le degré d'appartenance de chaque entrée à chaque fonction d'appartenance définie dans le système. Ces fonctions d'appartenance, qui utilisent en général entre deux et sept états flous pour chaque variable linguistique, possèdent différentes formes : en triangle, en cloche ou encore en trapèze.

La variable linguistique température et temps sont de bons exemples. Ces variables peuvent être composées de deux termes linguistiques froid et chaud, nuageux et dégagé tel que montré à la figure 2.2. La fuzzification consiste donc à faire passer les différentes valeurs discrètes en entrées à des valeurs floues. Lors de cette étape, le degré d'appartenance des mesures aux variables d'entrée est calculé. Dans l'exemple à la figure 2.2, la valeur 16 correspond à

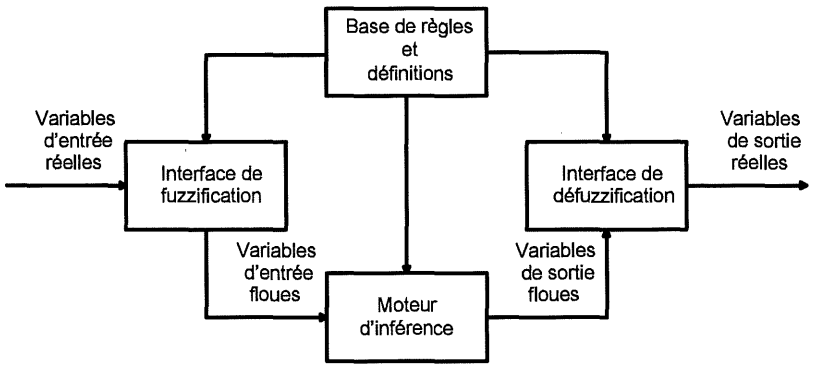


Figure 2.1 – Étapes du traitement flou

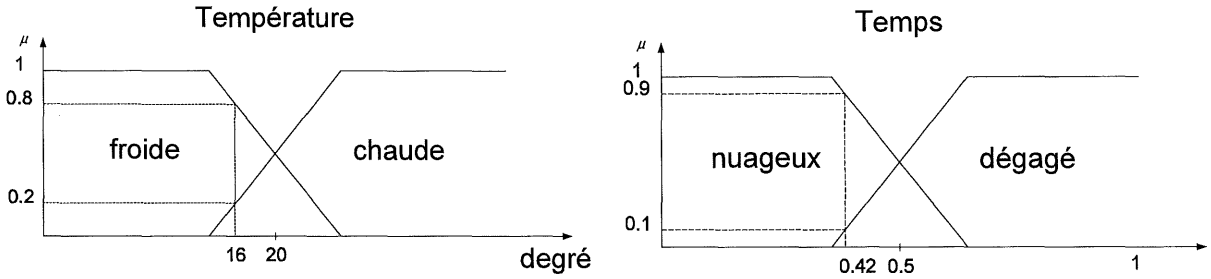


Figure 2.2 – Exemple des deux variables linguistiques régissant le système de la piscine

un degré d'appartenance de 0.2 à l'étiquette chaude et 0.8 à l'étiquette froide. De même, la valeur 0.42 a un degré d'appartenance de 0.9 à l'étiquette nuageux et 0.1 à l'étiquette dégagé. L'étape de fuzzification est alors terminée.

La variable température est décrit de la manière suivante :

- μ_F=0.8
- μ_C=0.2

et la variable temps comme cela :

- μ_N=0.9
- μ_D=0.1

L'inférence permet de déduire le degré d'appartenance des variables linguistiques de sortie à chacune des fonctions d'appartenance. Les degrés d'appartenance de chaque variable permettent d'appliquer les règles floues qui ont été préalablement définies.

Dans ce cas simple, 4 règles peuvent être considérées :

- Si la température de l'eau est froide et le temps est dégagé alors chauffer est moyen
- Si la température de l'eau est froide et le temps est nuageux alors chauffer est fort
- Si la température de l'eau est chaude et le temps est dégagé alors chauffer est faible
- Si la température de l'eau est chaude et le temps est nuageux alors chauffer est moyen.

Cela se traduit par :

- min(μ_F,μ_D)=min(0.8,0.1)=0.1
- min(μ_F,μ_N)=min(0.8,0.9)=0.8
- min(μ_C,μ_D)=min(0.2,0.1)=0.1
- min(μ_C,μ_N)=min(0.2,0.9)=0.2

Plusieurs techniques d'inférence sont possibles. Les plus courantes sont la coupure et la mise à niveau comme illustré à la figure 2.3. La coupure consiste à tronquer l'étiquette à la valeur

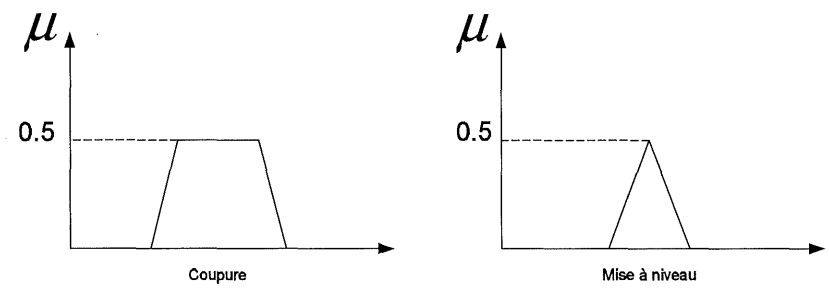


Figure 2.3 – Méthode de coupure et de mise à niveau

calculée alors que la mise à niveau consiste à réduire l'étiquette en conservant sa forme. L'opérateur max est ici utilisé pour chaque ensemble de sortie.

Cela implique :

- chauffer_faible =0.1
- chauffer_moyen =max(0.1,0.2)=0.2
- chauffer_fort = 0.8

La superposition des résultats obtenus en utilisant la technique de la coupure est montrée à la figure 2.4. La défuzzification consiste à prendre le résultat de la superposition des règles pour en ressortir une valeur non floue. Trois méthodes sont couramment utilisées lors de la défuzzification : la moyenne des maximums, le centre de masse et la méthode des hauteurs. La plus précise est celle du centre de masse. Cette méthode consiste en la recherche du point

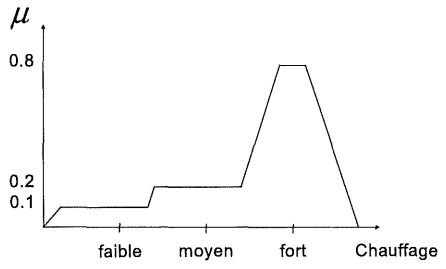


Figure 2.4 – Superpositions des résultats des règles floues

séparant l'aire sous la courbe de superposition en deux parties égales. La figure 2.5 montre la défuzzification de l'exemple étudié. Un inconvénient de cette méthode est qu'elle est très

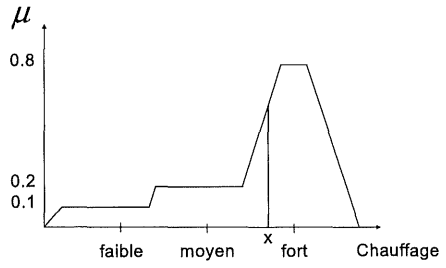


Figure 2.5 – Défuzzification avec la méthode du centre de masse

exigeante en temps de calcul, c'est pour cela que la méthode de la hauteur est préférée. Cette méthode consiste à pré-calculer le centre de masse de chaque étiquette. Ces centres de masse sont ensuite utilisés lors de la défuzzification afin de trouver la sortie du système. Cette méthode est souvent considérée comme une approximation de la méthode du centre de masse.

Les différentes étapes de la logique floue ont été décrites brièvement ce qui devrait permettre la compréhension des différents articles de la revue de la littérature. Les réseaux de neurones étant aussi très utilisés dans le traitement d'image, il est nécessaire d'en expliquer le fonctionnement.

2.2 Réseaux de neurones

Par analogie avec le cerveau humain, les chercheurs en intelligence artificielle ont élaboré un modèle de neurone artificiel inspiré des connaissances des neurobiologistes sur les neurones naturels. Un neurone artificiel est une unité de calcul munie de plusieurs entrées (dendrites) et d'une sortie (axone) comme présenté à la figure 2.6 . Chaque neurone a un état interne qui est

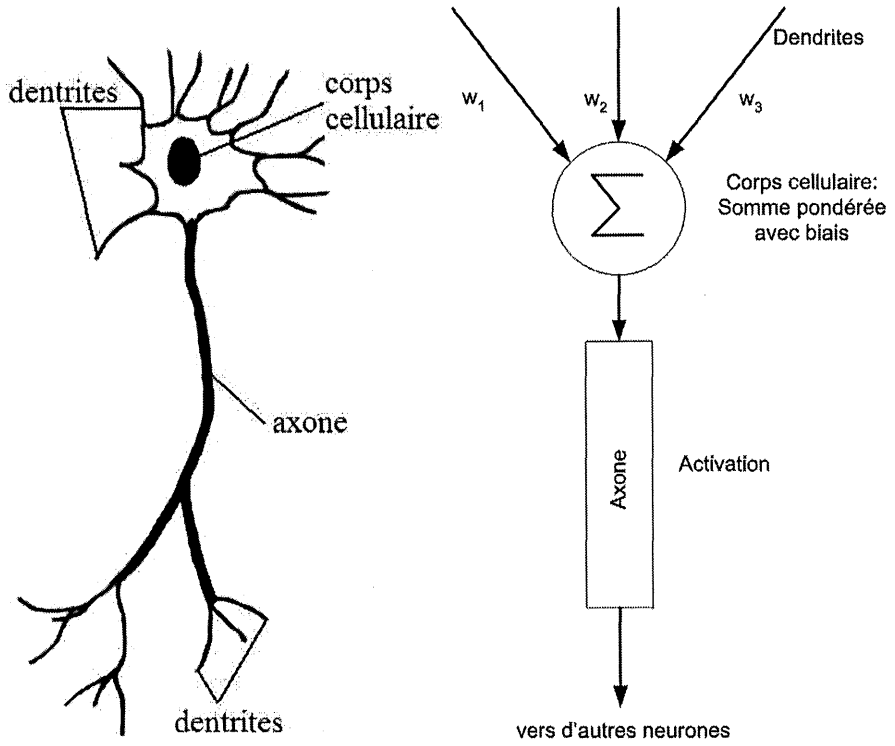


Figure 2.6 – Neurone biologique et son modèle mathématique (perceptron)

calculé comme la somme pondérée des valeurs d'entrée et possède une fonction d'activation. Cette fonction permet de décider l'amplitude du signal d'excitation qui sera propagé aux neurones suivants. Le type de fonction d'activation le plus utilisé est la sigmoïde. Ce calcul effectué, le neurone propage son nouvel état interne sur son axone.

Un réseau de neurones [Haykin, 1999a] est un ensemble de neurones connectés les uns aux autres par des liaisons axone-dendrites. Le réseau de neurones passe d'un état à un autre lorsque tous ses neurones ont recalculé leur état interne en fonction de leurs entrées. Ce processus peut être effectué séquentiellement ou parallèlement. L'ensemble des poids des liaisons

détermine le fonctionnement du réseau de neurones. Le réseau de neurones, sans connaissance au départ, réalise un apprentissage à partir d'exemples et devient, par modifications successives, un modèle rendant compte du comportement observé en fonction des variables descriptives. La construction du modèle est automatique et directe à partir des données. Elle ne nécessite pas d'intermédiaire, comme un expert. Les échantillons de l'ensemble d'entraînement sont présentés à la couche d'entrée du réseau de neurones. Lorsqu'un échantillon est appliqué au réseau, celui-ci cherche à atteindre un état stable. Lorsqu'il est atteint, les valeurs d'activation des neurones de sortie constituent le résultat.

Dans un réseau, chaque couche fait un traitement indépendant des autres et transmet le résultat de son analyse à la couche suivante. L'information donnée au réseau va donc se propager couche par couche, de la couche d'entrée à la couche de sortie, en passant soit par aucune couche, une ou plusieurs couches intermédiaires (dites couches cachées).

Un des algorithmes d'apprentissage les plus populaires fait une propagation de l'information à reculons (*back propagation*) lors de la phase d'apprentissage afin que le réseau puisse apprendre à donner les bonnes réponses. Un apprentissage est dit supervisé lorsque le réseau est obligé de converger vers un état final précis, tout en lui présentant un motif. C'est à lui de trouver l'état vers lequel il doit converger. À l'inverse, lors d'un apprentissage non supervisé, le réseau est laissé libre de converger vers n'importe quel état final lorsqu'un motif lui est présenté.

Les réseaux de neurones peuvent donc être classés en deux grandes catégories :

- Les réseaux à propagation avant (*Feed forward*), appelés aussi réseaux de type perceptron. Ce sont des réseaux dans lesquels l'information se propage de couche en couche sans retour en arrière possible.
- Les réseaux récurrents (*recurrent network*) sont des réseaux dans lesquels il y a un retour en arrière de l'information. Il y a des boucles, contrairement au *feed forward.*

Les types de réseaux de neurones diffèrent donc par plusieurs paramètres [Haykin, 1999a] :

- la topologie des connexions entre les neurones ;
- le type de neurones utilisé (perceptrons, *radial basis function*, etc.) ;
- l'algorithme d'apprentissage (*back propagation*, *Feed forward*, etc.).

Dans la revue de littérature deux types de réseaux de neurones se démarquent dans la problématique de recherche de la vitesse relative : les réseaux de Kohonen ou Self Organizing Map (SOM) et les réseaux d'Hopfield. Ce sont des réseaux de neurones fonctionnant avec un apprentissage non-supervisé. L'intérêt de ces types de réseaux sera discuté plus loin dans la revue de la littérature.

2.3 Réseaux d'Hopfield

Les réseaux de Hopfield [Haykin, 1999b] sont des réseaux récurrents entièrement connectés. Dans ce type de réseau, chaque neurone est connecté à tous les autres sauf lui-même et il n'y a aucune différenciation entre les neurones d'entrée et de sortie comme le montre la figure 2.7.

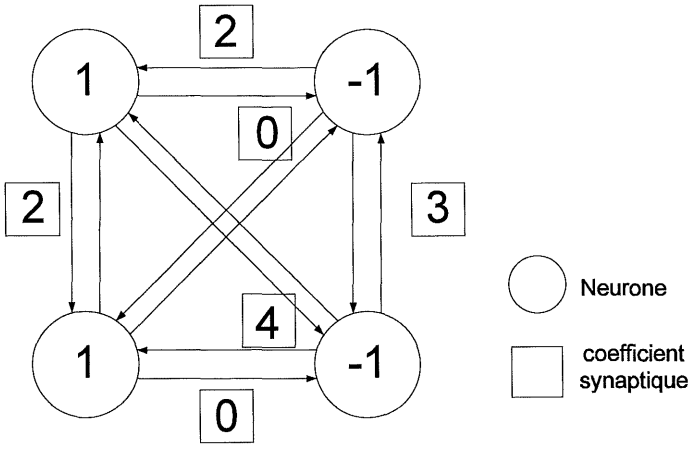


Figure 2.7 – Exemple d'un réseau d'Hopfield

Les connexions sont symétriques, c'est-à-dire que si un neurone i est connecté à un neurone j avec un coefficient synaptique de 2, le neurone j est connecté au neurone i avec un coefficient synaptique de même valeur. Les neurones sont binaires et peuvent prendre comme état +1 ou -1. Lorsqu'un stimulus est présenté au réseau, en phase d'apprentissage comme en phase de test, les neurones sont obligés de prendre les états correspondant aux valeurs du stimulus.

Ces réseaux fonctionnent comme une mémoire associative non-linéaire et sont capables de trouver des données stockées en fonction de représentations partielles ou bruitées. Un exemple

simple serait la reconnaisance de caractères manuscrits. Le rappel associatif se fait en minimisant une fonction d'énergie pour tomber dans l'un des attracteurs correspondant aux formes mémorisées. Dans une mémoire informatique classique, une information est retrouvée à partir d'une clé arbitraire. Par opposition, une donnée entreposée dans une mémoire associative est accessible à partir d'informations qui lui sont associées. La fonction d'une mémoire associative est de restituer une information en tenant compte de sa perturbation ou de son bruit. L'information doit alors se rapprocher d'une information apprise ou connue.

Le mode d'apprentissage du réseau d'Hopfield utilisé est le mode non-supervisé et suit la loi de Hebb :

$$w_{ij} = w_{ij} + (x_i * x_j) \tag{2.1}$$

où w représente le coefficient synaptique et x l'état du neurone.

L'efficacité de la synapse entre un neurone i et un neurone j est donc augmentée de 1 si ces deux neurones sont dans le même état (-1 * -1 = +1 , ou, 1 * 1 = +1), et diminuée de 1 si ces neurones sont dans des états différents (1 * -1 = -1 , ou, -1 * 1 = -1).

Les réseaux d'Hopfield opèrent de la manière suivante :
- les états du réseau sont fixés par un vecteur de valeur présenté au réseau ;
- les neurones calculent leurs sorties ;
- les sorties sont propagées entre les neurones. Le processus est répété jusqu'à ce que le réseau soit stabilisé, c'est à dire que les poids ne soient plus modifiés.

L'application principale des réseaux de Hopfield est l'entrepôt de connaissances mais aussi la résolution de problèmes d'optimisation. Ils peuvent aussi être utilisés comme algorithme de reconnaissance d'objet. Les propriétés des mémoires associatives permettent alors de reconnaître des objets fortement bruités parmi les objets appris lors de la phase d'apprentissage.

2.4 Réseaux de Kohonen

Les réseaux de Kohonen [Haykin, 1999c; Kohonen, 1982] sont composés de deux couches, une couche d'entrée et une couche de sortie (carte de Kohonen). Ce sont des réseaux à

apprentissage non-supervisé qui établissent une carte discrète, ordonnée topologiquement, en fonction des *patterns* d'entrée (figure 2.8). Le réseau forme ainsi une sorte de treillis dont chaque noeud est un neurone associé à un vecteur de poids. La correspondance entre chaque vecteur de poids est calculée pour chaque entrée. Par la suite, le vecteur de poids ayant la meilleure corrélation, ainsi que certains de ses voisins, vont être modifiés afin d'augmenter encore cette corrélation.

Il est utile pour la compréhension des réseaux de Kohonen de définir plusieurs termes qui seront utilisés :

Carte de Kohonen : ensemble de neurones disposés géométriquement en grille, souvent orthogonale. Tous les neurones de cette grille sont interconnectés et séparés de la même distance.

Fonction de voisinage ou fonction topologique de voisinage : cette fonction permet de déterminer le poids des neurones en fonction de la distance qui les sépare du neurone gagnant selon une fonction *chapeau mexicain* ou DOG (*Difference of Gaussians).*

Neurone gagnant : c'est le neurone dans la carte de Kohonen correspondant le mieux au motif en entrée du réseau.

Rayon d'influence : c'est le rayon du disque ayant pour centre le neurone gagnant. Seuls les neurones contenus dans ce disque subiront une modification de leurs poids.

Les prérequis nécessaires à l'algorithme sont :

- Un espace continu d'entrée des modèles d'activation générés suivant une distribution probabiliste.
- Une topologie du réseau de neurones sous la forme d'une grille de neurones (carte de Kohonen), définie comme un espace de sortie discret.
- Une fonction de voisinage $h_{j,i(x)}(n)$ dépendante du temps (n) définie autour du neurone gagnant $i(x)$, j représentant le neurone excité en entrée. Elle détermine la taille du rayon d'influence autour du neurone gagnant et suit deux critères :
    1. La fonction topologique de voisinage doit être symétrique et avoir son maximum défini pour le neurone gagnant.

2. L'amplitude de la fonction topologique de voisinage décroît inversement à la croissance de la distance latérale d_{ij}. d_{ij} est la distance qui sépare le neurone activé du neurone gagnant. $d^2_{j,i}$ est défini de la manière suivante :

$$d^2_{j,i} = \|r_j - r_i\|^2 \tag{2.2}$$

r_j et r_i représentent respectivement la position du neurone excité j et la position du neurone gagnant i. La fonction topologique est donc définie comme suit :

$$h_{j,i(x)} = exp(-\frac{d^2_{j,i}}{2\sigma^2(n)}) \tag{2.3}$$

avec

$$\sigma(n) = \sigma_0 exp(-\frac{n}{\tau_1}) \tag{2.4}$$

avec σ_0 la valeur de σ à l'initialisation de l'algorithme et τ_1 une constante de temps fixe.

– Un facteur d'apprentissage $\eta(n)$ qui est initialisé à une valeur η_0 et qui diminue graduellement en fonction du temps vers zéro sans jamais l'atteindre.

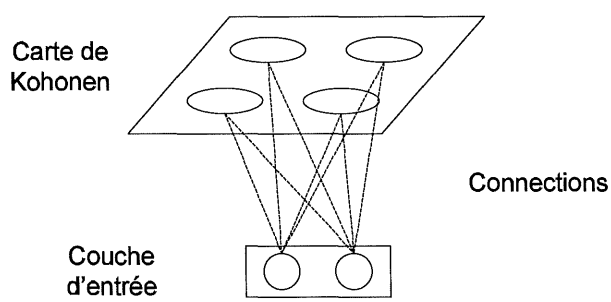


Figure 2.8 – Réseau de Kohonen

Les réseaux de Kohonen opèrent de la manière suivante :

Étape 1 : initialisation de la carte de Kohonen (le plus souvent à zero)

Étape 2 : sélection d'un neurone j de la couche d'entrée comme stimuli(le plus souvent àléatoirement)

Étape 3 : détermination du neurone gagnant i, c'est à dire celui qui se rapproche le plus du stimuli en entrée, en utilisant le critère de la distance euclidienne minimale :

$$i(x) = arg\ min\,||x(n) - w_j|| \tag{2.5}$$

x représente le vecteur des position des neurones.

Étape 4 : adaptation des neurones se situant dans le voisinage du neurone gagnant en utilisant la formule :

$$w_j(n+1) = w_j(n) + \eta(n)h_{j,i(x)}(n)(x(n) - w_j(n)) \tag{2.6}$$

avec

$$\eta(n) = \eta_0 exp(-\frac{n}{\tau_2}) \tag{2.7}$$

avec η_0, la valeur de η à l'initialisation de l'algorithme et τ_2 une constante de temps fixe.

Étape 5 : retourner à l'étape 2 jusqu'à ce qu'il n'y ait plus de changement notable dans la carte de Kohonen

Les réseaux de Kohonen sont principalement utilisés dans des applications de classification non-supervisées ou de réduction du nombre de dimensions d'un problème. L'exemple suivant montre l'entraînement d'un réseau de Kohonen pour générer des valeurs comprises entre 1 et 3. En entrée, les valeurs 1, 2 et 3 se succèdent donc à chaque itération lors de la phase d'entraînement.

La carte de Kohonen a une dimension de 4x4 neurones initialisés à 0. Les neurones sont distants de 1 orthogonalement les uns des autres. Les valeurs de η_0 et σ_0 sont égales respectivement à 0.2 et 0.6 et celles de τ_1 et τ_2 sont fixés respectivement à -3000 et 3000. La dixième itération au cours de l'apprentissage sera prise pour expliquer le fonctionnement.

À cette itération, la carte de Kohonen a déjà évoluée comme le montre la figure 2.9. Les valeurs réprésentent les poids de chaque neurones. La valeur 3 est sélectionné comme stimuli du réseau. La valeur qui dans la carte de Kohonen se rapproche le plus de 3 grâce à l'équation 2.5 est donc recherchée. Dans cet exemple, c'est le neurone ayant la valeur 2.6. Ce neurone

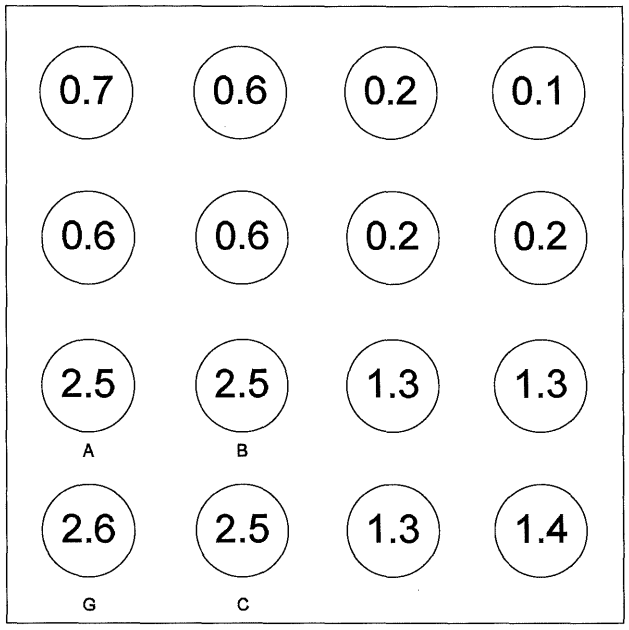


Figure 2.9 – Carte de Kohonen à la dixième itération

devient donc le neurone gagnant noté G. La valeur du neurone gagnant G se rapproche donc de la valeur 3 suivant l'équation 2.6 :

$$w_G(n+1) = 2.6 + 0.1993 * 0.2517 * (3 - 2.6) = 2.62$$

Il en va de même pour le neurone A et C.

$$w_A(n+1) = 2.5 + 0.1993 * 0.2517 * (3 - 2.5) = 2.5241$$

$$w_C(n+1) = 2.5 + 0.1993 * 0.2517 * (3 - 2.5) = 2.5241$$

w_B se rapproche aussi de la valeur 3 mais moins vite que A et C, car il est situé plus loin de G. En effet, $d^2_{G,B}$ est égale à 2. Cela modifie donc le résultat de la fonction de voisinage à la baisse.

$$w_B(n+1) = 2.5 + 0.1993 * 0.0633 * (3 - 2.5) = 2.51$$

À la fin du processus d'apprentissage, la carte de Kohonen sera composée de valeurs comprises entre 1 et 3. Chacune des valeurs (1, 2 et 3) sont alors sous la forme d'une zone incluant des valeurs proches respectivement de 1, 2 et 3. Chaque zone est alors considérée comme

une classe. Pour classer des nombres, il suffit alors de les présenter au réseau et de chercher le neurone gagnant. Selon la zone où le neurone gagnant se situe, le nombre en entrée sera dans la classe 1, 2 ou 3.

CHAPITRE 3

MÉTHODES POUR DÉTERMINER LA VITESSE RELATIVE

Ce troisième chapitre traite de la revue de littérature se rapportant au sujet de ce mémoire. Il se compose de quatre parties : la première partie traite des méthodes traditionnelles (par opposition aux méthodes dites d'intelligence artificielle) utilisée en aérospatiale pour faire un atterrissage autonome. La deuxième partie présente les méthodes d'intelligence artificielle permettant de simplifier une image, ce qui est nécessaire pour effectuer un bon *tracking*. La troisième partie expose les méthodes d'intelligence artificielle capable d'effectuer un *tracking* de cibles dans une séquence d'images. La dernière partie est une synthèse de cette revue de la littérature.

3.1 Méthodes traditionnelles

Les algorithmes les plus couramment utilisés pour effectuer le *tracking*, ou suivi de cible, sont basés sur la méthode Shi-Tomasi, du nom de leurs auteurs Shi et Tomasi [Shi et Tomasi, 1994]. Selon eux, la difficulté principale dans le *tracking* est la sélection de bonnes cibles dans l'image. Il faut, en effet, être sûr que les points sélectionnés de l'image soient des points marquants pour les retrouver facilement dans une suite d'image. Ils posent comme condition que, entre deux images successives fournies par la caméra, le déplacement des points sélectionnés est très faible et peut donc être représenté sous la forme d'une translation. Ils cherchent donc la translation permettant de superposer au mieux la fenêtre des pixels au voisinage du point d'intérêt dans l'image précédente avec celle dans l'image courante. L'intérêt de cette méthode est qu'elle détecte et prévient lorsqu'un point ne peut plus être suivi : les deux fenêtres n'ont pas pu être superposées correctement. Cette méthode permet de contrôler, pendant l'exécution de l'algorithme, la validité des points sélectionnés. La définition d'une

bonne cible n'est donc plus intuitive, elle se base sur un calcul concret. Une bonne cible est tout simplement une cible facile à suivre. Le problème de cette méthode, et de toutes celles qui en découlent, est que cet algorithme ne fonctionne bien que pour de petits déplacements et qu'il faut contrôler si les points suivis sont encore dans les images ou non. La méthode Shi-Tomasi est cependant très utilisée, comme dans l'article de Johnson et Matthies [Johnson et Matthies, 1999]. Cet article décrit une méthode pour faire un atterrissage de façon autonome sur de petits astres comme des satellites ou des astéroïdes au moyen de deux caméras et d'un altimètre laser. Cette technique est basée sur un *tracking* automatique de cible entre des paires d'images, suivi d'une estimation du mouvement et d'un calcul de la différence de hauteur de prise de vue entre les deux images. Pour cela, l'algorithme Shi-Tomasi légèrement modifié par Benedetti et Perona [Benedetti et Perona, 1998] est utilisé ; une estimation de mouvement est ajoutée ce qui permet de faire fonctionner l'algorithme en temps réel.

La méthode utilisée par Johnson et Matthies se décompose en quatre étapes :

Étape 1 : Traitement de deux images successives avec leur altitude respective. Ce traitement permet de détecter des cibles à suivre dans les deux images. Une bonne cible est une cible qui a de fortes variations de texture dans toutes les directions. Tous les points de l'image ne seront pas traités. Dès qu'il y a assez de points à suivre, l'algorithme s'arrête. Cela permet de garantir une assez bonne vitesse de traitement.

Étape 2 : Les cibles trouvées sont comparées dans les deux images. C'est le *tracking*. La méthode Shi-Tomasi est utilisée, car elle est très bien adaptée à de petits déplacements ce qui est le cas dans cet article.

Étape 3 : Cette étape estime le mouvement à l'aide du suivi de cible. Cela permet d'estimer le déplacement des cibles entre deux images, limitant ainsi la surface à traiter pour rechercher une cible pour les images suivantes. L'estimation dans la méthode de Johnson et Matthias se fait en deux phases :

1. Une estimation linéaire du déplacement développée par Weng [Weng et coll., 1993]. Cette étape permet d'avoir une estimation approchée du déplacement et aussi d'éliminer les mauvaises cibles.

2. Un algorithme non linéaire qui permet d'affiner les résultats de la première étape en minimisant l'erreur qui est une fonction des paramètres de mouvement.

Étape 4 : Cette dernière étape consiste à déterminer la différence de hauteur du vaisseau entre les deux prises de vue. Pour cela, l'altimètre laser est utilisé. Avec l'estimation de l'altitude prise par l'altimètre laser, le déplacement total effectué est déduit.

La méthode utilisée par Johnson et Matthies obtient de bons résultats et englobe une grande partie des techniques déjà développées pour un atterrissage autonome. Cependant l'utilisation de plusieurs appareils (altimètre laser et cameras) la rend plus sujette aux pannes matérielles et introduit un coût élevé. De plus, la méthode utilisée doit calculer sur plusieurs images les cibles qui sont bonnes à suivre ce qui peut engendrer un manque de stabilité. Finalement, elle est plus adaptée aux petits déplacements correspondant à une approche finale d'atterrissage.

Une autre méthode pour déterminer la vitesse est donnée par Misu et coll. [Misu et coll., 1999]. Cependant les différentes étapes de la méthode sont très similaires à la méthode de Johnson et Matthies. La différence majeure est qu'une méthode de corrélation est utilisée pour effectuer le *tracking* au lieu de la méthode Shi-Tomasi. Cela permet d'avoir de plus grands déplacements entre deux images successives, mais cette méthode utilise des transformées de Fourrier, ce qui s'avère lourd en calculs. Même si la rapidité de calcul n'est pas le premier critère de ces travaux de maîtrise, il faut rappeler que les puissances embarquées en aérospatiale sont limitées. Cette méthode n'est donc pas retenue pour ce projet.

Comme décrit dans l'introduction, le but de ce projet et de faire du *tracking* en utilisant des méthodes dites d'intelligence artificielle pour augmenter la fiabilité et la stabilité des méthodes d'estimation de mouvement tout en essayant de ne pas augmenter les temps de calcul. La méthode de Johnson et Matthies donnent les bases de l'algorithme développé dans ce projet de maîtrise. En effet, il semble intéressant de suivre les deux principales étapes que sont la simplification et la détection de cible, puis le *tracking*.

3.2 Méthodes de simplification d'image

Cette section traite des méthodes d'intelligence artificielle reliées au prétraitement des données. Le prétraitement des données permet de limiter le nombre de données à traiter pour le *tracking*. C'est un élément très important de la méthode proposée dans ce mémoire. En effet, un mauvais prétraitement peut engendrer trop de bruit ou un manque de détails générant de très mauvais résultats pour *tracking*. Les méthodes de classification ou de détermination de contours peuvent donner des pistes de réflexion en ce qui concerne le prétraitement des données du LIDAR, comme l'élimination de points ou la détection de formes à suivre. Cela permet de limiter le plus possible les informations nécessaires pour effectuer le *tracking*.

La logique floue semble être très efficace en ce qui concerne la détection de contours ou l'extraction de contours. La démonstration à la fin de l'article de Garcia-Barrosso et coll. [Garcia-Barroso et coll., 1995] est très convaincante. Ils définissent un contour comme une discontinuité locale de la fonction d'illumination. Leur algorithme se compose de 4 étapes :

1. obtenir le vecteur gradient pour chaque pixel de l'image ;
2. obtenir le CPL (*Contour Path Location*) ;
3. obtenir le MD (*Maximum Discontinuity*) ;
4. l'extraction de contours.

La deuxième et la troisième étape sont effectuées en parallèle en utilisant la logique floue. L'obtention des vecteurs gradients se fait grâce à la combinaison de trois convolutions (3x3, 4x4, 2x2) générant ainsi une carte dense de gradient de vecteur. La seconde étape consiste à déterminer la forme du contour avec le CPL. Le CPL dépend de l'homogénéité de l'argument du vecteur gradient en fonction des pixels voisins. Pour cela, quatre classes sont choisies :

a. Les contours droits, les arguments des vecteurs gradients sont tous constants.
b. Les contours courbes, les arguments des vecteurs gradients varient continuellement le long du contour.
c. Les points, il y a une discontinuité dans les arguments des vecteurs gradients sur les deux côtés du contour.

d. Pas de contour, les valeurs des arguments du vecteur gradient varient aléatoirement de 0 à 360.

La troisième étape consiste à trouver le MD. Pour cela, le maximum de la variation de la fonction d'illumination est recherché. Ce maximum correspond à un contour. Le calcul du degré d'appartenance à un contour de chaque pixel est régi par 2 règles :

- Un haut degré d'appartenance à un contour pour un pixel correspond à un maximum de variation de la fonction d'illumination proche du centre du pixel. Comme un contour contient plusieurs pixels, la première règle peut être modifiée : c'est la deuxième règle.
- Le degré d'appartenance du pixel sélectionné augmente quand le pixel a deux voisins qui n'ont pas de voisins eux-mêmes et qui ont une grande valeur à la première règle.

Finalement, l'extraction de contours de la quatrième étape se fait de la manière suivante :

- le pixel doit être situé dans le voisinage d'un contour.
- le pixel est situé dans une surface où se trouve le maximum de la fonction de variation de l'illumination dans la direction du gradient.

L'une des difficultés de cette méthode, relative à ce projet, est que les gradients peuvent être discontinus avec le LIDAR. Néanmoins, cette méthode semble très efficace.

Miosso et Bauchspiess [Miosso et Bauchspiess, 2001] montrent aussi une autre méthode pour faire de l'extraction de contours. Ils appliquent trois filtres à l'image :

- un filtre passe-haut ;
- un filtre de détection de contour du premier ordre ;
- un filtre passe-bas.

Le degré de gris est connu pour chaque pixel. Le traitement consiste à pondérer le niveau de gris en fonction des pixels qui l'entourent. Ainsi, en fonction des pixels autour du pixel étudié, le niveau de gris sera rehaussé ou réduit. Ils utilisent alors la logique floue sur les données fournies par les trois filtres pour déterminer les contours. Les résultats sont de très bonne qualité, la logique floue est donc performante en ce qui concerne le prétraitement des données. Cependant l'obtention des règles reste un obstacle, car une image est composée de beaucoup d'informations très variées.

Une autre méthode permettant de traiter une image est la classification grâce à la logique

floue. La classification floue ne cherche plus à détecter les contours comme Garcia-Barrosso et coll. ou Miosso et Bauchspiess mais à recomposer l'image d'origine avec des éléments connus. La classification floue permet de classer des éléments souvent bruités, comme un signal audio ou une image en fonction de règles. La méthode est la suivante : différentes classes représentant les cas les plus probables ou les sorties désirées d'un système sont définies. Les règles d'inférence sont écrites de telle façon qu'elles correspondent toujours à une classe de sortie. La classe ayant la valeur défuzzifiée la plus grande est celle qui ressemble le plus au signal d'entrée [Wei et Mendel, 1999]. Cette méthode, comme d'autres [Suzuki et coll., 2001], permet en fait d'éliminer le bruit et d'obtenir ainsi des données simples à traiter.

Wei et Mendel [Wei et Mendel, 1999] tentent d'identifier de cette manière le type de modulation utilisée au travers d'un canal de transmission bruitée par classification. L'avantage de cette méthode, pour ce projet de maîtrise, est la décomposition en éléments connus d'un signal. En effet, les auteurs arrivent à classer des constellations bruitées dans différentes classes prédéfinies, ce qui a pour effet d'éliminer le bruit. Chaque point composant la constellation est fuzzifié. Ils utilisent des ensembles flous en 2D ce qui implique que les fonctions d'appartenances sont définies dans un domaine complexe. La valeur de chaque fonction d'appartenance dépend de la distance du point fuzzifié avec le centre de l'ensemble flou. La décision floue de tous les systèmes flous (FLS : Fuzzy Logic system) est alors combinée avec une opération floue d'intersection pour former une décision floue finale. Cette décision est ensuite défuzzifiée pour avoir le résultat réel (figure 3.1).

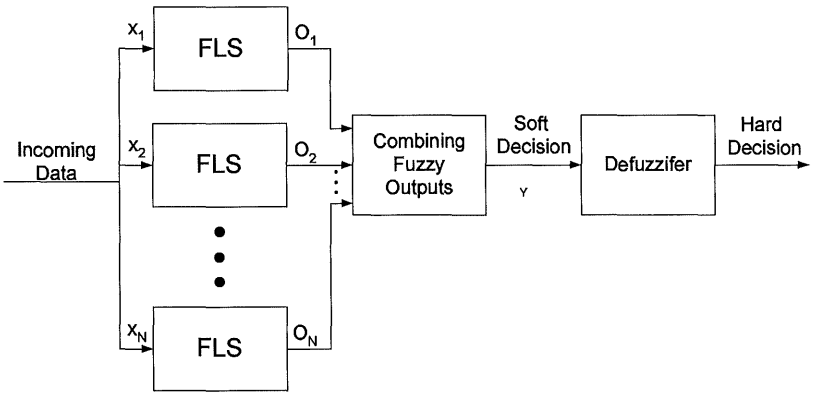


Figure 3.1 – Architecture de l'algorithme de Wei et Mendel

L'inconvénient de cette méthode est qu'il faut avoir des données d'entraînement pour générer les règles d'inférence, ce qui s'avère délicat dans ce projet de maîtrise. Malgré cela, la classification floue semble comporter beaucoup d'avantages, comme la réduction d'une image en éléments simples.

Les réseaux de neurones sont aussi utilisés dans la littérature pour simplifier des données. Ils peuvent effectuer de l'extraction de contours avec des informations supplémentaires provenant d'une base de connaissances. C'est notamment le cas des articles de Liu et coll. [Liu et coll., 1998a; Liu et coll., 1998b] qui décrivent une méthode pour extraire des données hydrographiques à partir d'images satellites, comme des fleuves et des lacs. Cette méthode permet de séparer l'eau des autres éléments de l'image à l'aide de deux réseaux de neurones. Ils utilisent un *two layer neural network.* Le premier réseau est récurrent et le second est un réseau de type LEGION (Locally Excitatory Globally Inhibitory Oscillator Network). Il a été démontré que le LEGION est très performant quant à la synchronisation à travers les oscillations d'une région et la désynchronisation des oscillations de régions différentes. Ces oscillations peuvent correspondre à une intensité ou une texture dans l'image. Il permet donc de séparer deux régions même s'il y a du bruit. Pour garder une certaine précision, cette méthode utilise une information contextuelle. L'idée est d'incorporer de l'information supplémentaire pour détecter les bordures complètes des bordures incomplètes quand il y a beaucoup de bruit dans l'image. Cet article démontre la grande robustesse des réseaux de neurones au bruit ce qui s'avère être un avantage dans ce projet, mais la nécessité des données d'entraînement et surtout d'une base de données contextuelles est un handicap.

Certains réseaux de neurones sont basés sur la similitude avec l'homme pour analyser une image. Ces réseaux sont utilisés, par exemple, pour déterminer les contours ou séparer les différents objets d'une image. Plusieurs théories s'opposent pour imiter l'intelligence humaine : soit l'étude porte sur le fonctionnement du cerveau humain, soit elle porte sur les méthodes analytiques qui sont différentes dans leur fonctionnement. Sadja et Finkel [Sadja et Finkel, 1992a; Sadja et Finkel, 1992b] réussissent, à l'aide d'un réseau de neurones, à simuler la segmentation faite d'une image par le cerveau humain. Ils utilisent les relations d'occlusions et de profondeurs entre les objets. Ils font apprendre au réseau de neurones la

relation qu'il y a entre l'intérieur d'une figure et l'extérieur, comme le fait le cerveau humain. Yen et Finkel [Yen et Finkel, 1997] décrivent une méthode pour faire de l'extraction de contours, toujours en utilisant une approche humaine. Chaque neurone fonctionne selon la méthode de *Linear quadrature steerable filter pyramids* [Adelson et coll., 1984]. C'est cet algorithme qui ressemble le plus au fonctionnement du cerveau humain. Le principal problème de ces méthodes est qu'elles semblent assez complexes à mettre en oeuvre : il parait difficile de simuler le fonctionnement du cerveau humain. Les résultats obtenus sont néanmoins très convaincants pour les deux méthodes et il n'est plus nécessaire d'avoir d'informations contextuelles. Il reste donc à s'affranchir du problème des données d'entraînement.

Les réseaux de neurones utilisant les architectures de type Hopfield [Haykin, 1999b] sont très courants. En effet, de nombreux réseaux de neurones utilisent ces types de réseaux pour leurs constitutions. Lu et Shen [Lu et Shen, 1996] utilisent les BP-net (Back Propagation Net) et un réseau de neurones d'Hopfield pour faire de l'extraction de contours. Le BP-net organise l'image en la classifiant en 8 classes. Le réseau de neurones Hopfield gomme les erreurs et rehausse les contours. Leur algorithme permet de rehausser les frontières, de retrouver les contours oubliés et d'éliminer les faux contours générés par le bruit. Cette méthode semble très performante à la vue des résultats, mais la classification ne semble pas pouvoir être appliquée à ce projet du fait de la discontinuité des gradients avec le LIDAR.

L'article de Evangelou et coll. [Evangelou et coll., 2001] montre l'utilisation d'un réseau de Kohonen pour effectuer une segmentation par *clustering* d'images satellitaires. La méthode décrite s'appliquait originalement à la segmentation automatique d'images du cerveau humain [Evangelou, 1999]. Le processus de compétition aboutit à ce que les vecteurs similaires en entrée soient regroupés ensemble dans la carte de Kohonen. Le rayon d'action du neurone gagnant est diminué à chaque itération du processus pour stabiliser l'algorithme. Le résultat est un centre de gravité pour chaque groupe de la carte de Kohonen qui minimise l'erreur de quantification. Cette erreur est définie comme la racine carrée de la somme des différences entre les entrées et les centres de gravité calculés. Cet article qui se base sur la méthode de Kohonen [Kohonen, 1982] montre les trois principaux avantages de cette méthode pour ce projet :

- Le fait que l'apprentissage soit non supervisé permet d'être indépendant au type d'images présentées aux réseaux.
- Ce type de réseau est robuste au bruit.
- Contrairement aux autres méthodes, ce réseau ne nécessite pas de données d'entraînement.

Si la partie de prétraitement des données du LIDAR décrite dans ce mémoire n'est pas vraiment basée sur la méthode de Evangelou et coll., ses avantages ont incité l'utilisation des réseaux de Kohonen pour cette partie du projet.

3.3 Méthodes pour effectuer le *tracking*

À priori, la logique floue ne semble pas posséder toutes les qualités pour faire du *tracking*, cependant H. Borotschnig [Borotschnig et coll., 1996] a réussi à appliquer du *tracking* sur des formes géométriques simples : cube, cylindre, lignes parallèles et perpendiculaires. Il utilise pour cela le *fuzzy graph tracking*. Le principe est de transcrire une image à l'aide d'un graphe. Les noeuds du graphe représentent les différents segments ou droites de l'image. Les liaisons entre les noeuds correspondent à la position entre les différents éléments de l'image, à savoir leur orientation respective : perpendiculaires ou parallèles à des degrés différents. C'est le seul algorithme trouvé qui n'utilise que la logique floue pour faire du *tracking*, et même s'il n'a pas les qualités requises pour ce projet, l'agencement des liaisons entre les différents objets de l'image en graphe semble être une idée intéressante.

D'autres chercheurs utilisent la logique floue et une base de connaissance pour faire du *tracking*. Agouris et coll. [Agouris et coll., 1998] emploient une méthode qui permet de faire de l'extraction et du *tracking* d'objets au moyen des informations pré-existantes et de la logique floue. Les informations sont sous forme de photos aériennes et de cartes topographiques. Ils font de l'identification de cible, ce qui permet d'améliorer l'interprétation et la classification des images mais aussi de faire du *tracking* en déterminant précisément le centre ou la bordure des objets. Ils sont partis du principe qu'une base de données sera toujours plus performante qu'un algorithme pour déterminer un objet, mais pour la comparaison, en elle-même, la logique floue demeure très performante. Il y a donc deux opérateurs : le premier est une base de connaissances, le second un opérateur logique floue. La logique floue permet, grâce aux

informations de la base de connaissance, d'extraire les contours d'un objet à suivre. La base de connaissances permet d'avoir des informations vagues sur un objet de la carte. L'exemple donné est celui de l'extraction d'une route dans une image satellite. La base de connaissance détermine la forme générale d'une route ; la logique floue permet d'analyser point par point l'image pour extraire avec précision les contours de la route, tout en étant guidée par les informations de la base de connaissance. Cette technique permet par la suite d'améliorer le *tracking* de la route dans les images successives. La principale limite de cette méthode dans le cadre de ce projet de recherche est, comme déjà mentionnée, le manque d'informations *a priori* pour créer une base de connaissances.

Les réseaux de neurones ont de grandes capacités de *tracking* ou de reconnaissance d'objets. Cependant, en grande majorité, ces réseaux sont entraînés pour suivre un seul objet. Ils utilisent pour cela des exemples de ces objets lors de la phase d'apprentissage. Zhao et Thorpe [Zhao et Thorpe, 2000] décrivent une méthode permettant de reconnaître des piétons dans la rue. Leur algorithme est divisé en deux parties. La première opère une segmentation de l'image en premier plan et en arrière-plan. La deuxième, grâce à un réseau de neurones, fait la reconnaissance des formes du premier plan et détermine s'il s'agit ou non de piétons. Ils utilisent deux caméras montées en stéréo et l'image est traitée de façon à avoir des ombres bien distinctes. En entrée du réseau, les auteurs utilisent le gradient de l'intensité de l'image pour encoder les informations sur les formes des objets. Chaque région de l'image est alors divisée en section de taille 30x65. Le réseau de neurones travaille donc sur une image simplifiée. L'algorithme d'apprentissage est de type retropropagation. En sortie, le réseau indique s'il s'agit d'un piéton ou non. Les résultats de cette méthode sont concluants puisqu'ils obtiennent en situation réelle 85.2 % de bonnes réponses. Les avantages de cette méthode sont sa rapidité et sa relative fiabilité. L'exemple de la détection de piétons montre que l'algorithme fonctionne en temps réel sur un Pentium 2 à 450 Mhz. Néanmoins, un problème majeur persiste dans ces méthodes : il faut avoir beaucoup d'exemples de l'objet à suivre. Dans le cas d'un atterrisage sur un corps planétaire, il n'y a pas d'objet connu à suivre. Il n'y a donc pas d'objet à faire apprendre au réseau.

Une autre méthode développé par Cirrincione et Cirrincione [Cirrincione et Cirrincione, 2003]

semble très intéressante : l'utilisation d'un réseau de Kohonen pour faire de l'estimation de mouvement par partie. En effet, ils utilisent un réseau de Kohonen pour faire du *clustering* des points qui se déplacent de la même manière. Cela revient à dire que les points de l'image qui se déplacent de la même façon sont regroupés côte à côte dans la carte de Kohonen. Ainsi par le principe de compétition, le neurone gagnant, qui est le neurone ressemblant le plus à l'entrée présentée au réseau, attire les points qui lui ressemblent. Une fois tous les points présentés au réseau, la carte de Kohonen regroupe les points similaires entre eux. Les résultats de cette méthode sont très convaincants. Cette technique montre une fois de plus l'avantage pour ce projet des réseaux de Kohonen, car il n'y a pas besoin de données d'entraînement. L'entraînement correspond en fait au regroupement des points similaires et est refait pour chaque image présentée. C'est un grand avantage, car entraîner un réseau de neurones avec des données artificielles ne garantit pas une stabilité parfaite avec des images réelles. Cependant dans ce projet, il n'est pas nécessaire d'avoir de la segmentation, le mouvement d'une image à l'autre étant uniforme. Une méthode effectuant juste du *tracking* serait suffisante.

Les travaux de Labonté [LabontÉ, 1998] réalisent du *tracking* dans un flux de particules. Deux réseaux de Kohonen interconnectés sont utilisés. Les images successives sont prises dans un intervalle de temps très court. Les distances entre deux images consécutives sont donc assez petites. À l'aide de la reconnaissance des particules de la première image dans la deuxième, il peut en déduire leur vitesse. La carte de Kohonen du premier sous-réseau est initialisée avec les coordonnées des particules de la première image, la carte de Kohonen du second sous-réseau est initialisée avec les coordonnées des particules de la seconde image. Les deux cartes de Kohonen servent d'entrée aux deux réseaux de neurones. Le processus de compétition entre les neurones permet de faire converger les neurones communs aux deux images vers la même position dans les deux cartes de Kohonen. Les neurones ayant le même poids correspondent à des neurones représentant la même particule. Il est à noter qu'au contraire d'un réseau de Kohonen classique, le rayon d'influence du neurone gagnant augmente avec le temps. Les résultats apparaissent très encourageants. Le réseau de neurones arrive assez bien à suivre chaque particule même quand le flux n'est pas constant. Dans le cas d'un flux uniforme, les résultats sont même excellents.

La méthode de Labonté semble performante. Les réseaux Kohonen sont très efficaces pour faire du *tracking*. Cependant, il reste un dernier point à éclaircir : l'auteur n'aborde pas la distance maximale permise entre deux images consécutives, un point clé pour ce projet.

3.4 Synthèse des différentes techniques

3.4.1 Prétraitement des données

En ce qui concerne le prétraitement des données du LIDAR, aucune technique ne semble avoir toutes les qualités requises. Les réseaux de neurones ont montré de grandes capacités de segmentation ou d'extraction de contours des images [Liu et coll., 1998a; Liu et coll., 1998b]. Leur vitesse de traitement semble rapide et ils sont robustes au bruit ce qui est un avantage dans ce projet. Cependant, ils nécessitent tous des informations *a priori* pour leur entraînement. Une autre méthode utilisant des réseaux d'Hopfield [Lu et Shen, 1996] semble être plus appropriée, leur apprentissage étant non supervisé. Cependant, leur adéquation avec un traitement efficace du LIDAR n'est pas évidente. La logique floue quant à elle semble mieux armée pour ce type de projet [Garcia-Barroso et coll., 1995; Miosso et Bauchspiess, 2001; Wei et Mendel, 1999] en particulier la classification floue, car il n'est pas nécessaire d'avoir de données d'entraînement pour faire une classification simple des gradients fournis par le LIDAR. De plus, la classification limite le bruit généré par celui-ci. Cependant, la discontinuité des gradients du LIDAR engendre de grandes difficultés : que faire quand d'une image à l'autre une infime variation fait passer par exemple un gradient d'une classe à une autre ? Il est très difficile d'avoir des règles d'inférence couvrant tous les cas possibles. C'est la difficulté majeure dans ce projet. La logique floue n'a effectivement pas besoin de données d'entraînement mais de règles écrites par un expert. Le problème résulte de la démultiplication de ces règles, leur nombre affectant d'autant la complexité de la conception.

Finalement la méthode d'Evangelou fournie l'idée de base pour ce prétraitement de données, car il ne nécessite ni de données d'entraînement, ni de règles à écrire. Les réseaux de Kohonen paraissent donc la technique la plus appropriée pour ce projet.

3.4.2 *Tracking*

L'état de l'art montre différentes techniques en ce qui concerne le *tracking*. Le premier point à relever est qu'aucune des méthodes trouvées jusqu'à maintenant n'utilise de LIDAR ou d'équivalent, excepté l'article de Johnson et Matthies [Johnson et Matthies, 1999] qui utilise un laser pour l'altitude. Le deuxième point à relever est que le cas de l'atterrissage d'une sonde sur une planète n'a pas été rencontré, même si beaucoup de méthodes de *tracking* utilisent de la logique floue ou des réseaux de neurones pour des situations précises. Si la méthode de Johnson et Matthies semble performante, elle trouve sa limite par la distance maximum entre deux images. Les autres méthodes dites intelligentes ont l'avantage d'être plus simples, certaines sont même explicitement rapides [Zhao et Thorpe, 2000]. Il semble qu'à elle seule, la logique floue ne puisse effectuer du *tracking*. En effet, à l'exception de l'article de Borotschnig et coll. [Borotschnig et coll., 1996], la logique floue n'est jamais utilisée seule. En effet, le *fuzzy graph tracking* semble très efficace, mais ne fonctionne que sur des images comportant des objets assez simples. Il parait impossible, au premier abord, de voir sur une image d'un relief quelconque, des parallèles et des perpendiculaires. La technique utilisée dans l'article de d'Agouris et coll. [Agouris et coll., 1998] semble plus prometteuse. Elle semble efficace pour faire du *tracking* ce qui pourrait être appliqué à un atterrissage autonome. Cependant, le fait qu'il faille avoir des informations sur les images à analyser est un handicap : en effet, par défaut, il n'y a que très peu d'informations sur le sol des corps planétaires du système solaire.

Finalement, pour les même raisons que dans le prétraitement des données, les réseaux de Kohonen sont la solution préférée. Labonté [LabontÉ, 1998] décrit une méthode qui semble très adaptée à ce sujet de maîtrise : en effet elle ne nécessite aucune information extérieure. D'ailleurs la partie *traking* de ce projet de maîtrise s'inspire grandement de cette méthode qui sera détaillée dans la partie conception de ce mémoire.

DEUXIÈME PARTIE

CONCEPTION

CHAPITRE 4

RÉALISATION

4.1 Description générale de l'algorithme ME-SOM

4.1.1 Contexte du projet

L'entrée de l'algorithme est composée de deux cartes consécutives représentant les mesures prises par le LIDAR. La valeur de ces mesures est en fait la distance séparant le LIDAR du sol. Dans ce document, le terme point d'élévation sera utilisé pour définir cette distance. Toutes les secondes, le LIDAR mesure 10 000 points d'élévation. Les images fournies sont corrigées de deux manières :

- L'angle d'incidence est annulé, toutes les images sont corrigées de façon à être perpendiculaires au vecteur gravitation.
- La rotation entre deux images est corrigée, la translation est le seul mouvement possible entre deux images.
- Le nuage de points est corrigé de manière à avoir toujours une surface balayée d'environ 100x100 unités de mesure.

De plus, la translation maximale entre deux images est fixée à 50 % de la taille d'une image.

4.1.2 Description de l'algorithme

L'algorithme proposé est composé de cinq modules principaux, comme présenté à la figure 4.1. Les mesures corrigées prises par le LIDAR représentent l'entrée de l'algorithme. La sortie de l'algorithme est la vitesse relative de la sonde par rapport au sol. Le temps t est de l'ordre de la seconde. Cet algorithme effectuant une estimation du mouvement, il a été nommé ME-SOM pour *Motion Estimation with Self Organizing Map.*

Les différents modules composant ce projet ont tous été entièrement codés dans Mat-

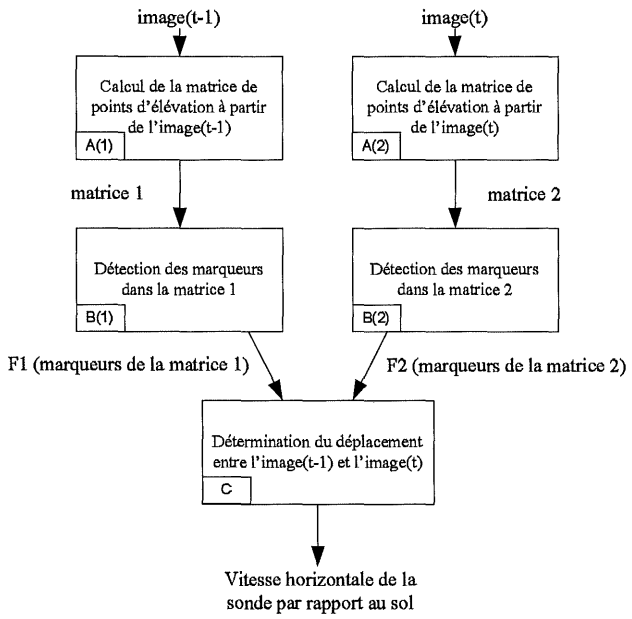


Figure 4.1 – Description générale de l'algorithme ME-SOM

lab [MATLAB, 2004]. Ce logiciel a été préféré pour son aptitude à travailler facilement avec des matrices. Dans la figure 4.1, les blocs A(1) et A(2) transforment les deux images successives composées de 10 000 mesures chacune en deux matrices de taille s'approchant de 35x35. Ces matrices sont appelées respectivement matrice 1 et matrice 2 pour l'image(t-1) et l'image(t). Il est à noter que les matrices 1 et 2 peuvent voir leur taille varier en fonction des mesures prises par le LIDAR. En effet, si le balayage du LIDAR n'est pas de forme carrée, les dimensions de la matrice 1 et 2 en seront affectées. Cependant, il y aura toujours une des deux dimensions proche de la valeur 35. Cette étape sera appelée le lissage des données. Dans les seconds blocs B(1) et B(2), les matrices 1 et 2 sont transformées à l'aide principalement d'un réseau de Kohonen pour faire ressortir des points marquants de l'image, ces points sont appelés marqueurs. Tous les marqueurs de la matrice 1 et 2 sont stockés respectivement dans $F1$ et $F2$ sous forme de vecteur colonne. Pour chaque marqueur, seule la position (x,y) dans la matrice (1 ou 2) est conservée. Le plan relatif à chaque matrice est défini de la manière suivante :

- Les coordonnées des points représentent en fait les indices de la matrice (1 ou 2)
- L'origine du plan est définie au point de coordonnées (1,1) au lieu de (0,0). Cela est dû à

la gestion des matrices dans le logiciel Matlab.

Dans tout le document, les coordonnées d'un point sont toujours exprimées dans le référentiel de la matrice contenant ce point. Il n'y a donc jamais de changement de repère ou de référentiel.

Le bloc C traite à l'aide d'un réseau de Kohonen les marqueurs ($F1$ et $F2$) détectés dans les deux images successives pour faire correspondre les marqueurs semblables dans les deux matrices 1 et 2. Ainsi, cela permet de déduire le déplacement effectué par la sonde d'une image à l'autre et donc sa vitesse relative. Dans la suite du document, les explications de la méthode proposée se limiteront aux blocs A(1), B(1) et C. En effet, les blocs A(2) et B(2), sont identiques, ils traitent uniquement une image différente.

4.2 Lissage des données du LIDAR

Cette partie montre le fonctionnement du prétraitement des données du LIDAR correspondant au bloc A dans la figure 4.1. Les données du LIDAR sont sous la forme d'un nuage de 10 000 points d'élévation répartis sur une surface de forme rectangulaire.

Comme le montre la figure 4.2, le balayage du LIDAR n'est pas régulier et présente plusieurs imperfections. Il faut donc réussir à traiter les données pour avoir une information utilisable. La méthode choisie est simple. Une fenêtre F balaye le nuage de points pour générer la matrice 1. À chaque déplacement ΔF de la fenêtre, la moyenne des points contenus dans la fenêtre est calculée et génère un élément de la matrice 1 à la position correspondant à l'emplacement de la fenêtre. La taille de la fenêtre F est fixée par le concepteur. L'algorithme est composé de 5 étapes suivantes :

Étape A_1 : Le nombre total de points mesurés par le LIDAR est déterminé. Le LIDAR est capable de prendre 10 000 points d'élévation en une seconde et cela de façon proportionnelle au temps : en une seconde, il prend 10 000 points, en une demi-seconde 5 000, en un quart de seconde 2 500, etc. Il faut donc s'assurer du nombre réel de point pris par le LIDAR.

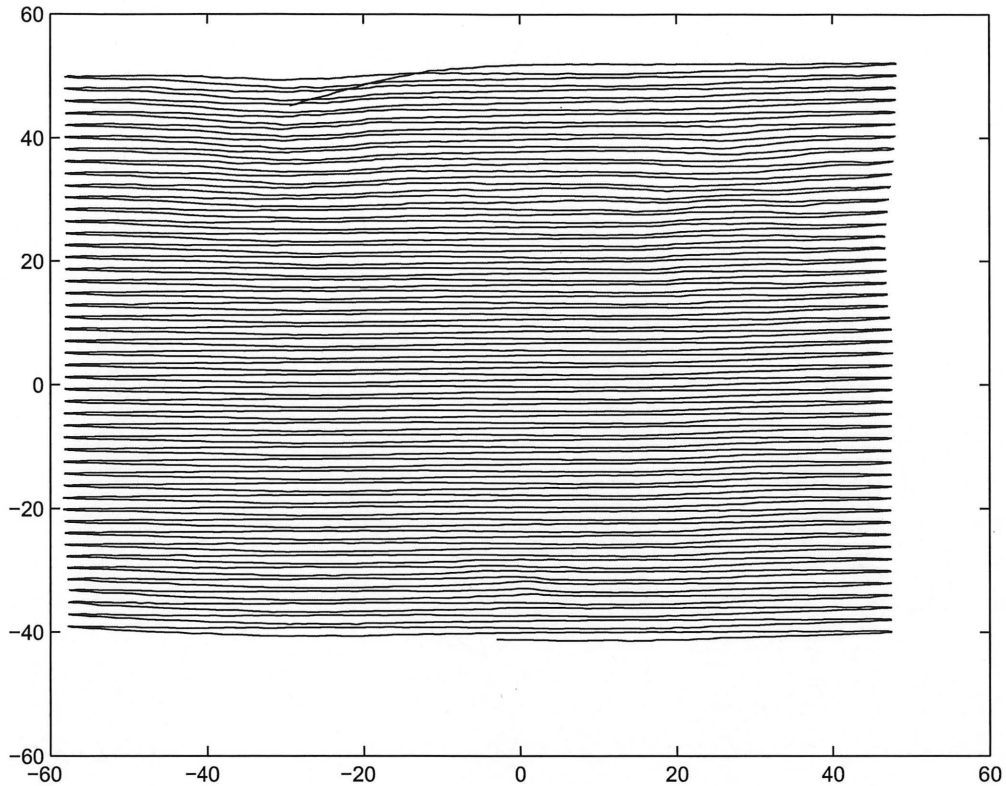


Figure 4.2 – Balayage d'une surface par le LIDAR, la composante Z a été annulée pour montrer la non-orthogonalité des mesures dans le plan horizontal. Les deux axes représentent des distances en unité de mesure

Étape A_2 : Les limites de la surface sont cherchées pour déduire la largeur et la longueur de la surface scannée.

Étape A_3 : Si le traitement est effectué pour l'image(t-1), la valeur du déplacement ΔF de la fenêtre F est calculée pour respecter une taille s'approchant de 35x35 pour la matrice 1. Si le traitement s'applique à l'image(t), le déplacement ΔF calculé pour l'image(t-1) est conservé pour avoir un traitement équivalent d'une image à l'autre. Le déplacement ΔF effectué sera conservé pour calculer, à la dernière étape du ME-SOM, la vitesse réelle entre deux images.

Étape A_4 : Le nombre de déplacements ΔF vertical et horizontal de la fenêtre F est déterminé. La taille de la fenêtre F est plus grande que la valeur de ΔF ce qui induit un chevauchement.

Étape A_5 : Pour chaque déplacement de la fenêtre, la moyenne des points contenus dans la fenêtre est calculée pour avoir les valeurs de la matrice de sortie.

La figure 4.3 montre le passage du nuage de points d'élévation en entrée à la matrice 1 en sortie du bloc A. L'algorithme est donc efficace pour recomposer une surface en trois dimensions. Il est à noter, comme précisé dans la partie description générale de l'algorithme, que les mesures prises par le LIDAR représentent la distance entre le LIDAR et la surface scannée : le relief est donc inversé. Les plateaux sont en réalité des plaines, les canyons des chaînes de montagne, etc. Il est donc inutile de recomposer le relief à l'endroit pour que l'algorithme ME-SOM fonctionne ce qui est un des avantages de cette méthode.

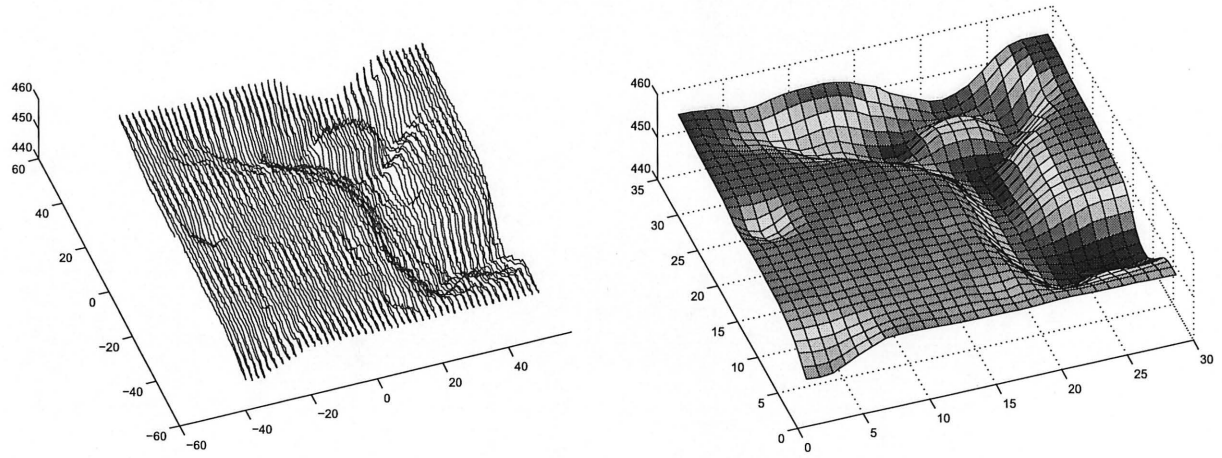


Figure 4.3 – Passage d'un nuage de points d'élévation générés par le LIDAR en entrée du bloc A à la matrice 1 en sortie

L'étape suivante consiste à trouver des points marquants de l'image ou marqueurs pour ensuite les suivre d'une matrice 1 à une matrice 2. C'est l'étape clef de la méthode proposée. En effet, si elle se déroule mal, il sera impossible même avec un excellent algorithme de *tracking* de suivre des points d'une image à une autre et donc de déterminer la vitesse relative.

4.3 Détection des marqueurs

Cette section décrit l'utilisation d'un réseau de Kohonen pour faire ressortir les marqueurs des matrices générées à partir des mesures du LIDAR. Cette étape correspond au bloc B dans la figure 4.1. Les réseaux de Kohonen ont été choisis de par leurs propriétés et surtout le contexte d'un atterrissage sur une planète. En effet, comme le souligne la revue de littérature, de toutes les méthodes d'intelligence artificielle, les réseaux de Kohonen se sont démarqués par leur méthode d'apprentissage ne nécessitant pas de données d'entraînement. Ces réseaux sont utilisés dans ce projet d'une façon particulière. Dans une utilisation standard, un nombre d'exemples conséquents est présenté au réseau pour que la carte de Kohonen converge vers une représentation ordonnée de l'espace d'entrée. L'utilisation de cette carte ordonnée permet, par exemple, de classer différents éléments de l'espace d'entrée. Dans ce projet de maîtrise, la convergence elle-même pour une entrée (la matrice 1) sur plusieurs itérations est utilisée pour faire ressortir par compétition les sommets ou les creux les plus marquants de la matrice. Ainsi, le problème de l'entraînement est résolu, puisque c'est la convergence du réseau pour un exemple de l'espace d'entrée qui donne la solution. Cette utilisation des réseaux de Kohonen réalise le prétraitement, quel que soit le type d'entrée, réelle ou artificielle. À l'inverse d'un réseau de neurones classique, il n'y pas de généralisation, il n'y a donc plus de source d'erreur. Même si une image jamais vue par le réseau est présentée, celui-ci va être capable de la traiter. Dans le cadre d'un atterrissage sur une planète, c'est un avantage majeur par rapport à d'autres méthodes.

4.3.1 Conception du réseau de Kohonen du bloc B

Le réseau permet de faire converger les minimums locaux (L_{min}) et les maximums locaux (L_{max}) de la matrice 1 vers un minimum ou un maximum global (G_{min} et G_{max}) fixé par le concepteur du réseau de neurones. L'organisation de la carte de Kohonen est obtenue par compétition entre les neurones qui la composent.

À la première itération, la matrice de poids W (la carte de Kohonen) est initialisée avec les valeurs de la matrice 1. La matrice W est ensuite normalisée par sa valeur maximum M.

La distance entre le LIDAR et la surface est toujours plus grande que le relief balayé. Cela implique que les valeurs de la matrice normalisée sont très proches de 1. En effet, la recherche de la vitesse relative est très importante pendant la phase de recherche d'un site sécuritaire. Pratiquement, cette recherche s'effectue entre 5km et 1km d'altitude. Le voisinage du neurone gagnant est défini par un disque D de rayon R (rayon d'influence) centré sur la position du neurone gagnant dans la carte de Kohonen. Pour chaque neurone, si le neurone est un minimum ou un maximum local, il devient le neurone gagnant.

Un fait marquant de la conception de ce réseau est que la carte de Kohonen sert de stimuli pour elle-même. L'entrée et la sortie du réseau sont composés uniquement de la carte de Kohonen. Seule la compétition entre les neurones de la carte est donc utilisée dans ce réseau. Le poids de tous les neurones inclus dans le disque D sont mis à jour pour converger vers la valeur G_{min} ou G_{max} dépendant si le neurone gagnant est un minimum ou un maximum local. Cette mise à jour de poids est conservée dans la matrice ΔW. À la fin de l'itération, W, le facteur d'apprentissage σ et R sont actualisés. La valeur de σ et R augmente permettant une meilleure compétition entre les neurones. Au fil des itérations, les plus grand minimums et petit maximums locaux (petits creux et petites bosses) sont éliminés par les plus importants. Voici la description des différentes étapes du bloc B présentées à la figure 4.4.

Étape B_1 : R_i, la valeur initiale du rayon R doit être fixée à 1, car l'algorithme travaille avec des indices de matrice entiers. R_f, la valeur finale du rayon R, est comprise entre la plus grande dimension de la matrice 1 et R_i. Si R_i et R_f ont des valeurs trop grandes, il n'y aura plus assez de points marquants dans l'image. G_{min} et G_{max} sont choisis comme le minimum et le maximum normalisés auxquels un biais est ajouté. Ce biais est négatif pour les minimums et positif pour les maximums. Ce biais permet de mettre en évidence les points marquants de la matrice par rapport au relief qu'elle décrit. Le choix de la valeur de ce biais sera expliqué dans la section 4.3.2. Le facteur d'apprentissage est difini par σ. Il est initialisé à la valeur σ_{init} :

$$\sigma_{init} = \frac{moy}{M \times MaxIters} \tag{4.1}$$

moy est la moyenne de toutes les valeurs normalisées de la matrice 1. $MaxIters$ est le nombre total d'itérations.

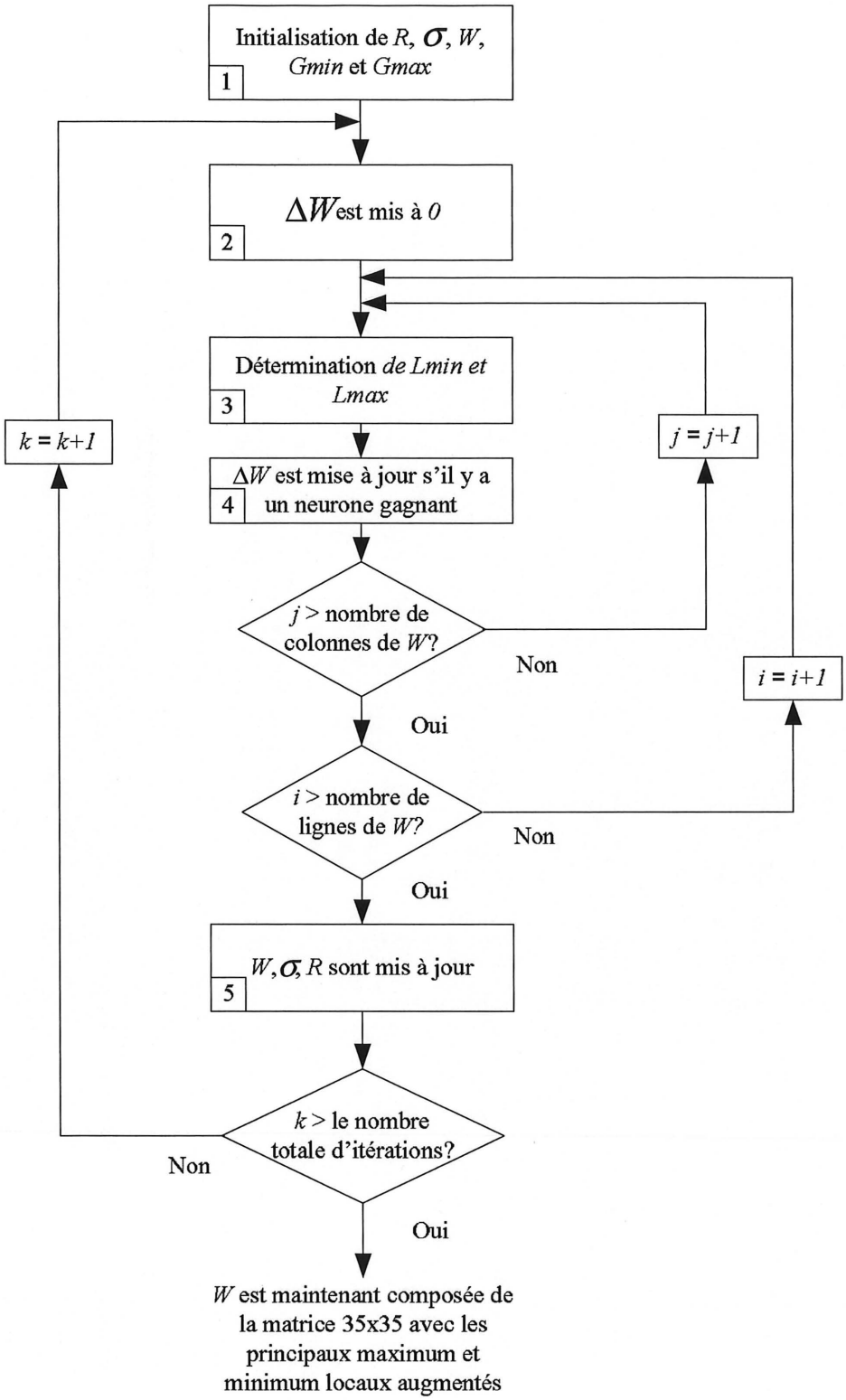


Figure 4.4 – Algorithme de détection des marqueurs (bloc B)

Une valeur trop grande pour σ_{init} et le réseau de Kohonen diverge, alors qu'une valeur trop petite ne démarquera pas assez les points significatifs de la matrice 1. La formule déterminant σ_{init} est donc dépendant des variations du relief. La solution pour en tenir compte a été de faire le rapport de la moyenne des hauteurs avec la hauteur maximale. La formule décrite pour σ_{init} semble être assez bonne pour fonctionner dans tous les cas étudiés. À la première itération, W est initialisée avec la matrice 1.

Étape B_2 **:** À chaque itération, la matrice de variation de poids ΔW est mise à zéro.

Étape B_3 **:** Une partie L_D de la carte de Kohonen est sélectionnée. L_D est définie comme le disque de rayon R centré sur le neurone courant $N(i,j)$. L_{max} et L_{min} sont déterminés dans la surface L_D comme un minimum ou un maximum local.

Étape B_4 **:** Si le neurone $N(i,j)$ est un minimum ou un maximum local, c'est à dire que la valeur de son poid est égale à L_{max} ou L_{min}, il devient alors le neurone gagnant. La matrice de variation de poids ΔW est modifiée avec les mises à jour des poids des neurones inclus dans L_D. Toutes les modifications de poids dans une itération sont additionnées dans la matrice ΔW comme :

$$\Delta W = \begin{cases} \sigma \times (\ G_{max} - L_{max}) & si\ N(i,j) = L_{max} \\ \sigma \times (\ G_{min} - L_{min}) & si\ N(i,j) = L_{min} \end{cases}$$

Étape B_5 **:** Cette étape met à jour W, R et σ. Cette étape est directement inspirée des travaux de Labonté[LabontÉ, 1998]. W est additionnée avec la matrice de variation de poids ΔW :

$$W_k = W_{k-1} + \Delta W \tag{4.2}$$

R est fixé comme suit :

$$R_k = R_{k-1} - \mu_R \tag{4.3}$$

où

$$\mu_R = \frac{(R_i - R_f)}{(MaxIters)} \tag{4.4}$$

σ est mis à jour suivant l'équation :

$$\sigma = \frac{(\sigma_{init} * \beta_{init})}{\beta_k} \tag{4.5}$$

où

$$\beta_k = \beta_{k-1} - \mu_\beta; \tag{4.6}$$

et

$$\mu_\beta = \frac{(G_{max} - G_{min})}{(\sigma_{bias} \times MaxIters)} \tag{4.7}$$

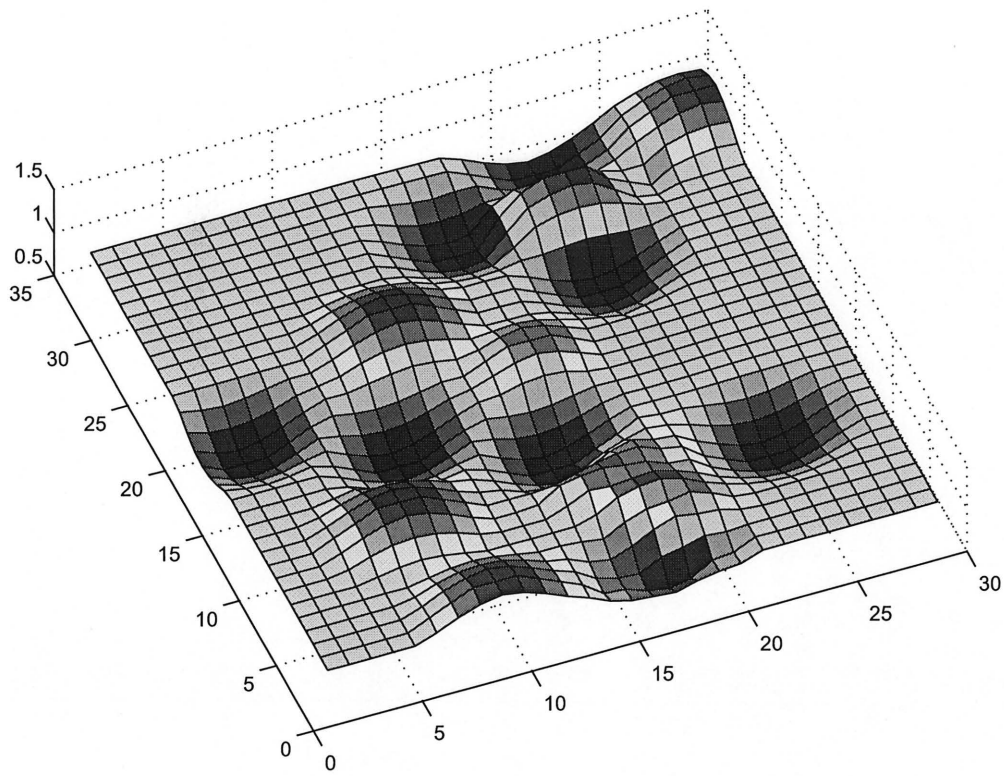


Figure 4.5 – Résultat de la construction des marqueurs pour la matrice de la figure 4.3

Le résultat du traitement est montré à la figure 4.5. Plusieurs parties de la matrice sont ainsi mises en évidence. Ces parties seront appelées tout au long de ce mémoire des taches. Un des avantages de cette méthode est que les parties qui semblent assez planes comportent aussi des tache à suivre. Des cas extrêmes seront présentés dans la section 5.1.6.

4.3.2 Seuillage de la matrice composée de taches

Le seuillage fait parti du Bloc B. Le seuillage de la matrice consiste à couper les valeurs proches de la moyenne de la matrice avec les taches. En effet, comme le montre la figure 4.5,

les taches se démarquent fortement du plan moyen. Il est donc facile de conserver seulement les points composant les taches. Cela permet de traiter, par la suite, seulement les points marquants de la matrice. Pour cela, ces 4 étapes sont effectuées :

Étape B_{S1} **:** Détermination du minimum P et du maximum G de la matrice 1 avec les taches.

Étape B_{S2} **:** Calcul de la moyenne MT des valeurs de la matrice 1 avec les taches.

Étape B_{S3} **:** Calcul de la limite haute LH et de la limite basse LB pour les taches représentant respectivement des maximums locaux et des minimums locaux. Ce calcul s'effectue de la manière suivante :

$$LH = MT + |MT - G| * S \tag{4.8}$$

et

$$LB = MT + |MT - P| * S \tag{4.9}$$

S correspond au seuil désiré par le concepteur, il s'exprime en pourcent.

Étape B_{S4} **:** Toutes les valeurs de la matrice dépassant la limite haute ou inférieure à la limite basse sont conservées et mises respectivement à 1 ou -1. Les autres sont mises à zéro.

Le choix du seuil S doit être fait en fonction du nombre d'itérations du réseau de Kohonen et de la valeur du biais choisi. La valeur du biais permet de calculer la valeur du maximum global et du minimum global. Il est donc impératif de bien choisir cette valeur. Bien que le choix soit fait de façon empirique, il est possible de suivre des règles pour optimiser la valeur. Le biais influence la hauteur des maximums et des minimums locaux lors de la compétition ; sa valeur ne doit donc pas être trop grande. Si c'est le cas, les maximums ou les minimums locaux les plus proches de G_{min} et de G_{max} seront nettement avantagés. Il faut cependant la choisir assez grande sinon il devient difficile de faire un seuil pour détecter simplement les marqueurs à conserver. Plus le biais est grand, plus la valeur des marqueurs s'éloigne de la moyenne du plan. Il devient donc plus aisé par un simple seuillage de récupérer les maximums et les minimums locaux. Il serait tentant de faire un seuillage adaptatif, cependant cette

méthode n'a pas été retenue, car il est plus intéressant d'utiliser tout le potentiel du réseau de Kohonen plutôt que d'ajouter une méthode complexe à mettre en oeuvre. Finalement, il faut toujours que le G_{min} et le G_{max} soient respectivement plus petit et plus grand que le maximum et le minimum de la matrice 1.

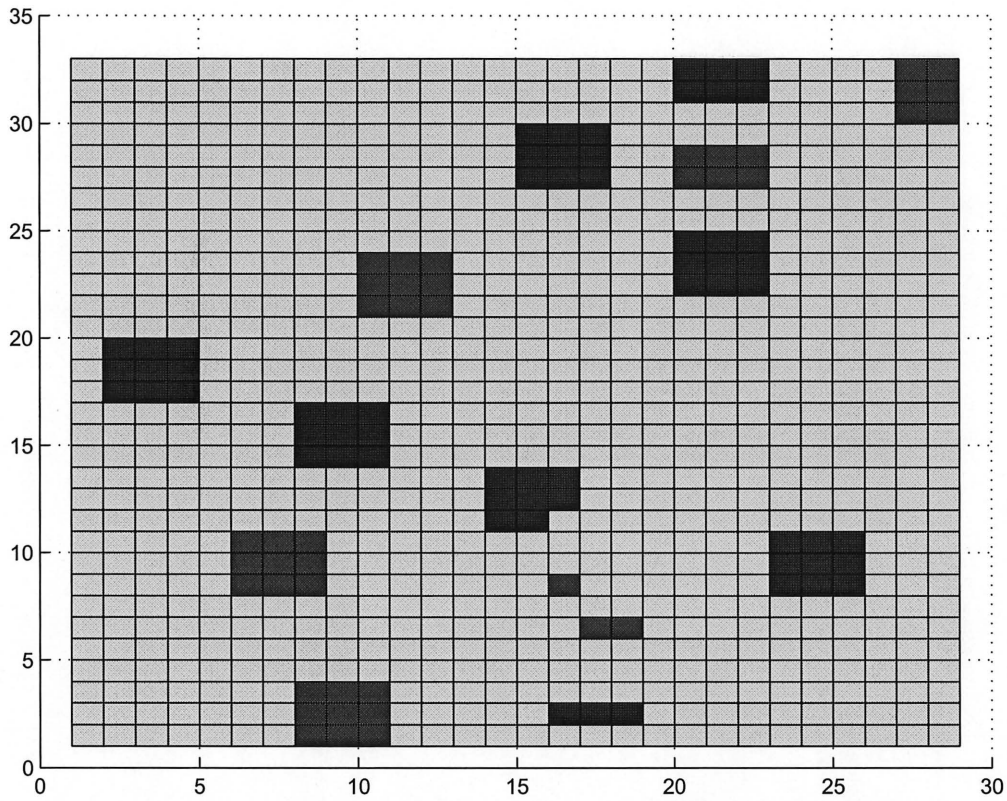


Figure 4.6 – Résultat du seuillage

Expérimentalement, le biais est fixé à 0.5, le nombre d'itération à 10 et le seuil à 60 %. La figure 4.6 montre le résultat du seuillage. Dans ce document, il faut différencier taches et marqueurs. Les taches sont un qualificatif visuel pour représenter un maximum ou minimum et les points qui les entourent. Les marqueurs représentent seulement le centre de gravité des taches. Après le seuillage, la complexité des données initiales a été fortement simplifiée. Il ne reste plus qu'une infime partie du contenu de la matrice 1. Pour accélérer le traitement, il faut encore réduire ce nombre de données. Ainsi, seul le centre de gravité des taches sera conservé avec leurs caractéristiques (minimum ou maximum). C'est la détection des marqueurs.

4.3.3 Détection des marqueurs

Ce dernier module du bloc B permet de déterminer le centre de gravité de chaque tache (figure 4.6). Pour cela, la matrice est parcourue ligne par ligne. À chaque valeur 1 ou -1 trouvée et non contiguë à une autre valeur 1 ou -1, un groupe est créé. Une fois la matrice totalement parcourue, tous les groupes composant la matrice avec leurs coordonnées sont détectés. Il suffit alors de faire la moyenne des coordonnées pour chaque groupe pour obtenir le centre de gravité. Tous les centres de gravité calculés sont mis dans le vecteur $F1$. Cette méthode simple évite d'avoir recours à un traitement d'image utilisant des filtres, traitement nécessaire si la matrice est plus compliquée ou bruitée. La figure 4.7 montre le résultat de la détection des marqueurs.

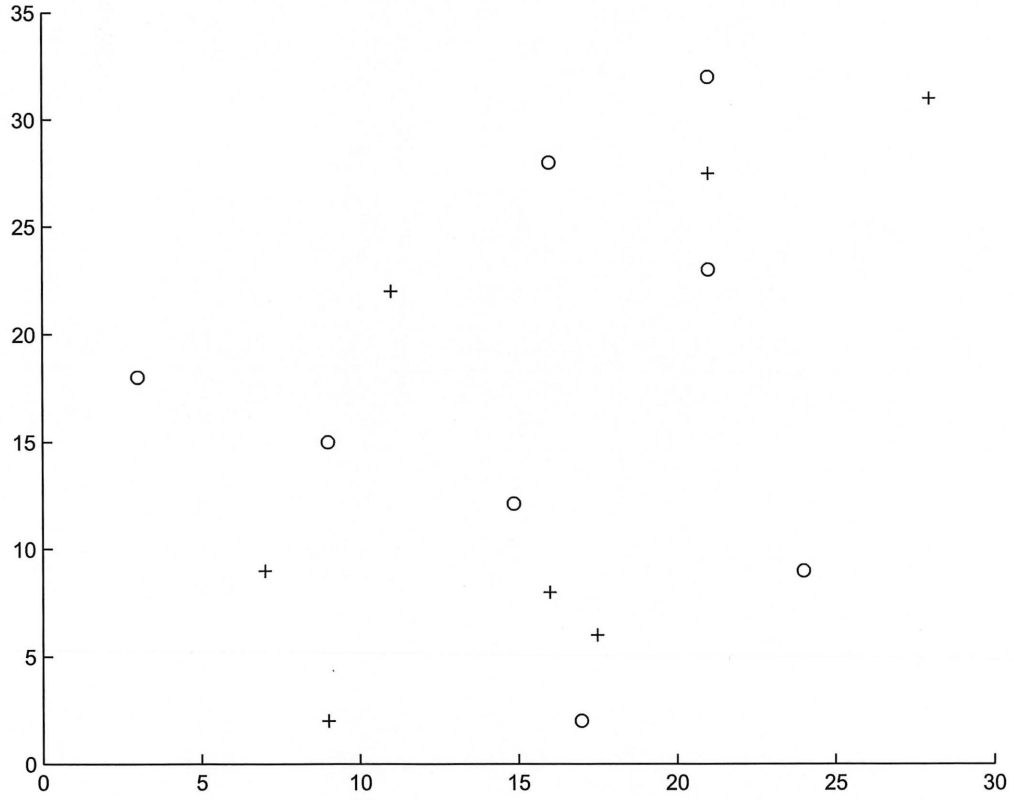


Figure 4.7 – Détection des marqueurs, les + représentent les maximums locaux, les o représentent les minimums locaux

La détection des marqueurs permet donc de réduire le nombre de données à de simples

points dans un plan. Typiquement, le nombre de marqueurs est de l'ordre d'une vingtaine par image, ce qui permet à l'algorithme de *tracking* d'être assez rapide.

En sortie du bloc B, les coordonnées des marqueurs sont normalisées avec les plus grandes coordonnées en x et en y parmi les marqueurs. Cela évite au réseau de Kohonen s'occupant de *tracking* de faire de trop grand mouvement à chaque itération. De grands mouvements sont source d'instabilité. En effet, le facteur d'apprentissage du réseau de Kohonen effectuant le *tracking* étant fixe quelque soit les positions des marqueurs, il est nécessaire de se rapprocher le plus possible d'un scénario semblable pour chaque cas. De plus, une trop grande valeur au facteur d'apprentissage donne une trop grande mobilité au réseau ce qui peut être la source d'oscillation et même de divergence. Toutes les coordonnées des marqueurs sont donc comprises entre 0 et 1 pour les x et les y.

4.4 Détermination du déplacement des marqueurs

4.4.1 Architecture générale du bloc C

Cette partie est basée sur les travaux de Labonté [LabontÉ, 1998]. L'idée générale est d'utiliser la distance entre les points d'une image pour les différencier les uns des autres. L'algorithme est composé en fait de deux réseaux de Kohonen interconnectés ($K1$ et $K2$). Par *interconnecté*, Labonté veut dire que la carte de Kohonen du premier réseau sert d'entrée au deuxième réseau et réciproquement. Les deux cartes de Kohonen des deux réseaux sont donc initialisées avec les vecteurs $F1$ et $F2$ (figure 4.1). Les deux cartes de Kohonen sont deux vecteurs-colonnes représentant des points dans un plan. Chaque neurone représente un couple de coordonnées (x,y) d'un marqueur. Dans la suite de l'explication, seul le premier réseau $K1$ qui effectue la mise à jour des poids des neurones de $F2$ sera décrit. Le second réseau $K2$ est identique à $K1$.

Pour chaque neurone de $F1$, le plus proche neurone dans $F2$ est recherché. Le neurone de $F2$ le plus proche pour un neurone de $F1$ devient alors le neurone gagnant NG si la distance euclidienne séparant ces deux neurones est inférieure à $LF2$. $LF2$ est fixé à 50% de la taille

d'une image pour suivre les spécifications du projet. Ainsi, les poids des neurones se situant dans le disque DF de centre NG sont tous mis à jour. Ils convergent vers la position du neurone gagnant. Quand tous les neurones de $F1$ ont été sélectionnés et que le processus a été effectué pour le deuxième réseau de neurones $K2$, les poids de $F1$ et $F2$ sont mis à jour. Quand l'évolution des deux réseaux de neurones s'arrête, les points communs aux deux matrices sont à la même position. La figure 4.8 décrit l'architecture générale d'un des deux réseaux de neurones interconnectés. Les deux réseaux ont strictement la même architecture. Le processus se compose des 5 étapes suivantes :

Étape C_1 : Il faut fixer le nombre d'itérations $KMaxIters$, la distance maximale $LF2$ existant entre un neurone en entrée et son neurone correspondant en sortie et la taille initiale et finale du rayon d'influence du neurone gagnant (KR_i et KR_f). Expérimentalement, le nombre d'itération a été fixe à 80. $LF2$ est fixé à 0.5 ce qui correspond à une translation maximale de la moitié d'une image, limite conceptuelle pour ce projet. KR_i est fixé à 1, car c'est la distance la plus grande possible, les coordonnées étant normalisées. KR_f est fixe à 0.01. Il est à noter que plus nombreuses sont les itérations plus la taille du rayon final peut être petite. Ceci implique que le rayon final détermine la précision de l'algorithme. En d'autres termes, pour un neurone correspondant à une position dans la matrice 1, il n'y aura qu'une seule zone de rayon 0.01 dans la matrice 2 dans laquelle idéalement il ne reste qu'un neurone. La carte de Kohonen doit être initialisée avec les valeurs du vecteur $F1$. Le facteur d'apprentissage α a été fixé par Labonté à la valeur suivante qui sera utilisée ici :

$$\alpha_{init} = \frac{0.005}{KR_f} \tag{4.10}$$

Étape C_2 : Il faut chercher le plus proche neurone dans $F2$ pour chaque neurone dans $F1$ représentant une position d'un marqueur. Pour cela, les distances euclidiennes séparant un neurone de $F1$ de tous les neurones de $F2$ sont calculées. Le neurone dans $F2$ le plus proche devient le neurone gagnant. En cas d'égalité entre deux neurones, le neurone gagnant est choisi au hasard parmi les neurones à égalité.

Étape C_3 : Il faut vérifier si la distance séparant le neurone gagnant et son neurone corres-

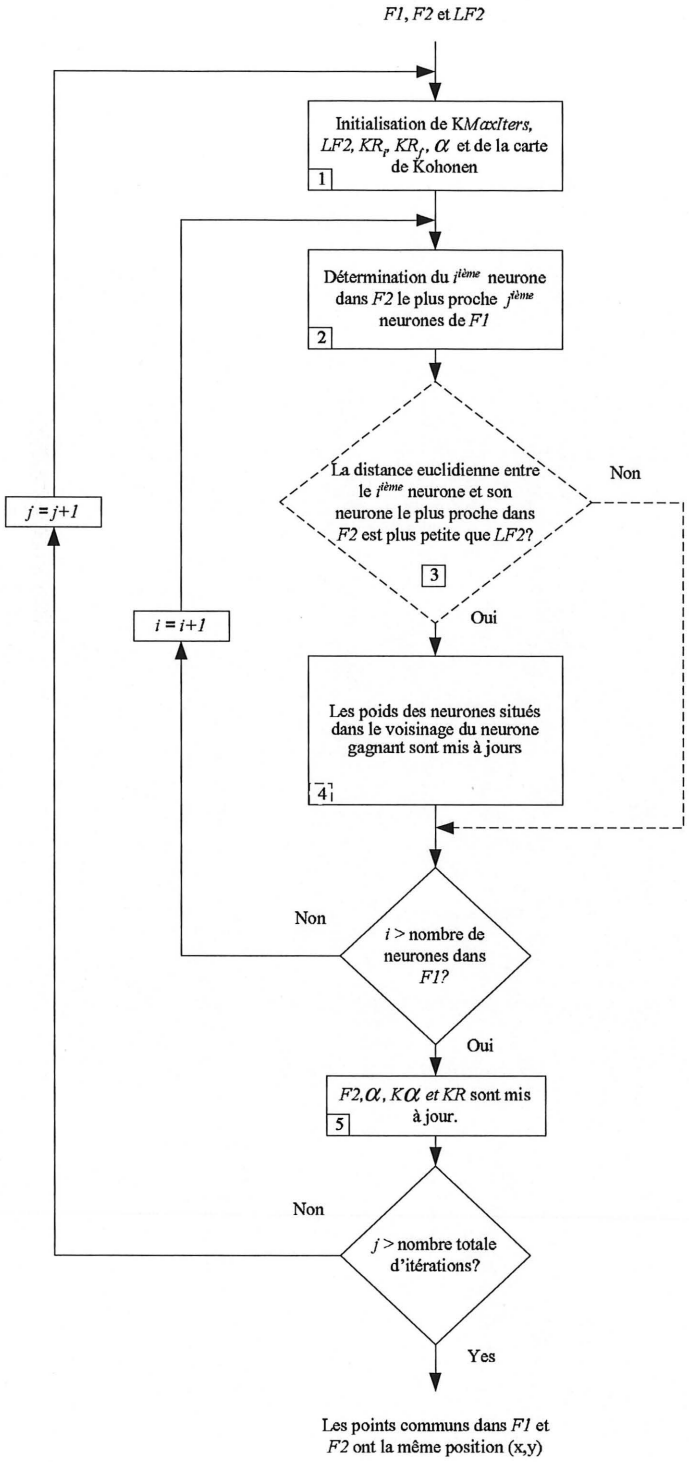


Figure 4.8 – Architecture de l'algorithme de *tracking* (bloc C)

pondant dans $F1$ est inférieur à $LF2$ sinon le neurone suivant est traité.

Étape C_4 : Les poids des neurones se situant dans le disque DF de centre NG sont tous mis à jour et conservés dans $\Delta F2$. Cette mise à jour des poids s'effectue de la manière suivante :

$$\Delta F2(k) = \alpha \times (F2(k) - NG) \ si \ F2(k) \in DF \tag{4.11}$$

Les indices k correspondent au parcours de tous les neurones de F2.

Étape C_5 : La taille du rayon d'influence KR est recalculée pour la prochaine itération de la manière suivante :

$$KR_j = KR_{j-1} - \beta \tag{4.12}$$

avec

$$\beta = (KR_i - KR_f)/(KMaxIters) \tag{4.13}$$

La taille du rayon KR_i est supérieure à la taille du rayon KR_f et β est positif. Le rayon d'influence diminue donc à chaque itération permettant ainsi de conserver seulement les neurones les plus proches jusqu'à ce qu'il n'en reste plus qu'un. Le facteur d'apprentissage α est aussi mis à jour. Plus le temps passe, plus le neurone gagnant NG est susceptible d'être le neurone homologue du neurone en entrée. La correction s'avère de plus en plus franche : α augmente. α évolue comme suit :

$$\alpha_j = \frac{KR\alpha_{j-1}}{KR} \tag{4.14}$$

avec

$$KR\alpha_j = KR_j \times \alpha_{init} \tag{4.15}$$

Le pas de convergence varie typiquement de 0.001 à 0.1.

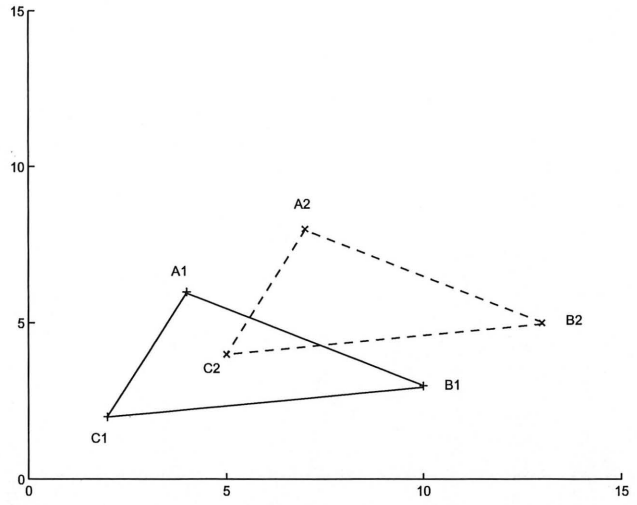


Figure 4.9 – Position des triangles à la première itération de l'algorithme de *tracking*

4.4.2 Explication du fonctionnement

L'évolution des poids des neurones représentant un triangle a été choisi pour l'explication. Cela donnera au lecteur un exemple pour bien comprendre le fonctionnement de la méthode décrite par Labonté. La figure 4.9 représente deux figures, le triangle 1 et le triangle 2, identiques translatées. Le triangle 1 en ligne pleine représente la position du triangle A, B, C dans l'image 1. Le triangle 2 en ligne pointillé représente la position du triangle A, B, C dans l'image 2. Le triangle 1 inclut les points A1, B1, et C1, le triangle 2 les points A2, B2, et C2. Le processus pour obtenir le déplacement est obtenu en 30 itérations. À la première itération (figure 4.9), le neurone gagnant de A1 est C2, celui de B1 est B2 et celui de C1 et C2 respectivement du triangle 1 au triangle 2. De même, du triangle 2 vers le triangle 1, le neurone gagnant de A2 est A1, celui de B2 est B1 et celui de C2 est A1. Le rayon étant de 1 soit la totalité de l'image tous les neurones vont bouger de la même manière. En effet, il faut bien retenir que ce n'est pas la différence entre le neurone gagnant et son homologue qui permet de calculer le mouvement des neurones mais bien la distance entre le neurone gagnant et les neurones qui l'entourent à l'intérieur du disque de rayon 1 pour cette première itération. Comme tous les neurones sont chacun inscrit dans le disque de rayon 1, l'addition de leur influence sera égale pour tous. Il ne faut pas oublier que l'opération de recherche du neurone gagnant s'exécute dans les deux sens, du triangle 1 au triangle 2 mais aussi du

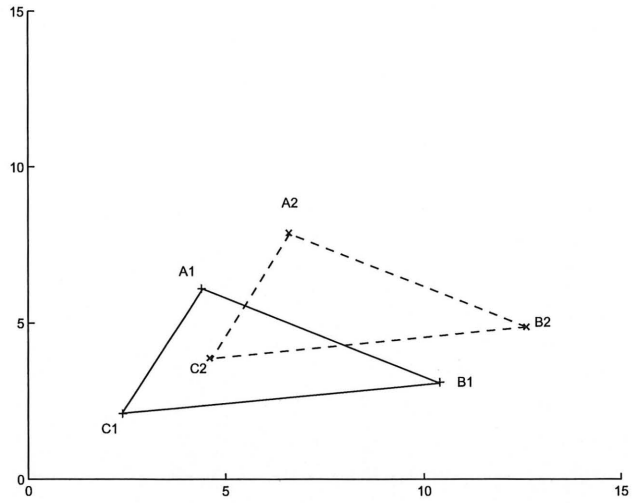


Figure 4.10 – Position des figures à l'itération 11 de l'algorithme de *tracking*

triangle 2 vers le triangle 1, les mouvements seront donc parfaitement opposés dans les deux processus. Ce processus va se poursuivre jusqu'à l'itération 12. En effet, le rayon d'influence du neurone gagnant est avant cela toujours assez grand pour englober tous les neurones de l'image. Les deux figures se déplacent donc de la même manière sans se déformer. La figure 4.10 montre la situation à l'itération 11. À cette itération, les deux triangles se sont légérement rapprochés par rapport à la figure 4.9, l'algorithme est donc arrivé au terme de la première phase. En effet, la distance [A1,A2], la distance [A1,C2] et la distance [C1,C2] sont presque identique. La réduction du rayon d'influence va permettre de les départager. La figure 4.11 montre la taille des rayons d'influence pour le triangle 2. Le rayon d'influence de B2 ne contient plus que B1, cela va donc avoir une influence directe sur la compétition entre les neurones. La compétition va faire en sorte que A2 et C2 vont encore bouger de la même manière mais B2 lui va commencer à se détacher. Le mouvement des neurones résultant de la somme des influences de chaque neurone entre eux, fait que B2 ne suit plus A2 et C2. En regardant de plus près les résultats, la première compétition du triangle 1 vers le triangle 2 montre que les neurones gagnant de A1, B1 et C1 sont respectivement C2, B2 et C2 alors que de la figure 2 à la figure 1 les neurones gagnant de A2, B2 et C2 sont A1, B1 et A1. Cela se traduit dans les faits par un rapprochement de B1 et B2, car dans les réseaux de neurones, ils sont bien retrouvés. Le problème se situe au niveau des 4 points (A1, C1, A2,

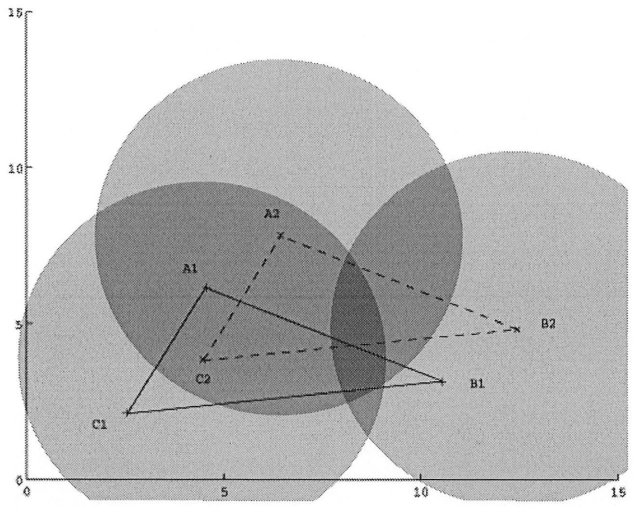


Figure 4.11 – Rayon d'influence pour la compétition du triangle 1 vers le triangle 2 à l'itération 12

C2). En effet, C2 est bien le neurone gagnant de C1 du triangle 1 vers le triangle 2, mais du triangle 2 au triangle 1, A1 est le neurone gagant de C2. Le neurone C2 pose donc un problème. Cependant, le fait que dans le premier réseau, C2 soit relié à C1 va rapprocher les neurones C1 et C2. De même, dans le deuxième réseau, le fait que A2 soit relié à A1 va rapprocher les neurones A1 et A2. De cette manière la somme totale des déplacements fera en sorte que C2 se rapproche de C1 et A2 se rapproche de A1.

Après 12 itérations, le rayon d'influence ne contient plus que les neurones homologues comme le montre figure 4.12. La simple convergence finit par rapprocher les points jusqu'à qu'ils soient confondus.

Cette exemple montre le fonctionnement de l'algorithme décrit par Labonté dans un cas simple avec peu de points. Cependant, il donne une assez bonne idée du processus. Dans un cas simple, cet algorithme semble très performant, dans des cas plus complexes avec notamment des points qui ne sont pas communs aux deux matrices ou à des places légèrement différentes, il peut occasionner des erreurs. C'est pour cela que le résultat est traité par un dernier module de bloc C qui récupère les valeurs générales du mouvement trouvées à partir de toutes les paires de points détectés par l'algorithme de *tracking*.

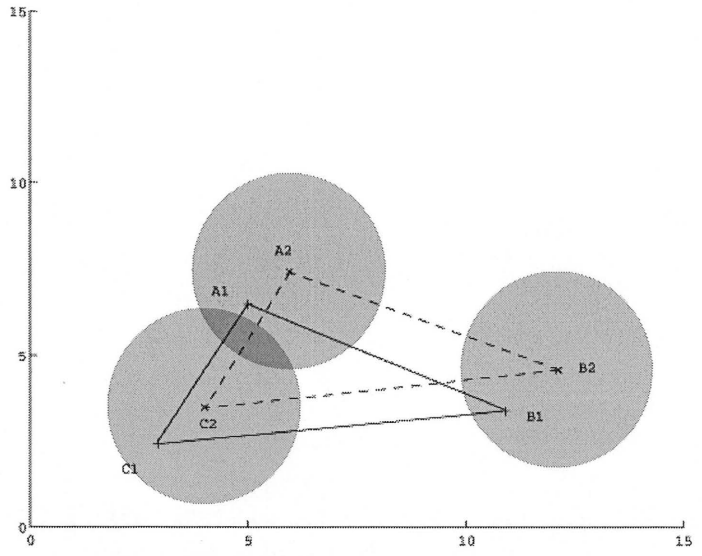


Figure 4.12 – Position des deux triangles à l'itération 24

4.4.3 Calcul de la valeur majoritaire

Cet algorithme permet de conserver les valeurs majoritaires d'un ensemble de valeurs. Il sera nommé BGA (Best Group Algorithm). Cela permet d'éliminer les valeurs qui s'éloignent trop de la majorité et donc de garder les valeurs les plus représentatives. Les valeurs sont composées de la direction et de la norme de tous les vecteurs déplacements découlant des paires déterminées par l'algorithme de *tracking*. Même si la direction et la norme majoritaire des vecteurs est bien définies, les paires de points en commun entre deux images ne sont pas toujours bonnes, surtout en présence de bruit important. C'est le cas à la figure 4.13.

La figure 4.13 montre que le déplacement est identifiable. Il n'est cependant pas possible de calculer une moyenne des distances obtenues sans engendrer une erreur qui peut être grande en fonction des erreurs commises par l'algorithme de *tracking*. Pour augmenter la robustesse de la méthode de Labonté, un filtrage de ces erreurs a été fait pour garder les distances qui semblent majoritaires. L'algorithme est inspiré de l'architecture d'un réseau de Kohonen, mais très simplifié. Il n'y pas d'itération et aucun pas de convergence à fixer. L'idée est de trouver un moyen de déterminer la distance majoritaire entre toutes les paires de points détectés par l'algorithme de *tracking*. Pour cela l'algorithme BGA se divise en 5 étapes :

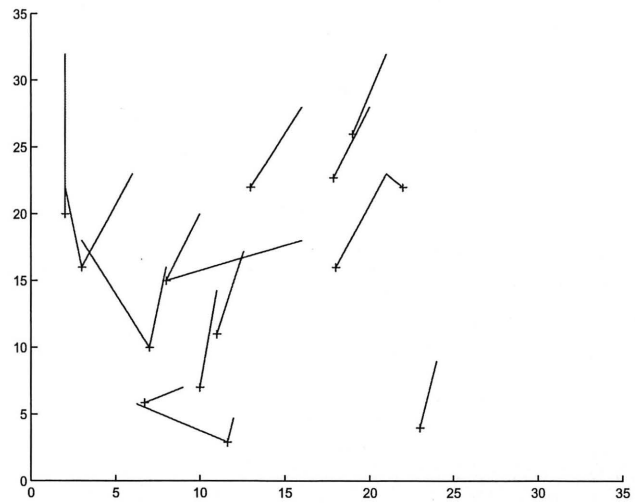


Figure 4.13 – Résultat de l'algorithme de *tracking* sans filtrage

Étape BGA_1 **:** Elle consiste à collecter toutes les distances et tous les angles des vecteurs formés par les paires de points en commun. Toutes les distances sont stockées dans d et tous les angles dans a.

Étape BGA_2 **:** L'écart type des valeurs des angles ET_a et des valeurs des distances ET_d est ensuite calculé de la manière suivante :

$$ET_d = \sqrt{\sum_{i=1}^{taille_d} \frac{(d_i - m_d)}{taille_d}} \tag{4.16}$$

où $taille_d$ représente le nombre total de valeurs de distance et m_d la moyenne des distances.

$$ET_a = \sqrt{\sum_{i=1}^{taille_a} \frac{(a_i - m_a)}{taille_a}} \tag{4.17}$$

où $taille_a$ représente le nombre total de valeurs d'angle et m_a la moyenne des angles.

Étape BGA_3 **:** Les valeurs de a sont normalisées grâce au rapport de l'écart type des distances sur l'écart type des angles. Cela permet de mettre les angles et les distances sur un pied d'égalité.

Étape BGA_4 **:** Pour chaque vecteur, les valeurs de la distance et de l'angle normalisé sont prises comme les coordonnées d'un point dans un plan (angle, distance). L'origine du

vecteur est en (0,0). Pour chaque couple distance-angle, la norme est calculée dans le plan défini plus haut et la somme des distances séparant tous les points du point défini par l'angle et la distance est aussi calculée. Deux nouvelles variables sont ainsi créées. La première da contient toutes les normes de vecteur défini dans le plan (angle, distance). La deuxième S_da contient la somme des distances entre tous les points.

Étape BGA_5 **:** L'écart type ET_{da} de da et l'écart type ET_{Sda} de Sda sont déterminés.

Étape BGA_6 **:** Les points selon la distance qui les séparent sur da et sur Sda sont regroupés. Les points composant un groupe sont au maximum séparés de Et_{da} et de ET_{Sda}.

Étape BGA_7 **:** Le plus grand groupe représente alors le groupe majoritaire. La distance majoritaire qui sépare les deux matrices en entrée est déduite en prenant la médiane de ce groupe majoritaire. Cette distance devient la distance entre les deux images.

Le résultat à la figure 4.14 montre l'efficacité de l'algorithme. Il y a néanmoins une prédominance de la norme des vecteurs sur la direction ce qui conduit l'algorithme à faire deux erreurs à la figure 4.14. Malgré ces deux erreurs, les résultats sont grandement améliorés ce qui est le but recherché.

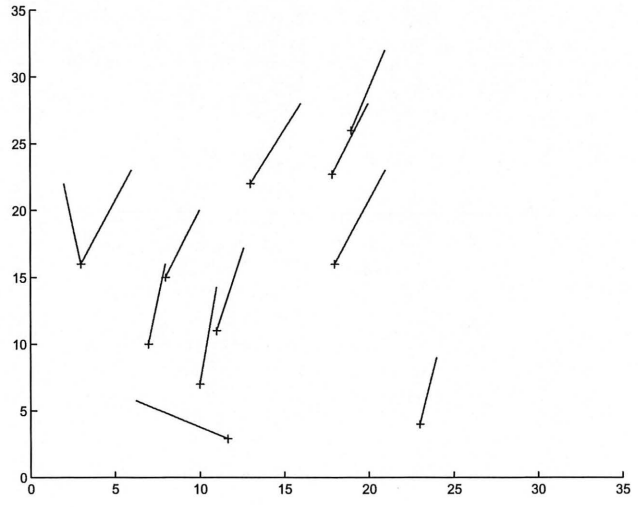


Figure 4.14 – Position des figures à l'itération 11 du processus

4.5 Calcul de la vitesse relative

Il reste alors à calculer la distance d'origine pour avoir le déplacement entre l'image 1 et l'image 2. Il suffit de multiplier la distance déterminée entre les deux matrices par le déplacement ΔF déterminé dans le module de lissage des données. Le temps entre deux images étant d'une seconde. La distance donne donc directement la vitesse en unité de mesure du LIDAR.

TROISIÈME PARTIE

RÉSULTATS ET CONCLUSION

CHAPITRE 5

RÉSULTATS

Cette partie traite de l'influence des différents paramètres de l'algorithme ME-SOM. En effet, les limites ainsi que les avantages de la méthode proposée dans ce mémoire seront étudiés. Les résultats de l'algorithme de détection des marqueurs et de *tracking* seront particulièrement étudiés. Les possibilités futures du ME-SOM seront aussi abordées.

5.1 Détection de marqueurs

Comme l'explique la partie conception, la méthode proposée pour détecter des marqueurs dans l'image est basée sur un processus de compétition entre les différents minimums et maximums locaux. Dans le cadre de ce projet, il est très important de vérifier la robustesse et la stabilité de la méthode proposée en fonction des différents terrains et conditions possibles lors d'une phase atterrissage. Les différents ajustements des paramètres ainsi que les facteurs extérieurs doivent donc être étudiés pour valider la méthode. Le premier critère est donc le traitement des données du LIDAR.

5.1.1 Effet du prétraitement

La première influence directe sur l'algorithme de détection est le lissage des données, plus particulièrement l'influence de la taille de la fenêtre F. En effet, plus la fenêtre est grande et plus la surface sera lissée. En augmentant la taille de la fenêtre, beaucoup de bruit est éliminé au détriment des détails.

Les conséquences sur l'algorithme de prétraitement sont assez importantes. Si la surface est trop lissée, l'algorithme de détection ne trouvera que très peu de marqueurs comme le montre la figure 5.1. L'algorithme de *tracking* se basant sur les relations entre les différents marqueurs d'une image ne peut alors plus fonctionner correctement. Un trop petit nombre

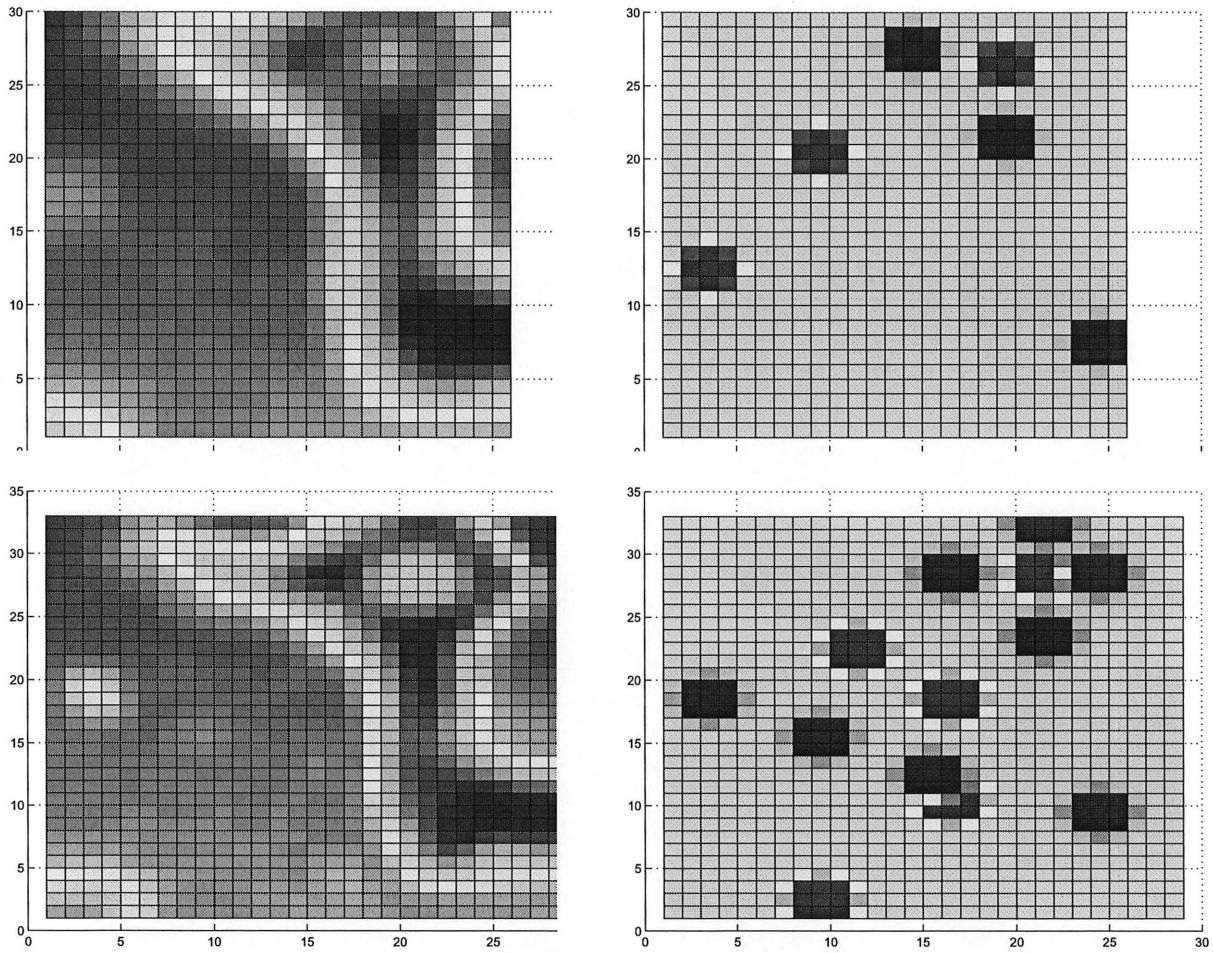


Figure 5.1 – Nombre de marqueurs avec une taille de la fenêtre F égale 12% pour la première rangée et 8% pour la seconde

de marqueurs peut, si les marqueurs ne sont pas communs aux deux images (par exemple de nouveaux points qui apparaissent dans la seconde image), le conduire à converger d'avantage vers une mauvaise solution. D'un autre coté, la taille de la fenêtre ne doit pas être trop petite du fait des spécifications du LIDAR. En effet, comme montré à la figure 4.2, les données du LIDAR ne sont pas réparties de façon uniforme, ce qui se traduit, si la fenêtre F est choisie trop petite, par des valeurs non définies pour la matrice 1(4.1). Pour une surface balayée de 100x100 unités de mesure, la taille de la fenêtre a été fixée expérimentalement à 8%. Ce choix est conservateur dans le sens où, dans la majorité des cas, 5% suffit pour avoir toutes les valeurs de la matrice 1 définies. En revanche, ce n'est que vers 14% que l'image devient vraiment trop lissée au point de ne plus avoir assez de marqueurs.

5.1.2 Effet du nombre d'itérations sur la compétition entre les neurones

La méthode de compétition employée pour faire émerger les marqueurs de la matrice 1 est basée sur la recherche des minimums et des maximums locaux. Du fait du bruit du LIDAR, plusieurs maximums ou minimums locaux se retrouvent en compétition. Il faut pouvoir distinguer quels sont les minimums et maximums locaux dûs au bruit et ceux dûs au relief. La compétition entre les différents neurones représentant ces points d'intérêt va donc permettre dans une majorité des cas d'éliminer le bruit. Les minimums ou maximums locaux les plus importants gagnent alors la compétition. Ce processus étant itératif, il est intéressant de voir l'influence du nombre d'itérations pour arriver au terme de la compétition.

La figure 5.2 montre l'évolution d'une matrice en fonction du nombre d'itérations. La première image est obtenue avec 5 itérations, la seconde avec 10 itérations et la dernière avec 15 itérations. Les taches représentent des maximums ou des minimums locaux. Les couleurs plus claires représentent des minimums ou des maximums locaux non affirmés ; c'est à dire qu'ils ne seront pas considérés comme des marqueurs par l'algorithme de *tracking*.

Les zones définies sur les figures montrent les changements entre les matrices en fonction du nombre d'itérations :

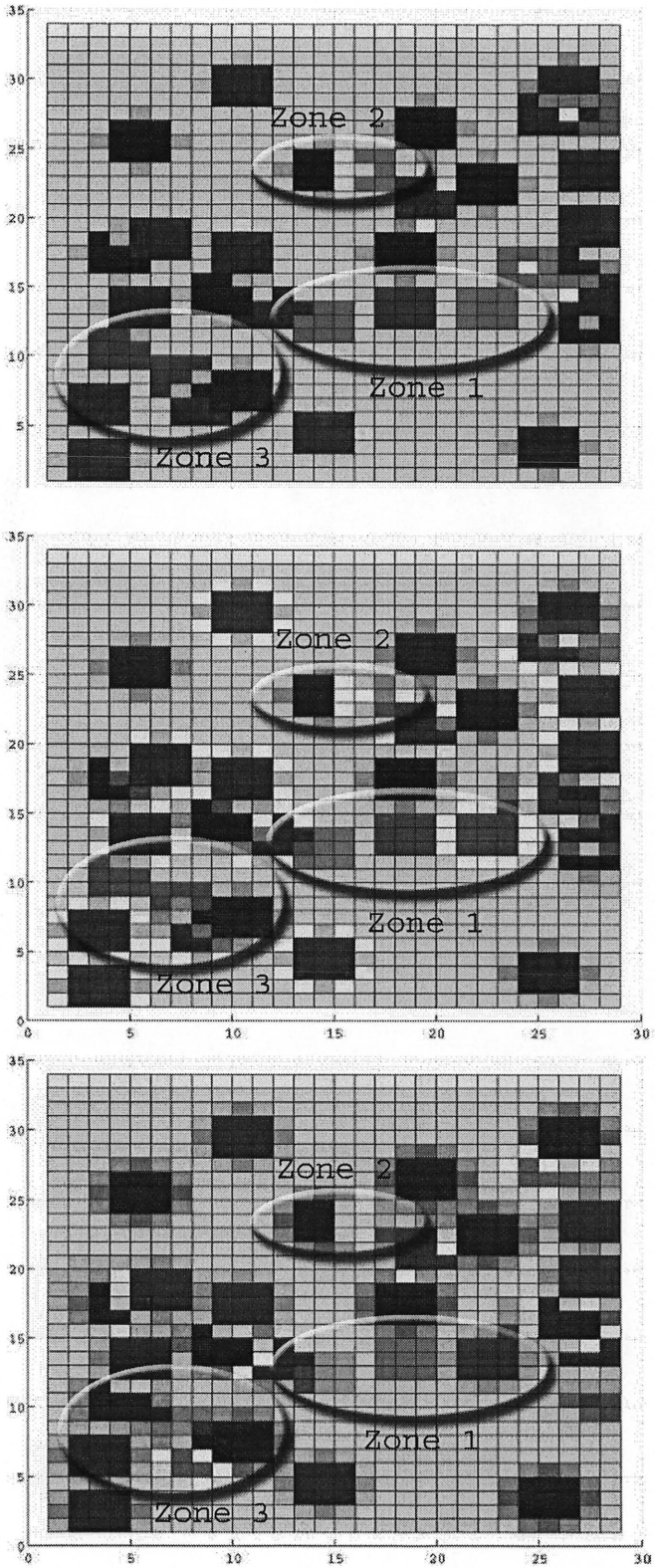


Figure 5.2 – Évolution des taches de la matrice 1 en fonction du nombre d'itérations

Zone 1 : Cette zone contient dans la première image 2 maximums locaux à gauche et à droite et un minimum local au centre. Cependant, les couleurs sont claires. Cela signifie que la compétition n'est pas achevée et que les valeurs contenues dans ces taches n'ont pas atteint le maximum et le minimum global. Ce n'est qu'à l'image 3, soit après 15 itérations, que la compétition s'achève dans cette zone. Finalement, le minimum local devient de plus en plus clair ce qui signifie qu'il sera éliminé. Le maximum local à gauche de la zone s'est affirmé, tandis que le maximum de droite a été totalement éliminé.

Zone 2 : Cette zone voit la disparition continue du maximum local à droite en fonction du nombre d'itérations. Là encore, il faut attendre 15 itérations pour avoir un processus quasi complet.

Zone 3 : Cette zone voit la séparation d'une zone de minimum locaux en deux zones bien distinctes. Ce sont les maximums locaux proches qui par leurs influences, ont scindé en deux cette zone.

Ces trois zones montrent l'évolution dans la plupart des cas des minimums et des maximums. Premièrement, la compétition est telle qu'un maximum local très petit est vite éliminé par les maximums ou minimums qui l'entourent comme le montre la Zone 2. Deuxièmement, il faut 15 itérations pour achever le processus de compétition. L'exemple utilisé est en fait parmi les pires conditions pour l'algorithme : un bruit blanc gaussien a été ajouté ce qui augmente le nombre de minimums et de maximums locaux. En fonction du lissage des données effectué et surtout de la taille de la fenêtre F, ce nombre d'itérations peut être divisé par 3. Expérimentalement, un nombre de 10 itérations donne toujours de bons résultats. Troisièmement, les taches de couleur finissent par avoir toutes la même taille. L'évolution de la zone 3 le montre très bien. Cela dépend en fait directement de la taille du rayon d'influence.

5.1.3 Effet de la taille du rayon d'influence

Le premier impact de la taille du rayon d'influence R est sur le nombre de marqueurs restant après la compétition. Il est facile de concevoir qu'un grand rayon produise de grandes taches, et la taille de la surface peut vite devenir un problème.

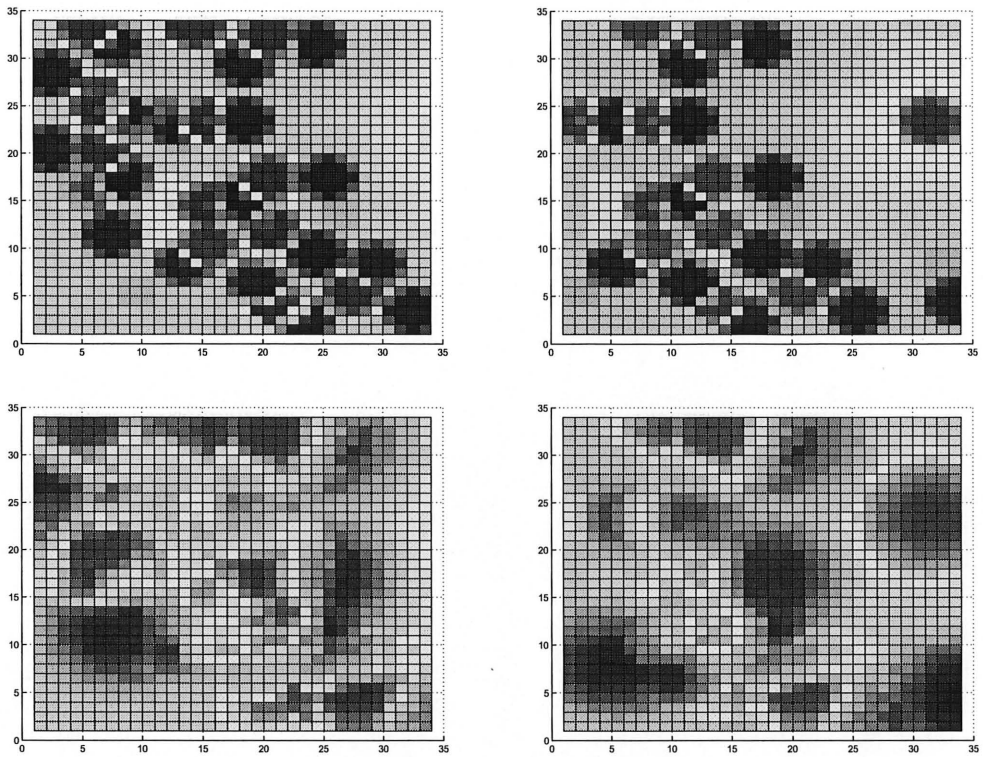


Figure 5.3 – Effet de la taille du rayon d'influence. La première rangée est obtenue avec un R égal à 10 %, la seconde rangée avec R égal à 25%.

Le deuxième impact est relatif à la compétition entre les neurones. Avec un rayon trop grand de véritables minimums et maximums locaux, et non juste du bruit, vont se retrouver en compétition. En effet, dans les premières itérations de l'algorithme de détection, le bruit qui se caractérise par de petites bosses ou de petit creux et en compétition avec le relief de la surface scannée. Comme les bosses et les creux du relief sont plus grands que le bruit, celui-ci finit par être éliminé. Si le rayon est choisi trop grand, le bruit va toujours être éliminé mais les bosses et les creux du relief lui-même, en fonction de la distance qui les sépare, vont aussi rentrer en compétition entre-eux ce qui va provoquer l'élimination de beaucoup de taches et donc une perte d'information. Ces creux et ces bosses ayant pris de l'importance au cours des itérations, le plus important l'emportera mais sera très fortement influencé. Ainsi le nombre de marqueurs sera très faible et leur centre de gravité sera altéré. Les marqueurs découlant de ces centres de gravité ne seront alors plus à la bonne place par rapport à la matrice 1 comme le montre la figure 5.3. Dans cet exemple le fait de passer le rayon R de 10% à 25%

diminue la précision de 99 % à 90.15% pour le ME-SOM, preuve que les centres de gravité des marqueurs restants ont été modifiés par rapport à leurs positions initiales. C'est l'un des inconvénients de la méthode, trop de compétition dénature le signal d'entrée. Le choix de la valeur de R ne peut se faire qu'empiriquement. Après de nombreux essais, un rayon égal à 10 % de la plus petite dimension de la matrice en entrée semble être le meilleur compromis.

5.1.4 Effet de la hauteur de la prise de vue

L'un des objectifs de ce test est de vérifier la stabilité de la méthode proposée en fonction de la hauteur de la prise de vue. En effet, ce projet, comme l'indique l'introduction, se déroule lors d'une phase d'atterrissage, ce qui signifie qu'à chaque prise de mesure du LIDAR le sol se rapproche. La conception de l'algorithme de prétraitement découle directement de cette constatation : il faut être robuste à la descente. De cette constatation est venue l'idée de travailler avec les minimums et les maximums locaux. En effet, quelle que soit la hauteur de prise de vue, un maximum ou un minimum local conserve souvent ses caractéristiques. Cependant, le rapprochement du sol les fait grossir. C'est pour cela que l'algorithme de prétraitement fixe, grâce à la taille du rayon de d'influence R, la taille des taches résultant d'un minimum ou d'un maximum local. De plus, le fait de conserver comme information le centre de gravité de la tache permet à l'algorithme de détection d'être robuste face à ce problème. La figure 5.4 montre le résultat avec une différence de 10000 mètres. C'est l'une des contributions de ce projet d'avoir un algorithme indépendant du rapprochement du sol pendant l'atterrissage. Les résultats sont excellents et démontrent que l'algorithme est totalement indépendant de la hauteur de prise de vue, car les deux matrices une fois traitées sont identiques.

Pour être sûr de la robustesse de l'algorithme de prise de vue, il est aussi nécessaire de regarder l'influence du zoom qu'engendre le rapprochement de la sonde à chaque image. Pour cela, le pire cas estimé a été généré. L'effet de zoom est dépendant de la distance à laquelle la sonde se trouve et de sa vitesse. Dans le projet LAPS [de Lafontaine et Gueye, 2004], cela se situe à une hauteur de 500 m avec une vitesse de 35 m/s. Avec ces paramètres, les dimensions de la surface scannée par le LIDAR diminuent d'environ 8 % entre deux

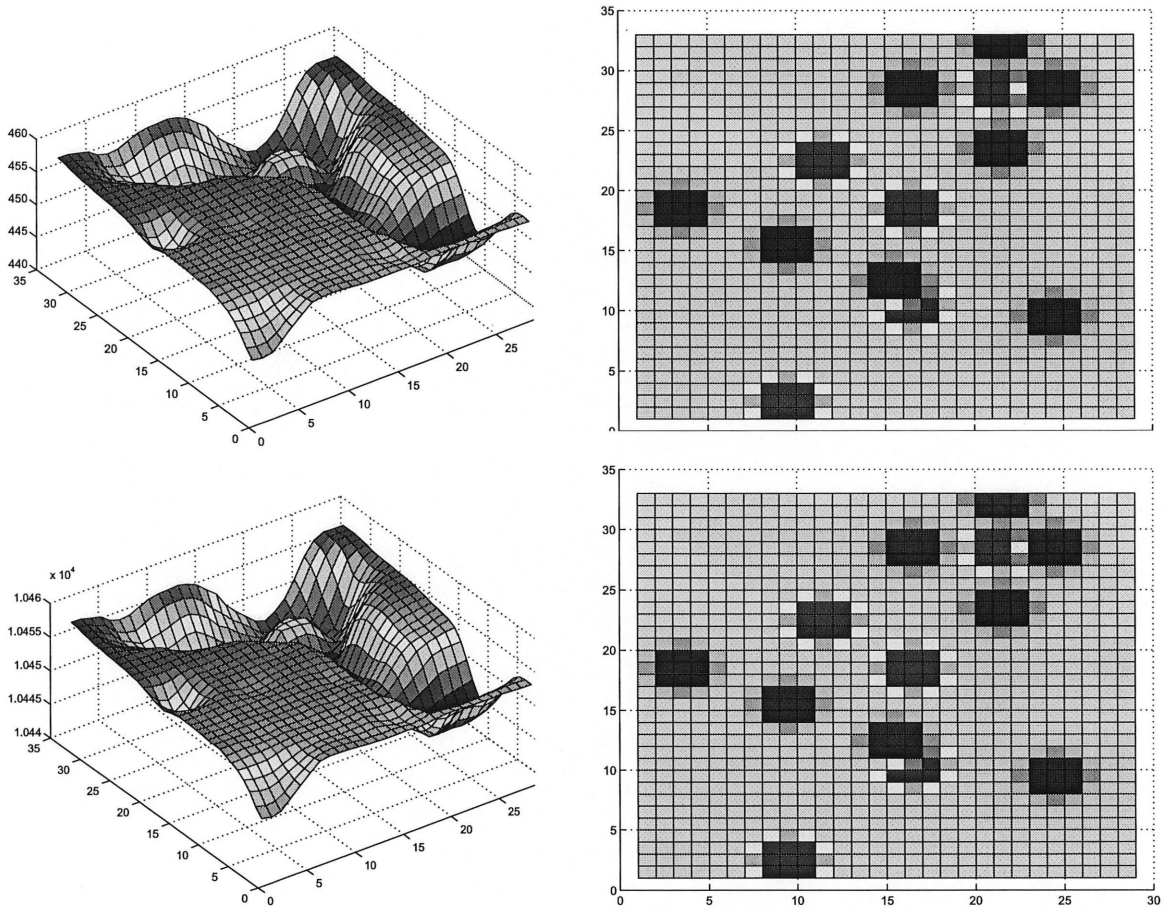


Figure 5.4 – Effet de la hauteur sur le fonctionnement de l'algorithme du bloc B. La rangée du haut est prise à 450 m, celle du bas a 10.5 km

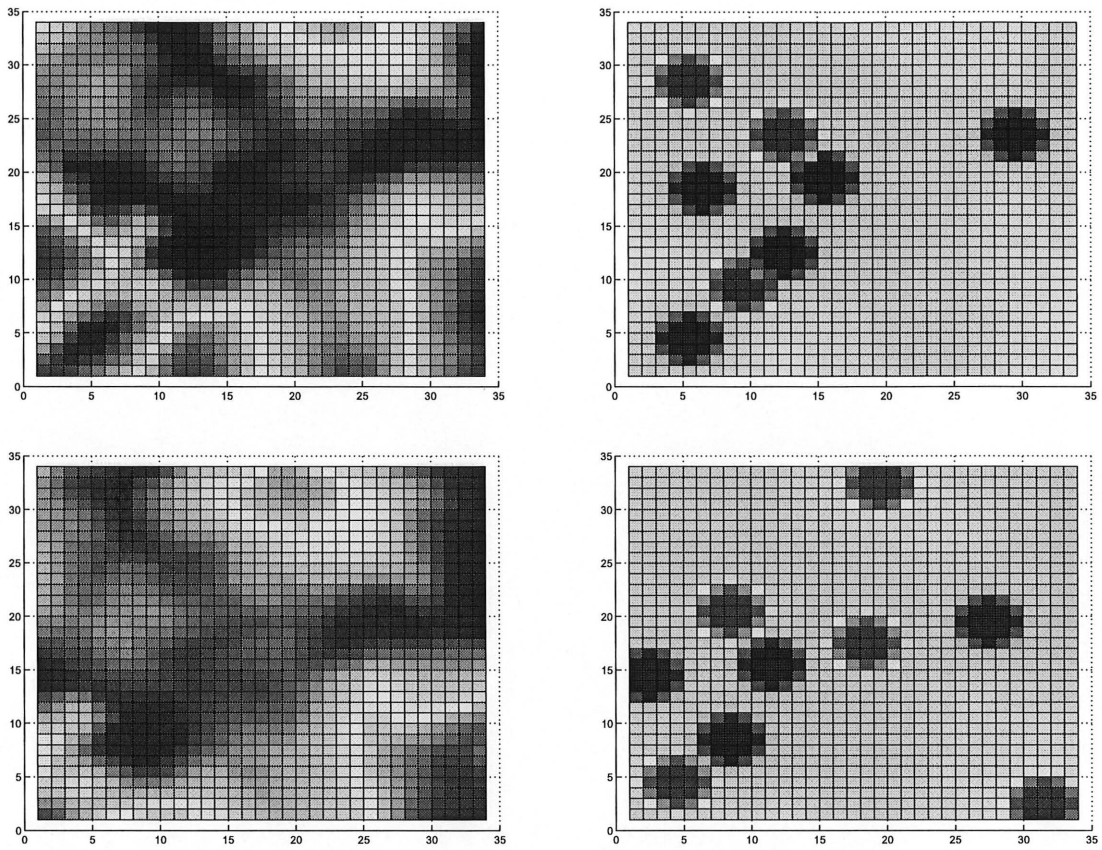


Figure 5.5 – Effet du zoom sur le résultat de la détection des marqueurs. La rangée du bas correspond à la rangée du haut après le rapprochement

images ce qui n'est finalement pas très important. Les résultats de la figue 5.5 montrent que l'algorithme de détection n'a aucune difficulté à détecter les marqueurs dans la même configuration ou presque ce qui est suffisant pour la réussite de l'algorithme de *tracking*.

Il faut bien noter que la robustesse à la différence de hauteur n'est pas obtenue grâce au LIDAR. Comme le montre la conception de l'algorithme, il n'est pas question ici d'estimer le plan moyen pour déterminer la différence de hauteur entre les deux images pour corriger les valeurs de la deuxième matrice par rapport à la première. L'algorithme n'a tout simplement pas cette information. Cette correction est inutile en raison de l'architecture de l'algorithme. Le fait de travailler avec des minimums et des maximums locaux permet à l'algorithme de détection des marqueurs de travailler en relatif. La première version de l'algorithme de prétraitement était basée sur une classification en logique floue des pentes du relief. Le calcul

des pentes entre les deux images a montré rapidement les limitations de cette méthode, car le rapprochement modifie le gradient des pentes. En effet, plus la sonde se rapproche et plus les détails du relief se distinguent, de plus la distance séparant deux prises de mesure par le LIDAR se réduit. Ces deux effets conjugués ont comme conséquence une variation à certains endroits de la valeur du gradient ce qui est assimilable à du bruit. De plus pour corriger le rapprochement, il est nécessaire de construire un plan moyen de la surface. Une mauvaise estimation de la distance séparant la sonde du plan moyen peut aussi engendrer des erreurs sur la valeur des gradients des pentes. Néanmoins, la différence de hauteur n'est pas la principale difficulté. Le bruit de mesure est, comme toujours dans ce type de projet, un handicap.

5.1.5 Effet du bruit

Le bruit est le paramètre ayant la plus grande influence sur la méthode proposée. Il est nécessaire de regarder, du fait de la méthode employée pour détecter les marqueurs, si un bruit important empêche le bon fonctionnement de l'algorithme. En effet, plus le bruit est important, plus la compétition entre les différents éléments de la matrice 1 est nécessaire. Le bruit a tendance à créer de petites bosses et des petits creux qui doivent être, idéalement, éliminés par les plus gros. La figure 5.6 montre le résultat de l'algorithme en fonction du bruit. Il faut noter que la première image prise contient déjà du bruit inhérent au LIDAR. C'est une image obtenue à partir de mesure effectuée en laboratoire. Les images suivantes ont été obtenues en rajoutant un bruit blanc gaussien de moyenne nulle. Les rapports signal sur bruit sont calculés pour les données en entrée de la manière suivante :

$$PSNR = 10 \times log10 \frac{MAX_{org}^2 \times NB_{pixel}}{\sum(MAT_{org} - MAT_{bruit})^2} \tag{5.1}$$

où MAX_{org} représente le maximum de la matrice originale MAT_{org}, NB_{pixel} le nombre de points de la matrice et MAT_{bruit} la matrice originale avec le bruit ajouté. La première ligne de la figure 5.6 correspond à un PSNR infini, la seconde à un PSNR de 18.14 dB et la dernière de 12.03 dB. Les résultats sont comparables jusqu'à PSNR de 18.14 dB, ce qui démontre

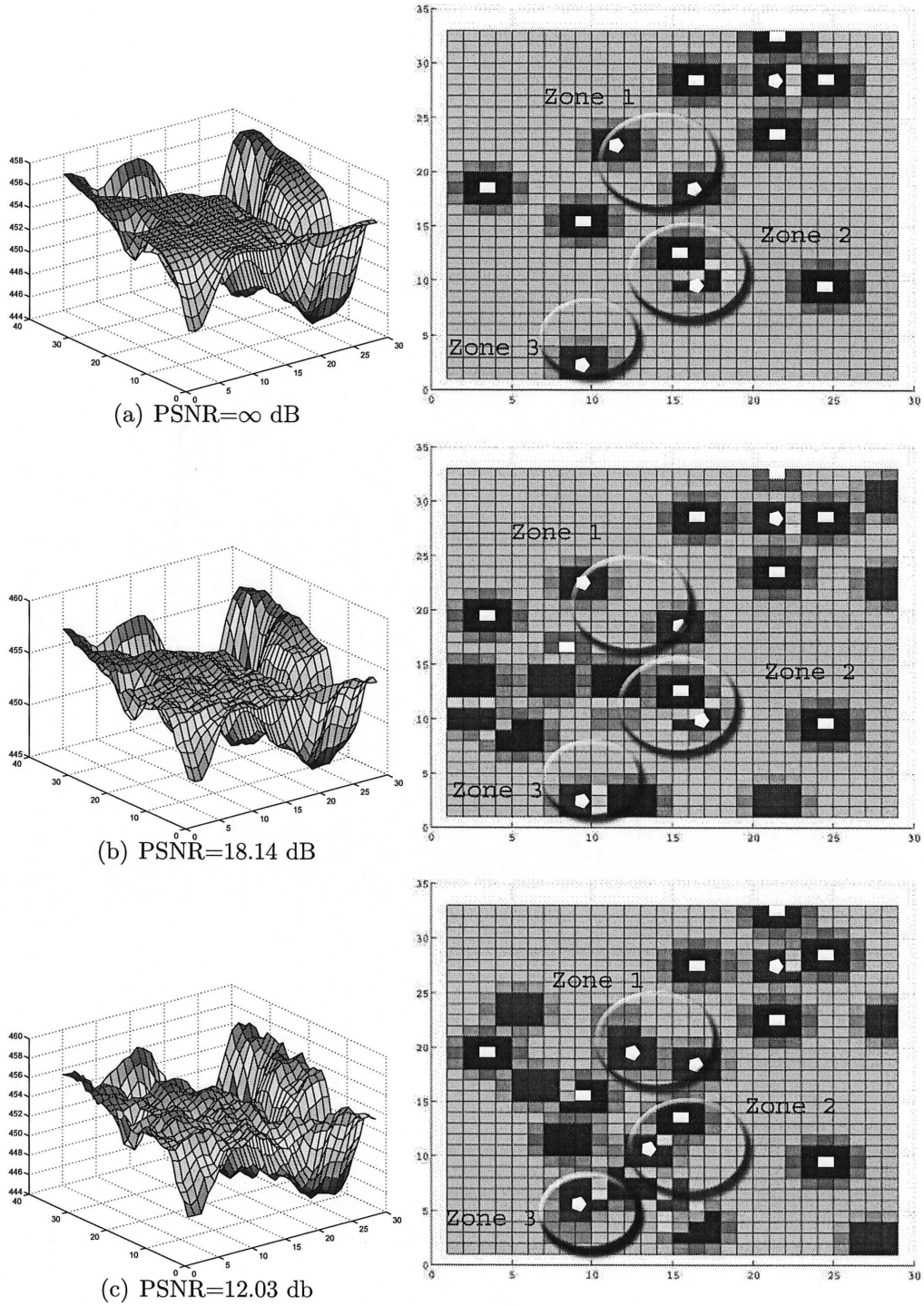
la grande robustesse au bruit de l'algorithme de détection des marqueurs. Les points blancs indiquent les marqueurs en commun entre les images. Ceci s'explique par le lissage des données qui a tendance à légèrement diminuer l'influence du bruit. En effet, le PSNR en entrée pour la deuxième et la troisième ligne est de respectivement 18.14 dB et 12.034 dB, ce qui est assez faible. Si cela affecte la qualité des images, les points principaux, eux, sont conservés, même avec un PSNR à 12 dB ce qui est vraiment excellent. Néanmoins, le dernier exemple illustre les limites de l'algorithme. En effet, de faux marqueurs sont rajoutés et dans les zones 1, 2 et 3, les principaux marqueurs commencent à avoir leurs positions légèrement modifiées : ceci entraînera nécessairement une erreur sur le calcul de la vitesse relative. Le bruit, devenant très important, peut pousser l'algorithme à détecter de faux marqueurs qui nécessairement vont influencer les vrais et donc modifier légèrement leurs positions. Le principe de compétition joue alors contre le but recherché. Cependant, les cas présentés ici sont extrêmes et ne devraient pas être rencontrés dans un atterrissage autonome réel. Ces résultats démontrent donc un autre avantage de la méthode développée dans ce mémoire : sa stabilité au bruit. Néanmoins, il semble qu'il ne faille pas dépasser en général un PSNR de 18 dB pour garantir de bons résultats.

5.1.6 Effet du type de terrain sur le résultat

Il est utile de voir si l'algorithme de détection des marqueurs est robuste à différents types de terrain. Ce serait un grand avantage si, indépendamment de la topographie du terrain, l'algorithme trouvait des marqueurs de façon uniforme dans l'image. Les résultats suivants sont deux cas extrêmes. La figure 5.7 représente une grande plaine plate avec une grande montagne. Cette image est totalement artificielle et a été générée à partir du logiciel Terragen [Terragen, 2004].

La simplicité de l'image comporte cependant une grande difficulté. En effet, la plaine étant très plate et la montagne très grande, il est difficile de détecter des points à suivre. Le fait d'avoir concu un algorithme de détection des marqueurs totalement relatif aux matrices va permettre d'être robuste à ce type de terrain.

La figure 5.8 montre que des marqueurs ont été détectés dans la plaine ce qui permet d'être



(a) PSNR=∞ dB

(b) PSNR=18.14 dB

(c) PSNR=12.03 db

Figure 5.6 – Effet du bruit sur l'algorithme du bloc B. Ces résultats ont été obtenus avec 10 itérations, 10 % pour la taille du rayon d'influence et une taille de la fenêtre F de 8 %. Les points blancs indiquent les marqueurs en commun au trois images

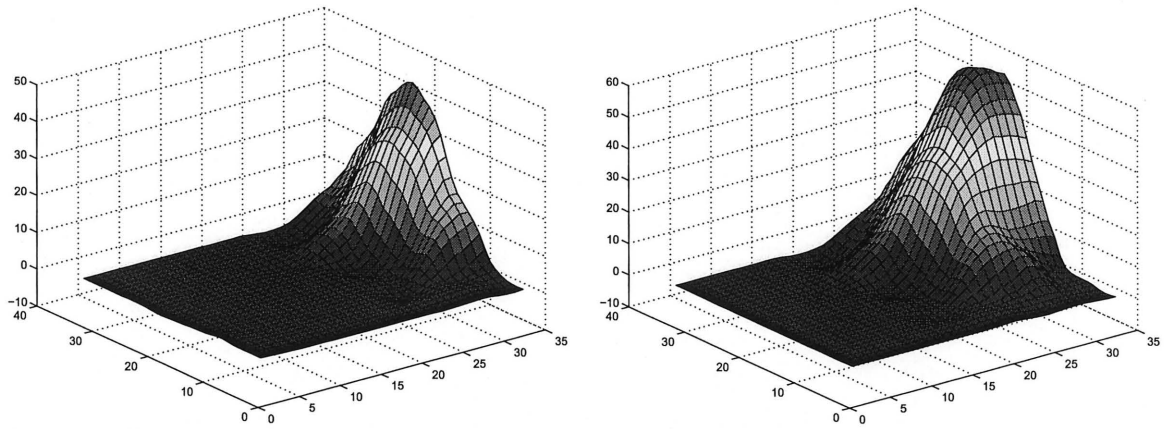


Figure 5.7 – Exemple de plaine plate avec une grande montagne

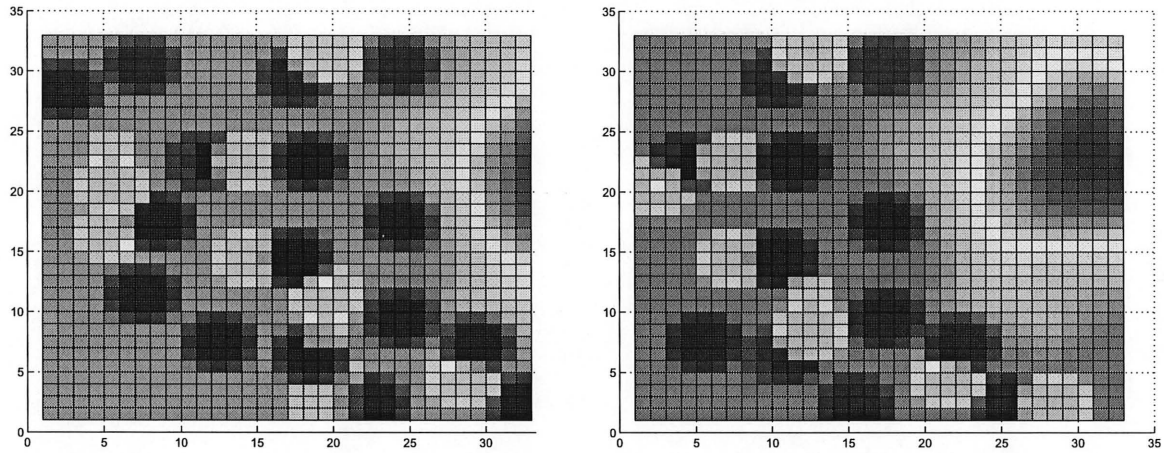


Figure 5.8 – Marqueurs détectés par l'algorithme dans une plaine avec une grande montagne

rassuré quant au bon fonctionnement de l'algorithme de *tracking*. De plus, ces marqueurs sont repérés à des endroits identiques dans les deux matrices consécutives ce qui permet d'être confiant en la précision de l'estimation de la vitesse. Les minimums et maximums locaux agrandis par l'algorithme de détection des marqueurs permet donc de mieux séparer les informations importantes de l'image. L'algorithme de détection permet donc de minimiser l'influence du relief de la surface scannée pour déterminer les points les plus remarquables dans la matrice.

Le prochain exemple montre un terrain très accidenté, comme une série de vallées encaissées, à la figure 5.9. La difficulté réside ici dans la recherche elle-même de marqueurs. En effet,

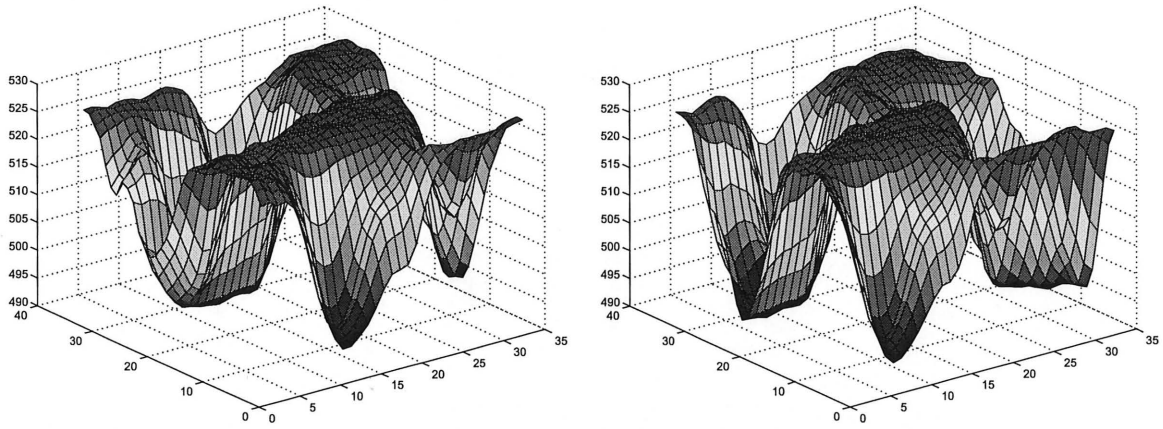


Figure 5.9 – Exemple d'une série de vallées encaissées

les flancs des pentes ne peuvent pas contenir de minimums ou de maximums locaux. Il y a donc un risque de ne pas trouver de marqueurs. L'algorithme semble cependant réussir à retrouver des marqueurs dans les deux matrices, comme le montre la figure 5.10. Ces résultats ne sont néanmoins pas parfaits. Cela permet de dire que l'algorithme développé est légèrement sensible au trop forte variation d'amplitude de la surface. En plus de contenir une surface essentiellement composée de pente forte, les amplitudes sont grande dans cette matrice. Le maximum global et le minimum global, G_{max} et G_{min}, sont donc très proches des valeurs de certains minimums et maximums locaux faussant ainsi légèrement la compétition. Néanmoins, il est peu probable qu'un atterrissage autonome d'une sonde soit décidé dans une telle zone, cela représente donc un cas extrême qui ne risque pas de se présenter en pratique.

Les résultats de l'algorithme de détection de marqueurs exposés dans ce mémoire montrent les différents avantages de cette méthode. Il est néanmoins difficile de déterminer avec exactitude le taux de réussite de l'algorithme. La plupart des tests de reliefs réalistes semblent indiquer que l'algorithme de détection est très bon et génère des marqueurs utilisables dans tous les cas étudiés. Il faudrait répéter de nombreux tests supplémentaires pour confirmer les bons résultats observés dans un niveau de confiance acceptable.

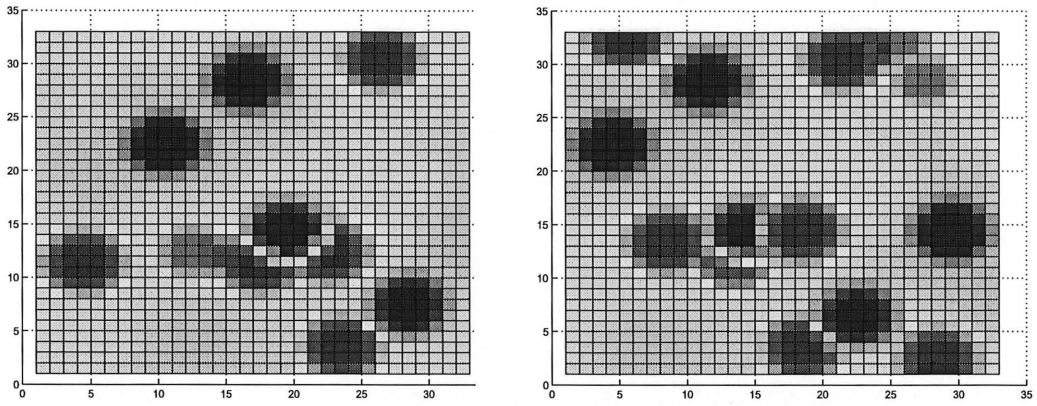
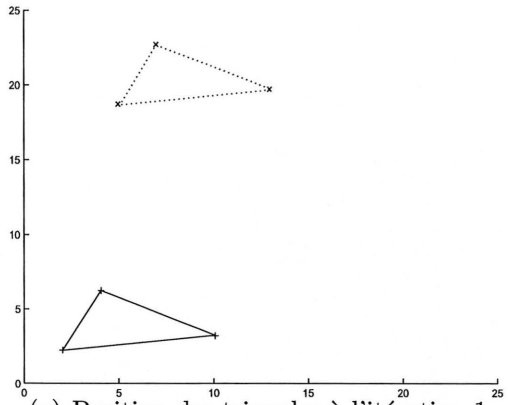


Figure 5.10 – Marqueurs détectés par l'algorithme dans une série de vallées encaissées

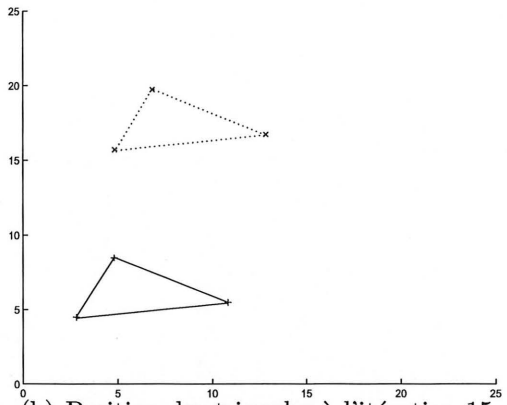
5.2 Résultats de l'algorithme de *tracking*

5.2.1 Limitation de la distance entre deux images

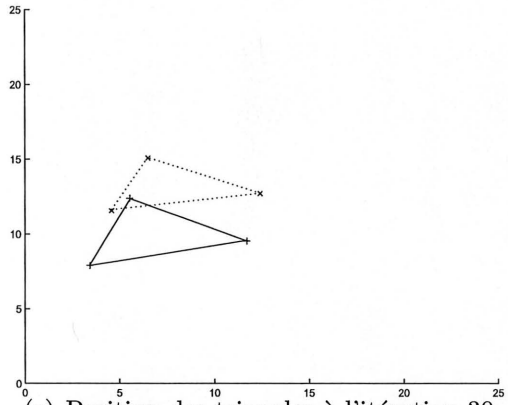
La limitation de la translation entre deux images prises par le LIDAR est le paramètre principal. De celui-ci dépend la vitesse horizontale limite au-delà de laquelle l'algorithme ne peut plus fonctionner. La séquence d'image à la figure 5.11, montre donc deux formes géométriques. La première, un triangle en ligne pleine, correspond à la position de la figure dans la première image. La deuxième, en ligne pointillée, correspond à la position de la figure dans la deuxième image. La distance entre les deux figures est de 17.26 unités ce qui correspond pour une matrice de dimension 35 à un déplacement total d'une demi-image, la limite conceptuelle pour ce projet. Cette séquence montre, qu'apparemment, l'algorithme de *tracking* n'a aucune difficulté à suivre des points séparés d'une distance égale à 50 % de la taille d'une image. La forme des figures est de plus parfaitement conservée tout au long du processus. À la figure 5.11(d) de la séquence, les deux triangles sont parfaitement superposés montrant donc que l'algorithme semble parfaitement tenir les spécifications. Néanmoins, des tests effectués sur des données réelles semblent indiquer que la limite réelle est plutôt de l'ordre de 20 % de la taille de l'image. L'explication semble être qu'une image réelle peut facilement contenir des constellations de points similaires, trompant ainsi l'algorithme si la différence entre deux images est trop grande.



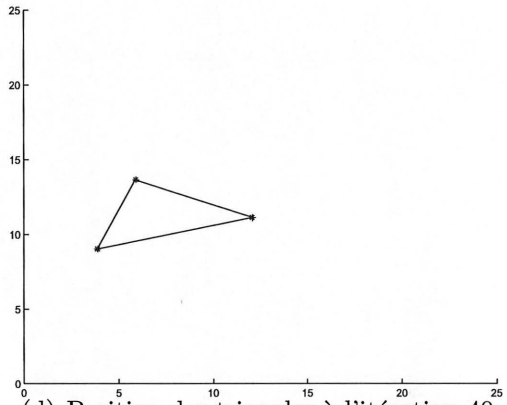
(a) Position des triangles à l'itération 1



(b) Position des triangles à l'itération 15



(c) Position des triangles à l'itération 30



(d) Position des triangles à l'itération 40

Figure 5.11 – Résultats de l'algorithme de *tracking* avec une différence entre les deux images de 50 %.

La figure 5.12 montre une autre série de tests qui essaie de tromper l'algorithme. Cette fois-ci, l'algorithme n'a pas été capable de retrouver tous les points communs aux deux images représentées par les figures en ligne pleine et en ligne pointillée. Le point qui est près de la position (5,5) dans l'image 5.12(a) est le responsable. Ainsi, l'algorithme de *tracking* ne semble pas, dans les premières itérations (image 5.12(b)), influencé par ce point mais finit par être déstabilisé (images 5.12(c), 5.12(d), 5.12(e) et 5.12(f)) dès que les points des deux triangles se rapprochent. Après de nombreux essais, il a été constaté que plus la distance est grande entre les deux images, plus l'algorithme a des chances de se tromper. Cette constatation est finalement très logique. En effet, plus la distance est grande entre deux images et plus il y a de chance d'avoir des points perturbateurs qui finissent par déstabiliser en partie ou complètement le processus de compétition. Dans l'image 5.12(f), l'algorithme

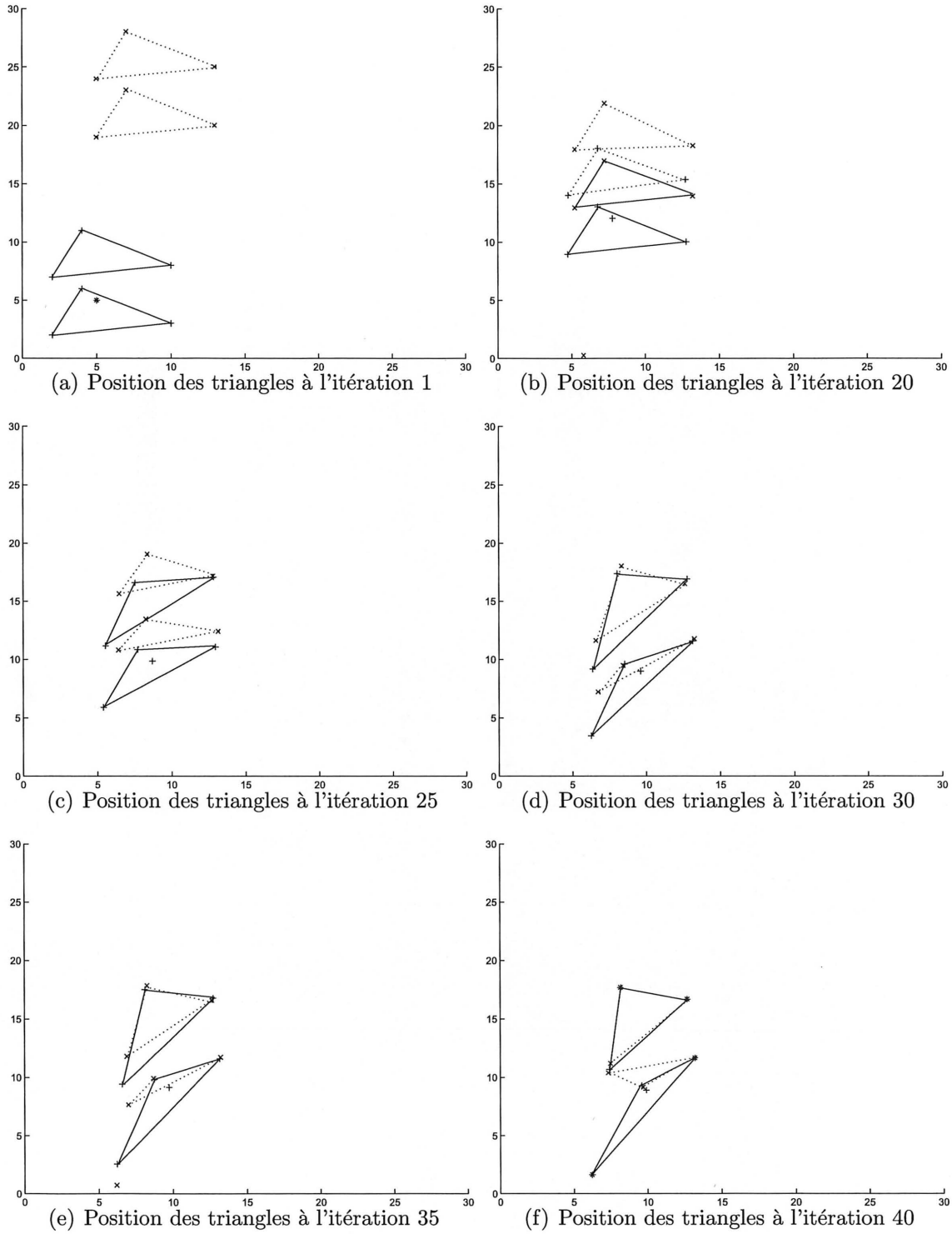


Figure 5.12 – Résultats de l'algorithme de *tracking* avec une différence entre les deux images de 50 %, composées de deux triangles et un point incorrect.

réussit cependant à donner la bonne réponse sur 3 des 7 points composant les deux images. Après de nombreux essais, il semble que la limite entre deux images se situe entre 20 et 30 pourcent de la taille d'une image dépendant du bruit qu'elle contient. La section suivante présentera les résultats finaux obtenus avec tous les modules. La précision de la méthode sera aussi évaluée.

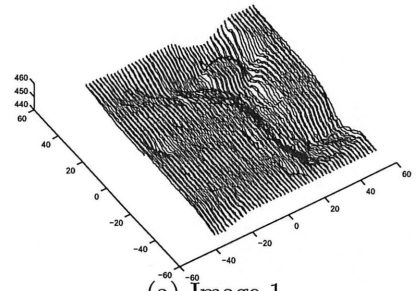
5.3 Résultats finaux

Cette partie décrit les résultats de l'algorithme ME-SOM selon différents tests. La figure 5.13 montre les principales étapes du ME-SOM pour obtenir la vitesse relative à partir des mesures du LIDAR effectuées en laboratoire. Cette simulation, comme la plupart des images dans ce mémoire, a été faite à partir de véritables données LIDAR et non des images générées par le logiciel Terragen. Il est à noter que l'image 5.13(b) comporte une erreur sur la partie supérieure. Cela est dû tout simplement au bord de la maquette scannée par le LIDAR. Ceci démontre une fois de plus que l'algorithme ME-SOM est robuste, puisqu'il trouvera le bon résultat.

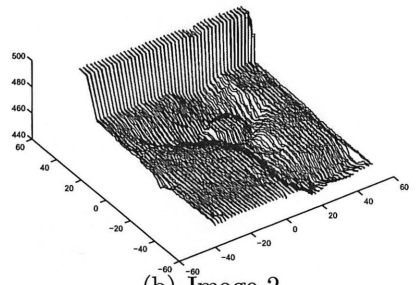
Une bonne précision de l'algorithme est nécessaire pour valider la méthode proposée. Une erreur de mesure supérieure à 10 % est ainsi considérée trop grande pour être utilisable. Il est donc intéressant de voir si la solution proposée respecte cette spécification.

5.3.1 Précision de l'estimation

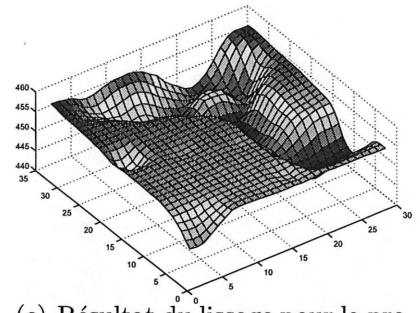
Il est délicat de donner la précision globale de l'algorithme ME-SOM, car plusieurs facteurs entrent en jeux. Le LIDAR prend des mesures discrètes et non continues. Il se peut très bien, par exemple, qu'un sommet de montagne soit présent dans la première image et que dans la seconde le LIDAR tape juste à coté. La précision de l'estimation est donc directement reliée à la résolution horizontale du LIDAR. De même, le fait de réduire l'image de dimension 100x100 à une matrice de 35x35 fait perdre encore de la précision. L'algorithme d'estimation ne peut donc pas être plus précis que la matrice 35x35. Le bruit de l'image peut aussi jouer sur la précision de l'algorithme. Là encore, il est possible d'estimer que le bruit des données



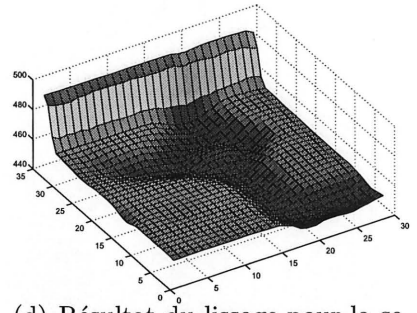
(a) Image 1



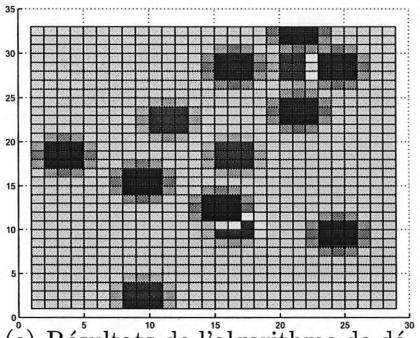
(b) Image 2



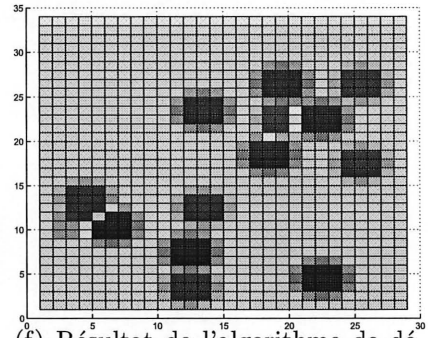
(c) Résultat du lissage pour la première image



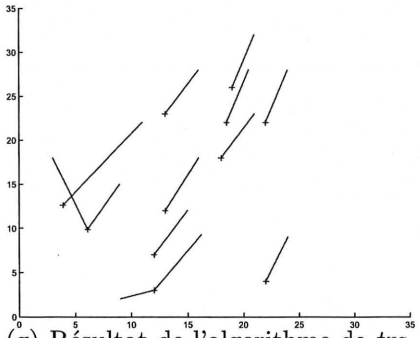
(d) Résultat du lissage pour la seconde image



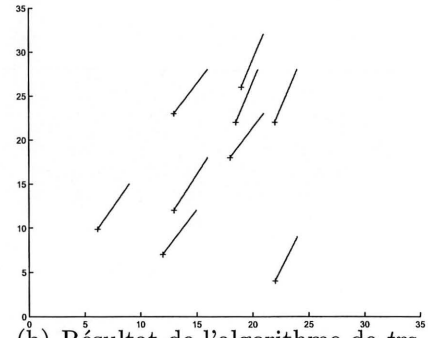
(e) Résultats de l'algorithme de détection des marqueurs pour la matrice 1



(f) Résultat de l'algorithme de détection des marqueurs pour la matrice 1



(g) Résultat de l'algorithme de *tracking* sans le BGA



(h) Résultat de l'algorithme de *tracking* avec le BGA

Figure 5.13 – Résultats obtenus avec les mesures du LIDAR

obtenues à partir du LIDAR dans le laboratoire est un cas extrême. L'une des propriétés du LIDAR est d'avoir un bruit de mesure constant quelque soit la distance de la cible. Il y a donc relativement plus de bruit dans les images en laboratoire que dans une situation réelle où la cible se trouve à plus de 1km contre à peine 10 m dans le laboratoire. Cependant même si ce bruit de mesure était plus faible, la précision de l'estimation serait toujours dépendante de la résolution horizontale. La résolution horizontale est donc le facteur limitant. Lors des tests avec des mesures prises dans le laboratoire, l'erreur de précision est en dessous de 10 %. Avec des images générées avec Terragen, l'erreur est réduite à moins de 8 %. Cela semble logique, car il n'y a alors plus de bruit. Les marqueurs bougent donc très peu pendant la phase de détection, la précision en est donc nettement améliorée. Les tests comportant un bruit rajouté, équivalent à un PSNR de 20dB, ont montré cependant que l'erreur de précision pouvait monter jusqu'à 15 %. Au-delà de cette valeur, le réseau de Kohonen s'occupant du *tracking* a souvent tendance à diverger.

La figure 5.14 montre l'erreur sur la vitesse en fonction de la distance horizontale parcourue entre deux prises d'image. Cette fois-ci, dans le but d'avoir des mesures précises, il a été décidé d'utiliser des images artificielles. En effet, il est plus facile de faire varier la différence entre deux images avec des images artificielles, le déplacement exact est connu. Dans cet exemple, la distance varie de 0 à 35 % de la taille des images. La taille des images est de 100x100. Les résultats obtenus sont illustrés avec la moyenne des distances trouvée par l'algorithme de *tracking* en noir et avec le BGA en gris.

La première constatation est que le traitement des données avec le BGA améliore la stabilité des résultats. La courbe grise représentant ces résultats est souvent contenue dans une erreur de 10 %. La stabilité est beaucoup plus grande surtout avec des distances faibles. En effet, dans la première partie de la courbe, même si l'erreur commise est grande, les résultats avec le BGA sont nettement supérieurs. Avec des grandes distances, l'algorithme avec BGA décroche à partir d'une distance de 27 % de la taille de l'image. Le graphe semble suggérer que la moyenne directe des résultats de l'algorithme de *tracking* est alors meilleure. En fait, il semblerait que cela soit une erreur, car les résultats de la figure 5.15, montre que le réseau de Kohonen s'occupant du *tracking* a complètement divergé.

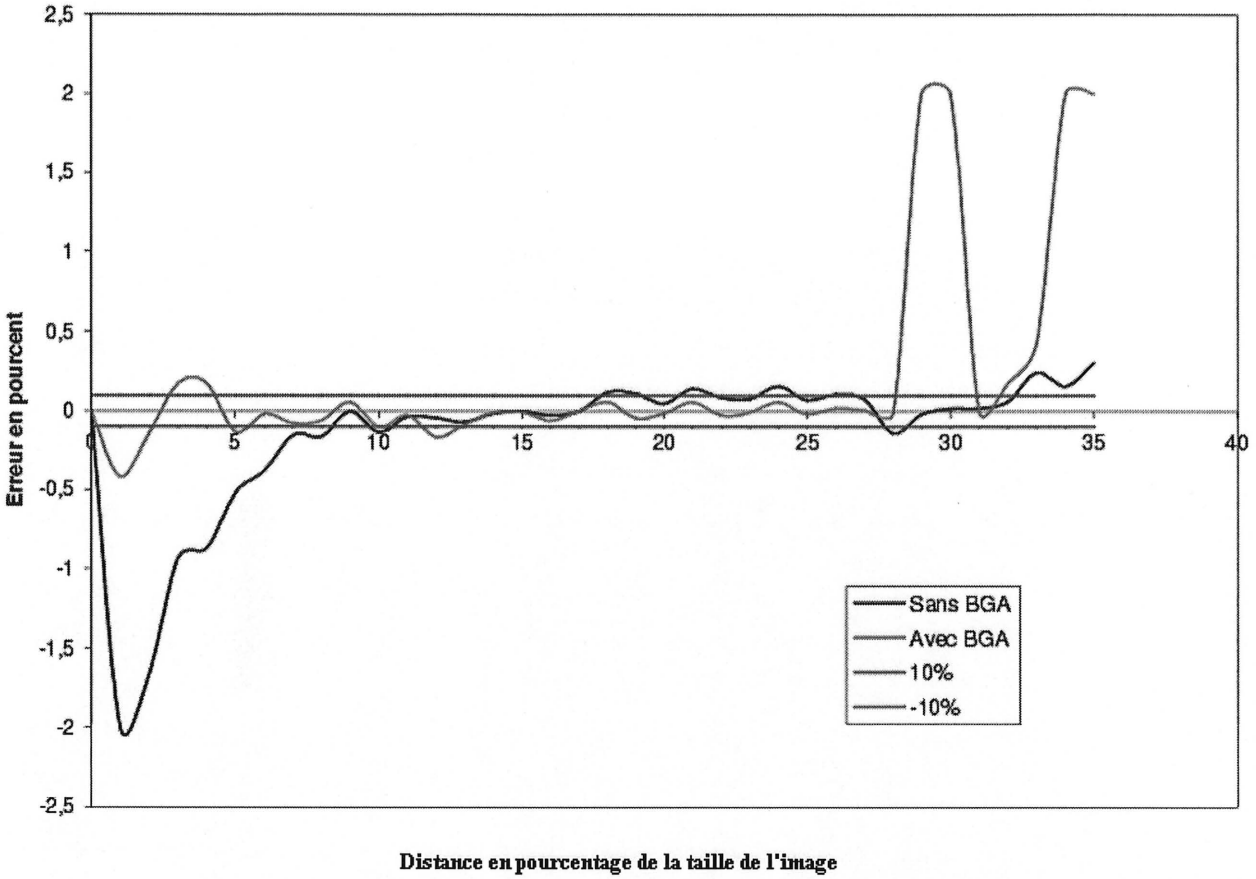


Figure 5.14 – Erreur en pourcent sur la vitesse en fonction de la distance parcourue entre deux prises d'images

La deuxième constatation est que l'erreur pour de faibles distances est grande. Cela peut en effet être un problème puisque les spécifications étaient d'avoir une erreur en pourcent toujours inférieure à 10 %. Cette erreur résulte du lissage des données et plus particulièrement de la réduction de la dimension de l'image d'origine en une matrice de dimension 35x35. En effet, avec un déplacement de 1 unité de mesure sur l'image originale, les marqueurs dans la matrice 35x35 vont réagir de façon différente : certains vont avoir leurs places modifiées d'autres non. Il est donc difficile pour l'algorithme d'être précis pour de petits déplacements inférieurs au déplacement ΔF qui permet de construire la matrice 35x35. De plus, si l'erreur en pourcent semble grande, elle est très faible en valeur. Par exemple, pour un déplacement de une unité de mesure d'une matrice de taille 100x100, le ME-SOM trouve une vitesse de 0.4714 unités de mesure ce qui donne une erreur de 0.5286 unité de mesure ce qui est assez

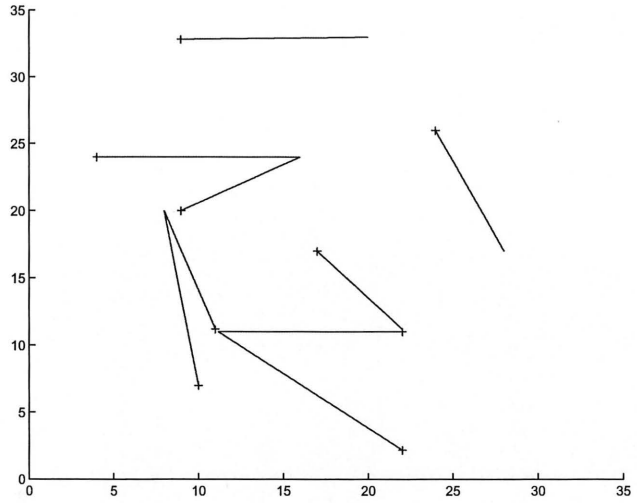


Figure 5.15 – Résultat de l'algorithme de *tracking* avec une distance de 27 % de la taille d'une image

faible en valeur.

La dernière constatation est que les résultats de l'algorithme semblent osciller autour de zéro. Là encore le passage à une matrice 35x35 en est la cause. Selon que la position du point étudié soit avant ou après un déplacement ΔF entier, le déplacement des marqueurs sera inférieur ou supérieur au déplacement réel. Par exemple, avec une matrice de taille 35x35 construite à partir d'une image originale avec un déplacement ΔF égale à 5, un déplacement de 3 engendrera une erreur positive alors qu'avec un déplacement de 6, l'erreur sera négative.

5.3.2 Taux de réussite de l'algorithme ME-SOM

Il est là aussi délicat, comme dans la plupart des méthodes dites d'intelligence artificielle, de donner un taux de réussite. En effet, ce taux peut facilement varier en fonction du bruit et surtout de la configuration des marqueurs, en particulier de la distance entre deux images successives. Lors des essais, le taux de réussite est de l'ordre de 97 % sur des mesures réelles si la distance entre les deux images est inférieure à 20 %. Cependant, ce taux diminue rapidement en fonction de la distance entre les deux images, à 35 % le taux de réussite est quasiment nul même avec des images sans bruit. Dans la pratique il est donc nécessaire de

se limiter à un déplacement inférieur à 20 % pour être sûr d'obtenir de bon résultat quelle que soit la qualités des images.

5.4 Réaction de l'algorithme à la rotation

Cette section ne fait pas partie du cahier des charges de ce projet. Néanmoins, il est utile de voir la réaction de l'algorithme face à une rotation entre deux images successives. En effet, il serait intéressant que la solution proposée soit robuste à la rotation résiduelle non-corrigée ou même puisse se passer de correction. Le LIDAR permettrait alors de calculer la vitesse relative, de chercher un site sécuritaire tout en calculant la vitesse de rotation de la sonde ce qui permettrait d'avoir la valeur de la rotation à partir de deux sources : le gyroscope et le LIDAR. Il n'est pas ici question de prouver l'efficacité du ME-SOM dans de telles conditions mais plutôt de montrer son potentiel. Les données utilisées ont été générées avec le logiciel Terragen et ne comportent pas de bruit. La figure 5.16 montre les résultats du ME-SOM avec une rotation de 0, 5, 15, 25 et 35 degrés.

Les résultats sont très bons, comme le montre la figure 5.16. En effet, l'algorithme de détection des marqueurs ne semble pas être influencé par une rotation et l'algorithme de *tracking* est sensé selon Labonté supporter de petites rotations. Dans les faits, le ME-SOM semble robuste jusqu'à un angle de 35 degrés. Il serait donc sans doute possible d'utiliser la méthode proposée dans ce mémoire pour corriger la rotation. Ceci abaisserait les temps de calculs et diminuerait les coûts d'un atterrissage sur une planète.

La section suivante montre les résultats du ME-SOM avec une rotation suivie d'une translation ce qui s'approche le plus d'une situation réelle. La figure 5.17 montre les différents résultats qui sont convaincants pour cet exemple. Même s'ils comportent quelques erreurs, les principaux points de l'image sont suivis.

Un angle de 30 degrés avec une translation de 15 % entre les deux matrices consécutives semblent être la limite après laquelle l'algorithme ME-SOM diverge. De nombreux tests avec des données réelles seraient encore nécessaires pour valider la méthode présentée dans ce mémoire en présence de rotation. La conception de l'algorithme est totalement relative à

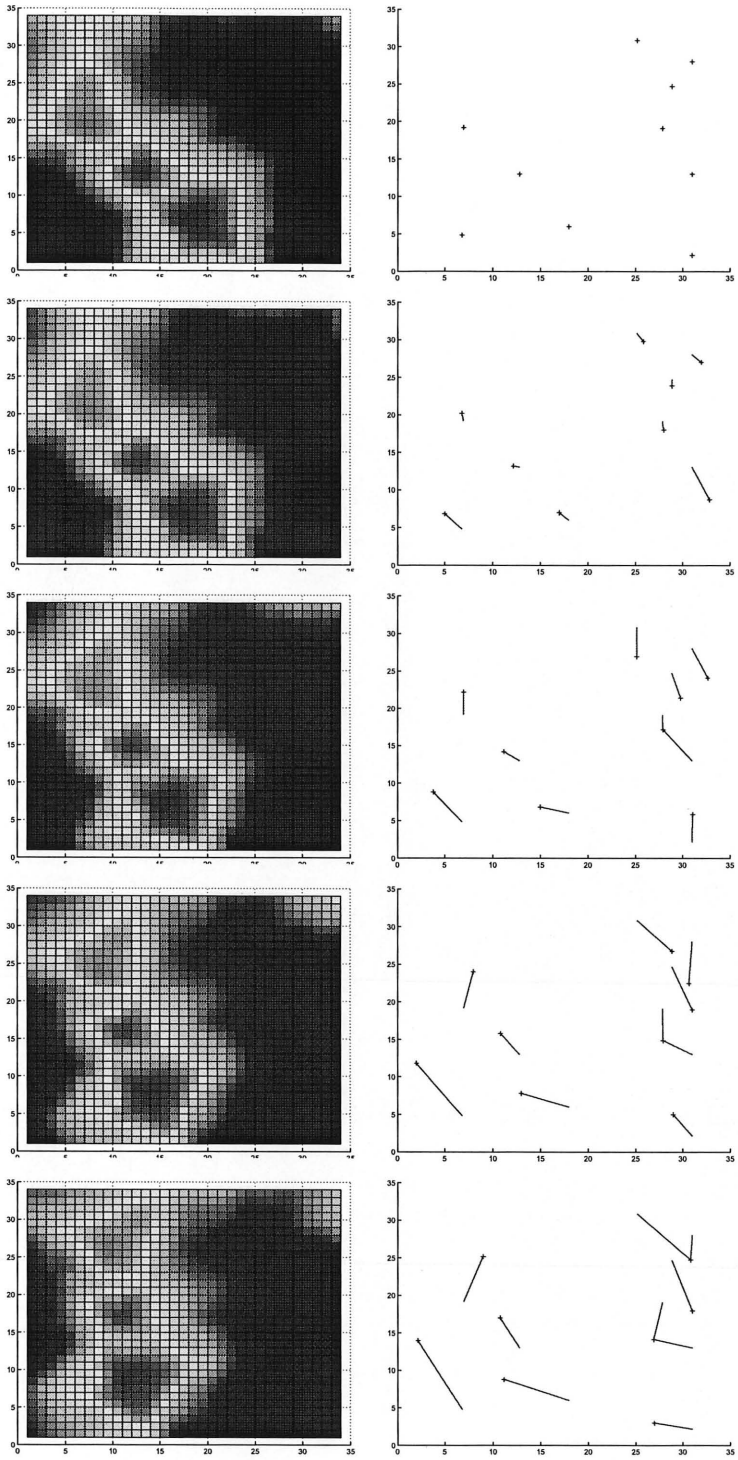


Figure 5.16 – Résultats du ME-SOM avec différents angles de rotation. La première ligne donne le résultat pour un angle de 0 degré, la deuxième un angle de 5 degrés, la troisième un angle de 15, la quatrième un angle de 25 et la dernière un angle de 35 degrés.

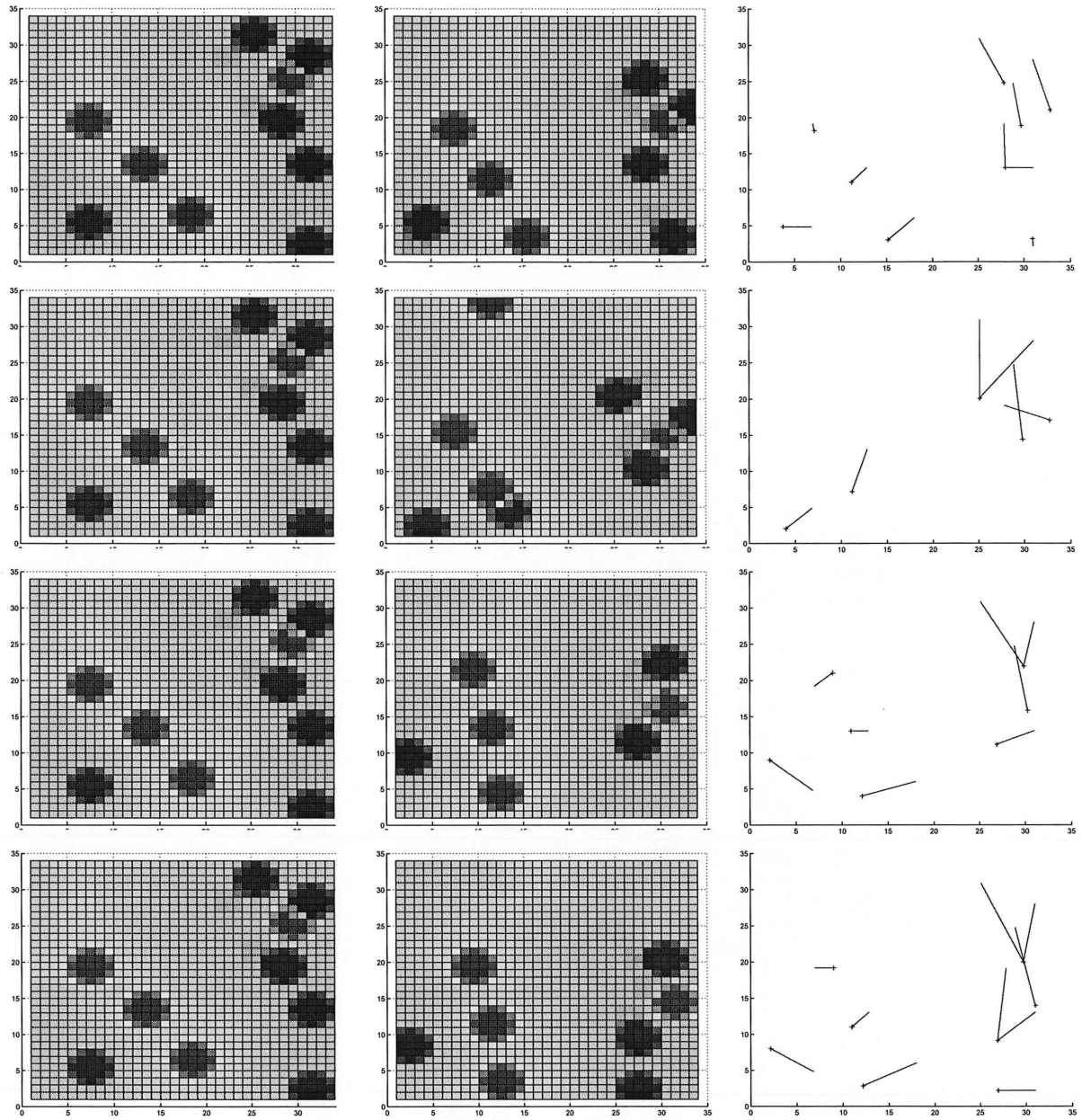


Figure 5.17 – Résultats du ME-SOM avec différentes rotations et différentes translations. La première ligne donne le résultat du ME-SOM pour un angle de 15 degrés et une translation de 10 %, la deuxième un angle de 15 degrés et une translation de 20 %, la troisième un angle de 30 degrés et une translation de 10 % et la dernière un angle de 30 degrés et une translation de 15 %

chaque image, et indépendante des paramètres extérieurs comme la différence de hauteur ou la rotation (inférieure à 30 degrés), il y a donc de bonnes raisons de croire que les résultats seront concluants.

CHAPITRE 6

CONCLUSION

Ce projet de maîtrise décrit une méthode originale permettant d'estimer la vitesse relative d'une sonde par rapport au sol afin d'atterrir sur le site choisi lors d'une phase d'atterrissage sur une planète telle que Mars. L'objectif principal est donc de concevoir un estimateur de vitesse avec des méthodes dites d'intelligence artificielle. Bien évidemment, la recherche de la solution a conduit à privilégier une solution autonome, l'interaction avec l'homme étant impossible lors de la phase d'atterrissage vu les temps de réponse.

Le LIDAR aidé de l'algorithme ME-SOM décrit dans ce mémoire, permet ainsi de déterminer la vitesse de déplacement. La méthode consiste à déterminer les points marquants de deux images consécutives et de trouver ensuite les points qu'elles ont en commun pour trouver le déplacement et la vitesse relative. L'utilisation de réseaux de Kohonen a démontré leur efficacité dans cette problématique. D'abord, le degré d'atteinte des objectifs est évalué, puis les principales contributions apportées sont mises en évidence .

6.1 Évaluation des objectifs

La méthode proposée dans ce mémoire avait pour but principal les objectifs suivants :

1. Déduire à l'aide seule des données du LIDAR la vitesse relative d'une sonde lors d'une phase d'atterrissage.
2. Être robuste aux différents reliefs que peut comporter une mission d'atterrissage.
3. Être robuste à la différence de hauteur de prise de vue entre deux balayages de surface consécutifs.
4. Avoir une erreur de précision sur la vitesse inférieure à 10 %.
5. Être capable de pourvoir estimer la vitesse avec un déplacement maximal entre deux images de 50 % de la taille d'une image.

Voici en résumé les avancées positives de ce projet de maîtrise :

Premièrement, l'utilisation d'algorithme d'intelligence artificielle et plus particulièrement de réseaux de Kohonen couplés au LIDAR, a démontré son efficacité pour ce problème. En effet, les résultats obtenus sont de très bonne qualité.

Deuxièmement, l'utilisation de réseaux de Kohonen a permis de se dégager du problème des données d'entraînement. Cela permet d'affirmer que les résultats de l'algorithme ainsi développé sont théoriquement indépendants du type de relief étudié, sauf les plus accidentés, mais la recherche de site d'atterrissage exclut ce type de relief risqué. Néanmoins une étude sur une plus grande quantité de données pourrait définitivement valider ce point.

Troisièmement, la différence de hauteur entre deux images consécutives peut être un handicap lors de la recherche de marqueurs communs entre deux images. Il était donc impératif d'être totalement robuste à cette situation bien particulière à une phase d'atterrissage. La conception de l'algorithme étant particulièrement bien adapté à ce problème, les différents tests effectués ont tous été concluants, car le système prend en compte uniquement le centre de gravité des taches construites par l'algorithme de détection de marqueurs.

Quatrièmement, l'erreur sur l'estimation a été vérifiée dans les différents tests effectués. La vitesse relative est donc dans tous les cas normaux d'utilisation inférieure au 10 % demandé dans les spécifications du projet. En mettant l'algorithme dans des situations extrêmes, l'erreur ne dépasse jamais 15 %. Si l'erreur dépasse cette valeur, c'est que l'algorithme a divergé. La précision de l'algorithme est donc suffisante pour contrôler une sonde.

Finalement, le dernier point semble être beaucoup plus difficile à obtenir. Si la condition d'un recouvrement minimum de 50 % semble possible sur des images parfaites, sans bruit et sans marqueurs perturbateurs, il semble que dans la pratique une distance supérieure à 20 % soit une limite difficile à dépasser. Il faut aussi considérer qu'avec le peu d'informations (seules les coordonnées des marqueurs sont connues) fournies à l'algorithme de *tracking* même un être humain ne pourrait pas toujours trouver la bonne réponse. Il faut donc sûrement reconsidérer une autre approche pour dépasser la barrière des 20%.

6.2 Travaux futurs

La méthode développée dans ce mémoire doit être plus profondément testée. La définition des nombreux paramètres la composant doit être aussi totalement confirmée et optimisée, comme la taille des rayons d'influence ou le nombre d'itérations qui ont un impact direct sur les résultats et la rapidité de l'algorithme. Le problème de la limite de déplacement entre deux images peut être évité en baissant le nombre de points par balayage et ainsi augmenter la vitesse d'acquisition : prendre 5000 points toutes les demi-secondes au lieu des 10000 toutes les secondes. Une autre méthode consisterait avec l'estimation de la vitesse faite à chaque mesure d'essayer de rapprocher les parties communes des deux matrices consécutives pour retomber dans un cas ou le déplacement est inférieur à 20 %.

Des travaux sur l'amélioration de l'algorithme dans le cas de rotation et surtout de combinaison de rotation et de translation amélioreraient beaucoup la méthode. Néanmoins, avec une limite sur l'angle de 30 degrés et un déplacement de 15 %, ces premiers résultats sont déjà très bons. L'amélioration de la méthode permettrait aussi de diminuer les corrections qui sont effectuées sur les images du LIDAR. Le gain de temps de calcul résultant réduirait le coût de la mission ce qui est indirectement aussi l'un des objectifs de ce projet de recherche.

Le dernier point sera d'adapter la complexité de l'algorithme pour une plateforme portable. En effet, l'analyse et l'amélioration de l'architecture de l'algorithme devraient permettre d'utiliser cet algorithme sur un système embarqué ayant une puissance de calcul raisonnable.

6.3 Conclusion

Ce mémoire propose une solution innovante pour déterminer la vitesse relative d'une sonde lors d'une phase d'atterrissage. L'utilisation des réseaux de Kohonen dans une application dans le domaine de l'aérospatiale est en effet très rare. Les résultats obtenus laissent cependant penser que l'intelligence artificielle a sa place dans ce domaine. Le LIDAR semble être aussi une technologie prometteuse. Elle possède en effet toutes les qualités requises pour permettre un atterrissage sur une planète. La méthode utilisée pour traiter les données pourra

remplacer une grande partie des instruments sur une sonde permettant ainsi une plus grande fiabilité aux pannes matérielles et peut être des coûts inférieurs.

En conclusion, ce projet ouvre la voie au développement de nouvelles technologies nécessaires à la conception de systèmes d'atterrissage, économiques, précis, efficaces et autonomes.

Références

ADELSON, E. H., ANDERSON, C. H., BERGEN, J. R., BURT, P. J., OGDEN, J. M. (1984). *Pyramid methods in image processing.* RCA Engineer, volume 29, numéro 6, pages 33–41.

AGOURIS, P., GYFTAKIS, S., STEFANIDIS, A. (1998). *Using a fuzzy supervisor for object extraction within an integrated geospatial environment.* International Archives of Photogrammetry and Remote Sensing, volume 32, pages 191–195.

BENEDETTI, A., PERONA, P. (1998). *Real-time 2-D feature detection on reconfigurable computer.* Proceedings of the IEEE Conference Computer Vision and Pattern Recognition, pages 586–593.

BOROTSCHNIG, H., PIMZ, A., SINCLAIR, D. (1996). *Fuzzy graph tracking.* Proceedings of the 5th Symposium for Intelligent Robotics Systems, pages 91–101.

CIRRINCIONE, G., CIRRINCIONE, M. (2003). *A novel self-organizing neural network for motion segmention.* Applied Intelligence, volume 18, pages 27–35.

DE LAFONTAINE, J., GUEYE, O. (2004). *Autonomous planetary landing using a LIDAR sensor : The navigation function.* volume 24, pages 7–18.

EVANGELOU, I. (1999). *Automatic segmentation of magnetic resonance image (MIRs) of the humain brain using self-organizing maps (SOMs).* Mémoire de maîtrise, Clarkson University.

EVANGELOU, I., HADJIMITSIS, D., LAZAKIDOU, A., CLAYTON, C. (2001). *Data mining and knowledge discovery in complex image data using artificial neural networks.* 17th International Conference on Logic Programming ICLP 2001.

GARCIA-BARROSO, C., SOBREVILLA, P., MONTSENY, E. (1995). *Step edge detection using fuzzy algorithms.* Proceedings of the 6th International Conference on Computer Analysis of Images and Patterns, pages 411–416.

HAYKIN, S. (1999a). *Neural networks, a comprehensive foundation*, Upper Saddle River, NJ, USA, International édition. Prentice Hall International, 842 pages.

HAYKIN, S. (1999b). *Neural networks, a comprehensive foundation*, International édition, chapitre 14.7, pages 680. Upper Saddle River, NJ, USA, Prentice Hall International.

HAYKIN, S. (1999c). *Neural networks, a comprehensive foundation*, International édition, chapitre 9.3, pages 446. Upper Saddle River, NJ, USA, Prentice Hall International.

JOHNSON, A. E., MATTHIES, L. H. (1999). *Precise image-based motion estimation for autonomous small body exploration.* Proceedings of the 5th International Symposium on Artificial Intelligence, Robotics and Automation in Space, pages 627–634.

KOHONEN, T. (1982). *Self-organized formation of topologically correct feature maps.* Biological Cybernetics, volume 43, pages 59–69.

LABONTÉ, G. (1998). *A SOM neural network that reveals continuous displacement fields.* Proceedings of the International Joint Conference on Neural Networks, pages 880–884.

LIU, X., WANG, D., RAMIREZ, J. R. (1998a). *Extracting hydrographic object from satellite images using a two layer neural network.* Proceedings of the IJCNN, pages 897–902.

LIU, X., WANG, D., RAMIREZ, J. R. (1998b). *A two-layer neural network for robust image segmentation and its application in revising hydro graphics.* International Archives of Photogrammetry and Remote Sensing, volume 32, pages 464–472.

LU, S. W., SHEN, J. (1996). *Artificial neural networks for boundary extraction.* Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, volume 3, pages 2270–2275.

MATLAB (2004). *http ://www.mathworks.com/*, The Mathworks.

MIOSSO, C. J., BAUCHSPIESS, A. (2001). *Fuzzy inference system applied to edge detection in digital images.* Proceeding of the Brazilian Conference on Neural Networks, pages 481–486.

MISU, T., HASHIMOTO, T., NINOMIYA, K. (1999). *Optical guidance for autonomous landing of spacecraft.* IEEE Transactions on Aerospace and Electronic systems, volume 35, numéro 2, pages 459–473.

SADJA, P., FINKEL, L. H. (1992a). *A neural network model of object segmentation and feature binding in visual cortex.* Proceedings of the IEEE International Joint Conference on Neural Networks, volume 4, pages 43–48.

SADJA, P., FINKEL, L. H. (1992b). *Object segmentation and binding within a biologically based neural network model of depth from occlusion.* Proceedings of the Computer Vision and Pattern Recognition, pages 688–691.

SHI, J., TOMASI, C. (1994). *Good features to track.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 593–600.

SUZUKI, H., MATSAKIS, P., DESACHY, J. (2001). *Fuzzy image classification and combinatorial optimization strategies for exploiting structural knowledge.* Proceedings of the International Conference on Fuzzy Systems, volume 1, pages 324 –327.

TERRAGEN (2004). *http ://www.planetside.co.uk/terragen/*, Terragen.

WEI, W., MENDEL, J. M. (1999). *A fuzzy logic method for modualtion classification in nonideal environments.* IEEE transaction on fuzzy systems, volume 7, numéro 3, pages 333–344.

WENG, J., HUANG, T., AHUJA, N. (1993). *Motion and structure from two perspective views : algorithms, error analysis and error estimation.* IEEE Pattern Analysis and Machine Intelligence, volume 11, numéro 9, pages 864–884.

YEN, S. C., FINKEL, L. H. (1997). *Salient contour extraction by temporal binding in a cortically-based network.* Advances in Neural Information Processing Systems, volume 10, pages 915–921.

ZADEH, L. A. (1965). *Fuzzy sets.* Information and Control, volume 8, pages 338–353.

ZHAO, L., THORPE, C. (2000). *Stereo and neural network-based pedestrian detection.* IEEE Transactions on Intelligent Transportation Systems, volume 1, numéro 3, pages 148–154.