Du parallélisme connexionniste à une pratique de calcul distribué numérique bio-inspiré

Bernard Girau

Nancy-Université

Du parallélisme connexionniste à une pratique de calcul distribué numérique bio-inspiré

Mémoire présenté et soutenu publiquement le 14 décembre 2007

pour l'obtention de l'

Habilitation à diriger des recherches de l'Université Nancy 2
(spécialité informatique)

par

Bernard GIRAU

Composition du jury

Rapporteurs :	Mathias QUOY
	Michel VERLEYSEN
	Thierry VIEVILLE
Examinateurs :	Frédéric ALEXANDRE
	François CHARPILLET
	Jean-François MARI

Département de formation doctorale en informatique — Ecole doctorale IAEM

Table des matières

Parcours scientifique

Curriculum vitæ résumé

Situation professionnelle actuelle

Maître de conférences en Informatique (section 27) à l'Université Nancy 2, Institut universitaire de technologie Charlemagne, Département d'informatique, actuellement en détachement auprès de l'INRIA comme Chargé de recherches.

Formation et diplômes

- **Doctorat en Informatique**, Ecole normale supérieure de Lyon (thèse soutenue le 26 mars 1999, intitulée *Du parallélisme des modèles connexionnistes à leur implantation parallèle*).
- **Agrégation de Mathématiques** (1995).
- **Magistère d'informatique et modélisation** de l'ENS Lyon (1994)

Principaux thèmes et résultats scientifiques

Mes recherches ont porté principalement sur les "Aspects parallèles (algorithmiques, computationnels, logiciels et matériels) du calcul connexionniste" :

- Parallélisation à grain très fin de réseaux connexionnistes. Mes travaux dans ce domaine montrent qu'une exploitation approfondie du parallélisme neuronal passe par une algorithmique des calculs induits qui ne considère pas le neurone comme le grain atomique de la structure neuronale.
- Implantations embarquées de modèles connexionnistes dédiés. Exploitation du parallélisme inhérent aux supports d'implantation connexionnistes. Mes travaux ont confirmé ici l'adéquation des calculs connexionnistes à la granularité des circuits numériques reconfigurables notamment.

Fruits de la combinaison de ces thèmes de recherche et de la dynamique de recherche de l'équipe Cortex, mes travaux actuels sont essentiellement axés autour de deux thèmes :

- Algorithmique et parallélisme matériel des réseaux de neurones et des modèles massivement distribués. Dans la continuité de mes travaux précédents, mes activités dans ce domaine se sont étendues à des modèles connexionnistes bio-inspirés, ainsi qu'à des modèles non connexionnistes.
- Modèles connexionnistes bio-inspirés massivement distribués pour la réalisation de tâches perceptives et de tâches cognitives complexes dans les systèmes autonomes. Mes travaux sur ce thème se sont concentrés autour de la conception d'un modèle complexe d'inspiration corticale pour la perception visuelle du mouvement.

Encadrements

Doctorants, post-doctorants

- Directeur de thèse de Mauricio Cerda, "Champs de neurones dynamiques massivement distribués pour la perception visuelle du mouvement", 2007-...
- Co-directeur de thèse de Mathieu Lefort, "Modules connexionnistes extensifs et perception multimodale", 2007-...
- Encadrement du stage de recherche international de Lucas Terissi (doctorant à l'Université de Rosario, Argentine, internship INRIA), "Perception visuelle du mouvement par champs de neurones dynamiques", mai-août 2007.
- Co-encadrement (dans le cadre d'un projet STIC de coopération internationale, sans accord de co-tutelle) de la thèse de Khaled Ben Khalifa, "Traitement matériel et logiciel des signaux physiologiques : application à l'étude de la vigilance", 2003-2006 (soutenue le 29 avril 2006).
- Directeur de thèse de Claudio Castellanos-Sanchez, "Modèles connexionnistes et perception visuelle dynamique pour systèmes autonomes embarqués", 2001-2005 (soutenue le 21 octobre 2005, direction administrative assurée par Frédéric Alexandre, DR INRIA).
- Responsable des travaux de Cesar Torres-Huitzil, post-doctorant INRIA, "Modèles connexionnistes embarqués pour la perception", 2004-05.
- Participation au co-encadrement CSEM des travaux de thèse ($1^{\text{ère}}$ année) de Fabio Restrepo, doctorant de l'Ecole polytechnique fédérale de Lausanne employé par le CSEM dans le cadre du projet "VLSI auto-structuré", 1997-98.

DEA/Master Recherche

- Encadrement des travaux de Master de Mauricio Cerda (Université du Chili, internship INRIA-Conicyt), "Rétroaction dans les modèles connexionnistes pour la perception du mouvement", 2007.
- Encadrement des travaux de Master Recherche d'Osman Kocak (UHP, ESSTIN), "Extraction visuelle du mouvement propre par réseaux de neurones", 2007.
- Encadrement des travaux de DEA (DEA MVA, ENS Cachan) de Florence Jacquey, "Parallélisme connexionniste et perception visuelle du mouvement", 2004.
- Encadrement des travaux de DEA (UHP, Supélec) de Nicolas Meurisse, "Implantation de réseaux de neurones sur système reconfigurable pour la perception visuelle en robotique autonome", 2001.
- Encadrement des travaux de DEA (UHP) de Samir Ferrat, "Etude d'un réseau de neurones impulsionnels et de son implantation FPGA pour la segmentation en robotique", 2001.

Ingénieurs

- Encadrement des travaux de stage ingénieur d'Adrien Gauffriau (Ecole centrale de Nantes), "Détermination automatique d'architectures FPNA pour l'implantation de réseaux multicouches", 2006.
- Encadrement des travaux de stage ingénieur de Damien Doiselet (UTBM), "Synthèse automatique de modèles neuronaux pour la perception visuelle sur FPGA", 2006.
- Encadrement des travaux de stage ingénieur de Frédéric Dhalleine, "Synthèse automatique de modèles neuronaux multicouches sur FPGA", 2003.

- Responsable des travaux de Philippe Beylik, ingénieur associé INRIA, "Implantation de calculs neuronaux sur circuits reprogrammables", 2001-02.

Stagiaires

- Encadrement des travaux de stage de Fabien Jacques (ESSTIN), "Contrôle harmonique distribué pour la planification de trajectoires", 2006.
- Encadrement des travaux de stage de Benoit Cado (IUT informatique) "Modélisation objet des architectures FPNA", 2005.
- Encadrement des travaux de stage de David Paris (Licence sciences cognitives), "Analyse statistique de modèles connexionnistes de prédiction du langage", 2004
- Encadrement des travaux de stage de Thomas Bronner (Maîtrise sciences cognitives), "Génération d'environnements synthétiques pour la perception du mouvement", 2002.
- Encadrement de différents projets d'initiation à la Recherche (Maîtrise d'informatique, ESIAL) 2000 à 2002.

Participation à des jurys

- Rapporteur et membre du jury de thèse de Benjamin Schrauwen, "Towards applied spiking neural networks in hardware", 29 novembre 2007, Universiteit Gent, Belgique.
- Membre du jury de thèse de Claudio Castellanos-Sanchez, "Modèles connexionnistes et perception visuelle dynamique pour systèmes autonomes embarqués", 21 octobre 2005, Université Henri Poincaré Nancy 1.
- Membre du jury de diplôme d'ingénieur d'Adrien Gauffriau, Ecole centrale de Nantes, 2006.
- Membre du jury de M2R d'Osman Kocak, Université Henri Poincaré Nancy 1, 2007.
- Membre du jury de DEA de Florence Jacquey, Ecole Normale Supérieure de Cachan, 2004.
- Membre du jury de DEA de Philippe Beylik, Université Henri Poincaré Nancy 1, 2003.
- Membre du jury de DEA de Nicolas Meurisse, Université Henri Poincaré Nancy 1, 2001.
- Membre du jury de DEA de Samir Ferrat, Université Henri Poincaré Nancy 1, 2001.

Projets et collaborations

Projets

- Collaboration avec la Faculté de Médecine de l'Université de Monastir et l'Institut Supérieur des Sciences Appliquées et de Technologie de Sousse, "Traitements connexionnistes embarqués de signaux physiologiques", projet STIC-Tunisie du programme de coopération bilatérale INRIA-DGRSRT, 2007-08. Responsable du projet.
- Collaboration avec le Laboratoire FPGA du Département Informatique de l'Institut national d'astrophysique, d'optique et d'électronique de Puebla (INAOEP, Mexique), "Hardware/Software Platform for Massive Parallel Applications using Reconfigurable Computing", projet Conacyt 59474, 2007-2008.
- Collaboration avec la Faculté de Médecine de l'Université de Monastir, "Classification des états de vigilance de l'homme par réseaux de neurones artificiels", projet STIC-Tunisie du programme de coopération bilatérale INRIA-DGRSRT, 2002 à 2006.
- Participation au projet européen NeuroCOLT sur la théorie de l'apprentissage des modèles connexionnistes, de 1994 à 1997, puis au projet européen NeuroCOLT 2 de 1998 à 2002.

- Collaboration avec le Laboratoire des systèmes logiques de l'Ecole polytechnique fédérale de Lausanne, sur l'application des principes bio-inspirés d'auto-structuration à la conception de circuits intégrés de très grande taille, dans le cadre du projet "Biologiciel" sur le matériel évolutionniste, 1997-98.

Collaborations diverses

- Collaboration avec H. Frezza-Buet (équipe ERSIDP, Supélec Metz), sur les modules connexionnistes à apprentissage non-supervisé dans des systèmes extensifs.
- Collaboration avec H. Berry (équipe Alchemy, INRIA-Futurs) et N. Fates (équipe MAIA, LORIA INRIA-Lorraine), sur l'aggrégation cellulaire pour les systèmes multi-agents et le calcul spatial bio-inspiré.
- Collaboration avec A. Boumaza et B. Scherrer (équipe MAIA, LORIA INRIA-Lorraine), "Contrôle embarqué distribué pour la planification de trajectoires".
- Collaboration avec C. Torres-Huitzil et M. Arias-Estrada (Institut national d'astrophysique, d'optique et d'électronique de Puebla, INAOEP, Mexique), sur l'implantation FPGA de modèles neuronaux pour la perception visuelle, et sur la synthèse automatique de réseaux de neurones sur FPGA.
- Collaboration avec Y. Bengio (Laboratoire d'informatique des systèmes d'apprentissage, Université de Montréal), "Apprentissage parallèle appliqué aux bases de données de grandes dimensions", 1999-2001.
- Collaboration avec F. Rossi, du laboratoire Ceremade (Centre de recherche en mathématiques de la décision) de l'Université Dauphine Paris IX, sur l'apprentissage généralisé des modèles connexionnistes.

Responsabilités

Organisation de projets et manifestations

- Co-organisateur de la conférence AMINA, Monastir, 2006. Organisateur et chairman de la session spéciale "Systèmes embarqués".
- Co-organisateur de la conférence NeuroComp (1ère conférence française de neurosciences computationnelles), Nancy, 2006.
- Chairman de la session "Neural Networks 1" de la conférence AIA, Innsbruck, 2006.
- Chairman de la session "Implantations matérielles" de la conférence AMINA (Applications médicales de l'intelligence neuro-artificielle), Monastir, 2004.
- Responsable du projet européen NeuroCOLT 2 pour le LORIA, 1998-2001.
- Co-organisateur du "Second Yearly NeuroCOLT Meeting", Villars-de-Lans, février 1996.

Activités au service de la communauté

- Responsable de la Commission ingénieurs du comité de recrutement INRIA Lorraine/Loria des personnels scientifiques contractuels (Comipers), membre du bureau du Comipers, 2004, 2005, 2006, 2007.
- Membre de la commission locale de recrutement d'ingénieurs sur postes d'accueil INRIA, 2001, 2002, 2003.
- Responsable des relations internationales pour l'équipe Cortex.

- Responsable de la Commission Liptools du Laboratoire de l'informatique du parallélisme, 1995-96.
- Membre de la Commission informatique du Laboratoire de l'informatique du parallélisme, 1995-96.
- Relecteur pour diverses conférences internationales : IWANN, ParCo, EuroPar, ICANN, HiPC, SPAA, I-SPAN, PACT.
- Relecteur pour divers journaux internationaux : IEEE Trans. on Neural Networks, IEEE Computer, Parallel Computing, Journal of VLSI Signal Processing Systems, Electronics Letters, IEEE Transactions on Computers, Iranian J. of Science and Technology, IEEE Trans. on Parallel and Distributed Systems, Int. J. of Neural Systems., J. of Applied Research and Technology, Neural Computing & Applications

Responsabilités pédagogiques

- Membre du groupe de réflexion sur la réorganisation pédagogique des enseignements en informatique de l'IUT Charlemagne autour du langage Java – responsable du sous-groupe de réflexion chargé de l'enseignement de Java et de la conception logicielle, 2001. Responsable de la réorganisation de l'enseignement de la programmation en $1^{\text{ère}}$ année puis en année spéciale.
- Responsable des projets tutorés de $2^{\text{ème}}$ année, IUT Charlemagne, département informatique, 2004-05-06.
- Membre de la commission de spécialistes, Université Nancy 2, section 27, 2004-05-06.
- Responsable de 20 modules d'enseignement (9 modules distincts).

Articulation des recherches, encadrements, collaborations

Avant de détailler l'ensemble de mes travaux sur le thème du parallélisme connexionniste comme base d'une pratique de calcul distribué numérique bio-inspiré, je souhaite décrire globalement comment se sont articulés les différents aspects scientifiques abordés, et comment mes différents encadrements et collaborations se sont insérés dans cette évolution de mes recherches.

Ma thèse, initialement orientée sur les réseaux de neurones et le parallélisme logiciel, s'est peu à peu focalisée sur un parallélisme à grain très fin, sur support matériel. J'ai été recruté comme Maître de Conférences à l'IUT Charlemagne en septembre 2000, et mes activités de recherche se sont immédiatement situées dans l'équipe Cortex du Loria, dont les travaux portaient alors essentiellement sur les modèles comportementaux d'inspiration corticale et l'intégration neuro-symbolique.

L'enjeu de ce recrutement, sur le plan de la recherche, était d'ouvrir peu à peu mes travaux sur les modèles neuronaux bio-inspirés, les faisant du même coup bénéficier d'une approche davantage orientée vers leurs capacités de modèles de calcul distribué.

Dès 2001, j'ai souhaité m'entourer rapidement d'**un ingénieur associé**, Philippe Beylik, et d'**un doctorant**, Claudio Castellanos-Sanchez. Durant deux ans, Philippe a eu pour rôle de m'aider sur le plan technologique dans mes travaux liés aux implantations matérielles sur FPGA. Il a pris en charge l'installation et la maintenance d'une carte reconfigurable et d'un environnement logiciel de gestion de cette carte. Il a également débuté la constitution d'une bibliothèque paramétrable de modules VHDL destinés à l'implantation de réseaux neuronaux, sur la base d'une partie de mes travaux de recherche. Cette bibliothèque constitue le socle initial du projet NNetWare auquel il est fait référence en 2.3.1. Enfin, en deuxième année, il a accompagné mon étude de l'adéquation

du calcul impulsionnel aux supports reconfigurables. En effet, le groupe Cortex avait étendu ses activités à ce type de calcul neuronal bio-inspiré, et pour les raisons évoquées en 2.1.3, ce type de calcul offre des perspectives particulières pour des implantations parallèles à grain très fin. J'ai donc initié, avec l'aide d'Olivier Rochel, doctorant de l'équipe qui travaillait sur ces modèles impulsionnels avec Dominique Martinez, une première étude exploratoire de l'implantation numérique de calculs impulsionnels. Là encore, il s'agit du socle initial des travaux décrits en 2.1.3. La thèse de Claudio Castellanos-Sanchez, quant à elle, s'est orientée vers la définition et l'implantation d'un modèle d'inspiration corticale pour la perception visuelle du mouvement (décrit en 4.1, thèse soutenue le 21 octobre 2005). Ces travaux ont été pour moi l'occasion de confronter ma connaissance du calcul connexionniste massivement distribué avec les calculs bio-inspirés fondés sur un flux très dense d'interactions excitatrices et inhibitrices. Le choix de la perception visuelle a été motivé par la structure bidimensionnelle des cartes corticales associées, dans la perspective d'implantations sur des supports matériels essentiellement bidimensionnels eux aussi. Le choix de la perception du mouvement a été motivé par un besoin de l'équipe, déjà largement engagée dans d'autres aspects de la perception visuelle. Enfin, le fil rouge de cette thèse, durant quatre ans, a été le maintien à tous les niveaux du modèle des contraintes de localité et de distribution assurant la continuité avec mon étude du parallélisme connexionniste. Ainsi, dès 2001, j'ai pu affirmer la poursuite conjointe des travaux liés au parallélisme connexionniste et de mes premiers travaux dans le domaine du connexionnisme d'inspiration biologique.

Outre plusieurs stages courts, principalement menés afin de compléter la bibliothèque de modules VHDL initiée avec Philippe Beylik, c'est en 2003 que se situe la prochaine évolution importante de mes travaux. Dans le cadre d'un projet STIC-Tunisie sur la "Classification des états de vigilance de l'homme par réseaux de neurones artificiels" en collaboration avec le laboratoire de biophysique de la faculté de médecine de Monastir, la volonté d'aboutir à un système ambulatoire de détection de l'hypovigilance a suscité ma participation à ce projet. J'ai ainsi été amené à superviser une partie des travaux de thèse de Khaled Ben Khalifa, **doctorant** de la Faculté des Sciences de Monastir, sur l'implantation embarquée des systèmes neuronaux élaborés dans le cadre de ce projet (thèse soutenue le 29 avril 2006). Outre une première confrontation, positive, avec les contraintes d'un système applicatif complet, ces travaux, en partie décrits en 2.1.3, ont marqué le début de la réflexion sur la nécessité d'aboutir à un environnement intégré dédié à l'implantation sur FPGA de calculs neuronaux (cf 2.3.1).

L'année 2004 a marqué le véritable début des travaux mêlant implantations parallèles et modèles d'inspiration corticale. Cela s'est traduit notamment au travers de l'encadrement d'**un post-doctorant**, Cesar Torres-Huitzil, avec lequel j'ai mené des travaux, décrits en 3.3.1, sur l'implantation parallèle matérielle des modèles bio-inspirés de perception visuelle, afin de valider l'approche défendue depuis le début de la thèse de Claudio Castellanos-Sanchez. Ce recrutement donnera par la suite lieu à une fructueuse collaboration avec le laboratoire d'informatique de l'INAOEP au Mexique auquel Cesar appartient. Par ailleurs, désireux de ne pas limiter l'étude des modèles d'inspiration corticale selon le point de vue du parallélisme connexionniste aux seuls aspects d'implantation embarquée, j'ai débuté une analyse plus fondamentale de la dynamique locale des interactions du modèle décrit en 4.1, m'appuyant alors sur l'encadrement du stage de **DEA** de Florence Jacquey. Ces résultats théoriques sont rapportés notamment dans le mémoire de thèse de Claudio Castellanos-Sanchez.

L'année 2005 a été marquée par la fin de la thèse de Claudio, le démarrage effectif du projet NNetWare (l'encadrement d'un **stage ingénieur** m'ayant donné l'occasion d'en définir l'archi-

tecture logicielle générale et les premiers éléments de l'interface graphique), et une collaboration accentuée avec Cesar Torres-Huitzil, de retour à l'INAOEP. C'est également au cours de cette année qu'ont débuté les travaux logiciels devant permettre à terme de relier le projet NNetWare et le concept FPNA initié lors de ma thèse.

2006 a été une nouvelle année charnière dans l'évolution de mes recherches. Devant l'avancement des différents travaux, j'ai demandé et obtenu mon détachement auprès de l'INRIA. A la suite de l'organisation d'une session "implantations embarquées des modèles distribués" à la conférence AMINA à Monastir, où Frédéric Alexandre et moi-même avons invité différents chercheurs concernés par la thématique d'une pratique de calcul distribué, j'ai mis en place différents projets. Le projet NNetWare a atteint une nouvelle phase, notamment avec l'aide de Cesar Torres-Huitzil et d'un **stage ingénieur**, où une chaîne fonctionnelle complète a été obtenue de la description du réseau de neurones à son implantation et son exécution sur carte FPGA. Au travers de ma collaboration avec l'INAOEP et du projet STIC-Tunisie accepté pour 2007-2008, principalement porté par moi-même côté français et Mohamed Hedi Bedoui et Khaled Ben Khalifa côté tunisien, une phase ultérieure débute qui vise à maîtriser l'automatisation de la gestion des différents niveaux de parallélismes. Par ailleurs au sein du Loria, j'ai débuté une première collaboration avec des chercheurs de l'équipe Maia, autre équipe défendant une pratique de calcul distribué, selon une inspiration néanmoins différente.

L'année 2007 est marquée par une activité nettement renforcée du côté des modèles d'inspiration corticale, autour de l'encadrement des **stages de master** d'Osman Kocak et Mauricio Cerda (travaux en liaison avec 4.4 et 4.3.1), ainsi que du stage internship de Lucas Terissi, **doctorant** argentin. Cette activité va servir de socle aux travaux de **thèse** de Mauricio que je vais encadrer. Cette année est également marquée par le démarrage de plusieurs projets, le projet STIC-Tunisie évoqué plus haut, mais aussi un renforcement des contacts avec l'équipe Maia et avec l'équipe Alchemy, toujours sur le thème affirmé d'une pratique de calcul distribué d'inspiration biologique mais non corticale, et une interaction de plus en plus forte avec les chercheurs de l'équipe Cortex impliqués dans les modèles comportementaux à base de champs neuronaux dynamiques (voir 3.1.1 et 3.3.2). Ces différents aspects seront davantage évoqués comme perspectives de recherches en 2.2.2, 3.3.2 et 4.5, et c'est dans le cadre général de ces modèles que vont se situer les travaux de **thèse** de Mathieu Lefort que je vais co-encadrer avec Yann Boniface.

En conclusion, depuis mon recrutement, au travers de différents projets ou collaborations et au travers d'un équilibre des recrutements sur les différents thèmes abordés, j'ai maintenu une volonté constante de défendre une approche globale d'un connexionnisme considéré comme pratique de calcul distribué à la fois algorithmique et implanté, tout en incluant peu à peu les travaux d'inspiration corticale dans l'ensemble de mon approche.

Principales publications

Chapitres de livres avec comité de lecture

[1] B. Girau. FPNA : Applications and implementations. In A. Omondi and J. Rajapakse, editors, *FPGA Implementations of Neural Networks*. Springer, 2006.

[2] B. Girau. FPNA : Concepts and properties. In A. Omondi and J. Rajapakse, editors, *FPGA Implementations of Neural Networks*. Springer, 2006.

Revues scientifiques internationales avec comité de lecture

[3] Bernard Girau and Cesar Torres-Huitzil. Massively distributed digital implementation of an integrate-and-fire LEGION network for visual scene segmentation. *Neurocomputing*, 70, 2007.

[4] Cesar Torres-Huitzil and Bernard Girau. Massively distributed implementation of a spiking neural network for image segmentation on FPGA. *Neural Information Processing Letters and Reviews, special issue on Bio-inspired Models and Hardware*, 10, 2006.

[5] Cesar Torres-Huitzil, Bernard Girau, and Claudio Castellanos-Sánchez. On-chip visual perception of motion : A bio-inspired connectionist model on FPGA. *Neural networks*, 18(5-6) :557–565, 2005.

[6] B. Girau. Conciliating connectionism and parallel digital hardware. *Parallel and Distributed Computing Practices,* special issue on *Unconventional parallel architectures*, 3(2), 2000.

[7] B. Girau. FPNA : interaction between FPGA and neural computation. *Int. Journal on Neural Systems,* special issue on *New trends in neural network implementations*, 10(3), 2000.

[8] B. Girau and A. Tisserand. MLP computing and learning on FPGA using on-line arithmetic. *Int. Journal on System Research and Information Science,* special issue on *Parallel and Distributed Systems for Neural Computing*, 9(2-4), 2000.

Actes de conférences internationales avec comité de lecture

[9] Cesar Torres-Huitzil, Bernard Girau and Adrien Gauffriau. Hardware-software codesign for embedded implementations of neural networks. In *Applied Reconfigurable Computing, ARC*, 2007.

[10] Bernard Girau and Amine Boumaza. Embedded harmonic control for dynamic trajectory planning on FPGA. In *Conf. on Artificial Intelligence and Applications*, 2007.

[11] Bernard Girau and Cesar Torres-Huitzil. FPGA implementation of an integrate-and-fire legion model for image segmentation. In *European Symp. on Artificial Neural Networks*, 2006.

[12] Bernard Girau and Khaled Ben Khalifa. FPGA-targeted neural architecture for embedded alertness detection. In *Conf. on Artificial Intelligence and Applications*, 2006.

[13] Cesar Torres-Huitzil and Bernard Girau. FPGA implementation of an excitatory and inhibitory connectionist model for motion perception. In *IEEE 2005 Conference on Field-Programmable Technology - FPT'05, Singapore*, 2005.

[14] Cesar Torres-Huitzil, Bernard Girau, and Claudio Castellanos Sánchez. Digital implementation of a bio-inspired neural model for motion estimation. In *International Joint Conference on Neural Networks, Montréal, Québec, Canada*, 2005.

[15] Claudio Castellanos Sánchez and Bernard Girau. Dynamic pursuit with a bio-inspired neural model. In *Advanced Concepts for Intelligent Vision Systems - ACIVS 2005, Anwertp, Belgium*, volume 3708 of *Lecture Notes in Computer Science*, pages 284–291, 2005.

[16] K. Ben Khalifa, B. Girau, F. Alexandre, M.H. Bedoui Parallel FPGA Implementation of Self-Organizing Maps In *International Conference on Microelectronics (ICM)*, 2004.

[17] C. Castellanos-Sanchez, B. Girau and F. Alexandre. A connectionist approach for visual perception of motion. Brain Inspired Cognitive Systems - BICS, pp.BIS3-1 1-7, 2004.

[18] B. Girau. On-chip learning of FPGA-inspired neural nets. In *Proc. Int. Joint Conf. on Neural Networks*, July 2001.

[19] B. Girau. Neural networks on FPGAs : a survey. In *Proc. Neural Computation*, May 2000.

[20] B. Girau. Simplified neural architectures for symmetric boolean functions. In *Proc. European Symposium on Artificial Neural Networks*, April 2000.

[21] B. Girau. Digital hardware implementation of 2D compatible neural networks. In *Proc. Int. Joint Conf. on Neural Networks*, July 2000.

[22] B. Girau. Building a 2D-compatible multilayer neural network. In *Proc. Int. Joint Conf. on Neural Networks*, July 2000.

[23] B. Girau, P. Marchal, P. Nussbaum, A. Tisserand, and H.F. Restrepo. Evolvable platform for array processing : a one-chip approach. In *Proc. MicroNeuro*, pages 187–193. IEEE Computer Society Press, 1999.

[24] P. Nussbaum, B. Girau, and A. Tisserand. Field Programmable Processor Arrays. In *Proc. ICES*, volume 1478 of *LNCS*, pages 311–322. Springer, 1998.

[25] R. Baron and B. Girau. Parameterized normalization : application to wavelet networks. In *Proc. Int. Joint Conf. on Neural Networks*, volume 2, pages 1433–1437. IEEE Computer Society Press, 1998.

[26] B. Girau and A. Tisserand. On-line arithmetic based reprogrammable hardware implementation of multilayer perceptron back-propagation. In *Fifth international conference on Microelectronics for Neural Networks and Fuzzy Systems - MicroNeuro'96*, pages 168–175. IEEE Computer Society Press, 1996.

[27] C. Gégout, B. Girau, and F. Rossi. Generic back-propagation in arbitrary feedforward neural networks. In *Artificial Neural Nets and Genetic Algorithms – Proc. of ICANNGA*, pages 168–171. Springer-Verlag, 1995.

[28] B. Girau. Mapping neural network backpropagation onto parallel computers with computation/communication overlapping. In *Proc. EuroPar*, pages 513–524. Springer-Verlag, 1995.

[29] B. Girau and H. Paugam-Moisy. Load sharing in the training set partition algorithm for parallel neural learning. In *Proc. IPPS* 9^{th} *Int. Parallel Processing Symposium*, pages 586–591. IEEE Computer Society Press, 1995.

[30] C. Gégout, B. Girau, and F. Rossi. NSK, an object-oriented simulator kernel for arbitrary feedforward neural networks. In *ICTAI Int. Conf. on Tools with Artificial Intelligence*, pages 95–104. IEEE Computer Society Press, 1994.

Introduction

De nombreux travaux portent sur les réseaux de neurones artificiels appliqués à des systèmes fortement contraints (systèmes ambulatoires, systèmes autonomes, systèmes adaptatifs, etc), pour lesquels le caractère élémentaire et massivement distribué des calculs neuronaux s'avère un atout. De façon plus large, ce caractère définit la nature même des calculs connexionnistes : une puissance de calcul et une robustesse fondées sur un parallélisme massif à grain très fin où les unités de calcul s'insèrent dans un flux d'informations très dense. Il est donc indispensable de pouvoir pleinement exploiter ce *parallélisme connexionniste*, à la fois en termes de potentiel de calcul et en termes de perspectives d'implantations embarquées efficaces.

La ligne directrice de mes travaux consiste à défendre cette forme de calcul distribué comme une véritable pratique de calcul constituant une alternative viable aux modèles plus classiques, principalement issus d'une approche séquentielle, procédurale, de l'algorithmique.

Le terme "pratique de calcul" recouvre dans ce manuscrit une approche qui se veut concernée à parts égales par tous les aspects du calcul, de la conception théorique des modèles à leur implantation. Une telle approche affirme l'interdépendance de toutes ces étapes, par opposition à une approche plus usuelle consistant à toujours traiter les aspects théoriques en amont des applications puis des implantations.

Les implications de cette approche vont être illustrées au sein des différents travaux décrits dans ce manuscrit. Ces travaux vont ainsi montrer l'influence mutuelle qu'ont les aspects théoriques du connexionnisme et les exigences liées à la problématique d'une implantation réellement distribuée, étape incontournable pour des modèles qui se veulent constituer ce qui a été précédemment appelé "pratique de calcul". Un aspect commun à tous ces travaux émerge pour constituer l'affirmation centrale de mes recherches : si les modèles connexionnistes sont constitués de nombreuses unités de calcul, leur puissance réside avant tout dans le flux des informations échangées par ces unités de calcul, et la perspective que ces modèles constituent une pratique de calcul distribué passe essentiellement par une recherche sur la gestion locale de ce flux massivement distribué.

Une approche globale de la pratique de calcul connexionniste ne peut pas faire l'économie d'une réflexion sur les fondements de la conception des architectures connexionnistes susceptibles de résoudre un problème donné. L'essentiel des travaux sur les réseaux de neurones s'est concentré sur la définition de familles de modèles neuronaux dont l'apprentissage permet de traiter des tâches telles que classification, régression, discrimination, auto-régression, etc. Ces modèles restent d'actualité, ne serait-ce que par la variété des applications dans lesquelles ils jouent encore un rôle central, et les travaux rapportés dans ce manuscrit portent en partie sur ces modèles. Néanmoins, ils n'apparaissent le plus souvent que comme des outils de traitement statistique de données, au même titre que de nombreux autres modèles. Ils se heurtent ainsi à une barrière de complexité dans les problèmes qu'ils sont susceptibles de résoudre. Franchir "à l'aveuglette" cette barrière, c'est à dire proposer ex nihilo des architectures connexionnistes capables de réaliser des tâches cognitives

complexes, ne semble pas raisonnable. De nombreux chercheurs suggèrent alors de prendre appui sur des exemples naturels de systèmes distribués capables de réaliser ces tâches. Les neurosciences sont une orientation possible. C'est cette inspiration sur la base des mécanismes observés dans le cerveau qui est proposée dans une partie des travaux rapportés dans ce manuscrit. L'objet de ces travaux n'est pas de faire le tour de l'apport possible des neurosciences pour la conception des modèles connexionnistes, mais de montrer que même dans le cas de modèles d'inspiration corticale, la défense d'une pratique de calcul connexionniste passe par une étude focalisée sur les mécanismes locaux de gestion du flux d'informations sous-jacent à ces modèles.

Le chapitre 1 va revenir sur la notion même de parallélisme connexionniste, de façon à en tracer les différentes évolutions et les différents enjeux. Le chapitre 2 a pour objet de montrer que la prise en compte de modèles connexionnistes comme architectures distribuées de calcul valide les arguments en faveur du parallélisme naturel des réseaux de neurones, mais pose simultanément des questions aussi bien sur l'algorithmique et l'apprentissage de ces modèles que sur la définition même du paradigme de calcul connexionniste. Le chapitre 3 se concentre ensuite sur la nécessité de prendre appui sur la bio-inspiration pour aboutir à des architectures connexionnistes traitant des problèmes de plus en plus complexes, tout en maintenant au cœur de ces recherches la réflexion sur la gestion locale et distribuée des interactions neuronales complexes. Le chapitre 4 illustre enfin l'application de cette inspiration corticale à l'élaboration d'un modèle connexionniste de perception visuelle du mouvement.

Chapitre 1

Parallélisme et connexionnisme

Les modèles connexionnistes, et notamment les réseaux de neurones artificiels, constituent les premiers modèles informatiques du calcul parallèle. Dans ces modèles, les unités de calcul distribuées effectuaient leurs calculs et échangeaient de manière directe les résultats de ces calculs. L'évolution des technologies, et notamment les progrès très rapides des architectures basées sur l'utilisation conjointe de processeurs séquentiels et de mémoires de données ont rendu ces modèles moins pertinents (voir 1.1). Le calcul parallèle s'est alors orienté vers les modèles PRAM et leurs descendants et variantes pour formaliser ses principes. Et pourtant le parallélisme est resté cité très souvent dans la recherche en réseaux de neurones comme un atout essentiel des modèles connexionnistes. Ce postulat, pourtant le plus souvent présenté de manière simpliste, a motivé une très grande variété de travaux sur la parallélisation des réseaux connexionnistes. Ces travaux, loin de justifier leur postulat de départ, ont peu à peu scellé l'existence d'un fossé entre le connexionnisme et les pratiques de calcul distribué. Le développement d'implantations neuronales sur circuits intégrés, bien que susceptible de maintenir la perspective de calculs connexionnistes réellement distribués, a tendu à confirmer la défaite du connexionnisme comme pratique *informatique* de calcul distribué. L'apparition des circuits reconfigurables, à mi-chemin entre la programmation et la conception matérielle a renversé cette tendance. Ces circuits sont parfois vus maintenant comme les précurseurs d'une évolution inévitable des circuits intégrés qui, devant l'incapacité de poursuivre indéfiniment la course à la loi de Moore, semblent s'orienter vers des systèmes de calculs constitués d'une multitude d'unités de calcul répartis (pas forcément régulièrement) sur une même puce, et fonctionnant en mode asynchrone. Cette évolution s'accompagne d'un regain d'intérêt pour le connexionnisme et ses sources d'inspiration. C'est cette évolution dans le temps du parallélisme connexionniste que ce chapitre s'efforce de résumer.

1.1 Le parallélisme et les réseaux de neurones, le divorce

L'architecture des réseaux connexionnistes autorise théoriquement un grand nombre d'éléments de calcul simples à travailler de façon concurrente. Cet aspect est ainsi sensé faciliter grandement l'obtention d'implantations très rapides. La question de la faisabilité de telles implantations est par exemple posée pour des applications ayant notamment des contraintes temps-réel dans lesquelles des architectures neuronales gourmandes en temps de calcul ont fourni d'excellents résultats (reconnaissance de codes postaux, reconnaissance de visages, séparation de sources, navigation, routage, etc). Le coût en temps de calcul de ces modèles est à la fois lié à leur architecture mettant en jeu

un grand nombre d'unités, mais aussi à leur apprentissage.

Le calcul neuronal parallèle, qu'il soit destiné à des machines massivement parallèles ou à des circuits intégrés, constitue donc dans la littérature connexionniste un argument important, mais souvent peu étayé, pour justifier en partie l'intérêt pratique des réseaux de neurones. Néanmoins l'ensemble des travaux d'implantation rapide de calculs neuronaux montre à quel point il est difficile de tirer directement profit de la structure parallèle des réseaux connexionnistes.

Ces travaux se heurtent à une mauvaise adéquation du parallélisme de la plupart des réseaux neuronaux aux contraintes, logicielles ou matérielles, des supports d'implantations. Si cette mauvaise adéquation justifie à elle seule une recherche sur la pertinence du connexionnisme comme pratique de calcul distribué, elle recouvre des disparités fortes suivant les supports parallèles considérés.

Plusieurs sortes de supports parallèles ont été utilisés pour développer des implantations neuronales rapides, tels que des ordinateurs parallèles ou grappes de PC, ou encore des circuits intégrés analogiques ou numériques.

Les grandes tendances des implantations sur machines parallèles sont les suivantes.

- Les implantations parallèles à grain fin sur ordinateurs massivement parallèles SIMD ou MIMD se sont heurtées aux nombreux échanges de données entre les neurones. Ceci peut se traduire par des efficacités faibles dues à un temps de communication élevé, ou bien par des méthodes de parallélisation qui se basent sur une réécriture algébrique des calculs, exploitant ainsi de potentielles régularités de structure, et s'écartant donc conceptuellement de la notion de parallélisme connexionnisme. La disparition progressive des machines parallèles SIMD a été un handicap pour ce type d'implantation. Et les évolutions vers des machines parallèles à mémoire partagée n'ont pas suffi à combler le fossé entre les grains de parallélisme des modèles connexionnistes et de ces supports.
- Les implantations parallèles à gros grain sur ordinateurs massivement parallèles s'appliquent principalement à la phase d'apprentissage évoquée plus haut. Les problèmes ne se situent alors plus dans la compatibilité entre le parallélisme de la structure neuronale et celui de la machine cible, mais dans la difficulté de concilier des algorithmes de prototypage essentiellement conçus de façon séquentielle et la distribution des données gérées. Ces approches restent d'actualité, notamment en lien avec les avancées récentes des technologies de programmation parallèle sur grappes de PC ou sur réseaux hétérogènes, mais elles portent sur des besoins ponctuels de calculs connexionnistes très lourds. Quelles que soient les potentialités de tels travaux, ils ne sont pas en relation avec la défense du paradigme connexionniste comme pratique de calcul distribué.
- Les neuro-ordinateurs ne correspondent pas à proprement parler à une méthode de parallélisation précise. Les implantations réalisées sur ce type de machine peuvent combiner les différentes parallélisations évoquées ci-dessus. Ils doivent cependant être considérés séparément des machines parallèles génériques. Il s'agit de systèmes parallèles dédiés au calcul neuronal intensif, qui sont bâtis à partir de circuits de type DSP (digital signal processors) ou neuroproceseurs. Le principal défaut de ces systèmes est lié à la spécificité de leur champ d'application. Les moyens mis en œuvre pour leur conception sont limités, ce qui se traduit d'une part par un temps de développement élevé pendant lequel les progrès des systèmes génériques concurrents sont considérables, et d'autre part par un manque de mises à niveau ultérieures qui sont un handicap pour l'intérêt à moyen terme des parallélisations ciblées sur de tels systèmes. Après une disparition quasi complète de ces machines, l'apparition des circuits reconfigurables (voir plus bas) a relancé les principes du développement de neuropro-

ceseurs. Mais encore une fois il s'agit là avant tout d'une technologie permettant des calculs connexionnistes rapides, et non d'une prise en compte du parallélisme connexionniste comme fondement de calculs distribués.

Les implantations parallèles ainsi obtenues ont un degré de parallélisme qui reste très inférieur aux possibilités offertes par le parallélisme des structures neuronales concernées. Et plus généralement, les travaux associés doivent avant tout être considérés comme des travaux de parallélisation efficace d'algorithmes dont la nature connexionniste n'est pas prise en compte.

Les implantations matérielles sont alors les plus à même d'offrir les meilleures performances, par une exploitation plus directe du parallélisme neuronal. En ce qui concerne ces implantations, les aspects principaux qui se dégagent sont les suivants.

- De nombreuses implantations sur circuits analogiques ont été réalisées. Des classes entières de modèles connexionnistes sont dédiées à ce type d'implantation, qui réunit les propriétés de vitesse, de densité et de consommation. Les neurones simples sont en effet proches des opérations réalisées par les circuits analogiques. Mais la réalisation de tels circuits est ardue et pose des problèmes spécifiques de précision, de stockage des données, de robustesse, etc. Le manque de flexibilité, le coût et le temps de développement contrebalancent les performances de tels circuits.
- La réalisation de circuits numériques pose moins de problèmes de précision et de robustesse, et est plus facile à adapter à une large variété de calculs neuronaux. Cependant, de tels circuits manquent aussi de flexibilité, et leur développement reste coûteux et nécessite malgré tout des compétences spécifiques, bien que plus abordables que celles liées à la technologie analogique.
- L'utilisation de circuits numériques configurables est encore restreinte, malgré un essor récent remarquable. Ces circuits peuvent permettre de contourner les principaux défauts des circuits numériques dédiés, grâce à une approche plus logicielle. L'atténuation progressive des frontières entre conception matérielle et logicielle par le biais des architectures reconfigurables (par exemple FPGA, field programmable gate arrays) permet d'envisager des parallélisations à grain très fin de réseaux de neurones, à la fois efficaces, flexibles, peu coûteuses et rapides à réaliser. Ce type d'implantation représente un compromis intéressant entre les performances d'un circuit intégré spécifique et la souplesse d'utilisation d'une approche logicielle. Elles sont également une véritable opportunité pour obtenir un prototypage rapide des réseaux de neurones au sein d'une application, tout en permettant ensuite d'aboutir à la conception d'un circuit intégré dédié.

Parmi toutes ces implantations logicielles et matérielles, celles qui utilisent un parallélisme à grain fin n'utilisent que rarement la structure naturellement parallèle des réseaux implantés, où le graphe des connexions représente les relations de précédence existant entre les différentes unités de calcul concurrentes. Les raisons qui expliquent cette faible utilisation du parallélisme neuronal le plus simple varient avec le support. Elles sont par exemple liées aux temps de communication sur les machines parallèles à mémoire distribuée, ou bien à la topologie non planaire des connexions lorsqu'un circuit numérique est visé.

1.2 Le parallélisme connexionniste

Si le tour d'horizon ci-dessus permet de mesurer le fossé plus ou moins large qui s'est creusé entre le parallélisme connexionniste et le parallélisme "informatique", et que l'apparition des circuits

reconfigurables a permis en partie de combler, il ne montre pas tout le chemin qui reste à parcourir pour faire passer le connexionnisme du statut de modèle distribué implantable au rôle de pratique de calcul distribué, au même rang que l'algorithmique séquentielle classique dans un contexte d'architectures schématiquement basées sur le modèle de Von Neumann. Le travail rapporté dans ce manuscrit a pour vocation de jeter quelques jalons sur ce chemin, ce qui passe par une définition plus précise de la notion de parallélisme connexionniste.

1.2.1 Les parallélismes des réseaux de neurones

Une excellente introduction aux différentes implantations parallèles du calcul neuronal a été proposée par Nordström et Svensson dans [NS92]. Les parallélismes relatifs aux applications à base de réseaux connexionnistes y sont classés en six catégories imbriquées :

1. Parallélisme de session d'apprentissage : plusieurs types ou tailles d'architectures neuronales, mais également plusieurs paramètres expérimentaux (liés à l'algorithme d'apprentissage, par exemple) peuvent être traités de façon concurrente.
    2. Parallélisme d'exemple d'apprentissage : lorsqu'un algorithme d'apprentissage supervisé gère simultanément plusieurs exemples, les calculs relatifs à chacun d'eux peuvent être traités de façon concurrente.
        3. Parallélisme de couche : les différentes couches d'un modèle connexionniste non récurrent peuvent traiter simultanément différents exemples dans le cadre d'un calcul avec pipeline.
            4. Parallélisme de neurone : les différents neurones d'une même couche peuvent travailler de façon concurrente. C'est cette forme de parallélisme qui est en général désignée lorsqu'on emploie l'expression *parallélisme neuronal*.
                5. Parallélisme de connexion : les différentes connexions d'un neurone peuvent travailler de façon concurrente. Cette forme de parallélisme est également souvent incluse avec la précédente dans l'expression *parallélisme neuronal*.
                    6. Parallélisme de chiffre : cette forme de parallélisme (principe des arithmétiques parallèles) n'est évidemment pas propre aux réseaux de neurones.

Cette classification peut varier d'une étude à l'autre [Sin90, NS92, SG93], mais il est en général aisé de se rapporter à celle de [NS92].

Si les aspects parallèles 1 et surtout 2 correspondent bien aux implantations parallèles à gros grain qui ont été réalisées [HA94, HT94, GPM94, FSS97, MI97]), les implantations à grain fin ne sont pas forcément basées sur les aspects parallèles 4 et 5. Beaucoup de ces dernières exploitent les régularités des calculs vectoriels induits, aussi bien dans le cadre d'implantations parallèles logicielles que matérielles (basées alors par exemple sur des notions complexes de pipeline au sein d'architectures systoliques).

L'objet de cette section n'est pas de répertorier et classifier les différents travaux d'implantation parallèle de réseaux connexionnistes en fonction de leur exploitation des différents niveaux de parallélisme cités ci-dessus (pour une telle étude, on se reportera à [Gir99], et plus récemment à [OR05]). Le but de cette discussion préliminaire est seulement de mieux situer le parallélisme connexionniste sur la base de la classification de [NS92].

1.2.2 Tentative de définition

De manière concrète, le cœur du parallélisme connexionniste est ce qui est classiquement désigné par parallélisme neuronal, c'est à dire le parallélisme de neurone et le parallélisme de connexion, autrement dit la capacité de plusieurs unités de calcul neuronal à travailler de manière concurrente, mais aussi leur capacité à gérer de manière simultanée les différents flux d'informations qui leur parviennent.

Une telle conception n'exclut pas d'exploiter d'autres types de parallélismes, et notamment le parallélisme d'exemples et le parallélisme de chiffres, ou encore le parallélisme de couches lorsque celui-ci est pertinent et se traduit par exemple par l'optimisation de calculs sous la forme de pipelines. Mais il est essentiel de ne pas mettre sur le même plan ce qui est purement technologique et ce qui fonde le paradigme de calcul distribué qu'est le connexionnisme.

Plus encore, réduire le parallélisme connexionniste au parallélisme neuronal tel qu'il vient d'être décrit reviendrait à le limiter à une vision descriptive de ses aspects distribués, alors qu'il s'agit davantage d'une approche globale de conception de modèles intrinsèquement distribués privilégiant l'échange d'informations par rapport aux calculs dans le but conjoint de résoudre des problèmes complexes et de les résoudre en bénéficiant concrètement de tous les atouts liés à un calcul numérique massivement distribué, modularisé, décentralisé et localisé.

1.3 Le calcul reconfigurable et les modèles massivement distribués

Les implantations de réseaux de neurones sur circuit ASIC (*application specific integrated circuit*) permettent d'obtenir les meilleures performances en termes de temps de calcul, taille, consommation. Elles bénéficient de la possibilité d'implanter directement, neurone par neurone, connexion par connexion, certaines structures neuronales. Dans ce cadre, les implantations analogiques ont une place privilégiée ([NL95, Mic97]), car elles s'adaptent bien aux fonctionnalités simples des neurones et permettent d'atteindre un rapport entre vitesse de calcul, densité et consommation hors de portée des implantations numériques. Cependant la conception et l'utilisation de ces circuits se heurtent à de nombreux problèmes, tels que leur manque de précision, leur manque de robustesse, leur sensibilité à la technologie utilisée et aux conditions d'utilisation, la difficulté de l'apprentissage sur circuit, et d'autres problèmes plus techniques tels que le stockage des poids, la difficulté de simuler de façon fiable le comportement électrique d'un circuit avant fonderie, le manque de cellules standards, etc ([CAL96, Mur92, MGP97, DC95, Dra00]). De plus, leur manque d'accessibilité aux non-experts de l'analogique limite fortement leurs domaines d'application, d'autant plus que les travaux d'implantation analogique sont encore peu réutilisables. En revanche il existe des modèles neuronaux qui semblent avoir été conçus exclusivement pour une implantation analogique, tels que les CNN à temps continu (cellular neural networks [CY88, APFP97, AFD$^+$02]). Plus récemment, l'émergence de l'intérêt de la communauté connexionniste pour les réseaux de neurones à trains d'impulsions, et les besoins très importants en calcul pour l'étude des dynamiques très complexes de ces réseaux, a motivé des travaux de grande ampleur pour la conception de circuits mixtes analogique-numérique insérés dans des systèmes capables de simuler en parallèle de grande population de neurones impulsionnels ([PGMS07, SGMM06]), avec plus ou moins de finesse dans la modélisation des mécanismes biologiques sous-jacents ([RTB$^+$07, ZBT$^+$06]). Il apparaît en effet que les circuits analogiques sont particulièrement bien adaptés à une implantation dense et efficace de ces neurones, même si des travaux récents à base de circuits numériques reconfigurables montre que l'analogique n'est pas incontournable [RHTF03, Hik05, UPRS05, SvC06, GTH07]. On notera que

les travaux de [PGMS07, SGMM06], ou même le projet *Blue Brain*[1] en ce qui concerne un niveau de parallélisme plus logiciel, sont essentiellement orientés vers les expérimentations en neurosciences, et non sur l'exploitation des réseaux impulsionels comme paradigme de calcul distribué.

Des implantations de réseaux de neurones sur circuits numériques ont également été réalisées (cf [Ham95, TCIF95, JPD$^+$92, FNH97]). La réalisation de circuits numériques offre l'avantage d'être plus tolérante vis-à-vis du type de modèle neuronal à implanter, réutilisable lors de changements de technologie sans travail excessif, robuste d'utilisation, et permet d'atteindre des performances très au-delà des programmations logicielles ([Ham95]). Par rapport aux implantations analogiques, la perte de densité peut amener à limiter le circuit à la réalisation d'une partie limitée d'un réseau de neurones (même si les tailles des réseaux implantés en analogiques sont réduites, mais pour d'autres raisons). Ceci implique une utilisation séquentielle du circuit pour calculer la totalité du réseau neuronal ([EFG$^+$97]).On aboutit au principe des neuroproceseurs, désormais principalement envisagés comme coprocesseurs neuronaux et non comme composants de neuro-ordinateurs ([WAK$^+$96]).

Les principaux inconvénients d'une implantation de réseaux de neurones sur circuits ASIC numériques sont les suivants (la plupart étant également des défauts rencontrés pour les circuits analogiques).

- Les coûts et temps de développement sont très importants (surtout pour un nombre limité de pièces) et ne cessent d'augmenter avec les technologies de plus en plus fines.
- Les circuits obtenus manquent d'adaptabilité. Il faut donc en général prédéterminer de façon logicielle le meilleur réseau de neurones pour l'application considérée.
- La phase de test reste nécessaire, même si elle est moins cruciale que pour des circuits analogiques, car des phénomènes matériels non prévus peuvent apparaître.
- Les structures neuronales les plus classiques ne permettent pas, essentiellement pour des raisons de régularité et de connectivité démesurée, des implantations numériques exploitant directement l'architecture connexionniste donnée.

Malgré les défauts cités dans les cas analogique et numérique, l'implantation de réseaux de neurones sur circuits intégrés reste essentielle pour certaines applications. Cependant, leur conception concrète n'est pas l'objet des travaux rapportés dans ce manuscrit. Proposer l'approche connexionniste comme pratique de calcul distribué suppose avant tout une grande flexibilité et une grande indépendance vis-à-vis des technologies.

1.3.1 Une nouvelle adéquation

Les principaux défauts cités pour les circuits ASIC numériques peuvent être contournés en utilisant un matériel reprogrammable, comme les FPGA (*field programmable gate arrays*), au prix d'une vitesse d'exécution plus faible, mais en progrès constant, et d'une taille disponible plus réduite, mais qui augmente sans cesse.

Les FPGA, tels que les FPGA Xilinx, sont basés sur une matrice de cellules appelées CLB (*configurable logic blocks*). Chaque CLB peut implanter des fonctions logiques simples (4 ou 5 entrées booléennes) avec quelques éléments de stockage (bascules par exemple) et des multiplexeurs. En fonction des capacités des CLB, certains opérateurs peuvent être implantés efficacement ou pas sur des FPGA. Les FPGA Xilinx (par exemple les différentes séries Virtex) sont ainsi parmi les mieux adaptés à l'implantation d'opérateurs sériels, en raison d'un nombre d'éléments de stockage relativement élevé par rapport à d'autres FPGA. Les CLB peuvent être connectés grâce à une structure

[1] `http ://bluebrain.epfl.ch/`

de routage configurable. Dans un Xilinx Virtex, les CLB peuvent être efficacement connectés aux CLB voisins et aux CLB de la même ligne ou colonne de la matrice de cellules. La structure de communication configurable peut également connecter des CLB à des blocs d'entrée/sortie qui gèrent les tampons d'entrée/sortie du circuit.

Une implantation à base de FPGA peut donc utiliser la flexibilité des configurations des CLB et de leurs communications pour s'adapter simplement à l'application visée, tandis que les implantations VLSI habituelles doivent être profondément modifiées puis refondues lorsque certaines caractéristiques de l'application changent. La conception sur FPGA demande la description de quelques blocs de base, d'élément de contrôle et d'un schéma de communication. Les outils de synthèse permettent ensuite de placer automatiquement et de façon fiable l'implantation décrite sur le circuit utilisé.

Les principaux atouts de cette approche intermédiaire entre matériel et logiciel par rapport aux circuits ASIC sont les suivants :

- Le coût des circuits reconfigurables est très inférieur à celui des circuits dédiés.
- Le temps de développement est également très inférieur.
- Certains supports sont réutilisables.
- Un prototypage direct est possible avec ces implantations matérielles, par reprogrammation du circuit.
- Il est possible, avec un même circuit, d'implanter des phases de calcul très diverses d'une même application embarquée.
- Une implantation sur matériel configurable peut bénéficier ultérieurement des progrès rapides des constructeurs de ces circuits.
- Une grande partie du code est réutilisable pour synthétiser des circuits ASIC si les performances obtenues sur FPGA justifient un tel investissement.
- Ce type d'implantation est plus rapidement accessible pour les non-experts de la conception de circuits intégrés.

1.3.2 Une tâche encore ardue

Malgré les avantages cités plus haut, implanter des modèles connexionnistes sur circuits reconfigurables nécessite encore un important travail. Les implantations de réseaux de neurones sur FPGA doivent résoudre de nombreux problèmes. Ces problèmes concernent également toute implantation matérielle numérique de réseaux connexionnistes. Mais ces difficultés sont exacerbées en raison des contraintes spécifiques des circuits FPGA, notamment lorsqu'il s'agit de problèmes liés à la taille des opérateurs. Ainsi les principaux obstacles sont :

1. Les supports matériels ont une topologie essentiellement bidimensionnelle, même si l'utilisation des couches de métal multiples permet une structure de communication nettement plus complexe qu'une simple grille. Une telle topologie convient mal aux structures complexes (même si elles sont souvent régulières) des graphes de connexion des réseaux de neurones. Ce problème topologique est renforcé pour les FPGA, pour lesquels l'utilisateur doit tenir compte d'une structure sous-jacente déjà spécifiée, sans pouvoir bénéficier d'une combinaison spécifique astucieuse des couches de métal.
2. Les neurones ont un nombre élevé de liens d'entrée parallèles dans la plupart des modèles, ce qui est incompatible avec les implantations matérielles numériques qui ne gèrent que des degrés entrants (fan-in) réduits. Dans le cas de FPGA, ceci peut se traduire par l'utilisation de

CLB exclusivement destinés à augmenter les capacités de routage, d'où une perte de ressources de calcul.

3. Les réseaux neuronaux mettent en jeu divers opérateurs coûteux en surface, surtout pour des calculs de précision suffisante. Bien que les réseaux de neurones n'exigent pas en général des calculs de précision élevée, la précision requise dépend de l'application.
4. Le stockage des poids ne peut pas toujours être assuré par le circuit FPGA pour les réseaux de neurones de grande taille. La gestion des accès mémoire est donc à prendre en compte. Les évolutions récentes des FPGA permettent de répondre partiellement à ce problème grâce aux blocs mémoires qu'ils contiennent.

Encore une fois, l'objet de cette section n'est pas de recenser les différentes techniques et méthodologies employées pour contourner ces difficultés d'implantation. On pourra se reporter à [Gir00c] pour une telle étude. L'évolution dans le temps des travaux mêlant circuits FPGA et réseaux neuronaux montre d'une part la place de plus en plus dominante de ces supports parallèles dans les travaux d'implantations parallèles de ces réseaux, et d'autre part la place toujours réduite dans ces travaux d'une réflexion approfondie sur la nature même du paradigme de calcul connexionniste et de ses liens avec les technologies reconfigurables. L'optimisation des implantations basées sur des modèles prédéfinis reste l'objet principal de la plupart de ces travaux, tandis que des approches plus en amont, bien que plus performantes et génériques, peinent à s'imposer.

1.3.3 Vers de nouveaux défis

Au-delà des difficultés citées précédemment, et inhérentes à la nature des calculs connexionnistes malgré l'adéquation entre le grain de parallélisme de ces modèles et celui des circuits reconfigurables, un des atouts principalement cités pour ces supports est fortement mis à mal par leurs évolutions récentes. En effet, les premiers travaux d'implantation, s'ils étaient conceptuellement capables de s'adapter à des architectures connexionnistes de taille quelconque, et donc capables de supporter le passage à l'échelle, pouvaient bénéficier ultérieurement des progrès rapides en taille et en vitesse de ces circuits. La tendance actuelle montre que ces progrès s'accompagnent de l'adjonction sur le même circuit de multiples ressources telles que des multiplieurs, des blocs mémoires, ou encore des DSP, encore en nombre réduit et plus ou moins répartis uniformément sur le FPGA. La gestion des ressources ainsi mises à disposition offre des capacités supplémentaires, mais elle impose une refonte générale de certaines méthodes d'implantation. De plus, la portabilité des travaux est remise en cause, puisque chaque famille de circuits définit ses propres ressources additionnelles et leur interaction avec les ressources logiques classiquement présentes dans les FPGA.

Cette évolution, si elle permet ponctuellement des gains importants en performance, constitue néanmoins un risque de retour à une certaine dépendance vis-à-vis des technologies. Il s'agit là d'un écueil que l'approche connexionniste se doit d'éviter pour prétendre constituer une pratique alternative de calcul distribué.

1.4 Connexionnisme et calcul spatial, le besoin d'inspiration

Sans avoir encore atteint une limite stricte dans les gains en finesse et en vitesse, la conception de circuits (et notamment de processeurs) semble devoir se heurter prochainement à l'impossibilité de suivre indéfiniment la loi de Moore compte tenu de la miniaturisation désormais quasi-extrême des transistors [Kis02, Int06]. Les principaux problèmes se situent autour de la dissipation d'énergie,

du synchronisme global (l'acheminement des signaux d'horloge constituant une part essentielle de la consommation des circuits actuels) et du coût induit [LW06]. Différents chercheurs et groupes de chercheurs se tournent désormais vers une conception de circuits constitués de multiples unités de calcul génériques plus ou moins régulièrement organisés [Deh02]. Les circuits reconfigurables peuvent être vus comme les précurseurs de cette évolution. Mais ils restent actuellement conçus dans une perspective d'universalité et de prédictibilité des configurations permises, au prix de ressources de routage sans cesse plus contraignantes, et au prix de logiciels de synthèse peinant à conserver un rôle de compilateur générique vis-à-vis des algorithmes à implanter.

Certains chercheurs estiment que la conception de circuits va devoir passer de la question "comment plaquer un algorithme donné sur un ensemble de ressources logiques" à la question "comment faire en sorte que les unités de calculs coopèrent pour résoudre une tâche donnée" [LW06]. Une des voies explorées est l'utilisation de l'algorithmique connexionniste. Néanmoins la complexité des problèmes en jeu exige de dépasser le contexte simplifié des modèles connexionnistes dédiés à la classification ou à la régression non linéaire. Les recherches évoquées se tournent vers le connexionnisme bio-inspiré, et notamment l'étude de phénomènes complexes d'émergence [BQ06]. L'idée d'une pratique de calcul distribué, et notamment d'une pratique de calcul distribué d'inspiration biologique, prend donc un relief nouveau à la lumière de cette évolution attendue.

Chapitre 2

Les modèles connexionnistes comme architectures de calculs parallèles

Les réseaux de neurones sont surtout comparés à d'autres modèles ou méthodes du point de vue de leur capacité à résoudre une tâche donnée. Un apport propre aux modèles connexionnistes se situe pourtant dans l'uniformité de leurs méthodes d'apprentissage, et dans leurs calculs basés sur la coopération d'unités élémentaires. Ces unités réalisent des opérations homogènes et simples, et leur schéma d'interconnexion leur permet de travailler de façon *concurrente*. Ce parallélisme connexionniste est un atout majeur, à condition notamment de le maintenir lors de l'implantation des réseaux de neurones.

Or ces implantations se heurtent principalement à une non adéquation entre le parallélisme des réseaux neuronaux et les contraintes topologiques des supports d'implantation matérielle. Cette inadéquation a pour effet de motiver parfois l'emploi de méthodes de parallélisation complexes qui n'ont aucun rapport avec le parallélisme naturel des réseaux connexionnistes. De telles difficultés rendent nécessaire le développement de méthodes de parallélisation directe des réseaux de neurones, ainsi que la définition de modèles connexionnistes cohérents avec les contraintes d'implantation. L'objet de ce chapitre est de montrer que ce travail passe avant tout par une étude approfondie de la gestion locale distribuée du flux des informations échangées par les neurones au travers de leurs interactions.

La section 2.1 a pour but de décrire les grandes étapes d'une implantation parallèle matérielle de réseaux neuronaux basée sur une exploitation directe du parallélisme connexionniste, puis de dresser un état des lieux de la place qu'occupe cette approche dans l'ensemble des travaux d'implantations de réseaux de neurones sur circuits reconfigurables, aussi bien parmi les techniques simples d'implantation que parmi les méthodes plus avancées.

La section 2.2 propose ensuite des approches plus globales, où l'idée même de contraintes d'implantation matérielle est modélisée en amont de la définition des paradigmes de calcul connexionniste, afin d'intégrer à ces modèles une meilleure tolérance à ces contraintes.

2.1 Les implantations neuronales sur FPGA

Comme évoqué précédemment, les FPGA constituent un choix peu coûteux, accessible et flexible pour les implantations matérielles. Ils ont également des avantages plus spécifique en ce qui concerne les implantations neuronales.

– Les FPGA reprogrammables facilitent le prototypage : dans la plupart des applications, plusieurs architectures de réseaux de neurones sont testées afin de trouver la plus efficace. Ce processus peut être directement réalisé avec l'efficacité matérielle d'une implantation sur FPGA, sans aucun coût additionnel. Et une architecture neuronale performante déterminée et implantée lors de ce prototypage peut toujours ensuite être remplacée plus tard par une meilleure architecture sans avoir à concevoir et fondre un nouveau circuit.
– L'apprentissage sur circuit est souvent considéré comme difficile et inutile. Il est implanté en effet assez rarement. Mais l'apprentissage sur circuit est habituellement la cause d'une perte d'efficacité dans une implantation matérielle, car il requiert des opérateurs spécifiques, une précision accrue, etc. C'est pourquoi l'apprentissage préalable sur PC est naturellement choisi lorsqu'aucun apprentissage dynamique n'est nécessaire. Au contraire sur un FPGA reconfigurable, un apprentissage sur circuit peut être réalisé avant qu'une implantation optimisée spécifique du réseau neuronal appris ne soit placée sur le même circuit.
– Les FPGA peuvent être utilisés pour des applications embarquées, lorsque la robustesse et la simplicité des calculs neuronaux est requise, même s'il s'agit de produire peu d'unités de ces systèmes embarqués.
– Les implantations conçues sur FPGA peuvent aisément être portées sur des FPGA plus récents améliorés. Il s'agit là d'un avantage majeur, puisque les performances des FPGA en termes de vitesse et de surface suivent (et même dépassent) la loi de Moore. Même des réseaux de neurones de très grande taille vont pouvoir être prochainement implantés directement sur un unique FPGA, pourvu que la méthode d'implantation tolère suffisamment le passage à l'échelle.

L'objet de cette section est de présenter très schématiquement les différentes questions qui se posent lorsqu'on souhaite réaliser une implantation parallèle d'un réseau de neurones sur FPGA, puis de faire un tour d'horizon très succinct des principales approches utilisées, et de situer parmi elles celles qui relèvent d'une exploitation voulue du parallélisme connexionniste.

2.1.1 Implanter un réseau de neurones sur FPGA

Les réseaux connexionnistes offrent des architectures distribuées qui peuvent être directement utilisées pour définir l'architecture des implantations parallèles matérielles. Explicitée telle quelle, cette assertion semble supposer que, pour des implantations basées sur ce principe, la définition du modèle neuronal est l'essentiel du travail et fournit directement l'implantation. Bien entendu, il s'agit là d'une vision extrêmement simplifiée. De nombreuses questions spécifiques à l'implantation se posent, plus ou moins dépendantes les unes des autres.

Détermination des enjeux de l'implantation Suivant les cas, les implantations matérielles visent à satisfaire une ou plusieurs contraintes telles que l'embarquabilité, la vitesse de prototypage, la décision en temps réel, une faible consommation, etc. Même si les enjeux liés ne sont pas forcément contradictoires, il faut clarifier les objectifs principaux du travail d'implantation pour guider les différents choix technologiques à faire.

Choix de la technologie reconfigurable Même si peu de fabricants de circuits reconfigurables existent, leurs circuits, et même leurs familles de circuits, sont fondés sur des principes assez différents, plus ou moins bien adaptés aux calculs connexionnistes prévus.

Prise en compte de l'apprentissage L'apprentissage embarqué n'est pas toujours pertinent. Il correspond à des situations de prototypage ou d'adaptation embarquée. Dans tous les cas, y compris lorsqu'un apprentissage sur le circuit n'est pas nécessaire, il faut accorder un intérêt particulier au positionnement de l'apprentissage par rapport à l'implantation, principalement en ce qui concerne l'adéquation des arithmétiques (paramètres, données, mises à jour, etc).

Choix de l'arithmétique Les choix arithmétiques sont cruciaux, et peuvent être guidés à la fois par les contraintes d'implantation et par la nature des calculs connexionnistes.

Détermination de la précision La surface des opérateurs constituant un inconvénient majeur dans la mise en place d'implantations neuronales massivement parallèles, l'optimisation de la précision suivant les performances attendues et suivant la nature des données manipulées, est une étape importante, qui a d'ailleurs motivé plusieurs études générales des besoins en précision des modèles connexionnistes [HHP90, HH93, Dra02, SMA07].

Estimation des performances Il est très difficile de réaliser une comparaison directe des performances des différentes implantations, en raison de la variété des architectures neuronales, des choix technologiques, des types de circuits reconfigurables, et de l'évolution dans le temps de leurs capacités. Différentes méthodes d'évaluation existent, plus ou moins pertinentes suivant les enjeux de l'implantation.

2.1.2 Les principales méthodes d'implantation

La méthode la plus aisée pour contourner les difficultés évoquées précédemment est d'implanter des réseaux de très petite taille. Plus généralement, les solutions habituelles incluent :

- l'implantation des seuls calculs propagés de généralisation ([BH94, SGMb94]),
- l'implantation d'un neuro-processeur sur le FPGA, destiné à être ensuite utilisé de façon séquentielle ([GSMb96, RHG$^+$96, GP07]),
- l'implantation d'un modèle connexionniste dont la connectivité reste limitée ([PUS96]),
- l'implantation d'un ensemble de neuro-processeurs sur différents FPGA, avec des communications entre ces circuits ([EH94, WMLB00, PKW$^+$01, RHTF03, UPRS05], ou même des systèmes dont la structure est plus proche d'un véritable neuro-ordinateur [KA97, Eli97, RVT07]),
- des calculs simplifiés, une faible précision, des fonctions de transfert fortement discrétisées, etc. ([BAA93]).

Des méthodes plus évoluées ont été envisagées.

- Implantation optimisée de la fonction de transfert par algorithme de type CORDIC (dans [ABDG$^+$97]), ou implantation de multiplieurs rapides ([SGMb94]).
- Utilisation partielle du parallélisme neuronal ([BAJS05, SAMA06]).
- Reconfiguration dynamique du FPGA ([Beu98]).
- Utilisation des mémoires embarquées pour gérer la connectivité ([GHSM00, GCBM00]).
- Utilisation d'une arithmétique à flots de bits aléatoires (bitstream neurons [BH94, vDJST93, NdMM03]). Les opérateurs sont de très petite taille, ne dépendent pas de la précision cherchée, et permettent donc l'implantation directe de réseaux de neurones de taille moyenne. Cependant les contraintes de routage des FPGA posent encore problème pour les réseaux de grande taille. Ce type d'arithmétique induit des problèmes de précision des calculs, surtout en ce qui

concerne les fonctions de transfert. De plus, il faut tenir compte des problèmes de corrélation entre les différents flots de bits lorsque le nombre de couches augmente.
- Utilisation d'une arithmétique sérielle simple ([SvC06]), ou en-ligne. Le choix de la meilleure arithmétique pour les implantations matérielles numériques est discuté dans [SJ95], mais sans que les spécificités des FPGA soient prises en compte, ni que le cas d'une arithmétique en-ligne soit considéré.
- Utilisation d'architectures pipeline systoliques complexes, permettant de maximiser le taux d'occupation des différents unités de calcul sans saturer les ressources de routage.
- Utilisation mixte sur les FPGA récents des ressources logiques et des ressources annexes (multiplieurs dédiés, blocs mémoire).

Il n'est pas question ici de détailler l'ensemble de ces méthodes et des travaux qui s'y rapportent. Il est possible de se référer à [Gir00c, ZS03] ou bien [OR06] pour un tour d'horizon plus approfondi.

2.1.3 Méthodes d'implantations évoluées

Les approches décrites dans la section précédente sont pour la plupart inspirées par un besoin d'optimisation ou de simplification des implantations des calculs connexionnistes en jeu, sans aucune remise en cause des fondements algorithmiques de ces calculs (à l'exception des travaux sur les réseaux de neurones bitstream). Sans aller jusqu'à profondément refonder cette algorithmique connexionniste en fonction des contraintes d'implantation (c'est l'objet de la prochaine section), cette section montre dans un premier temps qu'il est possible et souhaitable de réexprimer les algorithmes connexionnistes selon une perspective centrée sur les interactions locales afin d'aboutir plus directement à une implantation fondamentalement connexionniste.

Cette assertion sera d'abord illustrée par des travaux dans lesquels la prise en compte anticipée des contraintes d'implantation a pu guider le choix de l'algorithmique connexionniste. De façon plus avancée, c'est la nature même du connexionnisme, et donc la gestion locale et distribuée des interactions qui doit être au cœur de toute approche globale du calcul connexionniste. Cela sera illustré au travers de travaux spécifiques à des calculs neuronaux impulsionnels, qui montrent comment l'implantation de réseaux neuronaux à interactions locales récurrentes peut bénéficier d'une approche centrée sur les potentiels d'action échangés, et non sur les calculs opérés sur ces potentiels.

Les contraintes d'implantation comme facteur de choix algorithmique

Avant même de concevoir des modèles connexionnistes dédiés à une pratique de calcul distribué, il est possible, parmi la multitude des modèles et algorithmes connexionnistes classiques existants, de guider son choix en partie en fonction des contraintes d'implantation anticipées.

C'est ce qu'illustre par exemple le travail rapporté dans [GBK06]. L'approche développée dans ce travail concerne l'implantation d'un processus de décision fondé sur une carte auto-organisatrice (SOM : Self-Organizing Map). Cette implantation est effectuée sur un circuit FPGA programmable, en vue d'une future exploitation dans un système embarqué capable de détecter un état d'hypovigilance sur la base d'une dérivation électroencéphalographique (EEG) minimale. Ces travaux constituent davantage une application du parallélisme connexionniste qu'un apport véritable à son étude, et ils ont mené à différentes implantations qui mettent en jeu des modèles connexionnistes divers. C'est pourquoi l'ensemble de ces travaux sera résumé en annexes B et C.

En ce qui concerne l'implantation matérielle décrite dans [GBK06], et résumée en C.1, les

paramètres du réseau de neurones (i.e. les poids des connexions d'entrée/sortie) sont précalculés hors circuit. De façon à réduire la surface de notre implantation autant que possible, nous avons adapté le modèle SOM initial à un critère de distance basé sur une norme L_1, afin de supprimer les opérateurs quadratiques. Le principal avantage de cette implantation est la relation directe entre l'architecture neuronale et sa réalisation matérielle. Le circuit généré ne contient que les opérateurs et communications strictement nécessaires, quelle que soit la taille du réseau. L'implantation requise atteint donc l'objectif fixé de minimiser les ports d'entrée-sortie, et de maximiser le parallélisme interne en vue d'une consommation aussi réduite que possible. Un autre avantage de notre solution est d'être capable de gérer des précisions supérieures, ce qui assure une meilleure adaptabilité aux individus. La principale caractéristique de ce travail réside en un choix non usuel de la mesure de distance employée dans l'algorithme classique des cartes auto-organisatrices. Ce choix induit une étude spécifique, en ce qui concerne d'une part les performances en détection de l'hypovigilance, la sensibilité de l'apprentissage, ou encore l'étude de la précision minimale requise. Ce choix permet enfin une implantation entièrement basée sur une arithmétique sérielle en-ligne limitée à des opérateurs de surface très réduite.

Les calculs asservis aux interactions

Lorsque la topologie du réseau neuronal considéré ne constitue pas un obstacle à une implantation directe de l'architecture connexionniste distribuée, c'est parfois la nature même des calculs qui peut constituer un obstacle à une implantation simple. C'est par exemple le cas des réseaux neuronaux récurrents dont le potentiel de chaque neurone est continuellement mis à jour selon des équations dépendants d'un voisinage local. Dans ces modèles, la difficulté réside dans l'alternance apparemment nécessaire entre phases de calculs, phases de communications, phases de mises à jour. Néanmoins une approche à nouveau centrée sur la gestion des interactions permet de rapprocher l'implantation de la définition initiale du modèle.

Pour illustrer cela, on peut se référer aux travaux rapportés dans [GTH06, GTH07], sur l'implantation numérique massivement distribuée d'un réseau IF-LEGION pour la segmentation de scènes visuelles. Il s'agit là d'un modèle connexionniste utilisant des neurones impulsionnels. Actuellement, une des principales voies de recherche en neurosciences algorithmiques porte sur les propriétés des mécanismes de bas niveau qui sont utilisés par les systèmes neuronaux biologiques. Dans cette approche, le mode de calcul est organisé autour d'événements neuronaux appelés trains d'impulsions (spikes) [GK02]. Des arguments biologiques indiquent que certaines tâches de perception visuelle, ainsi que la plupart des tâches de perception olfactive, ne peuvent être réalisées de manière satisfaisante par les réseaux neuronaux classiques qui utilisent un mode de calcul analogique, où les valeurs représentent les taux moyens de décharge des neurones impulsionnels naturels [HM05]. Les modèles neuronaux impulsionnels sont différents des modèles connexionnistes classiques en ce sens que l'information est transmise au moyen d'impulsions réparties dans le temps. La relation impulsion-temps enrichit la dynamique des modèles impulsionnels qui exploitent le domaine temporel pour encoder et extraire l'information dans les impulsions échangées. Le codage temporel qui réside ainsi dans les trains d'impulsions doit être pleinement exploité, par exemple au travers de la notion de synchronisation neuronale [Sin03].

La plupart des travaux sur les implantations matérielles numériques de réseaux de neurones utilisent des groupes de chiffres binaires pour implanter les réseaux neuronaux classiques où toutes les interactions sont représentées par le taux de décharge moyen des neurones. Et pourtant, les modèles neuronaux impulsionnels, qui sont plus étroitement liés à l'idée de potentiel d'action et de commu-

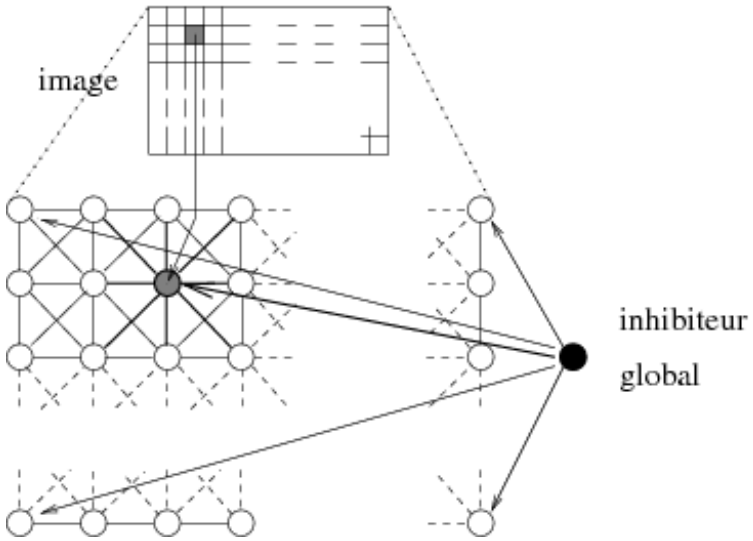


FIG. 2.1 – Architecture 2D du réseau LEGION : un oscillateur neuronal est connecté à ses huit voisins immédiats et à un inhibiteur global. Chaque oscillateur reçoit des stimuli visuels à partir des pixels de l'image d'entrée.

nication par train d'impulsions, semblent plus proches des mécanismes simples de transmission de signaux sur supports matériels numériques. C'est pourquoi l'implantation de réseaux neuronaux impulsionnels sur circuits numériques est devenue une voie de recherche active, comme le montrent de récents travaux [RHTF03, UPRS05, SvC06], alors que les implantations précédentes privilégiaient principalement la densité des supports analogiques. Ce changement est principalement lié à la flexibilité et aux progrès technologiques rapides des FPGA. Des méthodes avancées d'implantation sont maintenant nécessaires pour rendre les implantations numériques capables de manipuler des réseaux de neurones impulsionnels de grande taille. Cela requiert des implantations de surface réduite de ce genre de neurones, basé sur le fait que les calculs impulsionnels conviennent aux implantations numériques : ils sont intrinsèquement binaires, et le mode de communication par impulsion peut être géré par l'échange de simples bits entre neurones, au lieu de valeurs réelles codées. Autrement dit, un simple signal binaire est suffisant pour la communication entre deux neurones, et les besoins en routage pour l'interconnexion des neurones sur un circuit 2D peuvent être simplifiés. De plus, les neurones impulsionnels sont des intégrateurs pondérés qui peuvent décharger et être remis à zéro aussitôt qu'un seuil est atteint, ce qui évite des opérateurs gourmands en surface dans l'implantation matérielle. Le fait de combiner les modèles neuronaux impulsionnels avec des implantations matérielles spécifiques sur FPGA ouvre des possibilités intéressantes pour la conception de systèmes autonomes évolués.

A la suite de travaux sur l'adaptation de modèles neuronaux impulsionnels classique à la perception visuelle, nous avons d'abord décidé d'implanter un modèle impulsionnel, célèbre bien que fortement critiqué, pour la segmentation d'image sur FPGA : le modèle IF-LEGION (integrate-and-fire locally excitatory globally inhibitory oscillator network, [TW95a, TW95b], ainsi que [WT97, CWJ99] pour son adaptation aux images à niveaux de gris). Ce modèle est critiqué en raison de sa plausibilité biologique plutôt simpliste, ainsi que pour les problèmes mathématiques posés lors d'une

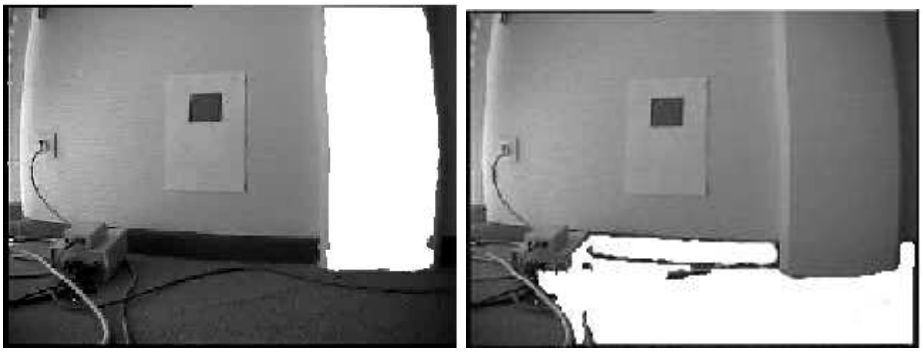


FIG. 2.2 – Segmentation d'image par un réseau IF-LEGION : les neurones émettant des trains d'impulsions simultanément correspondent aux pixels blancs.

implantation matérielle, mais sa structure 2D sous-jacente convient bien aux contraintes topologiques des implantations sur FPGA. Le modèle IF-LEGION est basé sur les notions d'oscillation et de synchronisation des neurones dans le cortex, dont plusieurs travaux montrent l'importance dans plusieurs fonctions perceptives. Il consiste en une grille 2D de neurones impulsionnels (cf figure 2.1 où chaque neurone est connecté à ses 8 voisins) qui reçoivent des stimuli visuels. Des neurones voisins sont couplés par des connexions excitatrices, et un inhibiteur global est activé dès qu'un neurone au moins émet un spike. Ce comportement dynamique est décrit par une équation différentielle qui met à jour le potentiel de chaque neurone en fonction du stimulus d'entrée et des excitations et inhibitions reçues. On trouvera le détail de cette équation en annexe D. Lorsque ce modèle est appliqué à la segmentation d'images, les neurones voisins recevant des entrées similaires se regroupent par synchronisation en raison des connexions excitatrices locales. Des groupes de neurones disjoints vont en revanche se désynchroniser de par le mécanisme d'inhibition globale. La figure 2.2 illustre ce phénomène sur l'environnement visuel d'un robot : les pixels blancs correspondent à un groupe de neurones synchronisés (émission simultanée) qui segmentent ainsi une partie du mur ou du sol.

Notre travail d'implantation s'est principalement concentré sur les solutions économiques en surface d'implantation, basées sur l'utilisation d'une arithmétique sérielle classique au sein d'une architecture de boucles pipelines entrelacées illustrées sur la figure 2.3 : cette figure schématise comment l'implantation d'un neurone détaillée en figure D.4 s'insère dans l'architecture générale de la figure D.3, en situant les chemins de données, les boucles, et les délais qui autorisent cet entrelacement. Ce principe général permet une implantation FPGA totalement parallèle d'un réseau IF-LEGION suffisamment grand pour traiter des séquences d'images issues d'une caméra basse résolution installée sur un robot. Cette implantation est décrite en détail en annexe D.

Ce travail a confirmé l'assertion de [NS92, Gir00c, Gir00a] : le grain de parallélisme des FPGA correspond bien aux calculs neuronaux, de telle sorte que des implantations FPGA de réseaux de neurones peuvent favoriser l'utilisation d'approches neuronales qui maintiennent un calcul totalement distribué et localisé. Dans notre cas, des canaux de communications permanents (ou synapses) ont été par exemple préférés à des systèmes événementiels.

Les résultats de [GTH07] montrent que des implantations efficaces de réseaux de neurones de grande taille peuvent être réalisées à travers l'utilisation de chemins de données spécialisés incluant une arithmétique sérielle. Par rapport aux travaux antérieurs, nous obtenons des améliorations de

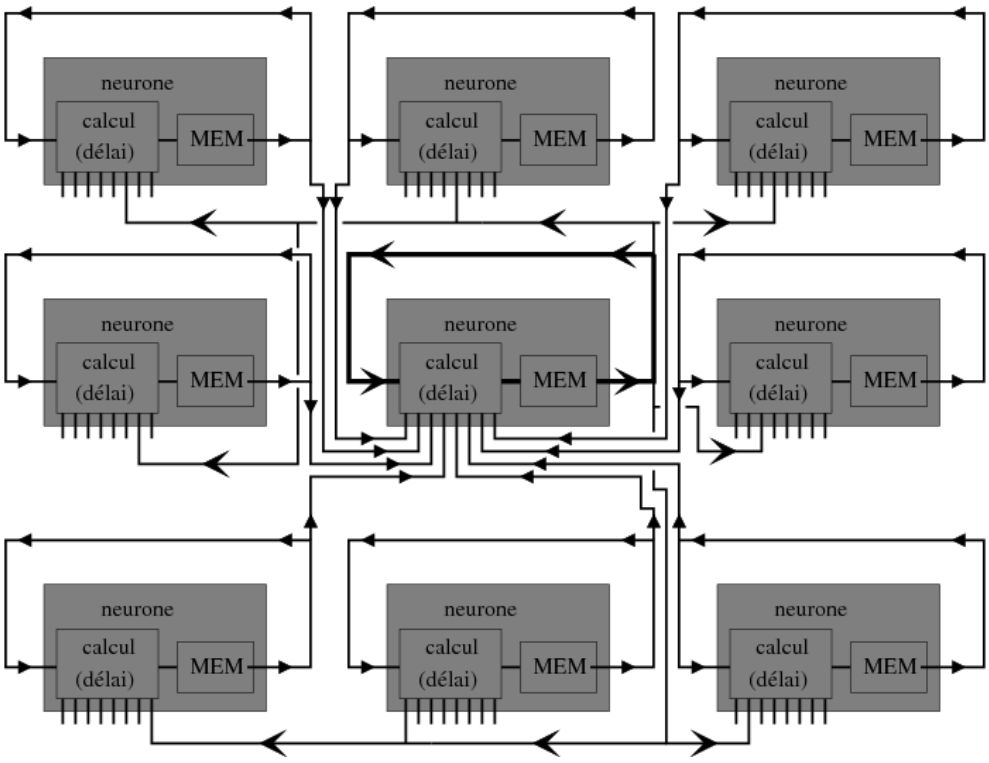


FIG. 2.3 – Boucles pipelines locales entrelacées dans l'implantation du réseau IF-LEGION.

performance significatives : ainsi les résultats indiqués dans le tableau D.1 dépassent largement ceux présentés dans [FSN02] où une architecture simplifiée est choisie et n'utilise que des poids binaires et une inhibition locale. Nos travaux permettent également d'atteindre des performances très légèrement supérieures à celles de [SvC06], les deux approches ayant d'ailleurs été développées simultanément. Mais cette comparaison indique principalement que les principes d'implantation communs qui ont été utilisés apparaissent ainsi comme des concepts fondamentaux pour de futures implantations, tandis que les principes complémentaires de généricité du modèle neuronal et de traitement intégré des entrées-sorties devraient être pris en compte pour fonder les prochaines améliorations importantes des implantations matérielles de réseaux impulsionnels.

Notre implantation est en mesure de traiter des résolutions d'images faibles ou modérées pour des applications embarquées avec un faible coût en surface dans la technologie FPGA la plus récente. Tous les calculs sont organisés au sein de ces chemins de données entrelacés, où des LUT (lookup tables) fournissent continuellement les entrées et sorties de tous les calculs parallèles et de tous les pipelines. Cette même approche a également été appliquée dans [GB07] à un modèle de contrôle harmonique distribué dont les interactions locales obéissent à des règles similaires et dont la topologie est également bidimensionnelle. Néanmoins ces travaux s'écartent d'une pratique purement connexionniste de calcul distribué.

2.2 Pratique connexionniste et nouveaux paradigmes de calcul

Parmi l'ensemble des approches évoquées dans les sections précédentes, la plupart optimisent leurs performances au détriment de leur généralité. Si cela est parfois suffisant, la gestion sur circuit de la complexité des connexions d'un modèle neuronal reste un obstacle majeur et empêche de profiter pleinement de l'augmentation importante du nombre de portes logiques des circuits FPGA. Les quelques travaux issus d'une réflexion sur les contraintes de l'implantation en amont de la détermination des modèles neuronaux et de leur apprentissage offrent les meilleures perspectives. Ces travaux se concentrent sur la nature des informations échangées par les neurones, et sur la gestion locale de ces échanges. C'est déjà le cas des modèles bitstream, fondés sur le souci de minimiser les surfaces d'implantations et les informations échangées, et qui ont fait l'objet d'une modélisation spécifique de leurs calculs et de leur apprentissage.

De telles approches en amont permettent de définir des paradigmes de calcul connexionniste destinés à résoudre les principaux problèmes rencontrés en phase d'implantation, et l'utilisation de tels paradigmes mène alors naturellement à des modèles neuronaux plus tolérants vis-à-vis des contraintes d'implantation. Au-delà de l'obtention d'implantations plus aisées et efficaces, ces approches permettent surtout de mieux cerner ce que peut être une pratique connexionniste de calcul distribué, par adéquation naturelle de l'algorithmique distribuée et de ses supports.

L'approche ainsi défendue dans cette section est donc de modifier la structure même du calcul neuronal en fonction du matériel cible, c'est à dire de proposer un paradigme de calcul connexionniste directement inspiré des contraintes matérielles et donc plus naturellement tolérant envers ces contraintes. Comme suggéré précédemment, les deux principaux défis restent la complexité des connexions et le passage à l'échelle, défis que tentent de résoudre les approches décrites respectivement en sections 2.2.1 et 2.2.2.

2.2.1 L'écueil des connexions

Un premier exemple est une classe de modèles connexionnistes qui autorise une parallélisation en temps et en espace de la gestion des connexions. Ces modèles utilisent un nouveau mode de calcul connexionniste qui permet de définir des réseaux fonctionnellement équivalents aux réseaux de neurones classiques, mais basés sur une architecture neuronale facile à implanter directement sur circuit. Cette approche est basée comme les FPGA sur le principe d'un ensemble de ressources dont les interactions sont à définir, ces ressources étant choisies de telle sorte que les calculs obtenus soient de même nature que dans les modèles neuronaux classiques. Par analogie, les modèles connexionnistes déduits de cette approche sont appelés FPNA (field programmable neural arrays). Les différents aspects théoriques et appliqués de ce concept ont été regroupés dans [Gir06a, Gir06b].

Principes

Les FPNA sont conçus comme des FPGA : ce sont des ensembles de ressources dont les interactions sont librement configurables. Ces ressources (d'une part les connexions, ou liens, et d'autre part les opérateurs neuronaux, ou activateurs) sont définis de façon à réaliser les calculs de réseaux de neurones classiques, mais ils se comportent de manière *autonome*. On peut ainsi localement *connecter* n'importe quel lien à n'importe quelle autre *ressource locale* (lien ou activateur). Le but de la contrainte de localité est de réduire les problèmes topologiques, tandis que les liens directement connectés génèrent des combinaisons de poids plus variées. La figure 2.4 illustre les ressources

locales d'un FPNA et leurs connexions possibles. En conséquence, de nombreuses connexions *virtuelles* (successions de liens directement connectés) peuvent être réalisées grâce à l'application d'un protocole simple de calcul aux ressources neuronales d'un réseau à la connectivité réduite. Ce paradigme de calcul neuronal permet à un réseau connexionniste simplifié de remplacer un réseau virtuellement complexe.

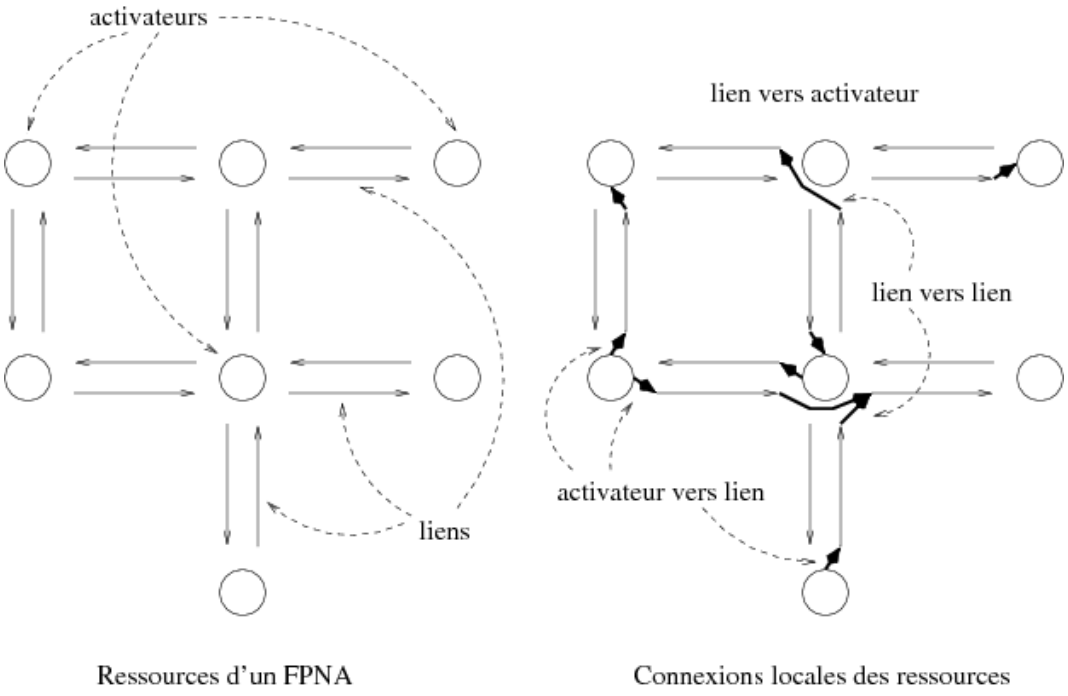


FIG. 2.4 – Ressources d'un FPNA et connexions locales

Dans un modèle connexionniste classique, chaque lien est une connexion entre la sortie d'un neurone et une entrée d'un autre neurone. C'est pourquoi le nombre d'entrées de chaque neurone est son degré entrant dans le graphe de connexion. A l'inverse, les ressources neuronales deviennent autonomes dans un FPNA : leurs dépendances sont fixées librement, et le traitement qui en résulte est plus complexe que dans les modèles neuronaux classiques. Comme dans les FPGA, la configuration des FPNA porte à la fois sur les interconnexions des ressources et sur les fonctionnalités de ces ressources.

Dans la plupart des cas, concevoir un réseau FPNA consiste à choisir une architecture neuronale classique qui convient à l'application visée, puis à déterminer un FPNA qui à la fois convient au support d'implantation choisi et est fonctionnellement équivalent[1] au réseau de neurones choisi. Finalement, le FPNA obtenu est directement placé sur le support matériel grâce à une implantation complètement modulaire : des blocs configurables prédéfinis sont donnés pour chaque ressource neuronale, et ils sont assemblés en accord avec l'architecture du FPNA naturellement tolérante aux contraintes matérielles.

[1]Le type de fonction réalisé est équivalent. Ensuite un algorithme d'apprentissage est utilisé pour régler les poids du FPNA en fonction de l'application.

Description formelle, applications et implantations

L'annexe A définit de manière précise les FPNA. Les principaux résultats théoriques et méthodes d'implantation sont également résumés en annexe A.

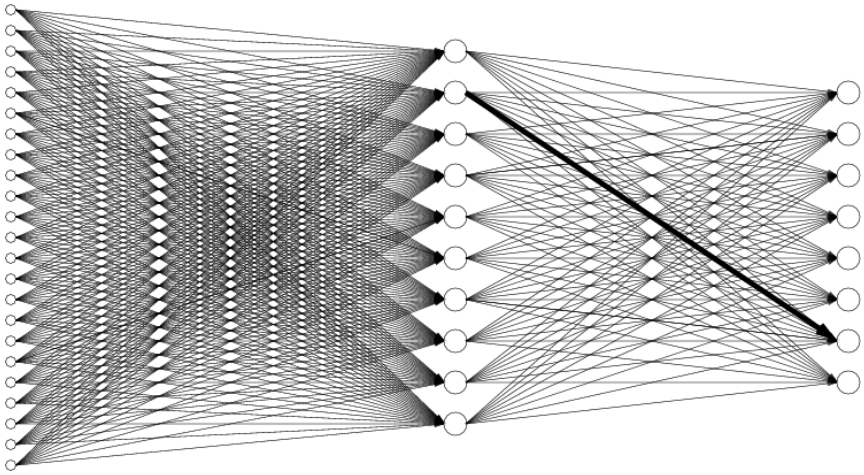


FIG. 2.5 – Perceptron multicouche pour la classification des états de vigilance

Un exemple d'application, décrit plus en détail en annexe C.2, se situe dans le cadre des travaux sur la détection de l'hypovigilance. Le paradigme FPNA a ainsi permis l'implantation embarquée de la classification des différents états de vigilance à partir de signaux électroencéphalographiques (travaux préalables à la détection de l'hypovigilance). Le modèle neuronal initialement employé était un classique perceptron multicouche à 23 entrées, une couche cachée de 10 neurones et 8 neurones de sortie (voir figure 2.5) dont l'implantation parallèle directe sur FPGA excédait les capacités en surface et en routage de ces circuits. Un réseau de neurones fonctionnellement équivalent à ce MLP a été construit sur la base du paradigme FPNA (voir figure 2.6), et son implantation décrite en annexe C.2 permet de satisfaire les contraintes matérielles d'un système ambulatoire. Ces résultats sont essentiellement dues à de fortes simplifications topologiques illustrées sur la figure 2.6 : la combinaison des liens en gras crée une connexion virtuelle qui correspond à la connexion en gras de la figure 2.5.

Apports

Plus généralement, cette approche répond partiellement aux défis variés des implantations neuronales sur FPGA (ou plus généralement sur circuits numériques) de la façon suivante :

Topologies 2D et degrés de connexion : Le paradigme de calcul FPNA permet de contrebalancer une topologie simplifiée de réseau de neurones par une complexité locale accrue des interactions entre les ressources neuronales. Cette complexité peut être réglée de façon à rendre la topologie simplifiée compatible avec les contraintes matérielles : un compromis doit être trouvé, puisque l'apprentissage des FPNA devient de plus en plus difficile quand la complexité des connexions locales configurées s'accroît.

Opérateurs consommateurs de surface : Les FPNA utilisent les mêmes opérateurs que les réseaux de neurones classiques, mais une topologie simplifiée se traduit par un nombre plus réduit de

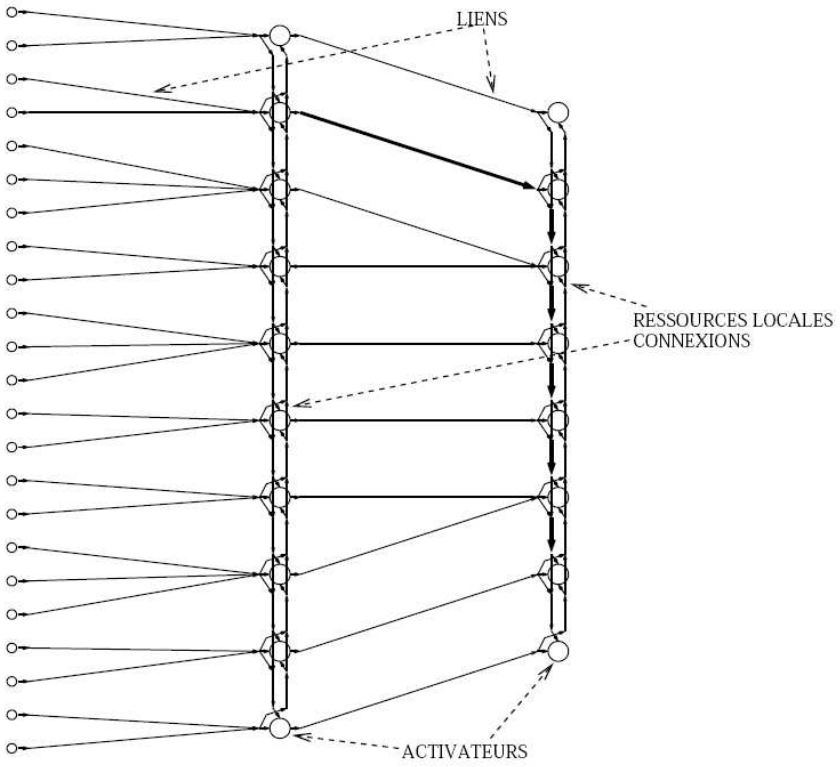


FIG. 2.6 – FPNA pour la classification des états de vigilance

tels opérateurs, de telle sorte que les problèmes de surface et d'interconnexion peuvent être simultanément résolus.

Stockage des poids : Le paradigme FPNA permet de créer de nombreuses connexions virtuelles malgré un nombre réduit de liens de communication. Par conséquent, la simplification de topologie résulte principalement en une diminution du nombre de liens, et donc moins de poids.

L'application des FPNA à des problèmes types a montré qu'une simplification topologique significative peut être obtenue avant d'être confronté aux limitations dues à la complexité de l'apprentissage combiné des connexions virtuelles. Le paradigme FPNA, centré sur la complexité des connexions, introduit donc une nouvelle conception du calcul neuronal qui permet de simplifier les topologies des réseaux de neurones classiques sans perte significative en capacité d'approximation. La ligne directrice de cette approche est l'autonomisation des interactions.

2.2.2 Le passage à l'échelle

Maîtriser la complexité topologique des modèles connexionnistes destinés à l'implantation est une étape importante vers une pratique de calcul distribué, mais une telle pratique doit pouvoir traiter des problèmes de plus en plus ardus. Dans le chapitre suivant, nous verrons comment la biologie peut constituer une source d'inspiration pour la définition de modèles connexionnistes de

plus en plus puissants. Néanmoins, du point de vue purement architectural, de tels modèles sont amenés à utiliser une quantité toujours croissante de ressources neuronales.

Inévitablement se posera donc le problème du passage à l'échelle. Là encore, une approche en amont est souhaitable. Plutôt que de constater a posteriori que les modèles proposés dépassent les capacités des supports d'implantation, et devoir alors développer des méthodes complexes et peu portables pour séquentialiser le calcul connexionniste massivement distribué proposé, il serait nécessaire de mieux formaliser les contraintes liées au manque de surface, puis de proposer des paradigmes de calcul connexionniste intrinsèquement tolérants à ces contraintes.

L'objet de cette section n'est pas de décrire un tel paradigme totalement abouti, mais de proposer différentes pistes de recherche permettant de répondre à terme à ce défi du passage à l'échelle. Là encore, la gestion locale et distribuée du flux d'information reste au cœur du problème.

Asynchronisme

Dans les architectures récurrentes, ne pas pouvoir assurer le calcul simultané de l'ensemble des neurones implique de respecter des règles de précédence lors de la séquentialisation (même partielle) des calculs. L'idéal est alors de disposer d'un modèle capable de s'affranchir de toute règle de précédence, laissant alors une liberté totale dans l'ordonnancement inévitable des calculs des différents neurones.

Une telle propriété peut être apparentée aux modèles asynchrones, dans lesquels les mises à jour incessantes des différents neurones se font dans un ordre aléatoire toujours renouvelé, par opposition aux modèles synchrones dont les calculs discrétisés dans le temps se font partout simultanément en fonction des états neuronaux observés au temps précédent. Mais les modèles asynchrones ne sont en fait pas plus libres que les modèles synchrones du point de vue des contraintes d'implantation. En effet, dans la plupart des cas, les dynamiques de ces modèles sont biaisées par les éventuelles synchronisations entre neurones, qui peuvent notamment résulter de calculs simultanés parmi des sous-parties fixes de l'ensemble des neurones. Par conséquent, il est souhaitable d'aboutir à des modes de calculs connexionnistes tolérant n'importe quelle combinaison entre la part de synchronisme et la part d'asynchronisme de leurs calculs implantés. c'est par exemple le cas du modèle de navigation distribué de [GB07], mais ce modèle résulte d'une discrétisation spatiale d'une équation différentielle de second ordre, et non d'une approche connexionniste. Cette question de l'asynchronisme est cruciale pour l'ensemble des modèles d'inspiration biologique qui seront notamment évoqués dans le chapitre suivant. Si ces modèles bio-inspirés sont généralement asynchrones, les travaux rapportés dans ce manuscrit n'ont pour l'instant pas abouti à une réponse satisfaisante à cet enjeu d'une indépendance partielle au degré de synchronisation des calculs implantés. Néanmoins, les travaux menés dans l'équipe et rapportés notamment dans [RV06] montrent d'une part que les simulations peuvent se contenter d'asynchronisme à l'intérieur de chaque carte neuronale, et non pas dans l'ensemble du réseau neuronal, et d'autre part qu'une rupture de la symétrie des interactions latérales entre neurones par l'introduction d'un bruit aléatoire est suffisante pour supprimer le biais inhérent aux simulations synchrones des champs de neurones dynamiques. Ce résultat, sans résoudre fondamentalement le problème posé, permet au moins d'y répondre en partie lors des implantations.

Modularité

Une solution technique parfois évoquée pour l'implantation de modèles connexionnistes de très grande taille est l'utilisation de systèmes reconfigurables utilisant plusieurs circuits FPGA. Si cette solution semble simple, elle se heurte encore davantage au problème de la densité et de la topologie des interactions dans les modèles connexionnistes, puisque les connexions entre circuits reconfigurables sont à la fois facteurs de ralentissement et surtout disponibles en nombre très réduit par rapport aux ressources logiques. Concevoir un paradigme de calcul connexionniste suffisamment modulaire pour donner lieu à de telles implantations multi-supports est à la fois un défi du point de vue des mécanismes d'interaction entre modules et du point de vue de la réalisation de tâches complexes sur la base de ces modules.

L'approche bio-inspirée présentée à partir du chapitre suivant se fonde en partie sur les connaissances physiologiques qui attribuent différents rôles à chaque aire corticale tout au long des processus perceptifs et cognitifs. C'est donc une approche qui mène "naturellement" à des architectures modulaires bien que massivement distribuées : les modules identifiés sont en nombre réduit, et chaque module est "réalisé" par une architecture connexionniste massivement distribuée. Néanmoins, cette approche de conception inspirée par le cortex ne permet de résoudre que partiellement le problème de la densité des connexions entre modules. Certes ces architectures réduisent le plus souvent le nombre de ces connexions en raison du principe de "champ récepteur" qui sera explicité en 3.1.2 et 3.2.2, et qui induit une notion de "localité à distance" des connexions entre modules, mais deux difficultés demeurent, à savoir un nombre de connexions inter-modules encore important (proportionnel au nombre de neurones présents dans les modules), et un schéma d'interconnexion particulièrement complexe à gérer dans une implantation directe en raison de champs récepteurs tous distincts mais avec un fort taux de recouvrement.

Ce principe de modularisation des architectures connexionnistes bio-inspirées se retrouve de manière encore plus poussée dans un projet visant à définir des modules connexionnistes autonomes extensifs capables de s'auto-organiser lorsqu'ils sont assemblés et interagissent avec différents types de données, par exemple au sein d'une boucle perception-action. L'inspiration corticale est fondamentale dans ce projet, compte tenu de la difficulté des propriétés visées, et de l'existence reconnue de modules satisfaisant ces propriétés dans le cerveau. Des travaux ont été ainsi initiés sur la base de modules connexionnistes d'inspiration biologique dont les propriétés fondamentales sont : la gestion combinée de deux niveaux de proximité topologique, des interconnexions majoritairement locales et bidimensionnelles, l'auto-recrutement des ressources, et l'homogénéité des calculs neuronaux. Ces calculs sont principalement basés sur les modèles des champs de neurones dynamiques.

La finalité des modules connexionnistes extensifs au coeur de ces recherches est de constituer des ressources de calcul physiquement assemblables. La compatibilité des concepts développés avec une implantation matérielle ultérieure constitue donc une contrainte forte. Les travaux préalables ont pour but de formaliser les différents types d'interactions dont doivent disposer de tels modules, cf 3.3.2. Les contraintes d'implantation des connexions entre modules y sont d'ores-et-déjà prises en compte dans la définition même des modèles.

2.3 Bilan

Au travers des différentes études présentées dans ce chapitre, la notion de parallélisme connexionniste comme pratique de calcul distribué (de la conception des modèles à leur implantation) apparaît à la fois comme une réalité et un défi.

Une telle pratique de calcul n'a de sens que si elle se traduit par une facilité de mise en œuvre comparable à ce qu'on peut attendre de l'algorithmique classique vis-à-vis des supports de calcul actuels. Cela pose donc la question des environnements de programmation qui peuvent accompagner cette évolution.

Il faut néanmoins rester conscient de la complexité des modèles mis en jeu, et donc de la difficulté de maîtriser leur conception en fonction des tâches à réaliser. Cela pose alors la question de l'inspiration nécessaire pour la conception de modèles connexionnistes puissants.

2.3.1 Vers des environnements dédiés

Bien que plus faciles que le développement d'ASIC, les implantations sur FPGA nécessitent encore une importante quantité de travail et une connaissance des outils tels que le langage VHDL, les outils de synthèse, etc. Par analogie, on pourrait dire que la programmation sur circuit reconfigurable commence à peine à franchir le pas de la conception logicielle par programmation directe en assembleur.

La plupart des implantations sur FPGA sont réalisées avec une approche ASPC (application specific programmable circuit), où la solution FPGA remplace la conception coûteuse d'un circuit ASIC. La démarche ASPC requiert une expertise matérielle poussée, même si elle permet de s'affranchir de certains problèmes spécifiques de la conception d'ASIC. C'est pourquoi une nouvelle tendance est apparue dans la conception de systèmes à base de FPGA, essayant d'éviter le long travail d'implantation et son manque partiel de flexibilité (certaines conceptions nécessitent une refonte profonde quand des changements sont opérés dans l'application visée). Ces travaux ont pour but de créer des environnements de haut niveau pour manipuler les FPGA, de telle sorte que ces supports matériels soient plus susceptibles d'être utilisés aussi simplement que des microprocesseurs. Ils peuvent être divisés selon trois principales démarches : des compilateurs matériels basés sur des versions simplifiées de langages logiciels classiques comme C/C++, des systèmes multiprocesseurs sur FPGA qui traitent principalement des applications dont le parallélisme réside dans des boucles imbriquées de très grande taille, et des environnements de conception spécialisés. Cette dernière approche apparaît comme plus accessible pour l'utilisateur final, et elle est également en mesure de tirer avantage du parallélisme spécifique des modèles implantés, mais sa généralité est fortement réduite aux modèles spécifiés explicitement dans l'environnement.

L'idée de développer des plateformes spécialisées pour le calcul connexionniste peut être donc vue actuellement comme une forme d'environnement de conception spécialisée. C'est par exemple le cas de notre plateforme NNetWare (http ://nnetware.gforge.inria.fr, [THGG07]), dont la finalité est de proposer pour un très large choix de modèles connexionnistes un outil de gestion automatique des différents aspects technologiques des implantations sur cartes FPGA (communications avec la carte, communication des différents éléments de la carte, détermination des choix arithmétiques, gestion de la précision, techniques avancées de placement avec pipeline, interprétation automatique de résultats typés, affectation automatique de ports, etc) et des différents niveaux de parallélisation (partielle ou complète) en fonction des contraintes spécifiées.

Au-delà de ce statut de plateforme spécifique, il s'agit d'un élément de l'ensemble de la chaîne de conception et d'implantation que doit recouvrir une pratique connexionniste de calcul distribué. La faisabilité de tels environnements de conception est donc une question pratique importante sur laquelle porte une partie de mes activités, même si elle n'est pas au cœur des thèmes défendus tout au long de ce manuscrit.

2.3.2 Concevoir des architectures connexionnistes complexes

La suite de ce manuscrit a pour but de proposer des voies de recherche pour répondre à la question suivante, inhérente à l'idée d'une pratique connexionniste de calcul distribué : par quels moyens peut-on obtenir un comportement robuste et cohérent de la part d'un système distribué gouverné par des règles uniquement spécifiées à un niveau local. En d'autres termes, imaginons qu'on ait besoin de programmer un large ensemble d'unités de calcul élémentaires qui n'ont que des facultés de communication locale ; comment obtenir un comportement global voulu tout en respectant de telles contraintes ?

Bien sûr ce problème est beaucoup trop large pour être traité dans toute sa généralité, et de nombreuses approches sont possibles. Notre équipe propose d'explorer la voie du calcul bio-inspiré. Plus précisément, prenant acte des capacités cognitives du cerveau humain pour résoudre une très large variété de problèmes complexes dans des environnements encore plus complexes, il apparaît prometteur de s'inspirer de connaissances neurophysiologiques pour déterminer de manière macroscopique les architectures connexionnistes susceptibles de réaliser certaines tâches cognitives particulièrement complexes. Il ne s'agit pas de reproduire finement les phénomènes biologiques, mais plutôt d'en extraire les principes anatomiques (architecturaux) et physiologiques (fonctionnels) à la base de leurs propriétés les plus remarquables de traitement de l'information, dont celles qui nous intéressent le plus ici : calcul local et adaptatif, gestion et synchronisation des flux d'informations. En ce qui concerne spécifiquement le parallélisme connexionniste, cette inspiration corticale doit permettre de mieux comprendre et de mieux déterminer l'organisation distribuée et l'émergence globale de décision issue du flux d'informations présent dans les modèles connexionnistes modulaires de très grande taille.

Quel que soit l'aboutissement des recherches évoquées jusque-là, il est nécessaire de préciser que la pratique connexionniste de calcul distribué défendue dans ce manuscrit n'a pas pour vocation à remplacer la conception algorithmique classique, mais bien de constituer une alternative complémentaire, avec une panoplie de champs d'applications privilégiés qui lui sera propre. Le fait de se concentrer sur des tâches à caractère cognitif dans le chapitre suivant est un premier choix pour la détermination de ces champs d'application.

Chapitre 3

Les modèles bio-inspirés comme architectures connexionnistes

Les évolutions récentes des recherches en réseaux de neurones tendent vers deux courants principaux. D'une part la confirmation de l'appartenance du connexionnisme "classique" à la thématique plus large de l'apprentissage automatique (machine learning), avec des évolutions vers des modèles tels que les SVM (support vector machines) ou les LSM (liquid state machines) par exemple, dont les liens avec l'inspiration biologiques sont de plus en plus ténus. D'autre part le renforcement de la branche connexionniste des neurosciences algorithmiques, avec principalement deux niveaux de réflexion, les mécanismes élémentaires (calcul impulsionnel) et les modèles comportementaux. La frontière entre ces deux courants n'est pas toujours stricte, avec notamment une grande part de l'étude du calcul impulsionnel qui s'écarte de la modélisation biologique et considère les réseaux impulsionnels comme des systèmes dynamiques complexes.

Sans écarter les liens possibles entre une réflexion sur le parallélisme connexionniste et le calcul impulsionnel, qu'illustre par exemple le travail rapporté en section 2.1.3, l'objet de ce chapitre est de montrer que l'approche bio-inspirée de détermination des modèles comportementaux peut permettre de répondre à la question de l'organisation globale du flux d'information dans des systèmes connexionnistes de grande taille capables de traiter des problèmes cognitifs complexes. La section 3.1 s'intéresse à la nature des calculs distribués inspirés des colonnes corticales et des interactions observées dans le cortex. La section 3.2 montre ensuite comment la structure et les fonctionnalités des différentes aires corticales peut permettre de concevoir une architecture connexionniste complexe, en illustrant cette approche par une étude très simplifiée des fondements biologiques de la perception visuelle du mouvement. Enfin la section 3.3 traite du problème de l'implantation matérielle de tels modèles bio-inspirés très massivement distribués et modulaires. Le chapitre suivant permettra d'illustrer comment l'ensemble des principes énoncés dans ce chapitre peuvent être exploités afin d'aboutir à un modèle bio-inspiré de perception du mouvement.

3.1 Calculs d'inspiration corticale

En combinant des connaissances et des travaux issus de domaines aussi variés que l'anatomie, la neurobiologie, la physiologie, l'imagerie, les études comportementales, les mathématiques ou encore l'informatique, les neurosciences algorithmiques offrent une coopération interdisciplinaire riche, et elles visent à comprendre les mécanismes qui sous-tendent les processus neuronaux tels que la

perception, l'action, la mémoire ou la cognition.

Les activités de notre équipe dans le domaine des neurosciences algorithmiques sont centrées d'une part sur l'étude des mécanismes de bas niveau qui sous-tendent le calcul neuronal (spikes) et d'autre part sur la compréhension des fonctions de haut niveau du cerveau à partir de modèles informatiques. Ces modèles sont fondés sur des paradigmes algorithmiques qui sont directement inspirés par plusieurs études du cerveau qui convergent vers un traitement distribué, asynchrone, numérique et adaptatif de l'information. Ces modèles comportementaux sont élaborés à partir de données intégrées et de fonctionnalités multimodales, et ont pour but de comprendre l'émergence de fonctions plus complexes, décrites en termes de flux d'informations, au niveau des populations de neurones.

L'objet de cette section est de décrire comment la gestion locale et distribuée de ce flux d'informations se situe au cœur de cette approche bio-inspirée, en se concentrant sur la nature des calculs et des interactions qui sous-tendent l'ensemble de ces modèles comportementaux.

3.1.1 Emergence fonctionnelle

De nombreuses données expérimentales montrent que le cortex est organisé spatialement et fonctionnellement. De plus, les interactions inter et intra zones fonctionnelles jouent un rôle prépondérant dans l'émergence d'une réponse stable à la fois liée aux stimuli et à la connaissance stockée dans la dynamique interne des aires corticales. La modélisation informatique des champs de neurones dynamiques permet de s'inspirer des connaissances neurophysiologiques afin de proposer des modèles numériques capables de réaliser certaines tâches cognitives complexes. La théorie des champs neuronaux continus (dans sa version néanmoins discrète en ce qui concerne nos travaux) fournit le cadre théorique pour concevoir des modèles de populations de neurones.

Champs neuronaux dynamiques

Nos travaux sur les modèles comportementaux sont en partie basés sur une famille de modèles neuronaux bien définie, même si elle englobe de nombreuses possibilités. Il s'agit de champs de neurones dynamiques, organisés selon des architectures multicartes, avec des cartes topologiquement régulières et en interaction mutuelle. Les unités de calcul constituant ces cartes (qu'on peut désigner sous le terme de neurones) sont une approximation du comportement de "petits" ensembles de neurones biologiques ("quelques" colonnes corticales[1]) dont on modélise ici l'activité moyenne. Cette activité dépend d'influences excitatrices et inhibitrices en provenance d'autres groupes de neurones (i.e. d'autres unités de calcul de la carte).

La connectivité interne à chaque carte est caractérisée par des noyaux latéraux à supports contraints par des relations de voisinage (localité) éventuellement adaptables et hétérogènes. D'un point de vue dynamique, de tels modèles évoluent selon les excitations et inhibitions mutuelles reçues, ainsi que selon une dynamique interne de relaxation. Dans une approche informatique, il est important de simuler numériquement de manière fidèle de tels phénomènes dynamiques, au travers d'une discrétisation asynchrone de l'évolution des différents neurones. Les aspects centraux de l'étude de ces modèles sont liés aux mécanismes de résonance, de rétroaction, de localité, de plasticité et d'adaptabilité. C'est donc bien la nature et l'organisation du flux massif d'informations propre au parallélisme connexionniste qui est au cœur des modèles bio-inspirés concernés par cette étude.

[1] On parle néanmoins ici déjà de plusieurs dizaines de milliers de vrais neurones.

Dans la suite du manuscrit, cette approche va être illustrée au travers de modèles d'inspiration corticale dédiés à la perception visuelle (attention visuelle, perception visuelle du mouvement). Dans ces travaux, l'organisation topologique distribuée des champs de neurones est à la fois guidée par et restreinte à des principes d'inspiration biologique qu'il est nécessaire de traduire au sein des modèles. Il ne s'agit pas de modéliser de manière précise les aires visuelles du cerveau, mais bien de s'inspirer de leur fonctionnement pour proposer des architectures connexionnistes capables de traiter des problèmes complexes liés à la perception visuelle.

Illustration : attention visuelle

Les travaux de [RV06] portent sur la définition de cartes neuronales modélisant les mécanismes d'attention visuelle, en exploitant principalement sous une forme discrétisée les phénomènes de bulle émergente identifiés dans les champs neuronaux continus [Ama77, Tay98]. Un premier modèle simple réalise une forme très rudimentaire d'attention visuelle sous la forme d'une propriété émergente. Ce modèle est très résistant à la fois au bruit et aux distracteurs potentiels. Il est également capable de dépasser la hiérarchie de saillance des stimuli en se concentrant sur n'importe quel stimulus de la scène visuelle, indépendamment de sa saillance intrinsèque. Ce modèle a ensuite été complété de manière à pouvoir changer de cible "à volonté", grâce à une architecture bio-inspirée plus complexe qui permet de mémoriser les stimuli déjà "vus". Ce modèle se fonde sur les interactions issues des aires visuelles supérieures.

Pour illustrer le principe général des champs neuronaux dynamiques et de l'émergence de comportement dans ces champs, on peut se contenter de présenter la partie la plus simple de ce modèle. Le but de ce modèle simplifié est de faire émerger une seule bulle d'activité, c'est à dire un ensemble de neurones voisins dans la carte corticale, dont l'activité est significative et décroît avec l'éloignement au centre de la bulle. Ce modèle est composé de trois cartes organisées de manière rétinotopique : une carte d'entrée, une carte visuelle et une carte de focalisation (focus). La première carte est une carte de reliefs, qui représentent l'intensité de la caractéristique recherchée (une couleur, un mouvement, etc.) dans le champ visuel. La deuxième carte reçoit les entrées excitatrices de la première. Cette deuxième carte ne garde que les entrées les plus saillantes grâce à un mécanisme d'interactions latérales locales. Finalement, la troisième carte (focus) reçoit les entrées excitatrices de la deuxième. Cette carte a une connexion inhibitrice latérale complète (i.e. les excitations sont seulement locales, tandis que les inhibitions se font selon un graphe de connexion complet). Ces propriétés génèrent une compétition globale dans la carte focus, permettant ainsi l'émergence et le maintien d'une seule bulle d'activité dans cette carte (voir figure 3.1).

D'un point de vue calculatoire, ce modèle est défini de la façon suivante. Les trois cartes ont la même taille $n \times m$ (taille de la carte d'entrée). On associe un potentiel $u(x, y)$ à un neurone (x, y) d'une carte corticale $\mathcal{C}$ (carte visuelle ou carte focus) recevant ses entrées E de la carte précédente $\mathcal{C}'$. Ce potentiel est mis à jour selon les équations différentielles suivantes :

$$\begin{aligned}\frac{\partial u}{\partial t}(x,y,t) &= -\frac{1}{\tau}u(x,y,t) \\ &\quad +\frac{1}{\tau}\left(\frac{1}{\alpha}\sum_{(x',y')\in\mathcal{C}} w\left(x,y,x',y'\right) f\left(u(x',y',t)\right)\right) \\ &\quad -\frac{1}{\tau}\left(\frac{1}{\alpha}\sum_{(x',y')\in\mathcal{C}'} s(x,y,x',y')E(x',y',t)\right) \end{aligned} \tag{3.1}$$

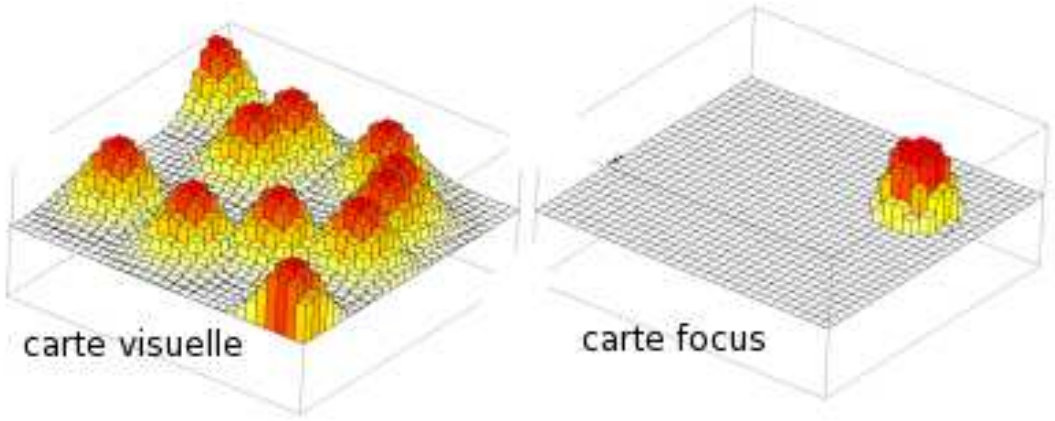


FIG. 3.1 – Modèle simplifié de focalisation de l'attention

où

$$w\left(x,y,x',y'\right) = Ae^{-\frac{||(x,y)-(x',y')||}{a^2}} - Be^{-\frac{||(x,y)-(x',y')||}{b^2}} \tag{3.2}$$

$$s(x,y,x',y') = Ce^{-\frac{||(x,y)-(x',y')||}{c^2}} \tag{3.3}$$

et $f\left(u(x,y,t)\right)$ représente le taux moyen de décharge du neurone de potentiel $u(x,y,t)$ (fonction d'activation). Ces équations sont une forme spatialement discrétisée des travaux sur les champs neuronaux continus bidimensionnels de [Tay98], et sont utilisées dans nos modèles sous une forme également discrétisée dans le domaine temporel. Elles fixent les rapports excitateurs et inhibiteurs des différentes unités des champs neuronaux selon des critères purement spatiaux.

- Les paramètres (A,a) et (B,b) sont liés à l'influence du voisinage du neurone. De façon à exciter les plus proches voisins et inhiber les neurones plus lointains, on choisit $A > B$ et $a < b$. De plus, la valeur de a est liée à la taille des bulles d'activité. D'autre part la valeur de b est liée au rayon d'inhibition de chaque neurone. Si l'objectif est de ne maintenir qu'une seule bulle d'activité dans la carte focus, cette valeur doit être choisie suffisamment grande pour couvrir la totalité de la carte neuronale. D'une certaine manière, b définit la distance minimum entre deux bulles d'activité distinctes compte tenu du mécanisme de compétition.
- Les paramètres (C,c) sont liés à l'influence des sorties de la carte précédente. Les valeurs de ces paramètres ont des effets similaires à ceux des paramètres A et a.
- α permet de contrôler l'influence respective du potentiel précédent du neurone et de ses entrées. Plus α est élevé, plus le neurone est soumis au mécanisme de relaxation et non aux excitations et inhibitions reçues.

Les travaux de [RV06] montrent bien la nécessité pour ce type d'approche d'une réflexion sur la coexistence de plusieurs types de flux d'information. En ce qui concerne les interactions locales, une partie du travail qui a suivi les premiers modèles d'attention visuelle a consisté à ne maintenir que des interactions locales au sein de chaque carte corticale [Rou06], alors même que ces cartes utilisaient initialement des relations inhibitrices globales. Ainsi, pour diffuser les inhibitions vers

des neurones lointains malgré des interactions locales, l'activité des neurones est bornée :

$$f\left(u(x,y,t)\right) = \begin{cases} 1 & \text{si } u(x,y,t) > 1 \\ -1 & \text{si } u(x,y,t) < -1 \\ u(x,y,t) & \text{sinon} \end{cases} \tag{3.4}$$

Cette approche permet de maintenir les principales propriétés des champs de neurones dynamiques tout en respectant davantage les besoins de localité inhérents au parallélisme connexionniste.

3.1.2 Topologie et sémantique des interactions

Une vision simplifiée du cortex consiste à le voir comme une couche bidimensionnelle de colonnes corticales, ces colonnes étant constituées elles-mêmes de neurones. Les interactions existantes sont majoritairement locales, aussi bien à l'intérieur des colonnes qu'entre colonnes voisines, mais de nombreuses interactions entre groupes de colonnes plus distantes existent également. Schématiquement, on distingue deux relations de proximité parmi les neurones. Une proximité sémantique, qui s'interprète par exemple en disant que des neurones d'une même colonne sont sensibles à des caractéristiques différentes mais proches pour un même percept ou stimulus. Et une proximité topologique, qui s'interprète par exemple en disant que des colonnes proches traitent des stimuli voisins dans l'espace "d'entrée". En résumé, on distingue principalement :

1. des interactions lointaines permettant à des cartes corticales disjointes de communiquer, avec une notion sous-jacente de champs récepteurs,
2. des interactions locales entre neurones ayant des champs récepteurs voisins,
3. et des interactions locales de compétition pour des neurones associés à un même champ récepteur.

Ce sont les deux premiers types d'interaction qui sont au cœur des modèles multicartes d'attention visuelle de [RV06], et notamment la notion d'organisation rétinotopique des aires visuelles corticales dans le cas de la perception visuelle.

Les travaux de [RV06] laissent de côté les interactions locales entre neurones associés à un même champ récepteur (interactions le plus souvent dédiées à une compétition locale entre différentes caractéristiques associées à ce champ récepteur). Lorsque de telles interactions sont à prendre en compte, il est possible de proposer de les insérer au sein de l'ensemble des autres interactions excitatrices et inhibitrices (ce qui est en quelque sorte ce qui se passe dans le cortex). Néanmoins, cela se fait au détriment de la distinction évoquée plus haut entre les trois types de flux d'informations des modèles comportementaux d'inspiration corticale. Ces interactions locales étant associées à une notion de proximité sémantique, une approche consiste à traduire cette notion à l'intérieur même du flux d'information distribué concerné.

Pour illustrer cette démarche, on peut citer les travaux de thèse de [Cas05], repris dans le chapitre suivant. Dans ces travaux, une architecture neuronale bio-inspirée a été développée pour détecter, extraire et segmenter les composantes de direction et de vitesse du flux optique dans une séquence d'images. La structure de ce modèle dérive directement du parcours des signaux optiques dans le cerveau humain, en commençant par la rétine et en recevant des traitements variés à chaque étape de son chemin cortical [CSGA04, CSG05].

La détermination de ce modèle s'est faite en maintenant à chaque étape des principes de calcul local et massivement distribué. Le modèle initial consiste en trois modules. Un premier module extrait les estimations spatio-temporelles des mouvements locaux. Le second module met en place

un mécanisme excitateur-inhibiteur dans lequel de très fortes inhibitions existent entre les neurones qui représentent des mouvements antagonistes, ce qui permet de renforcer la cohérence des zones de mouvement détectées. Le troisième module est issu des travaux réalisés sur la modélisation de l'attention visuelle et évoqués précédemment. On se reportera au chapitre suivant pour le détail du modèle.

Dans le second module de ce modèle, les interactions sont de trois sortes, à l'image de la distinction évoquée précédemment :

1. connexions issues du premier module (connexions lointaines organisées selon les champs récepteurs de manière rétinotopique)
2. connexions locales entre neurones associés à des champs récepteurs voisins
3. connexions locales entre neurones représentant l'intensité de chaque mouvement unitaire possible évaluées pour un même champ récepteur (la compétition qui en résulte permet d'estimer le mouvement local détecté dans ce champ récepteur)

Les travaux sur ce modèle indiquent que les trois types de flux d'information, mais aussi les flux d'information associés au module de focalisation de l'attention, doivent être gérés simultanément mais séparément, avec notamment des dynamiques temporelles propres.

3.1.3 Dynamiques et temporalité

Le modèle initialement conçu dans nos travaux sur la perception visuelle du mouvement fait une distinction entre la perception de bas niveau et son interprétation de haut niveau. Ainsi l'extraction des vitesses locales, bien que conforme aux observations biologiques de la sensibilité des neurones aux différents stimuli, ne se fait pas selon des mécanismes propres aux champs neuronaux dynamiques utilisés plus loin dans le modèle. Ce premier module, comme cela sera décrit dans le chapitre suivant, est constitué d'unités indépendantes qui effectuent chacune un filtrage spatio-temporel sur un champ visuel récepteur local.

Or l'idée d'une pratique connexionniste de calcul distribué suppose une grande homogénéité des calculs neuronaux (à l'instar de l'idée d'universalité des machines de Turing qui fonde l'algorithmique classique). Un effort particulier est donc fait dans l'ensemble de mes travaux sur les modèles massivement distribués d'inspiration corticale pour unifier les calculs et les interactions au sein de chaque modèle développé. Cette démarche se retrouve dans l'apport de la modélisation des effets centre-périphérie au sein du module chargé d'extraire les vitesses locales (voir section 4.3.1). Un des objectifs de cet apport est de ramener les calculs nécessaires pour cette extraction à une forme davantage compatible avec les mécanismes excitateurs-inhibiteurs des champs de neurones des modules suivants. Ainsi, le modèle présenté dans le chapitre suivant évolue vers une architecture très complexe dans laquelle la totalité des neurones suit des évolutions similaires à celle de l'équation 3.1.

Le problème de la temporalité des dynamiques neuronales apparaît alors. En effet, homogénéiser les mécanismes neuronaux n'implique pas nécessairement d'unifier la gestion temporelle des dynamiques. Si l'ensemble des neurones interagissent, des latences ou différentiations temporelles entre les dynamiques des neurones suivant leur rôle et leur proximité peuvent permettre de stabiliser et de guider la dynamique globale. Là encore, la réflexion doit se baser avant tout sur la présence de différents flux d'informations simultanés mais disjoints. Ainsi, l'approche développée dans le chapitre suivant consiste à "stabiliser" en partie les dynamiques neuronales en amont (neurones des premiers modules), ce qui correspond à une gestion temporelle plus "lente" des équations

différentielles dans les modules de plus haut niveau. Néanmoins, l'introduction massive d'interactions rétropropagées entre ces différents modules (justifiée en section 3.2.4) pose un problème dans une telle approche, aucun module n'étant plus strictement en amont des autres. L'utilisation de mécanismes de latence est une voie susceptible d'apporter des solutions dans ce cadre.

3.2 Architectures d'inspiration corticale

L'inspiration corticale n'est pas seulement présente dans la détermination des calculs et des interactions des différentes cartes neuronales de nos modèles. Elle intervient aussi dans l'organisation de ces différentes cartes au sein de l'architecture globale des modèles. L'objet de cette section est de préciser comment et dans quelle limite cette inspiration est prise en compte dans nos travaux, notamment au travers de l'inspiration biologique du modèle de perception visuelle du mouvement qui sera détaillé dans le chapitre suivant.

Ces travaux ne prétendent pas analyser finement l'anatomie et la physiologie des différentes aires corticales. C'est donc une approche très superficielle qui est utilisée, néanmoins suffisante pour poursuivre le but fixé en termes de calcul distribué bio-inspiré : exhiber plusieurs grands principes de gestion des flux d'informations et d'organisation des cartes neuronales utilisés dans le cortex, afin de maintenir leurs propriétés fondamentales au sein d'implantations distribuées ultérieures.

3.2.1 Inspiration corticale vs modélisation corticale

C'est avant tout l'étude des rôles respectifs et des influences mutuelles des aires corticales qui permet dans nos travaux de proposer des architectures multicartes. Le nombre de cartes, leurs tailles et leurs interactions sont choisis selon une vision simplifiée des aires corticales censées correspondre.

Il est néanmoins important de souligner ici qu'il ne s'agit en aucun cas d'un travail de modélisation du cortex et de ses différentes aires, avec une exigence de plausibilité biologique totale à la clé. L'approche considérée ici consiste seulement à dire que devant la complexité des tâches en jeu, il est difficile sinon impossible de proposer directement des architectures neuronales multicartes résolvant les problèmes posés, et qu'un ensemble de connaissances et d'hypothèses neurophysiologiques peuvent être retenues pour déterminer ces architectures. Le caractère irréfutable et surtout l'exhaustivité des hypothèses retenues et modélisées n'est donc pas un critère essentiel dans cette approche. On s'attache avant tout à exhiber des principes bio-inspirés modélisables.

Pour illustrer cette approche, la suite de cette section résume très succinctement l'ensemble des fondements biologiques qui ont été retenus pour élaborer le modèle de perception visuelle du mouvement qui sera détaillé dans le chapitre suivant. Les faits ou hypothèses biologiques présentés ci-dessous sont parfois très simplifiés, schématiques, ou même sujets à controverse dans les publications en neurophysiologie. Même si cela n'est pas rappelé à chaque fois, il faut donc considérer l'ensemble des affirmations présentées comme une description approximative des connaissances biologiques dans ce domaine.

3.2.2 Chemin cortical et signaux lumineux

Avant de présenter les différents traitements appliqués aux signaux visuels depuis la rétine jusqu'aux aires corticales supérieures, on peut très schématiquement situer les différentes parties du cerveau humain, et notamment celles qui sont impliquées dans la perception visuelle du mouvement.

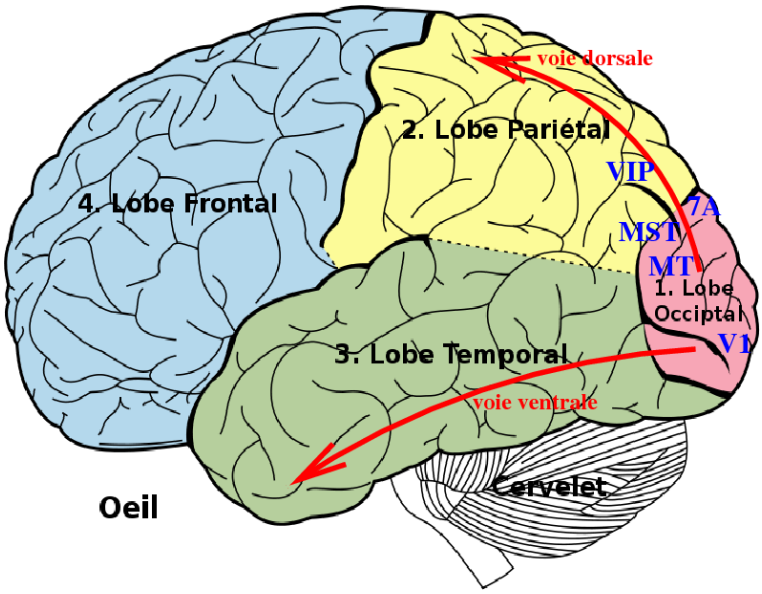


FIG. 3.2 – Schéma simplifié des lobes et des aires corticales

Le cerveau

La détection du mouvement chez l'être humain met en jeu plusieurs aires corticales anatomiquement distinctes. Les deux hémisphères du cerveau de l'être humain peuvent être divisées en lobes, eux-mêmes divisés en différentes aires fonctionnelles. qui traitent un ensemble de processus particuliers (voir figure 3.2) :

- Le lobe occipital assure le traitement des processus visuels.
- Le lobe pariétal traite principalement les processus sensitifs et de guidage.
- Le lobe temporal s'occupe principalement des processus auditifs et de la compréhension du langage.
- Le lobe frontal traite principalement les processus de l'action, de la décision et de l'abstraction.

La perception du mouvement est essentiellement prise en charge au niveau des lobes occipital et pariétal.

Chemin parcouru par les signaux visuels

Le chemin parcouru par les signaux visuels chez l'être humain peut être décomposé en quatre étapes :

1. Acquisition et compression des signaux lumineux au niveau de la rétine.
2. Intégration binoculaire au niveau du chiasme optique et des Corps Genouillés Latéraux (CGL) dans le thalamus.
3. Première analyse corticale au niveau de l'aire visuelle primaire (V1) dans le cortex occipital.
4. Traitements corticaux secondaires notamment dans le cortex temporal et le cortex pariétal.

Après des traitements locaux au niveau rétinien (voir ci-dessous), toutes les informations concernant l'image sont envoyées au cerveau par deux voies parallèles : la voie parvocellulaire et la voie magnocellulaire. Ces deux voies traversent les Corps Genouillés Latéraux (CGL) et se terminent

dans l'aire visuelle primaire (V1). Elles transportent différentes informations ; la voie parvocellulaire transporte l'information destinée au traitement de la forme et de la couleur alors que la voie magnocellulaire transporte l'information destinée au traitement du mouvement et de l'espace.

Les neurones de V1 extraient différentes représentations orientées et fréquentielles sur leurs champs récepteurs. Un champ récepteur correspond à la zone rétinienne (de forme et de surface variable) sur laquelle un neurone répond à la lumière par un changement de potentiel. Après le traitement en V1, l'information corticale est ensuite distribuée vers plusieurs aires par deux grandes voies (cf figure 3.2). La voie dorsale est impliquée dans l'analyse du mouvement et de l'espace, et la voie ventrale est impliquée dans la reconnaissance d'objets. Ces deux voies font des traitements fonctionnellement spécialisés et sont organisées hiérarchiquement. La voie dorsale est principalement la continuation au niveau cortical de la voie magnocellulaire et la ventrale de la voie parvocellulaire. Bien que les voies dorsale et ventrale ne soient pas totalement isolées l'une de l'autre, les différences fonctionnelles entre ces deux voies sont suffisamment marquées pour que le principe d'une perception du mouvement en grande partie indépendante de la reconnaissance d'objets soit retenu.

Les neurones de la voie magnocellulaire/dorsale ont une haute sélectivité à la direction du mouvement. Leurs réponses dépendent de la vitesse, du contraste et de la texture. Les signaux neuronaux transportant l'information de mouvement passent de V1 vers V2-V3, puis vers l'aire temporale moyenne (MT/V5) et l'aire temporale médiane supérieure (MST) avant de poursuivre leur chemin dans le cortex pariétal (cf figure 3.2). L'ensemble de ce chemin est davantage détaillé ci-dessous.

3.2.3 Aires corticales et perception visuelle du mouvement

Le but du modèle du chapitre 4 n'est pas de calquer la structure des différents aires corticales évoquées, mais de s'inspirer de leurs rôles fonctionnels et de leurs articulations, succinctement décrits ici.

La rétine et les Corps Genouillés Latéraux (CGL)

La rétine traite l'intensité des signaux lumineux dans l'espace et dans le temps. Ce traitement est organisé en trois niveaux : acquisition par les cellules photo-réceptrices, intégration par les cellules horizontales et bipolaires, puis compression et codage neuronal par les cellules amacrines et ganglionnaires. L'intégration peut être assimilée à l'obtention locales de signaux moyens (spatialement et temporellement). La compression est plus forte dans les zones les plus excentrées du champ visuel, principe qui sera exploité dans l'adaptation de notre modèle à une vision rétino-centrée en 4.2.

L'information visuelle est ensuite transportée par le nerf optique depuis les yeux vers les CGL qui réalisent une intégration binoculaire (avec croisement de l'information droite-gauche, mais la stéréo-vision n'est pas abordée dans nos travaux). Enfin l'information visuelle rejoint l'aire visuelle primaire.

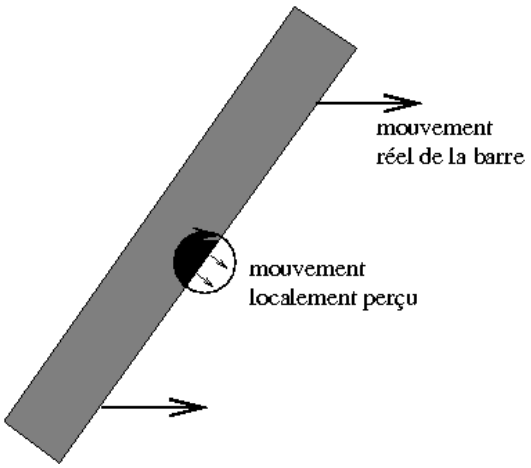


FIG. 3.3 – Illustration du problème d'ouverture

Le cortex visuel primaire (V1)

Le cortex visuel primaire (V1) possède beaucoup plus de neurones que les CGL (au moins deux ordres de grandeur). Toutes les cellules de V1 sont sélectives à une direction préférentielle[2], à une fréquence spatiale, et sont limitées à une extraction locale du mouvement. Des neurones voisins dans V1 ont leurs champs récepteurs (rétiniens) dans des régions voisines de la rétine (organisation rétinotopique). Ces différentes caractéristiques se retrouvent dans le modèle du chapitre 4.

Les cellules de V1 ne "voient" qu'une partie de la scène et, en conséquence, leurs réponses sont ambiguës. Cette ambiguïté est connue sous le nom de problème d'ouverture : compte tenu de leur perception locale, les neurones de V1 ne sont sensibles qu'aux mouvements perpendiculaires à l'orientation préférentielle des contrastes auxquels ils sont sensibles (figure 3.3). Un rôle important des aires corticales suivantes impliquées dans la perception du mouvement est de résoudre cette ambiguïté.

L'aire Médio-Temporale (MT)

Comme les cellules de V1, les neurones de l'aire MT répondent à la direction et à la vitesse d'un stimulus visuel et sont organisés de manière rétinotopique [SH98, BN04]. Les différentes représentations du mouvement produites par V1 sont collectées et combinées dans MT, notamment pour résoudre le problème d'ouverture. Les neurones de l'aire médio-temporale se distinguent des neurones de l'aire visuelle primaire par des champs récepteurs plus grands [DU86] et par leur sensibilité aux mouvements complexes et aux mouvements cohérents.

Il faut noter que le terme "neurones" employé ici dans la description des mécanismes corticaux recouvre en fait plutôt la notion de colonne corticale. Ces colonnes, elles-mêmes composées de différents neurones, peuvent être vues comme les unités élémentaires de calcul dans le cortex, lui-même étant assimilable à une structure plane composée de colonnes corticales. Ainsi, dans le cas de MT, les neurones ayant une même direction préférée de mouvement et une même vitesse préférée sont organisés en colonnes. Cette distinction entre colonne et neurone sort de l'objet de cette section,

[2]Un neurone est accordé à une direction préférentielle lorsque sa réponse est fortement liée à une orientation particulière du stimulus et diminue pour des stimuli qui s'écartent de cette direction.

dans la mesure où le modèle présenté dans le chapitre 4 considère les neurones des cartes comme les unités élémentaires de calcul.

L'aire Médio-Temporale Supérieure (MST)

Au-delà de l'aire MT se trouvent d'autres régions impliquées dans l'analyse du mouvement comme l'aire MST par exemple. Ses neurones ont des champs récepteurs de plus grande taille que ceux de MT, et la grande majorité de ces neurones répond à la direction et à la vitesse d'un stimulus visuel. Les neurones de MST sont non seulement sensibles aux déplacements plans comme dans l'aire MT, mais également aux déplacements radiaux (expansion ou contraction) ou encore aux mouvements circulaires (rotations). Cette sensibilité à différentes composantes de déplacement est notamment utile pour l'implication de la partie dorsale de MST (MSTd) dans la perception du mouvement propre (voir 4.4), les neurones de MSTd répondant à des stimuli visuels de grande taille. Cette implication de MST ne met d'ailleurs pas seulement en jeu des stimuli visuels, comme expliqué en 3.2.4.

D'autres aires corticales interviennent dans la perception visuelle du mouvement. On peut citer notamment l'aire VIP et l'aire 7A qui se situent dans le cortex pariétal, et pour lesquelles des sensibilités à des caractéristiques multimodales des stimuli ont été mises en évidence (informations visuelles, motrices, oculaires, tactiles, etc.). Sur le plan purement visuel, les grandes propriétés des aires MT et MST sont conservées dans ces aires (sensibilité à la direction et à la vitesse d'un stimulus visuel, sensibilité aux composantes radiales et circulaires). Le modèle du chapitre 4 ne prend pas encore en compte ces caractéristiques de haut niveau.

3.2.4 Bio-inspiration et flux d'informations

L'inspiration corticale ne constitue pas seulement une façon de proposer des architectures fonctionnelles de réseaux neuronaux. Elle justifie également l'importance centrale donnée aux flux d'informations selon l'approche proposée dans ce manuscrit. La complexité de ce flux au sein du chemin cortical parcouru par les signaux visuels ainsi que ses interactions avec des informations non visuelles doivent être pris en compte.

Rétroaction des aires supérieures vers les aires inférieures

Une vision classique de l'algorithmique est la décomposition procédurale des tâches et la séquentialité des traitements. Les observations neurophysiologiques montrent au contraire l'interdépendance des étapes de traitement perceptif dans le cerveau, et notamment l'importance fondamentale de la non-séquentialité au travers des interactions mutuelles des différents cartes corticales, ainsi que l'importance de la multiplicité des fonctions réalisées simultanément, et dont chacune influe sur l'ensemble.

D'un point de vue architectural, cette interdépendance se traduit par l'insertion des différentes aires neuronales au sein d'un flux aussi bien "montant" que rétropropagé. Dans le cas de la perception visuelle du mouvement, les recherches en neurobiologie montrent ainsi qu'il existe des connexions en rétroaction sur le traitement du mouvement dues à l'attention. De même, malgré la grande influence qu'a l'innervation en provenance de la rétine sur l'organisation des CGL, environ 80% des connexions excitatrices qui entrent dans les CGL ne proviennent pas de la rétine mais du cortex visuel primaire. D'une manière générale, l'influence des aires supérieures en rétroaction

sur les aires inférieures apparaît comme un aspect essentiel dans la détermination d'architectures bio-inspirées. Ce souci se retrouve notamment dans notre modèle (cf 4.3.1 et 4.3.2).

Bases corticales de la perception du mouvement propre

Le modèle d'inspiration corticale que nous proposons pour la perception visuelle du mouvement est essentiellement basé sur les traitements corticaux qui s'appliquent aux signaux visuels. Néanmoins, il est susceptible de prendre en compte des informations non visuelles, ce qui introduit un niveau supplémentaire de complexité. Là encore, il ne s'agit pas de modéliser les aires corticales qui traitent les informations visuelles et non visuelles, mais de proposer une architecture capable de résoudre un problème aussi complexe grâce à l'inspiration corticale. Si nos travaux en ce sens ne sont pas encore très avancés, on peut citer cette approche dans le cadre du problème de l'extraction du mouvement propre (cf section 4.4) qui doit permettre à notre modèle de distinguer les mouvements perçus dus aux déplacements de la caméra de ceux liés à des cibles en mouvement. Dans l'étude préalable de la perception du mouvement propre, il apparaît que des informations non visuelles jouent un rôle central. Si la prise en compte de ces informations n'intervient pour l'instant pas dans le modèle décrit au chapitre suivant, les éléments ci-dessous ont permis de mieux appréhender les conditions possibles d'une extraction purement visuelle du mouvement propre.

- Le système vestibulaire, situé dans l'oreille interne, est le principal capteur du mouvement propre. L'ensemble des capteurs vestibulaires dote le cerveau d'informations sur le mouvement de la tête dans l'espace. Ces capteurs sont basés sur des effets inertiels (rotation ou accélération linéaire), ils sont donc incapables de distinguer l'immobilité d'un mouvement à vitesse constante. En revanche, ils permettent de compléter l'information visuelle afin de distinguer un mouvement général de l'environnement d'un mouvement du sujet (même flux optique). Toujours actifs, ils ne délivrent néanmoins un message non nul qu'en cas de déplacement (non constant) dans l'espace.
- Après divers relais, l'information vestibulaire arrive au cortex. Elle est alors traitée par plusieurs aires visuo-vestibulaires, parmi lesquelles apparaissent notamment deux aires de la voie dorsale impliquées dans la perception visuelle du mouvement, l'aire MST et l'aire VIP. Pour la plupart des neurones de ces aires, la direction visuelle préférée est à l'opposé de la direction préférée de stimulation vestibulaire (réponse dite complémentaire, puisque, lors d'une rotation de la tête, le monde visuel subit un déplacement apparent dans la direction opposée à celle du mouvement de la tête).
- Si les informations visuelles et vestibulaires sont présentes dans ces aires, il n'est pas certain qu'elles soient "combinées" pour la perception du mouvement propre au niveau de ces aires.

Hormis les informations sensorielles vestibulaires et visuelles, il existe encore deux types de signaux principaux qui peuvent contribuer à la perception du mouvement propre, l'information somatosensorielle (informations sensorielles tendineuses, articulaires, tactiles, etc.) et les informations motrices efférentes (envoyées par le cerveau aux muscles), dont le système visuel reçoit une copie par le biais des aires associatives pariétales.

3.3 Implantations de modèles d'inspiration corticale

La pratique de calcul distribué numérique bio-inspiré proposée tout au long de ce manuscrit pose évidemment la question de l'adéquation des modèles d'inspiration corticale avec la notion de calcul parallèle connexionniste implanté. Cette section propose deux réponses très partielles.

Dans un premier temps, des travaux préliminaires d'implantation parallèle effective de modèles d'inspiration corticale sont décrits. Ils montrent à nouveau l'importance centrale de la gestion des flux d'information. Néanmoins les résultats atteints montrent à nouveau les limites d'une approche en aval, où l'effort d'implantation suit la conception d'un modèle certes distribué et modulaire, mais qui ne prend pas en compte l'ensemble des contraintes de l'implantation ultérieure. Dans un second temps, cette section décrit un projet encore naissant et qui vise à surmonter ces limites par une approche en amont. Ce projet a pour but de définir des modules connexionnistes auto-organisateurs d'inspiration corticale et dont la définition inclut la capacité à être implantés de manière totalement distribuée. Ces travaux montrent que l'inspiration corticale peut constituer aussi une piste pour organiser les architectures matérielles.

3.3.1 Implantation en aval

Inévitablement, les modèles d'inspiration corticale actuellement développés dépassent de très loin les capacités d'implantation parallèle directe des circuits reconfigurables actuels. La question du passage à l'échelle se pose donc. Sans apporter de réponse générique, les travaux rapportés dans [THGCS05a, THGCS05b, THG05] montrent qu'il est possible de séquentialiser partiellement les modèles d'inspiration corticale tout en respectant l'organisation corticale dont ils s'inspirent. Ainsi, le calcul des cartes neuronales effectuant les traitements visuels de bas niveau se retrouve décomposé par l'utilisation récurrente d'un même module qui implante une "sous-carte" neuronale. Ce module "glisse" de façon à parcourir peu à peu les cartes concernées.

Ces travaux portent sur une implantation matérielle modulaire d'une version préliminaire du modèle connexionniste bio-inspiré de perception visuelle du mouvement décrit en 4. L'architecture proposée utilise un parallélisme partiel au niveau des connexions et des neurones. Dans une première étape, reproduisant la sélectivité à l'orientation des neurones impliqués dans les premières étapes du traitement des stimuli visuels au sein du cortex visuel, un ensemble de composants matériels a été conçu de façon à traiter de manière concurrente la même image d'entrée mais selon des orientations privilégiées différentes. L'organisation spatiale du module matériel proposé ressemble à l'organisation en hypercolonnes des neurones, au travers d'un ensemble de grilles systoliques identiques d'éléments de calcul qui évoquent des grilles de neurones réglés pour la même orientation mais avec différents champs récepteurs. Dans une deuxième étape, un module connexionniste a été conçu sur la base d'un réseau excitateur-inhibiteur où un neurone interagit avec les autres à l'intérieur d'un voisinage ou rayon d'influence.

L'annexe E décrit plus en détail ces travaux d'implantation. Ce travail ne permet pas de conclure à la faisabilité des implantations directes d'architectures d'inspiration corticale. En revanche il propose des pistes pour la gestion des principaux flux d'informations propres à ces architectures au sein de chaque carte corticale, lorsque la taille de chacune des cartes dépasse les capacités du support.

3.3.2 Conception en amont

Parmi les pistes de recherche proposées en 2.2.2, la définition de modules connexionnistes autonomes extensifs capables de s'auto-organiser une fois assemblés constitue évidemment une première tentative globale de pratique de calcul distribué bio-inspiré modulaire.

Notion de modules connexionnistes autonomes extensifs

Les recherches que nous avons initiées visent ici à définir de tels modules comme des "briques" assemblables dont les mécanismes d'auto-organisation permettent l'apprentissage de leur sémantique au sein d'une architecture globale interagissant avec différents types de données au sein d'une boucle perception-action.

Ces modules sont basés sur des modèles de cartes corticales, dont les propriétés fondamentales sont : la gestion combinée de deux niveaux de proximité topologique (indices primaires et secondaires), des interconnexions majoritairement locales bidimensionnelles, l'auto-recrutement des ressources, et l'homogénéité des calculs neuronaux, qui sont principalement basés sur les modèles des champs de neurones dynamiques déjà évoqués en 3.1. Les modules sont assemblés selon des architectures multicartes, où les cartes sont composées d'un ou plusieurs modules, chacun d'eux étant une assemblée 2D de ressources neuronales interconnectées (unités élémentaires de calcul neuronal). Les modèles définis ont pour but de résoudre des tâches multimodales de perception-action, où l'apprentissage à base de récompense guide l'auto-organisation des modules connexionnistes. Les travaux de [OFB05] constitue une première approche possible pour cette auto-organisation et cet apprentissage. Certains de ces modules ont un rôle d'entrées-sorties vis-à-vis des informations perceptives et motrices traitées et générées par l'ensemble.

Gestion du flux d'informations

L'ensemble de ces recherches est encore peu avancé. Néanmoins, il est déjà possible d'illustrer l'importance de la conception conjointe des modules décrits ci-dessus et de leur capacité à être implantés de manière totalement distribuée. La pratique de calcul distribué bio-inspiré défendue dans ce manuscrit motive bien entendu cette conception conjointe.

Une caractéristique importante de ces modules connexionnistes est l'extensivité, que nous définissons selon deux niveaux : les extensions de cartes augmentent leur puissance de calcul et doivent satisfaire les propriétés de continuité des champs récepteurs, tandis que les extensions par adjonction de cartes ajoutent de nouvelles fonctionnalités au modèle complet et nécessitent la définition de relations inter-cartes spécifiques avant que l'apprentissage n'intervienne pour organiser les ressources neuronales. Cette extensivité peut être formalisée de la manière suivante :

Extensivité intra-carte : On considère des cartes rectangulaires de taille $n \times m$, constituées d'un ou plusieurs modules. On impose donc de toujours ajouter un nombre suffisant de modules à une carte pour assurer une forme rectangulaire. Si une carte de taille $n \times m$ est étendue à une carte $(n + n') \times m$ par adjonction de modules, les relations ci-dessous suffisent à redéfinir les interactions entre cartes

- On définit le champ récepteur des unités de calcul neuronal par une fonction $\rho : [0,1] \times [0,1] \longrightarrow \{0,1\}$
- La notion de régularité topologique, qu'on peut associer à un "glissement" du champ récepteur, peut être définie d'une carte de taille (n_1, m_1) vers une carte de taille (n_2, m_2) par des fonctions continues $\alpha : [0,1] \times [0,1] \longrightarrow [0,1]$ et $\beta : [0,1] \times [0,1] \longrightarrow [0,1]$, telles qu'une unité de coordonnées (x_2, y_2) de la carte de destination reçoit une connexion depuis une unité de coordonnées (x_1, y_1) de la carte d'origine si et seulement si

$$\rho(\frac{x_1}{n_1} - \alpha(\frac{x_2}{n_2}, \frac{y_2}{n_2}), \frac{y_1}{n_1} - \beta(\frac{x_2}{n_2}, \frac{y_2}{n_2})) = 1$$

Extensivité inter-carte : Lorsqu'une nouvelle carte (constituée d'un ou plusieurs modules) est rajoutée à l'architecture multicarte générale, des interactions sont imposées entre la nouvelle carte et les anciennes en déterminant pour chacune d'elles le triplet (ρ, α, β).

La gestion du flux d'information est un aspect central dans l'ensemble de ces travaux, principalement en lien avec les contraintes d'extensivité et les interactions denses entre cartes soumises au principe des champs récepteurs. De plus, dès leur conception, les modèles développés prennent en compte la volonté d'aboutir à des modules connexionnistes réellement implantés de manière parallèle sur des circuits.

La principale difficulté concernant le flux des informations échangées par ces modules est liée au glissement des champs récepteurs qui implique :

- des interactions nombreuses entre les unités de cartes différentes non situées sur les "bords" de ces cartes
- l'impossibilité d'utiliser un mécanisme apparenté à une diffusion partielle qui ne peut pas aisément différencier les groupes d'unités qui reçoivent les connexions issues d'unités neuronales voisines (recouvrement des champs récepteurs tous distincts)

Une des voies explorées est l'utilisation d'un mécanisme similaire aux connexions localement configurables des FPNA (cf annexe A) qui génère un graphe de connexions potentiellement complexe sur la base de mécanismes purement locaux. Afin de résoudre le second problème évoqué ci-dessus, il est possible d'adjoindre un mécanisme d'indexation des graphes de connexions tel que celui utilisé par le routage indexé des circuits développés dans [NGT98]. Très schématiquement, cela revient à munir les ressources de communication de différentes couches (un peu comme les différents couches métalliques d'un circuit intégré qui permettent aux signaux de se croiser). En combinant les propriétés des graphes de connexions virtuelles des FPNA, les heuristiques de minimisation du nombre d'indices pour le routage indexé, et des mécanismes de pipeline adéquats, il est possible de réaliser la transmission des informations entre deux cartes en un temps réduit et avec une surface d'implantation dédiée au routage relativement limitée.

Les éléments rapportés dans cette section ne sont encore que les prémisses des modules à définir. Ce projet constitue une voie de recherche prioritaire dans les perspectives de mon travail, nécessitant une étude poussée des mécanismes bio-inspirés d'auto-organisation et d'apprentissage de ces cartes connexionnistes, tout en constituant un défi du point de vue du calcul distribué. Si l'ensemble de ces difficultés combinées rend ce projet particulièrement complexe, c'est aussi un gage pour aboutir à des modèles constituant une véritable pratique de calcul distribué bio-inspiré, puisqu'il s'agit, comme dans le cas des FPNA, de prendre en compte les contraintes d'implantabilité distribuée dès la conception en amont des modèles. De plus, en réduisant le champ d'investigation au travers de contraintes additionnelles, cette approche permet de cibler davantage notre étude.

3.4 Bilan

Au cours de ce chapitre, différents travaux et études ont été évoquées afin d'identifier les fondements bio-inspirés pouvant nous permettre de définir des ressources neuronales à la fois distribuées et modulaires, en s'aidant pour cela de connaissances sur l'anatomie et la physiologie des aires corticales. La question sous-jacente est de déterminer les calculs élémentaires et les architectures macroscopiques inspirées du cortex pouvant constituer à terme une pratique de calcul numérique distribué.

Les principaux fondements identifiés sont centrés autour de la notion de champ neuronal, d'architectures multicartes, de champs récepteurs, et surtout de gestion conjointe de trois principaux flux d'informations gérés au travers d'interactions latérales ou inter-cartes.

Dans le cas plus spécifique de la perception visuelle du mouvement, une analyse superficielle des aires corticales impliquées a permis d'identifier les principaux modules (fonctionnalités), leurs ressemblances (organisation rétinotopique, sélectivité, etc.) et dissemblances (complexité des perceptions, taille des champs récepteurs, etc.), ainsi que leurs interactions avec d'autres aires corticales dans le cas particulier de l'extraction du mouvement propre). Le chapitre suivant va expliciter l'utilisation de ces concepts pour la mise au point de notre architecture neuronale multicarte pour la perception visuelle du mouvement.

Ces différents fondements doivent être maintenus lorsqu'on étudie la capacité de nos modèles à constituer une pratique de calcul numérique distribué. Suite à des travaux préliminaires d'implantation massivement distribuée de nos modèles, plusieurs pistes de recherche sont actuellement envisagées pour aboutir à des ressources neuronales distribuées et modulaires.

Chapitre 4

Modèle bio-inspiré de perception du mouvement

Ce chapitre résume l'ensemble des travaux de définition d'un modèle connexionniste bio-inspiré de perception visuelle du mouvement, dont les fondements biologiques ont été succinctement présentés au chapitre précédent. On se rapportera à [CSGA04, CSG05, THGCS05b] pour de plus amples détails sur ce modèle.

4.1 Modèle modulaire de perception visuelle du mouvement

Le modèle proposé dans [Cas05] et modifié ultérieurement repose sur une interprétation très modulaire des rôles respectifs des aires corticales V1, MT et MST, et s'inspire donc du traitement de l'information visuelle le long de la voie magnocellulaire dorsale au travers de trois modules. Le premier module a pour rôle d'extraire localement le flux optique (notamment en effectuant un filtrage spatio-temporel à l'aide de filtres de Gabor). Le second module a pour rôle d'extraire de manière plus cohérente le flux optique, en s'attaquant notamment au problème d'ouverture. Ce module met en oeuvre un mécanisme distribué de fortes interactions localisées en grande partie fondé sur un principe antagoniste. Finalement, le troisième module a pour rôle d'assurer le suivi de cible(s) dans la scène visuelle. Il est basé sur le modèle de focalisation de l'attention déjà évoqué en 3.1.1 et permet de faire émerger un objet en mouvement à travers l'évolution d'une population neuronale.

4.1.1 Extraction locale du flux optique

Basé sur la sélectivité à l'orientation et à la vitesse des réponses des neurones de V1, le premier module extrait au niveau de chaque champ récepteur une estimation de la vitesse et de la direction de mouvement perçu dans ce champ. Une hypothèse forte est ici une haute fréquence de capture d'images qui permet d'avoir une vitesse locale instantanée constante et ainsi d'assurer une détection locale du mouvement.

Le premier module est ainsi constitué de $\Theta * V$ différentes cartes neuronales rétinotopiquement organisées pour Θ directions et V vitesses distinctes localement estimées (voir figure 4.1). Les neurones de ces cartes reçoivent une estimation locale du mouvement dans la direction et à la

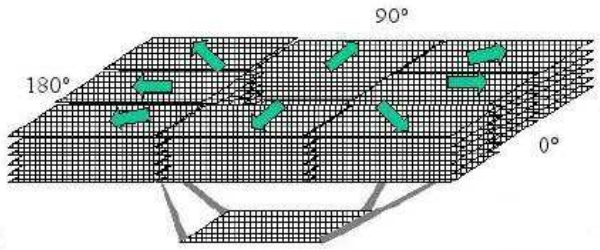


FIG. 4.1 – Organisation du module d'extraction locale.

vitesses associées au moyen d'un filtrage spatio-temporel de la séquence d'images reçue. Cette estimation est basée sur une méthode énergétique à base de filtres de Gabor.

Différentes approches ont été étudiées pour la détection locale du mouvement, principalement regroupées en quatre types : mise en correspondance (corrélation), différentielle, (fréquentiel) énergétique, (fréquentiel) fondé sur la phase, et stochastique [BP02, Cas05]. Notre modèle utilise les filtres énergétiques, qui peuvent être vus comme la recherche du plan qui correspond le mieux au contour local en déplacement de la séquence d'images dans le domaine spatio-temporel. Les filtres énergétiques combinent l'avantage d'exhiber des sorties qui peuvent modéliser les réponses des cellules de V1, et celui d'avoir une forme analytique relativement directe qui peut être potentiellement insérée assez aisément dans des calculs neuronaux à base d'excitations et d'inhibitions.

Mouvement 1D

Dans un espace 1D, détecter un mouvement revient à trouver un contour orienté dans les images (le temps étant la seconde dimension) ce que peut faire un filtre de Gabor (filtre passe-bande dans le domaine fréquentiel, réglé pour une fréquence donnée modulée par une gaussienne) :

$$g_s(x,t) = e^{-\frac{1}{2}(\frac{x^2}{\sigma_x}+\frac{t^2}{\sigma_t})} \sin(2\pi(w_x x + w_t t)) \tag{4.1}$$

Avec $\sigma = \sigma_x = \sigma_t$ et en travaillant après rotation de l'espace, on obtient :

$$\begin{aligned} g_s(x,t) &= e^{-\frac{1}{2}(\frac{x'^2+t'^2}{\sigma})} \sin(\frac{2\pi}{\lambda}(x')) \\ x' &= x\cos(\theta) - t\sin(\theta) \\ t' &= x\sin(\theta) + t\cos(\theta) \end{aligned} \tag{4.2}$$

Où λ est la longueur d'onde et θ l'angle de rotation (qu'on peut relier à la vitesse du mouvement 1D par vitesse= $\tan(\theta)$). Une relation utilisée dans [PK97] est $\frac{\sigma}{\lambda} = 0.56$, sur la base d'expériences en neurobiologie.

Le terme de filtre fréquentiel se retrouve dans son interprétation dans le domaine fréquentiel, où il s'agit d'un filtre passe bande orienté qui atteint son maximum en (w_x, w_t).

Bien que ce filtre puisse être réglé selon une certaine vitesse (ou orientation), il répond à différentes vitesses (orientations), ce qui est relié au problème d'ouverture. Pour résoudre ce problème, il faut utiliser des contraintes additionnelles [BP02], ou un processus de désambiguïsation éventuellement basé sur des informations rétropropagées [BN04].

Estimation de l'énergie

Avec le filtre ci-dessus, le signe de la réponse dépend du contraste du stimulus (un même mouvement donnera des valeurs opposés dans l'image "en négatif"). Pour construire un détecteur de mouvement indépendant de la phase, il est possible de combiner deux filtres en quadrature de phase [AB85].

$$g_s(x,t)^2 + g_c(x,t)^2 \tag{4.3}$$

où

$$g_c(x,t) = e^{-\frac{1}{2}(\frac{x'^2+t'^2}{\sigma})} \cos(\frac{2\pi}{\lambda}(x')) \tag{4.4}$$

L'utilisation d'égalités trigonométriques permet de ré-exprimer cette équation en séparant les termes spatiaux et temporels, exprimant l'énergie à base de filtres de Gabor sous la forme :

$$g_s(x,t)^2 + g_c(x,t)^2 = (g_s(t)g_c(x) + g_c(t)g_s(x))^2 + (g_s(t)g_s(x) - g_c(t)g_c(x))^2 \tag{4.5}$$

avec

$$g_s(t) = e^{-\frac{1}{2}\frac{t^2}{\sigma}} \sin(-2\pi\frac{\sin(\theta)}{\lambda}t) \tag{4.6}$$

$$g_c(t) = e^{-\frac{1}{2}\frac{t^2}{\sigma}} \cos(-2\pi\frac{\sin(\theta)}{\lambda}t) \tag{4.7}$$

$$g_s(x) = e^{-\frac{1}{2}\frac{x^2}{\sigma}} \sin(2\pi\frac{\cos(\theta)}{\lambda}x) \tag{4.8}$$

$$g_c(x) = e^{-\frac{1}{2}\frac{x^2}{\sigma}} \cos(2\pi\frac{\cos(\theta)}{\lambda}x) \tag{4.9}$$

Mouvement 2D

Détecter un mouvement 2D revient à trouver un plan orienté (la troisième dimension étant le temps), avec la contrainte d'un mouvement perpendiculaire à la direction du contour 2D en mouvement. On peut décrire le plan cherché au moyen de deux rotations.

$$x' = xcos(\theta) - ysin(\theta) \qquad x'' = t'sin(\phi) + x'cos(\phi) \tag{4.10}$$

$$y' = xsin(\theta) + ycos(\theta) \qquad y'' = y' \tag{4.11}$$

$$t' = t \qquad t'' = tcos(\phi) - x'sin(\phi) \tag{4.12}$$

L'expression du filtre de Gabor 2D est alors :

$$g_s(x,y,t) = e^{-\frac{1}{2}(\frac{x''^2}{\sigma_x}+\frac{y''^2}{\sigma_y}+\frac{t''^2}{\sigma_t})} \sin(\frac{2\pi}{\lambda}(x'')) \tag{4.13}$$

De la même manière qu'en 1D, en supposant $\sigma_x = \sigma_y = \sigma_t$, et avec des notations similaires on obtient finalement la forme suivante pour l'estimation de l'énergie :

$$g_s(x,y,t)^2 + g_c(x,y,t)^2 = (g_s(t)g_c(x,y) + g_c(t)g_s(x,y))^2 + (g_s(t)g_s(x,y) - g_c(t)g_c(x,y))^2$$

Afin de rendre robuste ce filtrage aux changements de luminosité, la sortie des filtres est normalisée par rapport à la luminance moyenne locale de l'image d'entrée (la sortie est divisée par

le niveau de gris local moyen). Une partie des travaux menés consiste à supprimer le recours à cette normalisation au travers d'un mécanisme excitateur-inhibiteur cohérent avec les mécanismes neuronaux mis en jeu dans les modules suivants (cf 4.3.1).

En ce qui concerne la capacité de ce premier module à être implanté de manière distribuée (cf annexe E), on notera que les travaux rapportés dans [THGCS05b] sont fondés sur une version initiale différente de ce filtrage énergétique : des filtres de Gabor spatiaux sont utilisés pour extraire les contours locaux, mais c'est une approche de mise en correspondance qui est appliquée de manière causale au niveau temporel (recherche locale des contours en des positions déduites des vitesses supposées). Les techniques et conclusions de [THGCS05b] peuvent être néanmoins étendues au filtres présentés ci-dessus.

4.1.2 Module de désambiguïsation

Le but de ce module est de résoudre l'ambiguïté des réponses du module d'extraction locale du flux optique (et ainsi de résoudre le problème d'ouverture). Il met en jeu un mécanisme d'interactions locales dépendant non seulement de la distance entre les champs récepteurs, mais aussi de l'antagonisme sémantique entre les différentes cartes neuronales du premier module (chacune étant associée à un vecteur élémentaire de mouvement spécifique). Ce mécanisme permet une compétition locale entre les neurones associés à un même champ récepteur, de façon à faire émerger un mouvement local prépondérant, tout en tenant compte dans cette compétition de la cohérence des mouvement détectés dans le voisinage locale du champ récepteur concerné.

De manière plus précise, des neurones voisins de la même carte neuronale reçoivent des excitations mutuelles, et sont inhibés par l'ensemble des neurones des autres cartes dans un voisinage déterminé par un rayon d'influence dépendant de l'activité de chaque neurone. La force des inhibitions est modulée suivant l'antagonisme des vecteurs élémentaires de mouvement. Ainsi des cartes représentant des mouvements de directions opposées s'inhiberont plus que des cartes dont les directions préférentielles sont proches (de même pour les vitesses). Ces principes ont été initialement proposés dans [Mog00].

Mécanisme excitateur-inhibiteur

On définit l'évolution des potentiels des différents neurones de ce module de désambiguïsation sur la base des champs neuronaux présentés au chapitre précédent.

L'actualisation à l'itération T du neurone situé en (x, y), de direction préférentielle θ et de vitesse préférentielle v lors du traitement de l'image située au temps t dans la séquence est définie par :

$$\begin{aligned}\frac{\partial H}{\partial T}(x,y,t,\theta,v,T) &= -A \cdot H(x,y,t,\theta,v,T) \\ &+ \big(B - H(x,y,t,\theta,v,T)\big) \cdot Exc(x,y,t,\theta,v,T) \\ &- \big(C + H(x,y,t,\theta,v,T)\big) \cdot Inh(x,y,t,\theta,v,T)\end{aligned}$$

Cette équation définit le comportement global des neurones et concentre l'activation des neurones autour de l'intervalle $[-C, B]$. Le premier terme définit l'évolution du potentiel du neurone (qui sera discrétisée dans le domaine temporel), et le deuxième est le terme d'actualisation, décomposé lui-même en un terme de relaxation, une contribution excitatrice du voisinage (et du neurone d'entrée dans le module d'extraction locale du flux optique) et une contribution inhibitrice du voisinage.

A, B et C sont des constantes réelles et η le coefficient d'apprentissage. Le voisinage d'un neurone est défini par son rayon d'influence qui est calculé en fonction de son état interne. Ainsi, plus un neurone est actif, plus son rayon d'influence est grand, et plus il a tendance à imposer sa direction et sa vitesse préférée à la zone recouverte par son champ récepteur. Pour résumer, les contributions excitatrices et inhibitrices sont issues des interactions locales dont la force entre deux neurones dépend de :

- la distance (euclidienne) entre les champs récepteurs des deux neurones,
- l'activation du neurone source de l'interaction,
- l'antagonisme de leur direction et de leur vitesse préférées (en général basé sur un produit scalaire des vecteurs élémentaires de mouvement associés).

Interactions avec les autres modules

Dans la sortie du module d'extraction locale du flux optique, des mouvements locaux sont détectés dans des zones fortement contrastées pourtant statiques. Ce problème apparaît souvent lorsque des images naturelles sont traitées. De tels pseudo-mouvements perturbent le module de désambiguïsation, et surtout le module de suivi décrit ci-après. Ce problème peut être résolu en supposant que les neurones du module de désambiguïsation ne répondent qu'aux objets dont le mouvement produit des réponses locales variables (l'objet ayant bougé à l'image suivante), ce que notre modèle modélise de façon analytique au travers de l'initialisation (itération à $T = 0$)des neurones de ce module :

$$H(x, y, t, \theta, v, T = 0) = F(x, y, t, \theta, v) - F(x, y, t - 1, \theta, v) \tag{4.14}$$

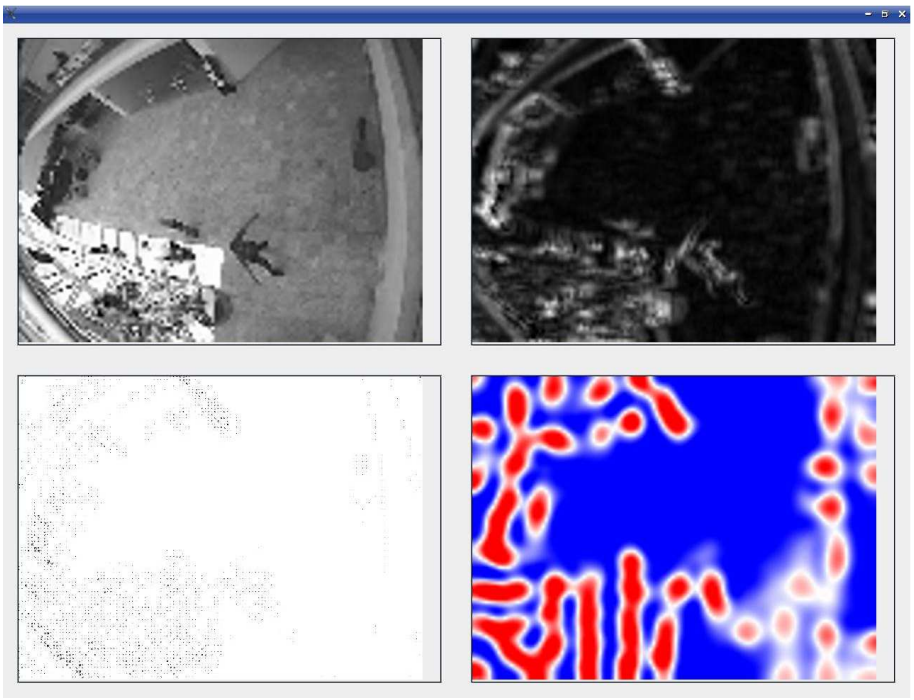
où $F(x, y, t, \theta, v)$ est l'estimation locale du mouvement issue du premier module pour le champ récepteur centré en (x, y), au temps t, pour la direction θ et pour la vitesse v. Les figures 4.1.2 (a) et (b) montrent l'influence de cette initialisation sur la présence de pseudo-mouvements. La séquence d'images est tirée d'une caméra de surveillance, et montre 4 personnes se déplaçant dans des zones plus ou moins contrastées. La figure 4.1.2 (a) montre les résultats obtenus lorsque les sorties des neurones du module d'extraction locale sont directement utilisées pour initialiser les neurones du module de désambiguïsation. La figure 4.1.2 (b) montre les résultats obtenus avec l'initialisation de l'équation (4.14). On notera qu'un des personnages en mouvement se retrouve sous la forme de deux bulles d'activité (buste et jambes).

L'architecture générale des deux premiers modules du modèle de [Cas05] est illustrée sur la figure 4.3. Prenant en entrée les valeurs calculées par les neurones du module d'extraction du flux optique, le module de désambiguïsation fournit les sorties de $\theta \times v$ neurones pour chaque champ récepteur initial. Le module suivant, i.e. le module de suivi des cibles en mouvement, se base sur une seule estimation de mouvement pour chaque "pixel". Les sorties des différentes cartes du module de désambiguïsation sont donc intégrées pour constituer la carte d'entrée du module de suivi, par exemple par un processus winner-take-all :

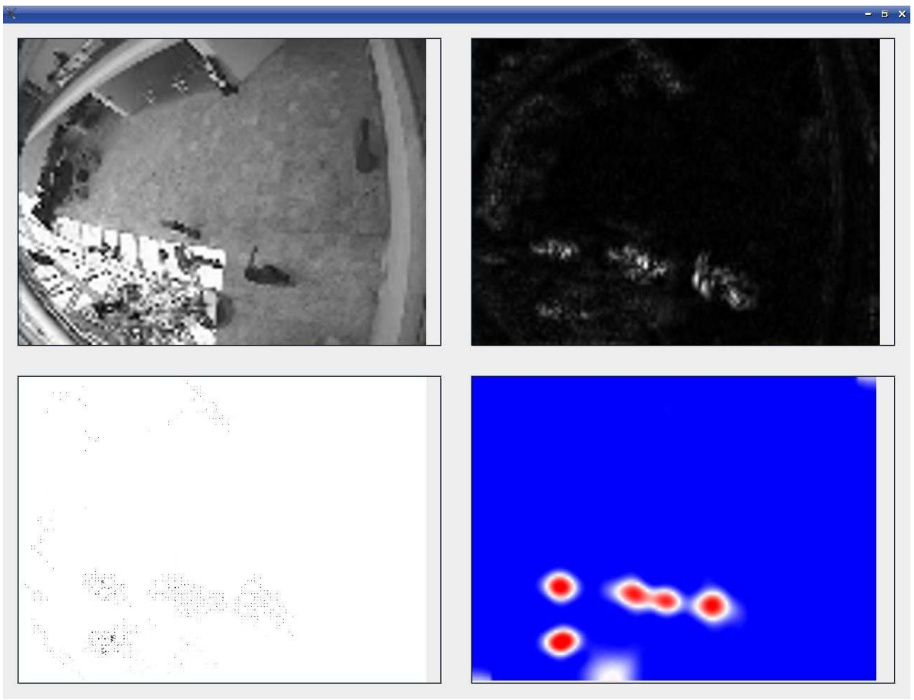
$$\hat{H}(x, y, t) = max_{v \in V, \theta} \left(H(x, y, t, \theta, v, T) \right) \tag{4.15}$$

4.1.3 Module de suivi

Le rôle du troisième module est d'assurer le suivi au travers d'un mécanisme d'attention visuelle. L'hypothèse la plus généralement répandue pour la focalisation de l'attention est l'existence d'une



(a)



(b)

FIG. 4.2 – (a) Initialisation directe. (b) Initialisation de l'équation (4.14). Pour chaque capture d'écran : en haut à gauche = image d'entrée, en haut à droite = extraction locale du flux optique, en bas à gauche = sortie du module de désambiguïsation, en bas à droite = carte visuelle du module de suivi

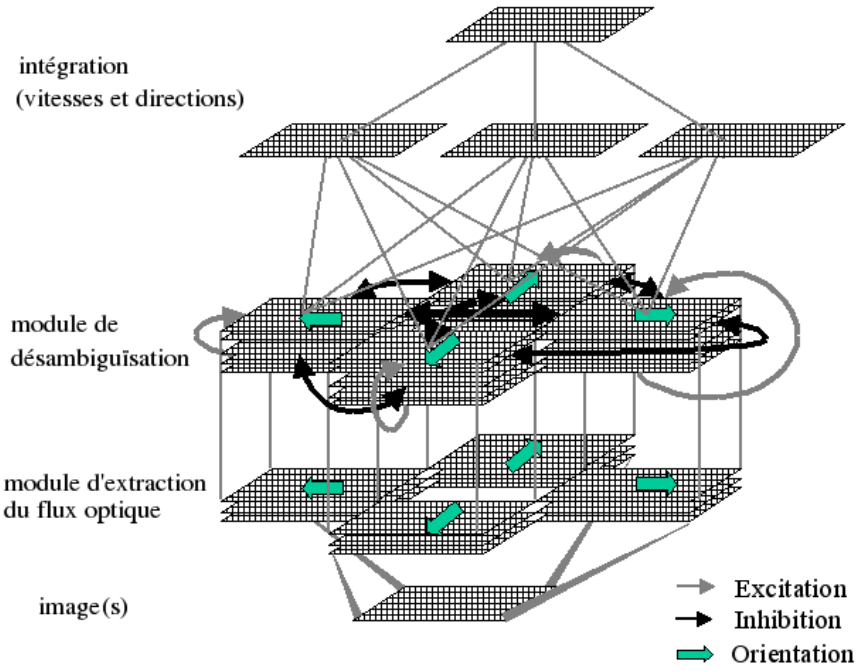


FIG. 4.3 – Architecture générale du modèle de [Cas05] (hors suivi).

carte de saillance. Notre modèle utilise ici une partie du modèle d'attention visuelle de [RV06], déjà présentée en 3.1.1. En utilisant comme carte d'entrée la carte issue de l'intégration des différentes cartes du module de désambiguïsation, la carte finale de focalisation fait émerger une seule bulle d'activité, représentant un mouvement dans la région de l'image qu'elle recouvre.

Les figures 4.4 et 4.5 illustrent le suivi de mouvement finalement obtenu par ce modèle modulaire sur deux séquences d'images. La première ligne montre deux images de la séquences. La seconde montre le flux optique extrait par le modèle aux mêmes instants de la séquence. La troisième ligne montre d'abord les cartes d'entrée du module de focalisation obtenu, puis le graphique final donne les différentes positions du centre de la bulle de la carte de focalisation tout au long de la totalité de la séquence. Dans la séquence de la figure 4.4, on peut observer deux personnes en train de marcher l'une vers l'autre, s'agripper puis s'éloigner l'une de l'autre. L'attention se porte d'abord peu de temps sur une des deux personnes, puis sur l'autre, avant de très vite revenir sur la première, et de se maintenir dessus après éloignement. Dans la séquence 4.5, une personne marche, avec une occultation peu avant la fin de la séquence. On notera que le suivi effectué "reproduit" les oscillations de la marche.

La suite du chapitre s'attache à décrire certains évolutions proposées de notre modèle connexionniste bio-inspiré de perception visuelle du mouvement. Ces travaux sont toujours en cours.

4.2 Perception rétino-centrée

Cette section décrit l'adaptation de notre modèle à un champ visuel non plus directement issu d'une caméra, mais fondé sur une vision rétino-centrée, c'est à dire où les champs récepteurs grandissent avec l'éloignement au centre de la rétine, permettant ainsi une plus grande précision de

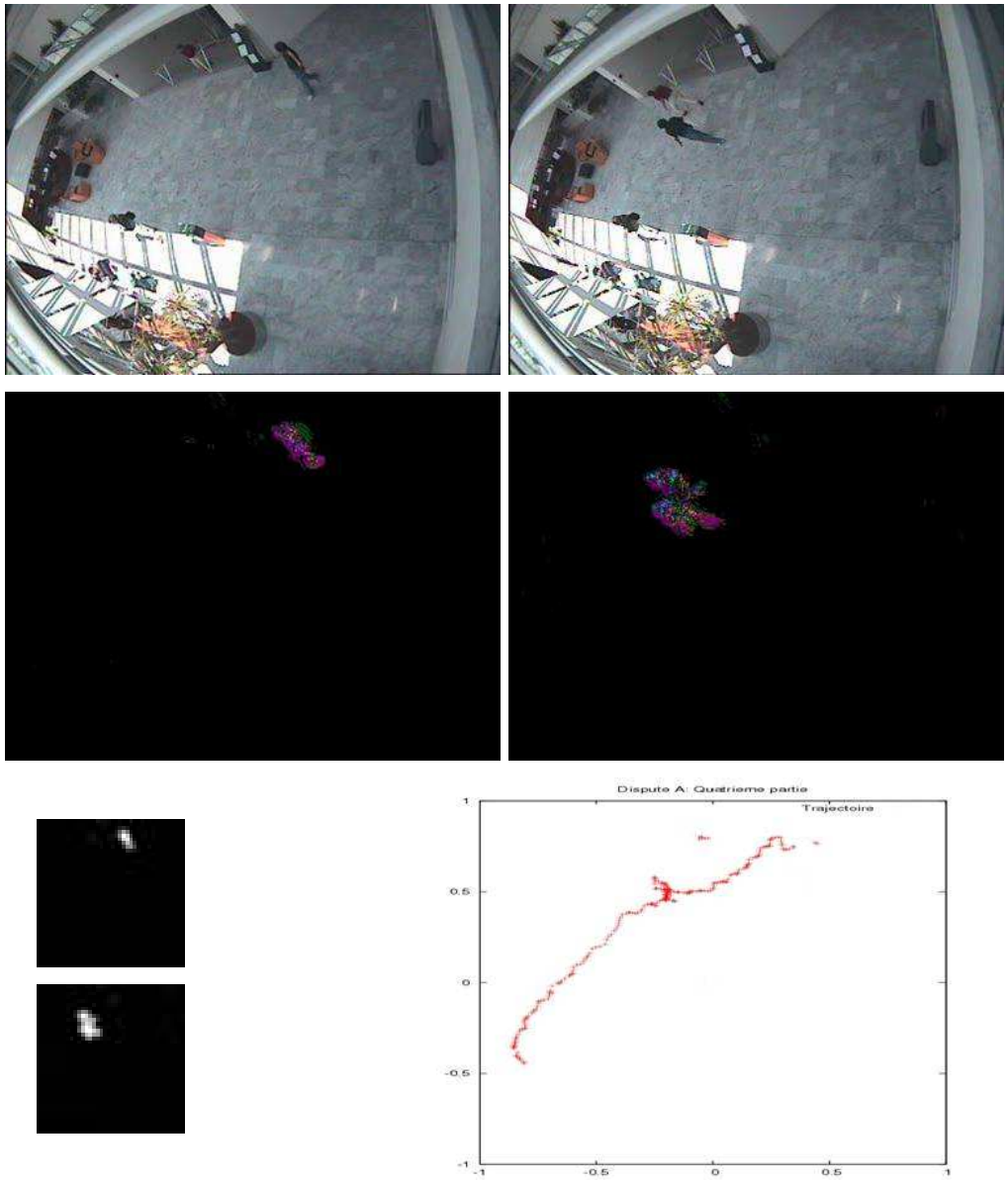


FIG. 4.4 – Extraction et suivi de mouvement (séquence 1)

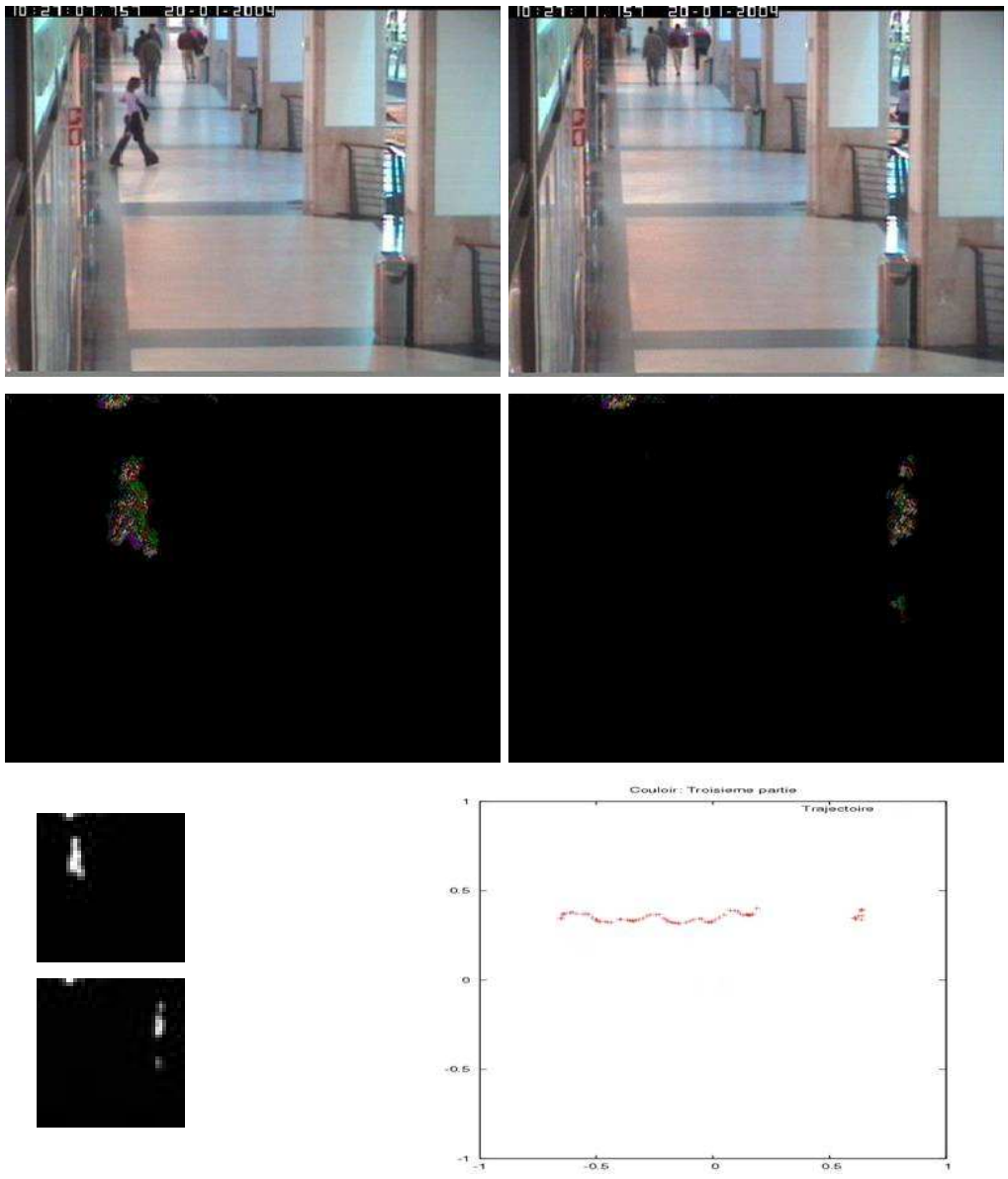


FIG. 4.5 – Extraction et suivi de mouvement (séquence 2)

la perception visuelle centrale tout en assurant une perception, même peu précise, des mouvements dans la périphérie du champ visuel. Cette adaptation n'est pas seulement motivée par les arguments neurophysiologiques sur la rétine. Elle permet en effet, à ressources de calcul égales, de couvrir un champ visuel large tout en maximisant la précision centrale (les cibles suivies pouvant ensuite être recentrées pour les percevoir avec plus de précision). Cet aspect calculatoire est important dans un modèle massivement distribué comme le nôtre.

La répartition des champs récepteurs dans une vision rétino-centrée peut être modélisée par une transformation log-polaire [TS93, JBO87]. Nous nous inspirons du modèle de [BL98]. Ce modèle divise l'image d'entrée en deux régions, produisant ainsi deux images qui se complètent. L'image dite intérieure contient la région centrale de l'image d'entrée, avec une résolution maximale (celle de l'image d'entrée). L'image dite extérieure utilise un système de coordonnées log-polaire et est caractérisé par une réduction de la quantité de données entourant la région centrale par rapport à l'image d'entrée.

4.2.1 Définition

Il est nécessaire de définir la transformation en coordonnées log-polaire, mais également la transformée inverse qui permet de retrouver l'image dans le système de coordonnées d'origine.

Transformation log-polaire

Soit $I(x, y)$ une image dans le plan euclidien, $\mathbb{R}$ x $\mathbb{R}$, alors la transformée log-polaire $I_{LP}(\xi, \psi)$ avec origine en (x_c, y_c) est définie par le changement de coordonnées suivant :

$$\xi = log_a(\rho.k) \tag{4.16}$$
$$\psi = \theta \tag{4.17}$$

où k est un nombre réel et (ρ, θ) sont les coordonnées polaires définies par :

$$(\rho, \theta) = \left(\sqrt{(x - x_c)^2 + (y - y_c)^2}, \arctan\left(\frac{y - y_c}{x - x_c}\right)\right) \tag{4.18}$$

Dans le domaine discret, les coordonnées (ξ, ψ) deviennent :

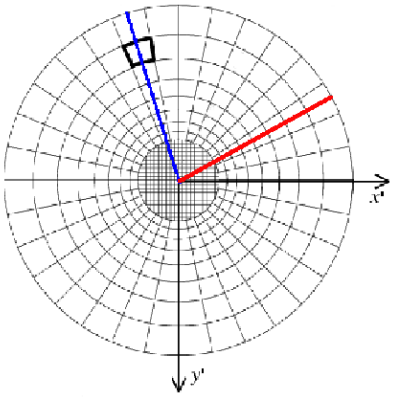
$$r = \lfloor log_a(\rho.k) \rfloor \tag{4.19}$$
$$t = \lfloor \theta.\Delta t \rfloor \tag{4.20}$$

où $0 \leq r < R$, $0 \leq t < T$, $\Delta t = \frac{2\pi}{T}$, avec R et T étant respectivement le nombre d'anneaux et de secteurs par anneau du "quadrillage" log-polaire de l'image (voir figure 4.6).

En associant le rayon discret $r = 0$ au rayon continu ρ_0 de la zone centrale de l'image, on peut exprimer k de la façon suivante :

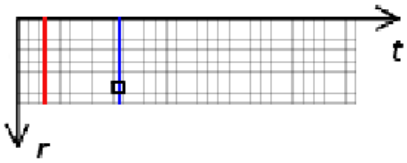
$$log_a(\rho_0.k) = 0 \Rightarrow \rho_0.k = 1 \Rightarrow k = \frac{1}{\rho_0} \tag{4.21}$$

Il faut noter que contrairement à une transformation log-polaire de toute l'image, le maintien au centre d'une image non transformée ayant la résolution initiale de l'image d'origine permet de contourner le problème de la singularité au centre dans une transformation log-polaire totale (θ étant alors indéterminé au centre, et les pixels de taille infiniment petite).



(a)

$$\begin{aligned} x' &= x - x_c \\ y' &= y - y_c \end{aligned}$$



(b)

FIG. 4.6 – Transformation log-polaire. (a) Nouvelle distribution des pixels. (b) Représentation de l'image extérieure, où $r = 0$ est associé à ρ_0 et $r = R - 1$ à ρ_{max}.

De la même manière pour R :

$$R = log_a \left(\frac{\rho_{max}}{\rho_0} \right) \tag{4.22}$$

où ρ_{max} est le rayon maximal du champ visuel considéré. Cette équation permet d'exprimer a en fonction du choix de ρ_0, ρ_{max} et R :

$$a = e^{\ln\left(\frac{\rho_{max}}{\rho_0}\right)/R} \tag{4.23}$$

En appliquant cette transformation, l'image originale est représentée par deux images, l'image intérieure de taille $(2\rho_0 \text{x} 2\rho_0)$[1] et l'image extérieure de taille $(R\text{x}T)$, qui représente la zone transformée comprise entre les rayons ρ_0 et ρ_{max} (voir figure 4.6). Les valeurs des pixels de l'image extérieure sont calculées comme les moyennes des pixels de l'image initiale qui se trouvent dans le secteur associé.

[1]En fait seuls les pixels contenus dans le disque de rayon ρ_0 sont utilisés.

Transformation inverse

Etant données les coordonnées (t, r) dans l'espace log-polaire discret, les coordonnées cartésiennes associées peuvent être calculées par :

$$x = \lfloor \rho.\cos(\theta) \rfloor + x_c \quad (4.24)$$
$$y = \lfloor \rho.\sin(\theta) \rfloor + y_c \quad (4.25)$$

où

$$\rho = \rho_0.a^r \quad (4.26)$$
$$\theta = t.\Delta t \quad (4.27)$$

Les pixels de la région centrale sont directement obtenus à partir de l'image intérieure par :

$$x = x_I - \rho_0 + x_c \quad (4.28)$$
$$y = y_I - \rho_0 + y_c \quad (4.29)$$

où x_I et y_I sont les coordonnées dans l'image intérieure.

4.2.2 Propriétés et paramétrisation

Surface des secteurs

La surface des secteurs de l'anneau numéro r est :

$$A(r) = \frac{\pi\rho_r^2 - \pi\rho_{r-1}^2}{T} = \frac{\pi\left(\rho_r^2 - \rho_r^2/a^2\right)}{T} = \frac{\pi\rho_r^2(a^2-1)}{a^2T} \quad (4.30)$$

Facteur d'aspect

Un facteur d'aspect des secteurs peut être défini comme :

$$\gamma(r) = \frac{w(r)}{h(r)} \quad (4.31)$$

où $w(r)$ et $h(r)$ sont la largeur et la hauteur du secteur. En évaluant la largeur comme la longueur de son arc intérieur :

$$w(r) = \frac{2\pi}{T}.\rho_{r-1} = \frac{2\pi}{T}.\rho_0.a^{r-1} \quad (4.32)$$
$$h(r) = \rho_r - \rho_{r-1} = \rho_0.a^{r-1}(a-1) \quad (4.33)$$

Le facteur d'aspect est alors :

$$\gamma(r) = \frac{w(r)}{h(r)} = \frac{\frac{2\pi}{T}.\rho_0.a^{r-1}}{\rho_0.a^{r-1}(a-1)} = \frac{2\pi}{T(a-1)} \quad (4.34)$$

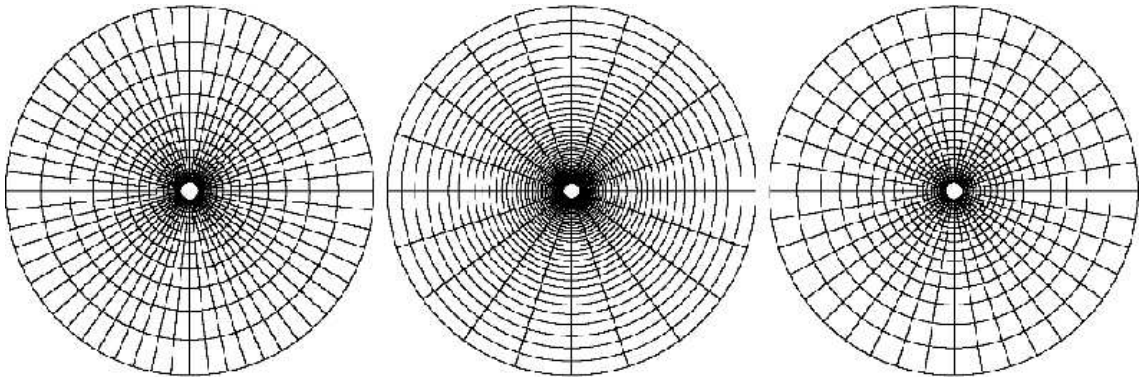


FIG. 4.7 – Distributions log-polaires avec différents facteurs d'aspect. (à gauche) $\gamma(r) < 1$. (au centre) $\gamma(r) > 1$. (à droite) $\gamma(r) = 1$.

Sur-échantillonage

On est en présence de sur-échantillonage quand un pixel de l'espace cartésien correspond à plusieurs pixels de l'espace log-polaire. Le sur-échantillonage est alors maximal sur l'anneau le plus proche de la région centrale, et il peut être quantifié comme le rapport entre les surfaces des pixels cartésien et log-polaire.

$$O(r_0) = \frac{1}{A(r_0)} \tag{4.35}$$

Choix des paramètres

La transformation log-polaire dépend de cinq paramètres : R, T, ρ_0, ρ_{max} et a. Compte tenu des propriétés évoquées ci-dessus, le choix des paramètres peut être contraint selon différents critères.

Facteur d'aspect unitaire : Des exemples de distributions log-polaire sont montrés en figure 4.7 pour $\gamma(r) < 1$, $\gamma(r) > 1$ et $\gamma(r) = 1$. Afin de maintenir des pixels log-polaires ni "allongés", ni "aplatis", le choix de $\gamma(r) = 1$ peut être fait afin d'obtenir la même résolution dans les directions radiales et angulaires. Ceci implique :

$$\gamma(r) = 1 \Rightarrow T = \frac{2\pi}{a-1} \tag{4.36}$$

Taille de l'image rétino-centrée : La taille totale de l'image rétino-centrée sur laquelle le modèle de perception du mouvement va s'appliquer peut être bornée :

$$N = R.T + \pi\rho_0^2 < N_{max} \tag{4.37}$$

Ceci permet de choisir N_{max} en fonction des ressources en calcul et mémoire disponibles, et d'optimiser les choix de R, T et ρ_0 qui en découlent pour une précision centrale souhaitée.

Echantillonage : Pour éviter le sur-échantillonage, il faut vérifier :

$$O(r_0) \leq 1 \Rightarrow T \geq \frac{\pi\rho_0^2(a^2-1)}{a^2} \tag{4.38}$$

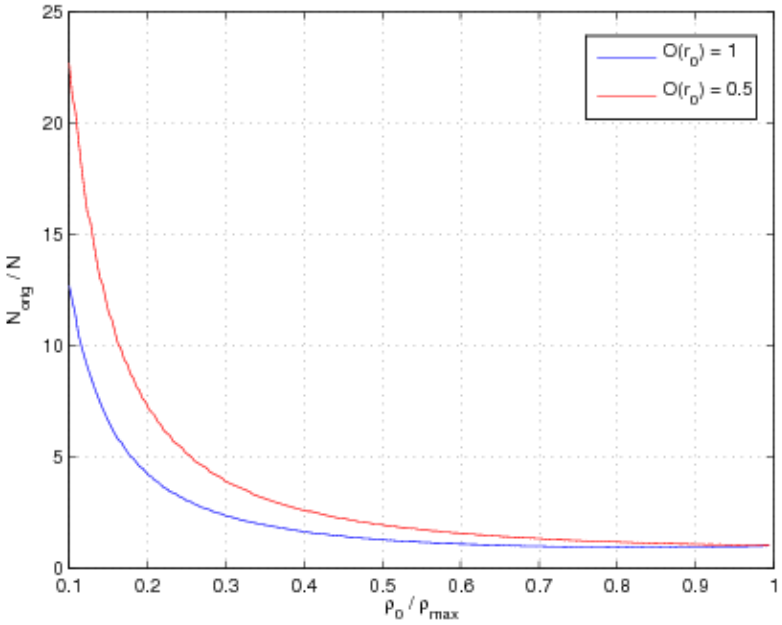


FIG. 4.8 – Relation entre N_{orig}/N et ρ_0/ρ_{max} pour $O(r_0) = 1$ et $O(r_0) = 0.5$.

où $O(r_0) = 1$ correspond à une continuité d'aspect et de surface entre les pixels de la région centrale et ceux de la périphérie immédiate.

Rayon maximal du champ visuel : Dans les expériences rapportées, ρ_{max} est choisi de manière à maximiser l'image rétino-centrée à l'intérieur de l'image initiale (et non pas à recouvrir cette image).

$$\rho_{max} = \min\left(\frac{F}{2}, \frac{C}{2}\right) \tag{4.39}$$

où F et C sont le nombre de lignes et de colonnes de l'image initiale.

En tenant compte de ces critères, on peut paramétrer la transformation log-polaire au moyen de deux paramètres libres, R et ρ_0. Pour tenir compte de l'équation (4.38), le choix de ρ_0 est d'abord effectué (taille désirée pour la région centrale), puis R est minimisé de manière à atteindre le facteur de sur-échantillonage désiré.

La figure 4.8 montre la réduction du nombre de neurones N dans le modèle après transformation log-polaire des images d'entrée pour différentes valeurs de ρ_0 par rapport au nombre initial de neurones N_{orig}.

4.2.3 Interactions entre les zones centrale et périphérique

Lorsque les champs récepteurs ou les rayons d'influence des neurones associés aux pixels de l'image rétino-centrée recouvrent à la fois des pixels de l'image intérieure et de l'image extérieure, les interactions des différentes cartes sont redéfinies pour gérer simultanément les deux zones. Ainsi,

chacune des cartes neuronales du modèle présenté en section 4.1 est désormais associée à deux cartes qu'on désignera par carte intérieure ou carte extérieure de la même manière que pour les images.

- Lorsqu'un neurone n_1 d'une carte intérieure doit exciter ou inhiber un neurone associé à un pixel situé en dehors du disque de rayon ρ_0 centré en (ρ_0, ρ_0), alors le neurone réellement influencé est pris dans la carte extérieure correspondante. Un neurone d'une carte intérieure peut ainsi exciter ou inhiber plusieurs fois un neurone d'une carte extérieure.
- Lorsqu'un neurone d'une carte extérieure doit exciter ou inhiber un neurone associé à un pixel de coordonnées (t_1, r_1) situé en dehors de l'image extérieure, plusieurs possibilités apparaissent :
  1. Si $t_1 < 0$, le neurone correspondant est en position $(T + t_1, r_1)$ de la carte extérieure.
  2. Si $t_1 \geq T$, le neurone correspondant est en position $(t_1 - T, r_1)$ de la carte extérieure.
  3. Si $r_1 \geq R$, le neurone correspondant est en dehors de l'image.
  4. Si $r_1 < 0$, le neurone correspondant est dans la carte intérieure correspondante. Dans ce cas, tous les neurones de la carte intérieure qui sont associés aux coordonnées log-polaires (t_1, r_1) sont excitées ou inhibées.
- Les poids des interactions sont les mêmes que dans le modèle initial, qu'il s'agisse de neurones d'une carte intérieure ou extérieure. Cet aspect sera remis en cause.

4.2.4 Résultats expérimentaux

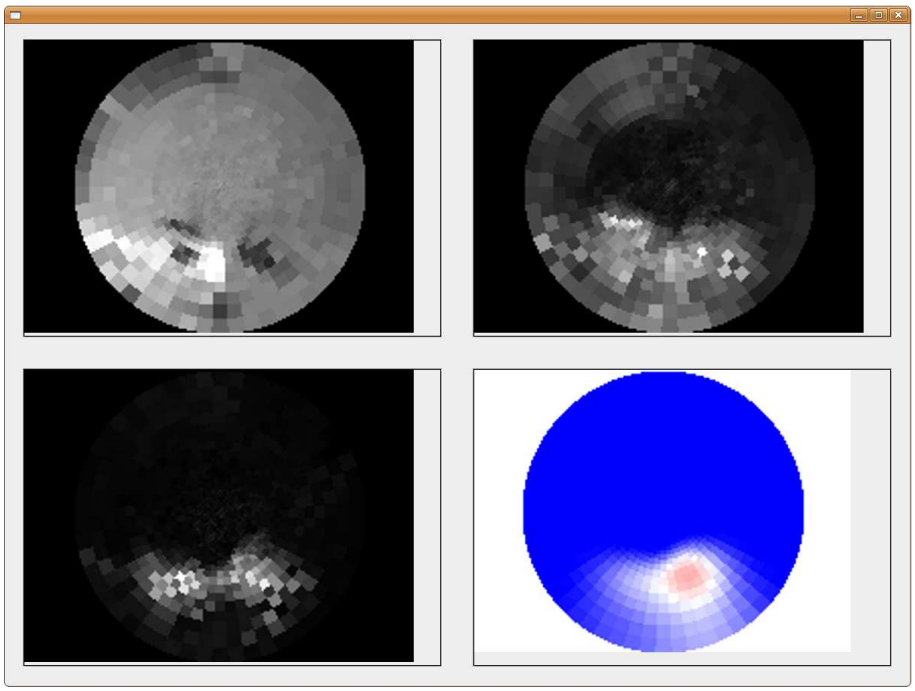
Les résultats sont présentés ici pour deux séquences d'images, *Hall* et *Magasin*. La séquence *Hall* présente 3 personnes marchant dans des directions opposées. La séquence *Magasin* présente une personne marchant perpendiculairement à la caméra. Différentes valeurs de ρ_0 sont testées. Dans toutes les figures, l'image en haut à gauche correspond à la transformation rétino-centrée d'une image de la séquence en entrée, l'image en haut à droite correspond à la sortie du module d'extraction locale du flux optique, l'image en bas à gauche correspond à la sortie du module de désambiguïsation, et l'image en bas à droite correspond à la carte visuelle du module de suivi.

Les figures 4.9, 4.10 et 4.11 montrent les résultats utilisant la séquence *Hall*. Lorsque la valeur de ρ_0 augmente, la résolution augmente dans la zone extérieure et les bulles d'activités sont mieux maintenues (et sont moins soumises à un certaine diffusion) dans les cartes extérieures. Lorsque les mouvements sont très proches des limites de l'image, les bulles d'activité s'étalent et parfois disparaissent. Ce dernier effet est dû au nombre de neurones actifs nécessaires pour maintenir une bulle compte tenu des paramètres choisis dans le module de suivi. Une adaptation continue de ces paramètres avec l'éloignement au centre est donc souhaitable.

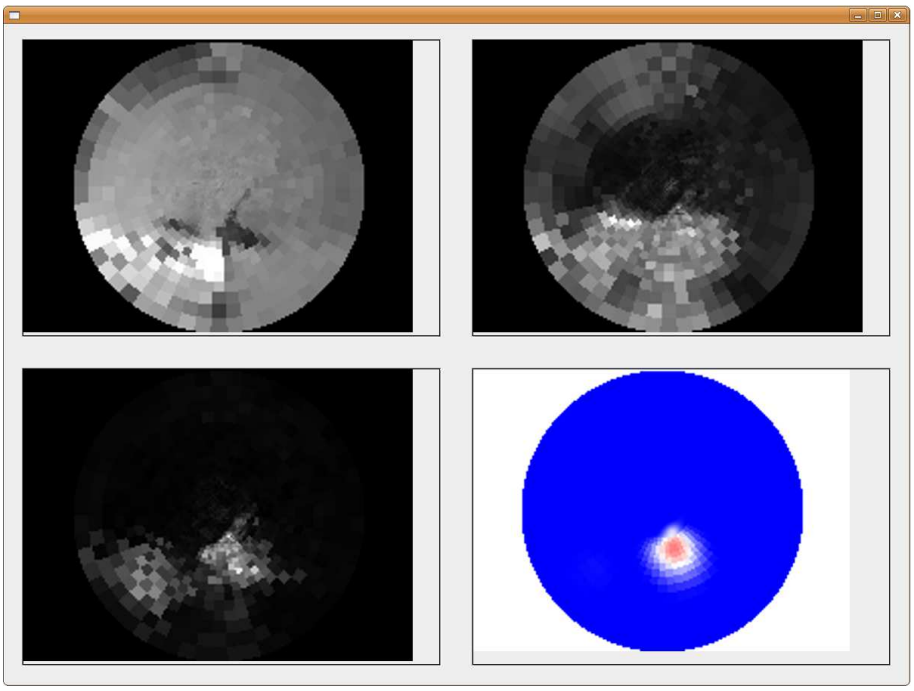
Des résultats similaires sont obtenus avec la séquence *Magasin* (figures 4.12 et 4.13). L'augmentation de la résolution se traduit dans la figure 4.13 par des bulles d'activité correspondant aux jambes de la personne. De façon plus générale, les tests ont montré que si la diminution de ρ_0 peut engendrer des résolutions trop faibles, elle diminue aussi les bruits de haute fréquence introduits par les tremblements de la caméra.

4.3 Interactions rétropropagées

Une partie des travaux actuels porte sur l'introduction massive de connexions rétropropagées (cf section 3.2.4). Ces connexions apparaissent à deux niveaux. D'une part au sein même du module d'extraction locale du flux optique, en se basant sur une amélioration de la détection par

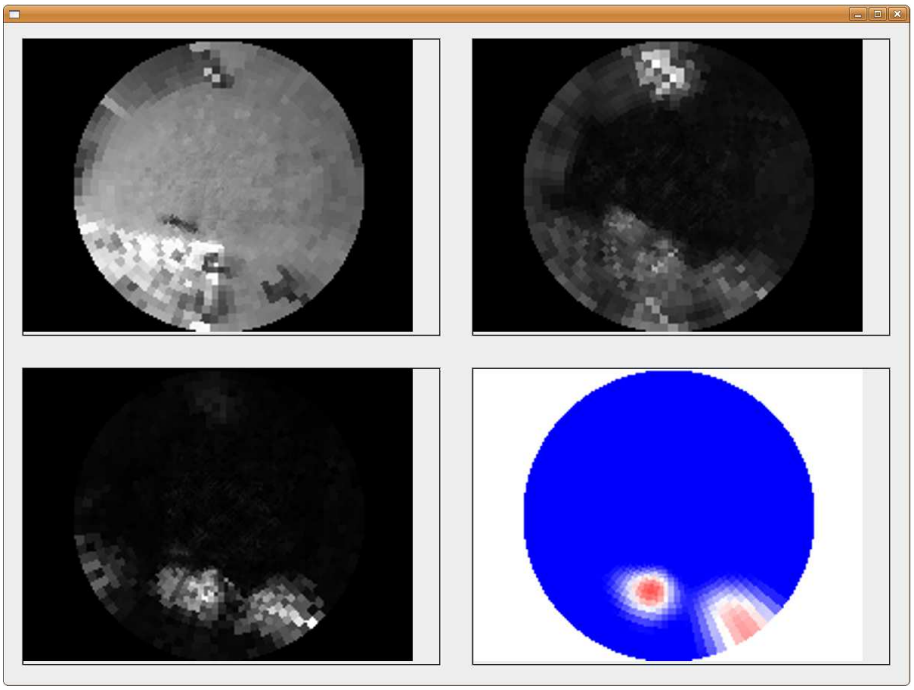


(a)

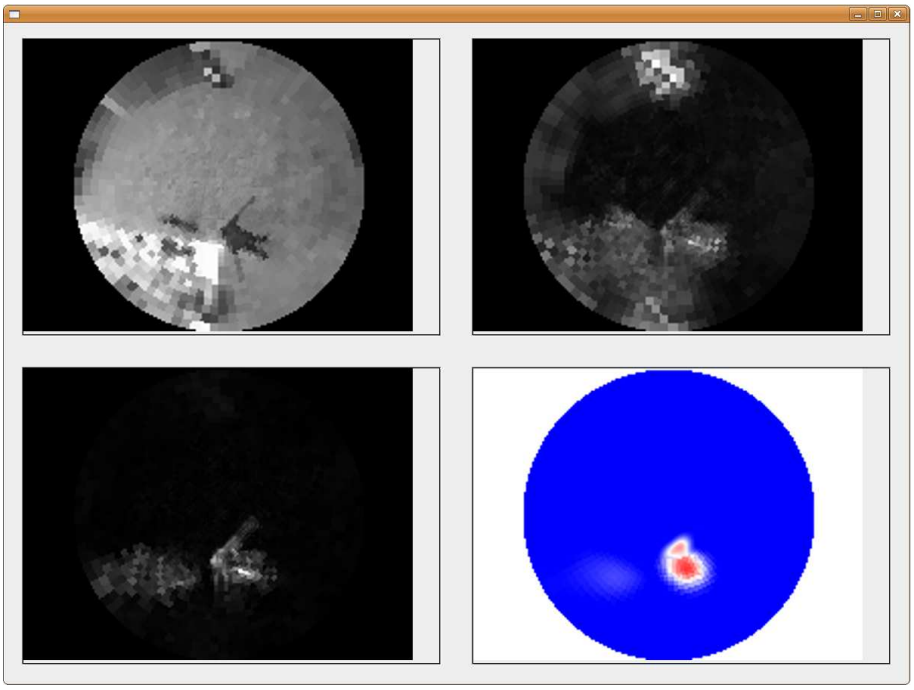


(b)

FIG. 4.9 – Séquence *Hall* : simulation avec $\rho_0 = 0.2\rho_{max}$. Dans la figure (a) on peut voir l'effet de diffusion des bulles d'activité. Dans la figure (b) les mouvements sont plus proches de la région centrale, avec moins de diffusion.

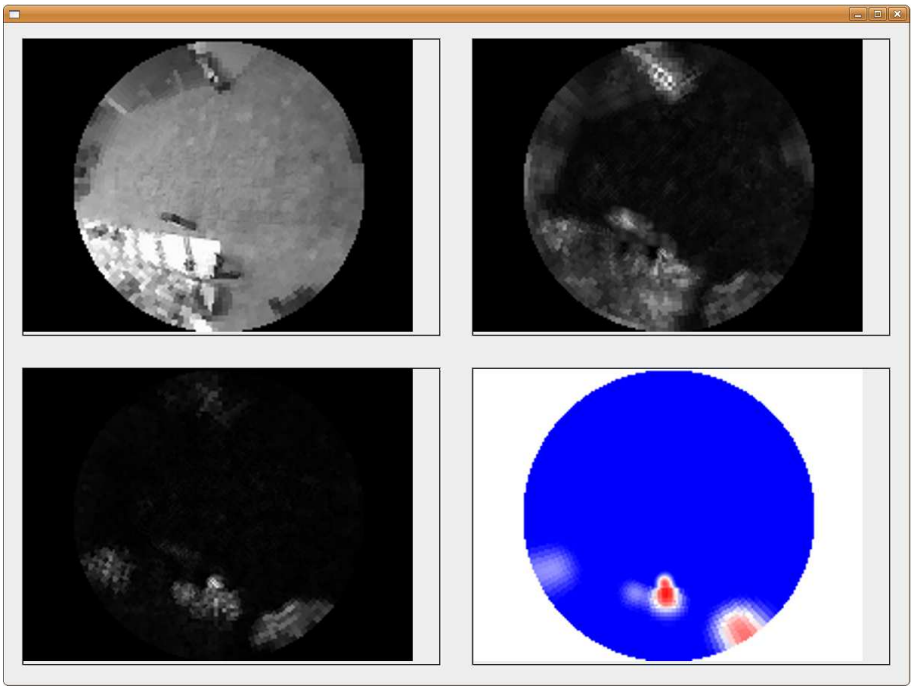


(a)

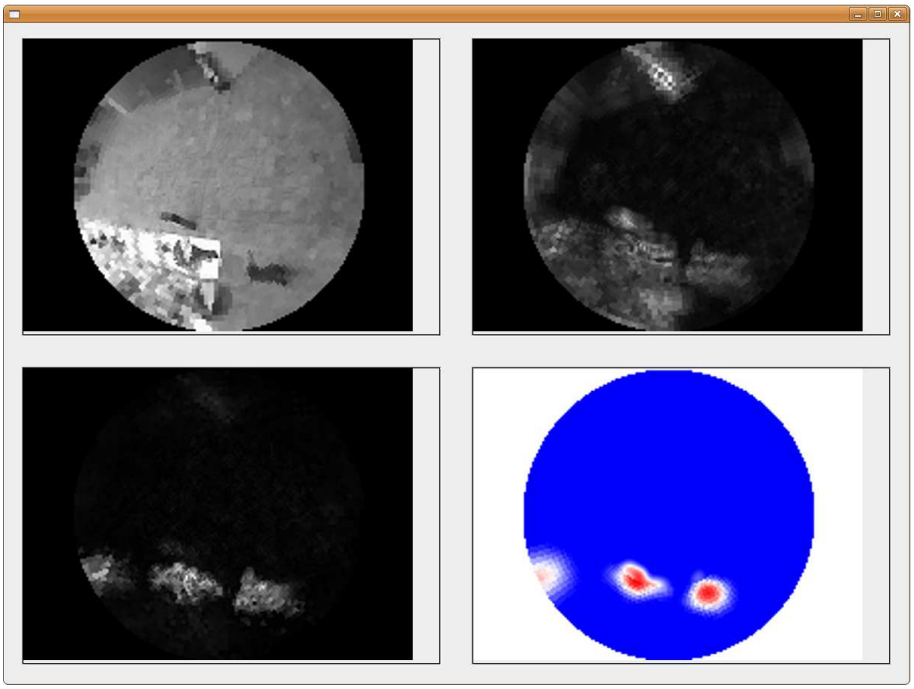


(b)

FIG. 4.10 – Séquence *Hall* : simulation avec $\rho_0 = 0.3\rho_{max}$. (a) Deux personnes sont détectées. (b) La bulle associée à la personne évoluant dans la zone bruitée en bas à gauche de l'image commence à disparaître en raison du faible contraste local.

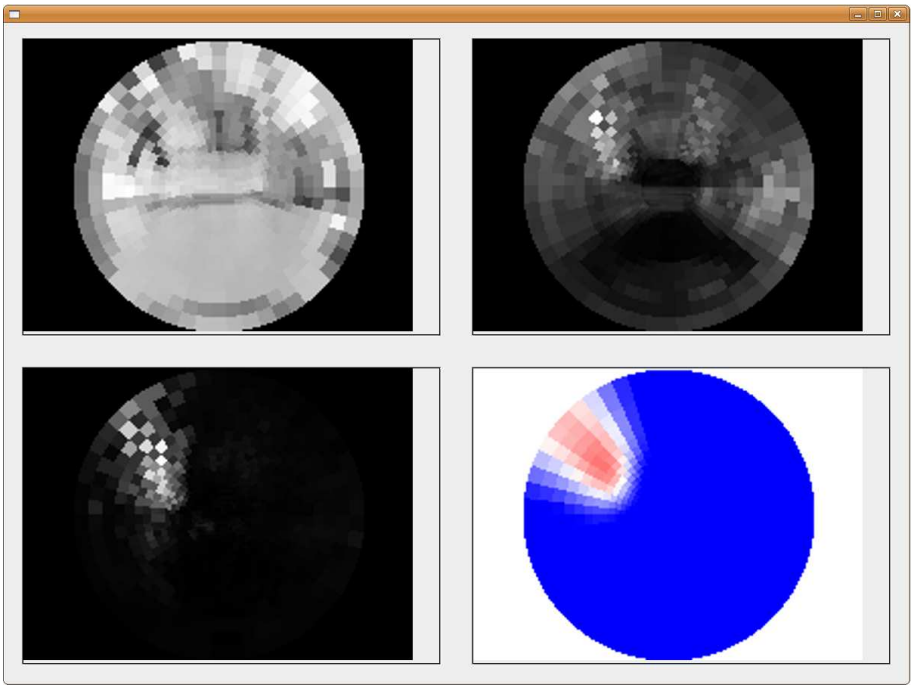


(a)

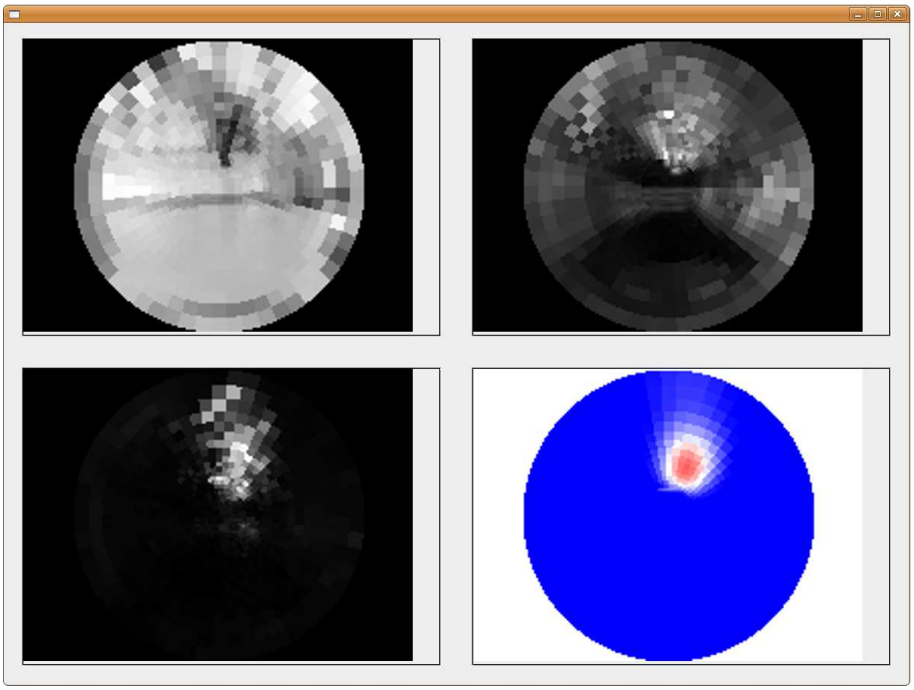


(b)

FIG. 4.11 – Séquence *Hall* : simulation avec $\rho_0 = 0.5\rho_{max}$. Sur (a) et (b) les bulles associées aux trois personnes apparaissent.

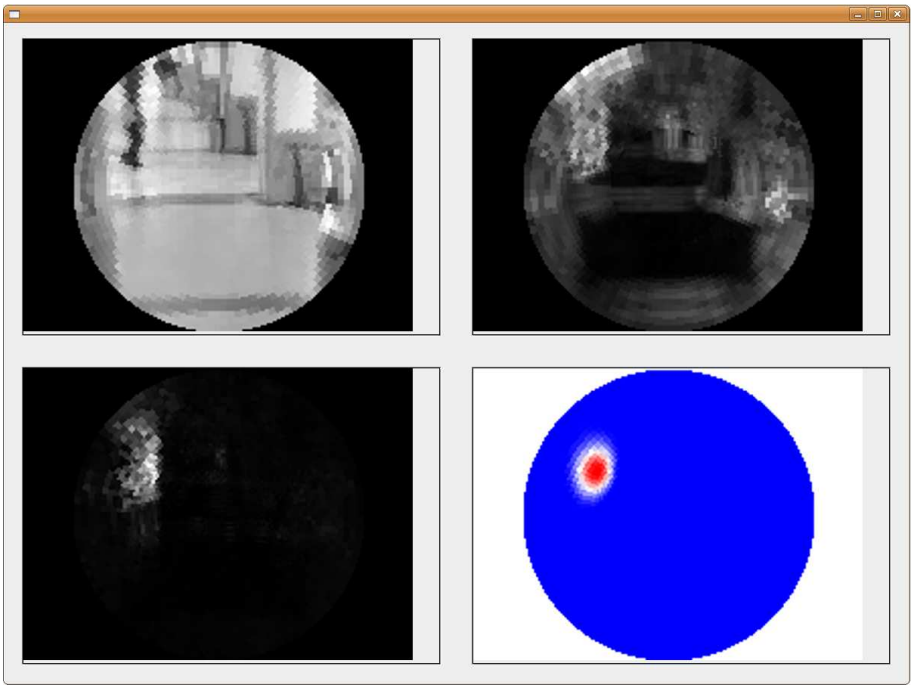


(a)

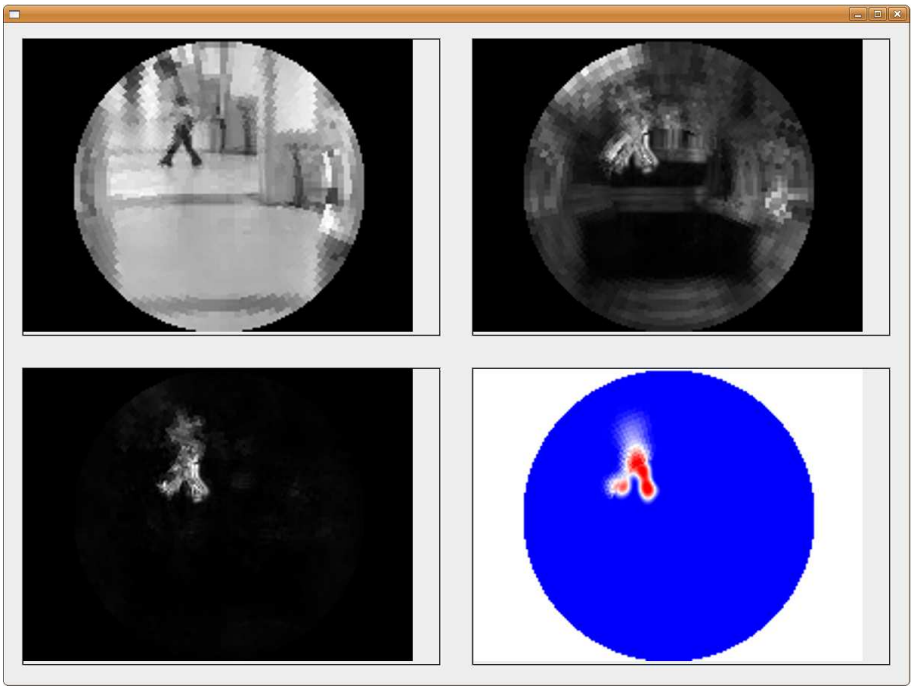


(b)

FIG. 4.12 – Séquence *Magasin* : simulation avec $\rho_0 = 0.2\rho_{max}$. (a) Effet de diffusion des bulles en périphérie. (b) Diffusion réduite plus près du centre.



(a)



(b)

FIG. 4.13 – Séquence *Magasin* : simulation avec $\rho_0 = 0.4\rho_{max}$. (a) Bonne détection de la personne bougeant en périphérie. (b) Détection précise en région centrale.

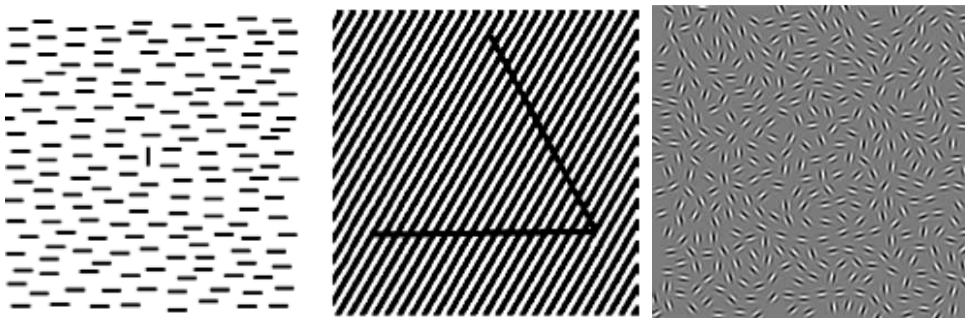
effet centre-périphérie. D'autre part entre les modules d'extraction locale et de désambiguïsation, afin d'améliorer la précision des mouvements détectés tout en effectuant une désambiguïsation des mouvements détectés au niveau de chaque contour.

4.3.1 Modélisation des effets centre-périphérie

Afin d'améliorer la détection locale de mouvement dans les zones bruitées et dans les zones faiblement contrastées (notamment en vue de la suppression des mécanismes de normalisation utilisés dans le module d'extraction locale), nous avons introduit dans le premier module des interactions rétropropagées modélisant les effets dits centre-périphérie. Cette modélisation se base sur des constatations expérimentales.

Effets centre-périphérie dans les images statiques

L'observation de réponses de cellules de V1 dans [PMPK98] indique qu'il existe une relation entre la direction détectée au centre et celle détectée sur la périphérie d'un champ récepteur : excitation du centre lorsque la direction en périphérie est opposée (effet "pop-out"), excitation du centre lorsque la direction détectée en périphérie est la même qu'au centre le long de la direction ainsi détectée (effet de colinéarité). La figure 4.14 illustre ces effets.



(a) Effet pop-out. (b) Effet de colinéarité.

FIG. 4.14 – Exemples d'effets centre-périphérie.

Amélioration de la détection de mouvement en faible contraste

Les travaux de [HJP+98] indiquent qu'une désactivation par refroidissement de MT produit une baisse de l'activité dans l'aire V1, et ceci de manière plus marquée pour les stimuli faiblement contrastés. Les interactions rétropropagées de MT vers V1 semblent être majoritairement excitatrices.

Le principal point commun entre les conclusions de [HJP+98] et [PMPK98] est le fait que l'état des neurones associés au centre des champs récepteurs est modifié par celui des neurones associés à la périphérie. De nombreux autres travaux confirment cette assertion, mais semblent indiquer que ces effets vont au-delà de la portée des interactions latérales, et que la vitesse de ces interactions est insuffisante pour expliquer les effets constatés. L'existence de connexions rétropropagées apparaît comme une hypothèse intéressante pour expliquer les effets centre-périphérie.

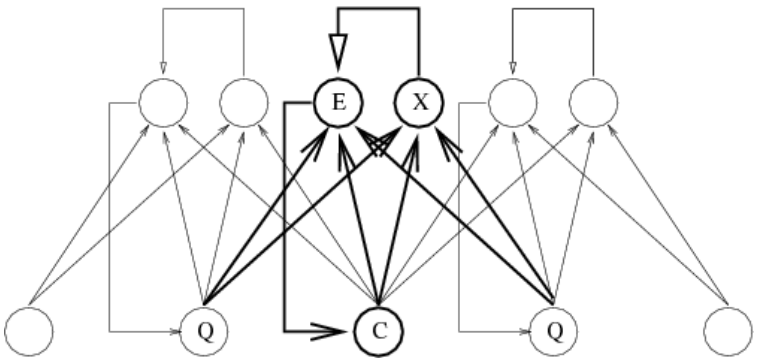


FIG. 4.15 – Modélisation des effets centre-périphérie par connexions rétropropagées.

Modélisation

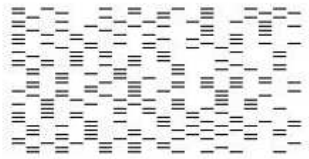
Différentes modélisations ont été proposées pour les effets centre-périphérie, par exemple dans [XH01] avec des interactions latérales et dans [SdS06] avec des interactions rétropropagées. La nature des calculs proposés dans ces modèles est difficilement compatible avec les neurones excitateurs-inhibiteurs de notre modèle. C'est néanmoins sur la base des idées de [SdS06] que nous avons proposé un modèle pour la prise en compte des effets centre-périphéries. Le principe consiste à insérer dans le module d'extraction locale du flux optique une double couche de neurones (couche excitatrice, couche inhibitrice), comme illustré en figure 4.15.

Les différents neurones associés à un champ récepteur de détection locale du mouvement émettent des interactions vers les neurones excitateurs et inhibiteurs associés au centre du champ récepteur. Le neurone central reçoit en retour l'interaction issue du neurone excitateur. Une modélisation générale est la suivante :

$$\begin{aligned}
\frac{\partial C}{\partial t} &= -\alpha C + \widetilde{E} + k * Input \\
\frac{\partial X}{\partial t} &= -\beta X + \eta \widetilde{C} + \Sigma_i \lambda_i Q_i \\
\frac{\partial E}{\partial t} &= -\gamma E + \mu \widetilde{C} + \Sigma_i \omega_i Q_i - \widetilde{X} \\
\widetilde{C} &= Max(C, 0) \\
\widetilde{X} &= Max(X, 0) \\
\widetilde{E} &= Max(E, 0)
\end{aligned}$$

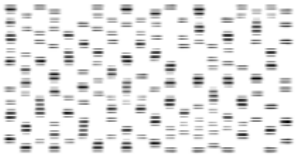
où C est l'activité du neurone central, X celle du neurone inhibiteur associé, E celle du neurone excitateur associé, et Q_i représente les activités des neurones associés au même champ récepteur.

Ce cadre général a été plus précisément paramétré pour modéliser les mécanismes d'inhibition iso-périphérique permettant d'expliquer les effets pop-out : si le stimulus est le même au centre et à la périphérie du champ récepteur, cela accroît l'activité du neurone inhibiteur et diminue



(a) Image d'entrée.

(b) Détection horizontale.



(c) Après 5 itérations d'inhibition iso-périphérique.

(d) Après 20 itérations (et plus).

FIG. 4.16 – Inhibition iso-périphérique par interactions rétropropagées.

l'excitation rétropropagée vers le centre. Les équations simplifiées obtenues sont :

$$\begin{aligned} \frac{\partial C}{\partial t} &= \widetilde{E} + k * Input \\ \frac{\partial X}{\partial t} &= \eta \widetilde{C} + \Sigma_i \lambda_i Q_i \\ \frac{\partial E}{\partial t} &= -\widetilde{X} \end{aligned}$$

où les coefficients η et λ_i sont obtenus au moyen d'une différence de gaussienne.

Les travaux sur les effets centre-périphérie sont en cours. La figure 4.16 illustrent les effets d'inhibition iso-périphérique obtenus dans le module d'extraction locale, avec mise en valeur du contour horizontal isolé et élimination des autres. L'ensemble des résultats obtenus jusque-là montre essentiellement une diminution du bruit dans la détection opérée. Les résultats liés à l'obtention des autres effets centre-périphérie sont beaucoup plus mitigés.

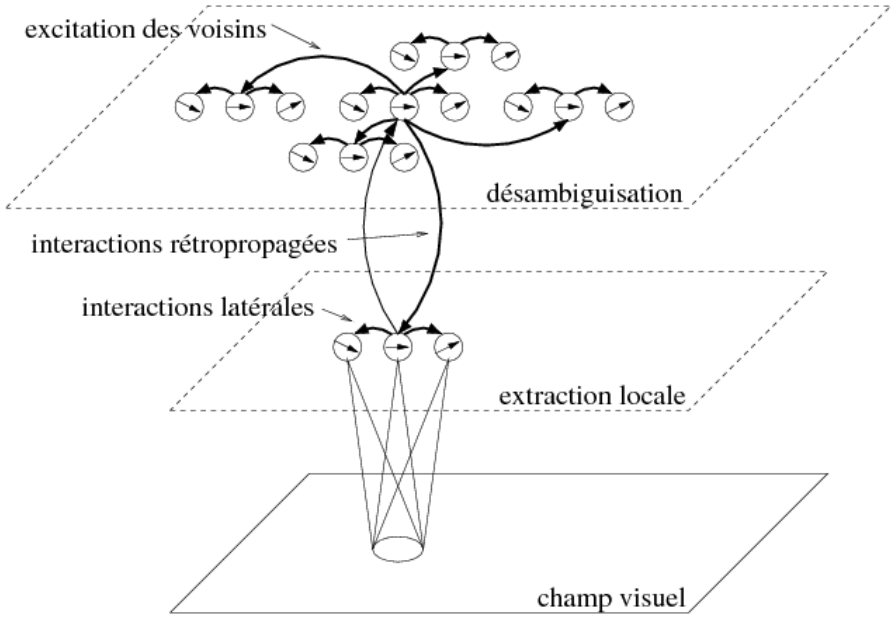


FIG. 4.17 – Désambiguïsation par interactions rétropropagées : modèle.

4.3.2 Désambiguïsation améliorée et stabilisation de la détection de mouvement

La nature locale des traitements opérés par les neurones sélectifs à la vitesse dans des aires corticales telles que V1 constitue depuis longtemps un argument supplémentaire en faveur des modèles simplement propagés, dans lesquels les aires de haut niveau du chemin visuel devraient être responsable de la résolution du problème d'ouverture, principalement parce que ces aires du cortex visuel traitent des zones plus larges du champ visuel. Néanmoins, même en considérant que les aires de plus haut niveau (comme MT) sont le lieu de la résolution du problème d'ouverture par désambiguïsation des mouvements détectés localement, cela n'indique pas comment cette résolution s'opère précisément. Dans [BN04], les auteurs proposent l'utilisation de connexions rétropropagées de MT vers V1 pour permettre cette désambiguïsation. Sur la base de ces travaux, nous avons proposé l'introduction de telles interactions dans notre modèle de perception visuelle du mouvement. Les premiers résultats indiquent une résolution satisfaisante du problème d'ouverture dans des séquences d'images artificielles, ainsi qu'une diffusion stabilisée des informations de mouvement autour des contours des objets, permettant ainsi le maintien plus aisé de bulles d'activité neuronales de la taille des objets en mouvement détectés.

Les modifications d'architectures ainsi proposées sont (cf figure 4.17) :

- introduction de connexions excitatrices latérales dans le module d'extraction locale du flux optique entre des neurones sélectifs à des orientations voisines associés à un même champ récepteur,
- introduction de connexions excitatrices multiplicatives rétropropagées du module de désambiguïsation vers le modules d'extraction locale, entre neurones associés à la même orientation, la même vitesse et le même champ récepteur,
- choix des poids des connexions latérales du module de désambiguïsation de manière à renforcer l'excitation des neurones topologiquement voisins.

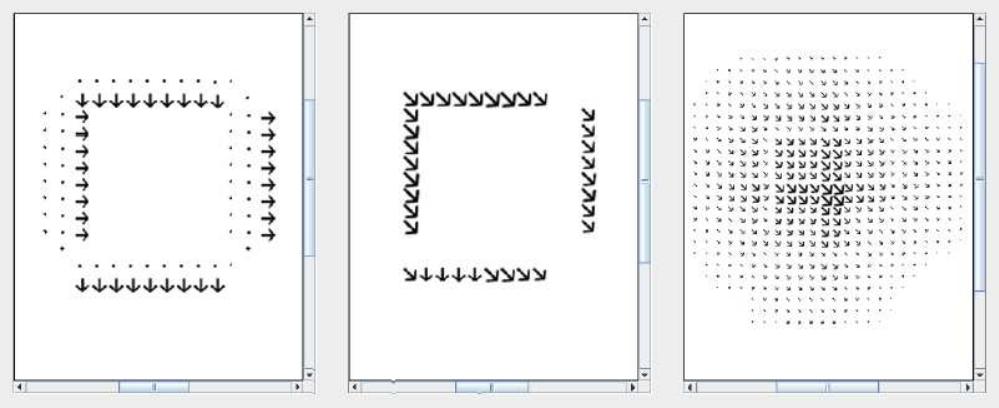


FIG. 4.18 – Désambiguïsation par interactions rétropropagées : résultats.

La figure 4.18 montre sur un exemple artificiel (carré en mouvement) le mouvement désambiguïsé obtenu. De gauche à droite, on trouve le flux optique extrait sans interactions rétropropagées, puis le flux optique extrait avec interactions rétropropagées (après 16 itérations de la boucle de rétropropagation), puis la bulle de mouvement désambiguïsé associée (après ces mêmes 16 itérations). Les travaux se poursuivent sur ces interactions rétropropagées afin de résoudre les problèmes de normalisation et de synchronisation observés.

4.4 Perception du mouvement propre

L'étude de la perception visuelle du mouvement propre peut être selon l'information qu'on cherche à établir : existence d'un déplacement propre, direction du déplacement, gestion des obstacles. Des champs de recherche distincts s'occupent de ces différentes questions : un champ de recherche sur les indices disponibles qui caractérisent le mouvement propre, un champ consacré à la perception de la direction de déplacement (heading), et un champ consacré aux aspects de prédiction du temps de contact avec un obstacle.

Compte tenu de la complexité des traitements corticaux impliqués dans la perception du mouvement propre, et évoqués au chapitre précédent, nos premiers travaux se sont limités à la perception purement visuelle du mouvement propre, et plus précisément à l'extraction visuelle de la direction de déplacement.

4.4.1 Extraction visuelle du mouvement propre

Le mouvement visuel peut être provoqué soit par le mouvement du sujet, soit par un mouvement externe au sujet, ou encore les deux à la fois. Le but de l'extraction du mouvement propre est la ségrégation du flux optique en deux composantes relatives au mouvement des objets d'une part, et au mouvement propre d'autre part.

Flux optique et mouvement propre

Le champ visuel contient des informations pertinentes pour percevoir un mouvement propre. Par exemple, l'ensemble des lignes reliant un objet dans l'environnement au centre de l'oeil du sujet

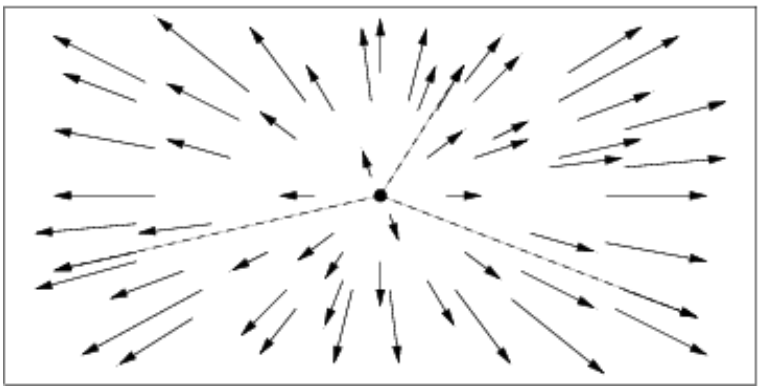


FIG. 4.19 – Représentation d'un flux optique simulant un mouvement propre vers l'avant. Au centre, le foyer d'expansion. Chaque vecteur de mouvement (lignes fléchées) est caractérisé par sa ligne de direction (ligne pointillée) le reliant au foyer d'expansion.

subit un mouvement d'expansion lorsque le sujet se déplace vers l'avant. Seule la ligne qui coïncide avec la direction de mouvement reste inchangée. Ceci génère un flux optique radial, comme illustré sur la figure 4.19. Le mouvement propre peut générer d'autres composantes de flux optiques, les flux optiques plans (translation parallèle au plan frontal) et circulaires (rotation dans le plan frontal). Les mouvements naturels comprennent souvent translation et rotation à la fois, ajoutant un degré de complexité supplémentaire au problème d'extraction des paramètres du mouvement propre à partir des informations du flux optique. La situation se complique encore dans la réalité parce que le champ visuel ne couvre pas 360°, ce qui d'une part réduit les informations disponibles, et d'autre part peut se traduire pas une confusion entre les différentes composantes du mouvement, compte tenu de l'incertitude de mesure des directions, par exemple.

Perception de la direction de déplacement

Un flux radial correspond à un mouvement propre vers l'avant (expansion) ou vers l'arrière (contraction). Il existe alors une singularité dans la distribution des vecteurs de mouvement, qui est le point unique où le vecteur de mouvement est nul, autrement dit, où la vitesse locale est nulle. C'est cette singularité du flux optique, encore appelée foyer d'expansion, qui permet de déterminer la direction de déplacement de l'observateur. Par ailleurs, le mouvement de translation a un effet sur le flux optique inversement proportionnel à la distance entre l'observateur et les objets qui génèrent ce flux.

Les mouvements des yeux et de la tête génèrent quant à eux un flux optique (rotation et translation) qui s'ajoute au flux provenant du déplacement de l'observateur, ce qui provoque un déplacement du foyer d'expansion sur la rétine, voire sa disparition. Le flux optique sur la rétine peut devenir alors très différent d'un flux radial caractérisant le mouvement propre en question. La relation directe entre la position du foyer d'expansion sur la rétine et la direction de déplacement est potentiellement rompue, ce qui complique la détermination de cette dernière à partir du flux optique. On distingue ainsi le flux optique, qui dépend uniquement du mouvement propre, et le flux rétinien.

Eléments neurophysiologiques de la perception de la direction de déplacement

La plupart des neurones de l'aire MST (ainsi que de l'aire VIP) paraissent moduler leurs réponses à un stimulus de flux optique en fonction de la position de la singularité du flux (foyer d'expansion pour les stimuli radiaux). Leur activité est ainsi renforcée avec une préférence pour la position de la singularité dans une certaine région de la rétine. En considérant l'ensemble des positions préférentielles de la singularité pour ces neurones, toute la rétine est couverte. L'interprétation de ces observations est variable. Certains biologistes soutiennent que la direction de déplacement peut être déduite à partir de la réponse d'un seul neurone. D'autres biologistes soutiennent que la direction de déplacement est codée par la population neuronale toute entière.

Il apparaît également que les neurones de MST peuvent compenser la perturbation du flux optique provoquée par les rotations de la tête et des yeux et pourraient ainsi être à la base de la décomposition du flux optique. De plus, les effets sur la perception de la direction de déplacement des objets se déplaçant indépendamment sont très réduits, ce qui suggère que les mouvements de ces objets sont largement supprimés lors de l'analyse de la direction de déplacement.

L'ensemble de ces résultats conforte le rôle de MST dans la perception du mouvement propre, et particulièrement dans la détermination de la direction de déplacement. De nombreux désaccords subsistent néanmoins entre les spécialistes (encodage de la direction de déplacement, combinaison des signaux visuels et vestibulaires dès MST, etc.).

4.4.2 Modèle pour l'extraction du mouvement propre

L'extraction du mouvement propre est une tâche de haut niveau relativement complexe. Nos premiers travaux se limitent à la détermination de la direction de déplacement, et ils tiennent compte d'hypothèses simplificatrices permettant d'éliminer les conditions évoquées précédemment dans lesquelles cette estimation est biaisée.

Nous supposons pouvoir confondre le flux optique et le flux rétinien (yeux et tête fixes par rapport au corps). La direction de déplacement est alors déterminée par le foyer d'expansion, que nous supposons présent dans le champ visuel.

Détermination de la direction de déplacement

Pour déterminer le foyer d'expansion, il faut détecter un renversement de direction ponctuel dans le flux optique. Si $V(x,y)$ est le vecteur vitesse détecté en (x,y), on définit le scalaire $\delta(x,y)$ pour chaque point (x,y) par :

$$\begin{aligned} \delta(x,y) &= \frac{\partial}{\partial x} sign(V_x) + \frac{\partial}{\partial y} sign(V_y) \\ \text{où } sign(x) &= \begin{cases} 1 & \text{si } x > 0 \\ -1 & \text{sinon} \end{cases} \end{aligned}$$

avec une interpolation simple :

$$\begin{aligned} \frac{\partial}{\partial x} sign(V_x) &\simeq sign(V_{x+1}) - sign(V_{x-1}) \\ \frac{\partial}{\partial y} sign(V_y) &\simeq sign(V_{y+1}) - sign(V_{y-1}) \end{aligned}$$

En fait, δ correspond à la divergence du champ de vecteurs (flux optique) que nous avons modifié en appliquant la fonction $sign(.)$, qui pourra être approchée avec des neurones saturés. Ainsi, δ est positif pour un flux radial d'expansion et il est négatif pour un flux de contraction. De plus, pour un flux optique radial (parfait), δ est maximal (en valeur absolue) au foyer d'expansion.

On peut noter que les travaux de [LTYS06] indiquent l'intérêt d'une vision centrée au foyer d'expansion mais exprimée en coordonnée log-polaire pour extraire ensuite les paramètres précis du mouvement prore. Ce recentrement et cette estimation précise n'est néanmoins pas encore envisagée dans nos travaux.

Insertion dans l'architecture générale

Le module de désambiguïsation fournit pour chaque champ récepteur de l'image une information de mouvement sous la forme de l'activation des neurones associés aux différentes directions et vitesses. Afin de déterminer une seule direction et une seule vitesse pour chaque champ récepteur centré en (x, y), on peut calculer une moyenne des directions et des vitesses pondérée par les activations des neurones :

$$\overrightarrow{V}(x,y) = \frac{H(x,y,\theta,v)}{\sum_{\theta\in\Theta, v\in V} H(x,y,\theta,v)} * v * \vec{v}_\theta$$

où $H(x, y, \theta, v)$ est l'activation du neurone en (x, y), de direction préférentielle θ et de vitesse préférentielle v. Θ est l'ensemble des directions, V est l'ensemble des vitesses. $\vec{v}_\theta$ est un vecteur unitaire dans la direction θ.

La connaissance du foyer d'expansion constitue la première étape pour la ségrégation du flux optique en deux composantes relatives au mouvement des objets d'une part, et au mouvement propre d'autre part. Nous concevons actuellement une architecture dans laquelle les informations extraites par le calcul ci-dessus sont prises en compte dans une carte organisant une compétition (sur le mode de la carte focus du module de suivi) afin de faire émerger le foyer d'expansion. Cette carte permet ensuite d'inhiber les neurones compatibles avec sa direction de déplacement dans une carte de détection des mouvements des objets indépendants.

4.5 Bilan

Outre de nombreux aspects traités de manière encore incomplète et très imparfaite dans ce modèle (résolution du problème d'ouverture, prise en compte du flux d'informations rétropropagées, etc.), les perspectives de ce travail sont nombreuses et trouvent leurs sources dans les nombreux aspects neurophysiologiques de la perception visuelle du mouvement que notre modèle ne prend pas en compte.

Néanmoins, une difficulté majeure est de maintenir dans ces travaux une réflexion centrée sur le caractère local et massivement distribué des modèles de calcul sous-jacents. Si l'annexe E montre le souci de confronter nos modèles à la problématique d'une pratique de calcul distribué par l'implantation sur FPGA des modules initiaux de ce modèle, il apparaît que les évolutions les plus récentes privilégient l'inspiration corticale dans un but fonctionnel et non calculatoire. L'approche défendue en 3.3.2 relève d'une volonté affirmée de replacer la pratique bio-inspirée de calcul distribué au cœur de l'ensemble de nos travaux sur les modèles à base de cartes neuronales.

Bilan et perspectives

Les réseaux de neurones constituent un domaine d'étude très théorique et dont l'évolution est indissociable de ses applications. Les aspects parallèles du calcul neuronal sont présents aussi bien dans ses propriétés théoriques que dans son intérêt pratique. Les réseaux neuronaux tirent profit d'un principe général que je désigne sous le terme de parallélisme connexionniste, qui utilise la gestion distribuée d'un flux d'informations comme puissance de calcul à part entière. Il m'apparaît nécessaire de mieux identifier ce parallélisme et de l'exploiter pleinement, dans le domaine de l'implantation des réseaux de neurones comme dans la détermination et l'étude de modèles connexionnistes d'inspiration biologique permettant de traiter des tâches cognitives complexes.

Tout au long de ce manuscrit, c'est l'idée de promouvoir une véritable pratique de calcul distribué qui guide mes travaux, avec deux principes fondamentaux. Il apparaît d'une part qu'une telle pratique ne peut pas se contenter d'une étude théorique en amont suivie de travaux d'implantations, et que l'ensemble des aspects de ce calcul distribué, des modèles jusqu'aux supports, sont interdépendants. D'autre part, la gestion du flux d'information distribué connexionniste est au cœur des difficultés comme des solutions. Par ailleurs, la bio-inspiration constitue dans ces travaux une opportunité essentielle pour que la pratique de calcul distribué proposée soit en mesure d'être confrontée à des problèmes sur lesquels les approches plus classiques ont montré leurs limites.

Les travaux décrits vont de la conception théorique de paradigmes de calcul connexionniste tolérants aux contraintes d'implantation à la définition et à l'implantation embarquée de calculs neuronaux complexes d'inspiration corticale. Ils ouvrent surtout des voies de recherche à la fois difficiles et prometteuses pour les années à venir.

Cheminement scientifique

L'ensemble de mes travaux est fondé sur la conviction selon laquelle les modèles connexionnistes ne sont pas seulement des outils de traitement de données parmi d'autres, mais constituent un mode de calcul à part qui doit nous amener à réfléchir d'une manière "naturellement" connexionniste aux problèmes à résoudre. Il ne s'agit pas de dire que le connexionnisme est fondamentalement plus puissant que d'autres modes de calcul, ou qu'il n'a pas d'équivalent (analytique, algorithmique, etc.). Il s'agit avant tout d'une façon de concevoir des solutions et de les mettre en œuvre directement sous la forme de modèles numériques massivement distribués. C'est pourquoi le parallélisme connexionniste doit être non seulement une propriété des modèles ainsi conçus, mais aussi leur essence. Le besoin de pousser cette approche du connexionnisme jusqu'à la réalisation d'implantations réellement distribuées peut être vu dans ce contexte comme la confrontation à un principe de réalité qui écarte tout recours caché à une pratique de calcul non distribué.

Mes travaux de thèse sont l'histoire d'un cheminement de cette réflexion d'un parallélisme logiciel vers un parallélisme matériel, c'est à dire vers une pratique de calcul connexionniste à

grain très fin. Mes travaux depuis la thèse, rapportés dans ce manuscrit, sont la prolongation de ce cheminement vers la nécessité d'étendre le parallélisme connexionniste à la conception de modèles pour des tâches particulièrement complexes telles que la perception et le raisonnement. L'inspiration corticale n'est pas la seule solution permettant une telle extension, mais elle est une solution qui s'appuie sur les très larges capacités observées du cortex. Dans ces travaux, l'inspiration corticale n'est pas une contrainte de plausibilité biologique, mais elle constitue la référence sur laquelle il est possible de se fonder pour proposer de manière directe des solutions connexionnistes à des problèmes complexes.

Il est tout à fait possible de développer des modèles connexionnistes d'inspiration corticale sans y voir un mode de calcul distribué particulier. La spécificité de mes recherches, et des axes de réflexion que je souhaite poursuivre, est de chercher à maintenir ce lien privilégié entre les modèles connexionnistes et les conditions d'une pratique de calcul distribué à grain très fin. Cette spécificité induit plusieurs défis scientifiques complémentaires, qu'on peut principalement diviser en deux catégories : ceux liés à l'adéquation entre calcul connexionniste et support de calcul massivement distribué, et ceux liés à la prise en compte de l'inspiration biologique dans le développement d'une pratique de calcul connexionniste.

Défis pour un calcul connexionniste embarqué

Au-delà des nombreux travaux en cours ou à venir à la suite des travaux rapportés au chapitre 2 sur les implantations parallèles matérielles des modèles connexionnistes, trois enjeux majeurs apparaissent.

Environnements dédiés

Le premier enjeu concerne la nécessité de concevoir des environnements dédiés propres à une pratique connexionniste du calcul distribué. De tels environnements sont indispensables pour s'affranchir des spécificités de chaque support matériel. Ce besoin devient d'autant plus criant à la lumière des évolutions récentes des circuits FPGA par exemple, et de la difficulté qu'il y a désormais à maîtriser toutes les possibilités offertes par les unités arithmétiques, les mémoires, et autres coprocesseurs embarqués aux côtés des ressources logiques de ces circuits. Devant la variété des technologies et des solutions à prendre en compte simultanément, on peut légitimement se demander si un tel effort de conception d'environnements peut être pris en charge dans les activités d'équipes de recherche. Néanmoins, si un travail conjoint avec des équipes de développeurs est souhaitable et probablement inévitable à long terme, de nombreux défis scientifiques préalables restent du ressort de la recherche. Ces défis scientifiques liés à la conception de tels environnements sont nombreux, de l'optimisation des choix arithmétiques à la gestion multicritère des contraintes de surface, de temps ou de consommation, en passant par la prise en compte des principaux modes de calcul neuronaux. Un des principaux défis réside dans l'optimisation automatisée de la gestion des différents niveaux de parallélisme lorsque le support ne permet pas une implantation totalement distribuée. Nos recherches sur ce sujet s'orientent notamment vers une approche à base de factorisation partielle de graphes de flots de données. A plus long terme, le défi principal sera sans doute la possibilité d'aborder directement le connexionnisme d'inspiration corticale dans ces environnements dédiés. Malgré quelques travaux préalables, notamment autour du calcul connexionniste impulsionnel, le manque de généralité des modèles et des méthodes d'implantation est pour l'instant un obstacle majeur.

Calcul embarqué asynchrone et aléatoire

Le second enjeu est lié à la nature fondamentalement asynchrone du calcul neuronal. S'il est relativement aisé de maîtriser le degré d'asynchronisme dans une implantation logicielle (au moyen de tirages aléatoires), une implantation matérielle privilégie naturellement un système totalement synchrone basé sur une horloge commune et permettant d'optimiser le gain en performance autorisé par un parallélisme massif. Une façon d'aborder cet enjeu est de privilégier des modèles tolérants à n'importe quel degré d'asynchronisme (cf 2.2.2). Mais du point de vue d'une pratique de calcul connexionniste bio-inspiré embarqué, la question de parvenir à des implantations réellement asynchrones reste posée. Elle constitue sans doute un pas essentiel à franchir pour que cette pratique de calcul connexionniste défendue dans ce manuscrit soit clairement identifiée et valorisée.

Cette question est en partie reliée au problème de la gestion distribuée de calculs de nature stochastique [NdMM03, GP07]. En effet de tels calculs peuvent, toujours sur la base synchrone des circuits numériques, permettre d'obtenir des comportements de nature asynchrone. Ce lien entre asynchronisme et calcul aléatoire se retrouve dans la définition même des modèles neuronaux massivement distribués [RV06], comme évoqué en 2.2.2, ou encore dans les enjeux généraux du calcul spatial évoqués en 1.4 et ci-dessous [LW06]. C'est peut-être ce dernier domaine de recherche qui permettra d'aboutir à des supports de calcul tolérant à la fois la distribution massive et l'asynchronisme, et pour lesquels les approches neuronales seront d'autant plus pertinentes.

Interprétation distribuée de haut niveau

Un troisième pas important à franchir pour asseoir le connexionnisme comme une pratique de calcul distribué porte sur l'écueil de l'interprétation centralisée des calculs réalisés. Il est nécessaire de montrer que les modèles connexionnistes peuvent maintenir le principe de distribution massive depuis la prise en compte des données jusqu'à l'expression de la "décision" à laquelle le traitement distribué aboutit.

Les travaux sur l'attention visuelle apportent une réponse partielle, puisque c'est la totalité de l'activité de la carte de focalisation qui représente l'interprétation obtenue de la scène visuelle. Il reste néanmoins nécessaire de mieux cerner ce que peut être une interprétation ou décision distribuée, afin d'en systématiser l'usage. Ce souci rejoint le besoin de définir des abstractions spatiales valides pour différents modes de calcul spatial (ou massivement distribué) afin d'en formaliser l'usage algorithmique. De plus, les aspects temporels doivent être pris en compte dans une telle notion. C'est à nouveau le problème de la perception visuelle du mouvement qui peut offrir le cadre d'une première approche de cette étude. Au-delà des nombreux travaux qui restent à mener pour améliorer et étendre les capacités de notre modèle, une des perspectives importantes est d'aboutir à l'extraction de motifs spatio-temporels de mouvement. Ces motifs, naturellement distribués, ont pour vocation d'associer une sémantique unique à différentes séquences de mouvement similaires, et peuvent être étendus à l'identification des propriétés fonctionnelles des objets environnants par exemple. L'inspiration corticale constitue là encore une aide essentielle pour aborder une problématique aussi difficile.

Défis pour une pratique de calcul connexionniste bio-inspirée

Si les défis sont donc encore nombreux pour confirmer le connexionnisme comme une pratique de calcul massivement distribué, d'autres défis apparaissent lorsqu'il s'agit plus particulièrement

de prendre en compte l'influence de l'inspiration biologique, notamment corticale, sur la nature des calculs et des architectures au cœur de cette pratique. Il est utile de rappeler ici que la démarche de ce manuscrit, bien qu'orientée vers une inspiration corticale, ne tend pas à la compréhension et à la modélisation du cerveau. Il s'agit d'identifier les apports possibles de cette inspiration dans l'extension de la pratique de calcul connexionniste défendue vers des problèmes de nature cognitive particulièrement complexes. Ces apports doivent donc être confrontés aux contraintes spécifiques d'une pratique véritablement distribuée de calcul.

Flux d'informations massivement récurrents et entrelacés

Les travaux rapportés dans les chapitres 3 et 4, ainsi qu'en annexe E indiquent que les difficultés rencontrées dans la définition et dans l'implantation massivement distribuée de modèles d'inspiration corticale ne résident que partiellement dans la taille de ces modèles. C'est bien à nouveau dans la complexité du flux d'informations que se trouvent les difficultés principales, et notamment dans la prééminence des connexions latérales et rétropropagées au sein de calculs massivement récurrents. Les réponses apportées en ce qui concerne la gestion des connexions latérales locales (par les travaux de l'annexe D notamment) restent très largement insuffisantes quand des flux tels que ceux du modèle de perception visuelle du mouvement sont en jeu. Les sections 4.3.1 et 4.3.2 confirment l'importance des flux rétropropagés, en liaison directe avec les flux latéraux intenses au cœur des champs neuronaux dynamiques. Le défi est donc majeur, et ne peut pas se contenter de solutions spécifiques à tel ou tel modèle.

Auto-organisation et autonomie

A long terme, l'inspiration corticale est non seulement susceptible de fournir les modes de calcul et les architectures générales des modèles recherchés, mais aussi des moyens pour parvenir à apprendre et auto-organiser ces modèles pour aller au-delà de la définition de modèles spécifiques à une tâche donnée. Ce besoin d'auto-organisation est présent dans l'ensemble des approches du calcul spatial [Zam04]. Là encore, la spécificité de mes recherches est de placer les contraintes d'une pratique de calcul massivement distribué au cœur de la réflexion sur l'auto-organisation des champs neuronaux dynamiques.

Ainsi, au cœur des recherches à venir, la définition de modules connexionnistes autonomes extensifs capables de s'auto-organiser une fois assemblés (cf 3.3.2) peut être vue comme la fédération des différents objectifs de mes recherches, car de tels modules constituent une véritable pratique de calcul distribué, dans laquelle la définition du modèle adapté à un problème donné fournit directement son implantation matérielle. Les défis posés par un tel projet sont nombreux et particulièrement ambitieux, et dépassent largement la problématique de la gestion distribuée bidimensionnelle des flux d'informations évoquée en 3.3.2. La maîtrise des champs neuronaux dynamiques, leur extension à des calculs neuronaux plus complexes, incluant possiblement une part temporelle (calculs impulsionnels), la définition d'un apprentissage piloté par les récompenses issues de la boucle de perception-action, ou encore la question de la détermination des interactions globales entres cartes, sont tout autant de sujets de recherche interdépendants qui expliquent que ce projet se situe à l'échelle d'une collaboration intense entre différents chercheurs de l'équipe.

Calcul spatial bio-inspiré

Du point de vue de la capacité des modules évoqués ci-dessus à être physiquement assemblables, ce projet s'inscrit plus globalement dans la volonté de définir une forme possible de calcul spatial bio-inspiré. Si les évolutions récentes des supports matériels de calcul distribué incitent à être prudents sur l'effort nécessaire pour poursuivre les travaux d'implantation connexionniste, les tendances actuelles de la conception de circuits s'orientent de nouveau vers les fondements connexionnistes, notamment bio-inspirés, pour gérer la multiplicité et l'hétérogénéité des unités de calculs présentes sur les futurs circuits, comme évoqué en 2.3.2. C'est l'émergence et la maîtrise de comportements collectifs cohérents qui constituent l'enjeu de ces recherches.

Sans être au cœur de mes recherches, cette problématique recouvre de nombreux points communs avec la notion de pratique de calcul distribué d'inspiration corticale défendue dans ce manuscrit (localité des calculs, asynchronisme, décision distribuée, etc.). Néanmoins l'inspiration corticale n'est pas la seule piste possible pour une pratique de calcul distribué. D'autres modèles naturellement distribués existent. Mais l'inspiration biologique dans son ensemble reste une voie privilégiée pour maîtriser le passage à des tâches de grande complexité et pour maîtriser l'émergence des comportements globaux issus des interactions locales. Des recherches ont ainsi été entamées en collaboration avec des chercheurs de deux autres équipes sur la définition de paradigmes de calcul bio-inspirés qui se concentrent sur la capacité de certains systèmes biologiques à faire émerger une coordination de type multicellulaire parmi des entités individuelles. Ce processus est un exemple de réponse auto-organisée par émergence d'un consensus parmi les cellules individuelles sur leurs conditions environnementales. Cette étude inclut le développement d'implantations matérielles distribuées, mais ce développement ne se réduit pas à des problèmes technologiques. La question fondamentale est à nouveau de valider nos modèles comme pratique de calcul distribué numérique. Il s'agit donc d'une approche de conception conjointe de l'implantation et du modèle, dans laquelle c'est à nouveau la compréhension et la gestion du flux d'informations massivement distribué qui focalise mes recherches.

Annexe A

Résumé des travaux sur les FPNA

Cette annexe est extraite de [Gir06a, Gir06b]. Pour une introduction sur les motivations et les principaux apports de ce travail, se reporter à la section 2.2.1.

[...]

A.1 FPNAs, FPNNs

This section describes the FPNA/FPNN concept. These models appear as parameterized task graphs that are specialized so as to perform neural computations, but they differ from standard neural models that are graphs of non-linear regressors.

The distinction between FPNAs and FPNNs is mainly linked to implementation properties. An FPNA is a given set of neural resources that are organized according to specific neighbourhood relations. An FPNN is a way to use this set of resources : it only requires local configuration choices. Therefore, the implementations of two different FPNNs exactly require the same mapping on FPGA, as long as they are based on the same FPNA.

A.1.1 FPNAs

From FPGAs to FPNAs

The first aim of the FPNA concept is to develop neural structures that are easy to map directly onto digital hardware, thanks to a simplified and flexible topology. The structure of an FPNA is based on FPGA principles : complex functions realized by means of a set of simple programmable resources. The nature and the relations of these FPNA resources are derived from the mathematical processing FPNAs have to perform.

To summarize, in a standard neural model, each neuron computes a function applied to a weighted sum of its inputs : if $\vec{x_i}$ is the input vector of neuron i, and $\vec{w_i}$ is its weight vector, it computes $f_i(\vec{w_i}.\vec{x_i})$. See [GGR95a, GGR95b] for a unified theoretical approach of the computation of standard neural networks. The input vector $\vec{x_i}$ may contain neural network inputs as well as outputs of other neurons, depending of the *connection graph* of the neural network. In such a standard model, each link is a connection between the output of a neuron and an input of another neuron. Therefore the number of inputs of each neuron is its fan-in in the connection graph. On

the contrary, neural resources become *autonomous* in an FPNA : their dependencies are freely set, and the resulting processing is more complex than in standard neural models. As in FPGAs, FPNA configuration handles both resource interconnections and resource functionalities.

FPNA resources

An FPNA is a programmable set of neural resources that are defined to compute partial convolutions for non-linear regression, as standard multilayer neural networks do. Two kinds of autonomous FPNA resources naturally appear : *activators* that apply standard neural functions to a set of input values on one hand, and *links* that behave as independent affine operators on the other hand.

These resources might be handled in different ways. The easiest scheme would allocate any link to any activator, with the help of an underlying programmable interconnection network. This would lead to massively pruned standard neural networks, or to multiple weight sharing connections. Topological problems would still appear (such as high fan-ins), and weight sharing would lead to few different $\vec{w_i}$ weight vectors. Therefore, another principle has been chosen for FPNAs : any link may be *connected* to any *local resource*. The aim of locality is to reduce topological problems, whereas connected links result in more various weight vectors.

More precisely, the links connect the nodes of a directed graph, each node contains one activator. The specificity of FPNAs is that the relations between *any* local resources of each node may be freely set. A link may be connected or not to the local activator *and to other local links*. Direct connections between affine links appear, so that the FPNA computes numerous composite affine transforms. These compositions create numerous *virtual neural connections*, so that different convolution terms may be obtained with a reduced number of connection weights.

Definition of FPNAs

The following definition specifies the structure of an FPNA (directed graph), as well as the functional nature of each individual neural resource.

An FPNA is defined by means of :

- A directed graph $(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is an ordered finite set of nodes, and $\mathcal{E}$ is a set of directed edges without loop. $\mathcal{E}$ may be seen as a subset of $\mathcal{N}^2$.
  For each node n, the set of the direct predecessors (resp. successors) of n is defined by $Pred(n) = \{p \in \mathcal{N} \mid (p, n) \in \mathcal{E}\}$ (resp. $Succ(n) = \{s \in \mathcal{N} \mid (n, s) \in \mathcal{E}\}$). The set of the input nodes is $\mathcal{N}_i = \{n \in \mathcal{N} \mid Pred(n) = \emptyset\}$.
- A set of affine operators $\alpha_{(p,n)}$ for each (p, n) in $\mathcal{E}$.
- A set of activators (i_n, f_n), for each n in $\mathcal{N} - \mathcal{N}_i$: i_n is an iteration operator (a function from $\mathbb{R}^2$ to $\mathbb{R}$), and f_n is an activation function (from $\mathbb{R}$ to $\mathbb{R}$).

To simplify, (p, n) and (n) now stand for the corresponding links and activators.

Interpretation

Resources are associated with the nodes, whereas locality is defined by the edges. For each node $n \in \mathcal{N}$, there is one activator and as many communication links as this node has got predecessors. Each link is associated with an affine operator. An activator is defined by (i_n, f_n), since it will handle any neuron computation as in a sequential program. Indeed, any standard neuron computation may be performed by means of a loop that updates a variable with respect to the neuron inputs, and a final computation that maps this variable to the neuron output. The iteration function i_n stands for

the updating function inside the loop. The neuron output is finally computed with f_n. See [Gir99] for the definition of (i_n, f_n) so as to obtain most standard neurons.

When the FPNA graph and the different operators have been defined, a general implementation can be given : each resource corresponds to a basic block, and these blocks are organized according to the graph. This implementation may be used by any FPNN derived from this FPNA. Some of these FPNNs compute very complex functions (equivalent to standard neural networks), though the FPNA graph is made simple (reduced number of edges, limited node fan-ins and fan-outs, so that the FPNA is easily mapped by the compiler onto the hardware device).

A.1.2 FPNNs

An FPNN (field programmed neural network) is a configured FPNA : an FPNA whose resources have been connected in a specific way (furthermore, some parameters must be given to specify the computation of each resource).

In other words, the underlying FPNA of an FPNN is a topologically fully specified architecture of neural resources, whereas this FPNN is a given way to specify how these resources will interact to define a functional behaviour.

Definition of FPNNs

An FPNA configuration requires local connections (from link to link, link to activator or activator to link) as well as precisions about the iterative process performed by each activator (initial value, number of iterations). Therefore an FPNN is specified by :

- an FPNA (available neural resources),
- for each node n in $\mathcal{N} - \mathcal{N}_i$,
  - a real value θ_n (initial value of the variable updated by iteration function i_n)
  - a positive integer a_n (number of iterations before an activator applies its activation function)
  - for each p in $Pred(n)$, two real value $W_n(p)$ and $T_n(p)$ (coefficients of affine operator $\alpha_{(p,n)}(x) = W_n(p)x + T_n(p)$),
  - for each p in $Pred(n)$, a binary value $r_n(p)$ (set to 1 *iff* link (p, n) and activator (n) are connected),
  - for each s in $Succ(n)$, a binary value $S_n(s)$ (set to 1 *iff* activator (n) and link (n, s) are connected),
  - for each p in $Pred(n)$ and each s in $Succ(n)$, a binary value $R_n(p, s)$ (set to 1 *iff* links (p, n) and (n, s) are connected),
- for each input node n in $\mathcal{N}_i$,
  - a positive integer c_n (number of inputs sent to this node),
  - for each s in $Succ(n)$, a binary value $S_n(s)$ (see above).

Computing in an FPNN

Several computation methods have been defined for the FPNNs in [Gir99]. Their common principles may be described as follows :

- All resources behave independently.
- A resource receives values. For each value,
  - the resource applies its local operator(s),

– the result is sent to all neighbouring resources to which the resource is locally connected (an activator waits for a_n values before sending any result to its neighbours).

The main differences with the standard neural computation are :

– A resource may or may not be connected to a neighbouring resource. These local connections are set by the $r_n(p)$, $S_n(s)$ and $R_n(p,s)$ values.
– A link may directly send values to other communication links.
– A resource (even a link) may handle several values during a single FPNN computation process.

A sequential version of the most general FPNN computing method [has been defined]. This method clearly illustrates the above principles. Moreover, it stands as a reference computation scheme when establishing theoretical properties.

A parallel version is described [below]. This computation scheme is aimed at being directly implemented onto FPGAs.

A.1.3 Asynchronous parallel computation

The asynchronous parallel computation scheme illustrates best the resource autonomy. It is based on an asynchronous local handling of requests : there is no global list of requests. A request $req[\varrho_1, \varrho_2, x]$ is created when a value x is exchanged by two connected resources ϱ_1 and ϱ_2. Therefore, this parallel computation is performed at resource-level instead of node-level. The same local computations are performed as in the sequential algorithm, except that resources behave independantly, and this behaviour depends on the type of resource (link or activator).

The corresponding algorithm is as follows :

Initialization **:**

– For each input node n in $\mathcal{N}_i$, c_n values $\left(x_n^{(i)}\right)_{i=1..c_n}$ are given (FPNN inputs), and the corresponding requests
$req[(n),(n,s),x_n^{(i)}]$ are created for all s in $Succ(n)$ such that $S_n(s)=1$.
– Each node n in $\mathcal{N}-\mathcal{N}_i$ has got a local counter c_n and a local variable x_n, initially set as $c_n=0$ and $x_n=\theta_n$.

Parallel processing **:**

All resources work concurrently. Each one of them sequentially handles all requests it receives. Resource ϱ_2 chooses a request $req[\varrho_1,\varrho_2,x]$ among the unprocessed requests that have already been received (with a fair choice policy). This request is processed by ϱ_2 as follows :

1. an acknowledgement is sent to ϱ_1
2. if ϱ_2 is activator (n)
   – c_n and x_n are updated : $c_n = c_n + 1$, $x_n = i_n(x_n, x)$
   – if $c_n = a_n$ (the local activator computes its output)
     – for all s in $Succ(n)$ such that $S_n(s)=1$,
       create $req[(n),(n,s),f_n(x_n)]$
     – wait for all acknowledgements
     – reset : $c_n = 0 \quad x_n = \theta_n$
   else (resource ϱ_2 is link (p,n))
   – compute $x' = W_n(p)x + T_n(p)$
   – for all $s \in Succ(n)$ such that $R_n(p,s)=1$,
     create $req[(p,n),(n,s),x']$

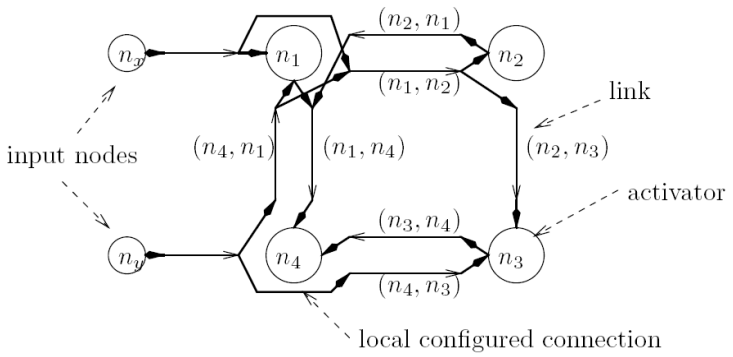


FIG. A.1 – FPNN ψ (resources and local connections)

– if $r_n(p) = 1$ then create $req[(p, n), (n, n), x']$
– wait for all acknowledgements

A general implementation architecture has been directly derived from this parallel computation. [Moreover, this computation has been extended so as to correctly handle recurrent FPNNs.]

Example

The following example is a detailed description of a whole parallel computation process. It illustrates how requests are handled at a resource-level that is purely local and that does not require any global scheduling. Most of all it shows that the FPNN computation paradigm permits a given set of neural resources to behave as a standard neural network with a very different architecture (here a multilayer perceptron).

Let Ψ be the FPNA defined by :
– $\mathcal{N} = (n_x, n_y, n_1, n_2, n_3, n_4)$
– $\mathcal{E} = \{(n_x, n_1), (n_y, n_4), (n_1, n_2), (n_2, n_1), (n_1, n_4), (n_4, n_1), (n_4, n_3)$
$, (n_3, n_4), (n_2, n_3)\}$
– $i_{n_1} = i_{n_2} = i_{n_3} = i_{n_4} = ((x, x') \mapsto x + x')$
– $f_{n_1} = f_{n_2} = f_{n_3} = f_{n_4} = (x \mapsto \tanh(x))$

Figure A.1 shows the activators, the links and the configured local connections of a Ψ-derived FPNN ψ. The binary parameters of ψ are equal to 0, except :
– for input nodes n_x and n_y : $S_{n_x}(n_1)$, $S_{n_y}(n_2)$,
– for node n_1 : $r_{n_1}(n_x)$, $r_{n_1}(n_4)$, $S_{n_1}(n_4)$, $R_{n_1}(n_x, n_2)$, $R_{n_1}(n_2, n_4)$, $R_{n_1}(n_4, n_2)$,
– for node n_2 : $r_{n_2}(n_1)$, $S_{n_2}(n_1)$, $R_{n_2}(n_1, n_3)$,
– for node n_3 : $r_{n_3}(n_2)$, $r_{n_3}(n_4)$, $S_{n_3}(n_4)$,
– for node n_4 : $r_{n_4}(n_1)$, $r_{n_4}(n_3)$, $R_{n_4}(n_y, n_1)$, $R_{n_4}(n_y, n_3)$.

Moreover : $a_1 = 2$, $a_2 = 2$, $a_3 = 3$, $a_4 = 3$, and $c_{n_x} = c_{n_y} = 1$.

The asynchronous parallel computation method may apply to ψ in *different ways* that depend on the scheduling policy of each neural resource. Figure A.2 sketches the nine steps of a possible parallel computation. Processing resources are grey filled. Processed requests are easily identified thanks to the configured connection (dark grey thick arrows) between their sender and their receiver.

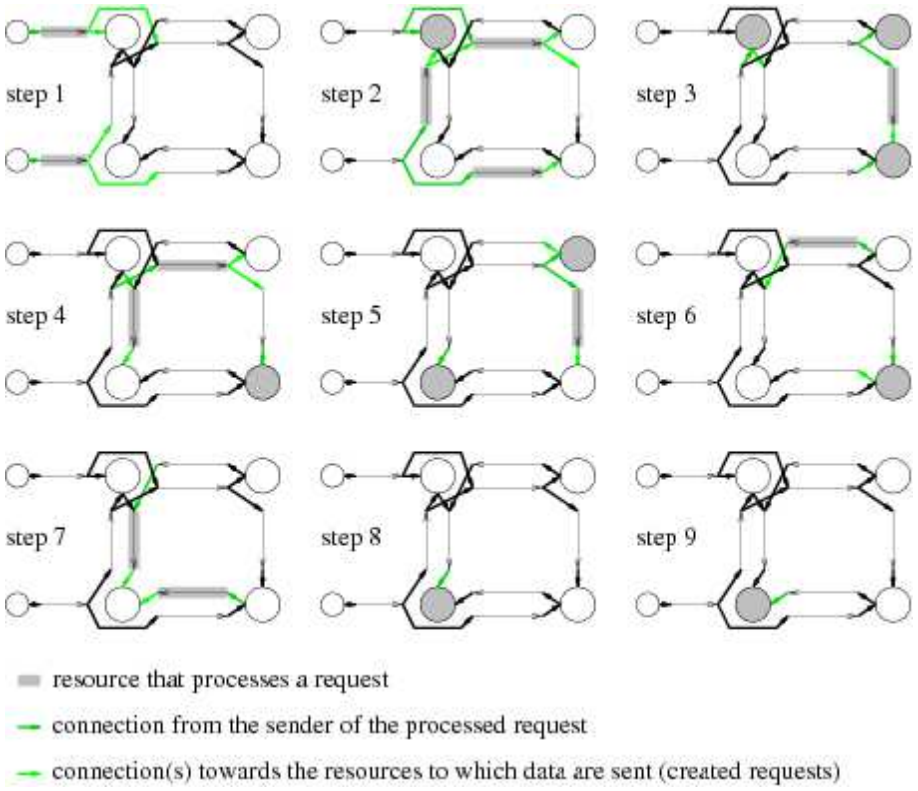


FIG. A.2 – A parallel computation of FPNN ψ

The number of created requests becomes apparent with the corresponding configured connections (light grey thick arrows). The main aspects of this parallel processing are :

***Initialization* :** FPNN inputs are assigned to input nodes : $x_{n_x}^{(1)} = x$, $x_{n_y}^{(1)} = y$. The initial set of requests is

$$\{req[(n_x), (n_x, n_1), x], req[(n_y), (n_y, n_4), y]\}.$$

Moreover $\forall\ i \in \{1, 2, 3, 4\}\ \ c_{n_i} = 0$ and $x_{n_i} = \theta_i$.

***Parallel processing progress* :**

1. Resources (n_x, n_1) and (n_y, n_4) work concurrently, so that both initial requests are simultaneously processed :
   - Request $req[(n_x), (n_x, n_1), x]$ is processed : an acknowledgement is sent to (n_x), request
     $$req[(n_x, n_1), (n_1, n_2), \alpha_{(n_x,n_1)}(x)]$$
     is created since $R_{n_1}(n_x, n_2) = 1$, and $\alpha_{(n_x,n_1)}(x)$ is also sent to activator (n_1) since $r_{n_1}(n_x) = 1$, i.e.
     $req[(n_x, n_1), (n_1), \alpha_{(n_x,n_1)}(x)]$ is created.
   - Request $req[(n_y), (n_y, n_4), y]$ is processed : an acknowledgement is sent to (n_y), requests
     $$req[(n_y, n_4), (n_4, n_1), \alpha_{(n_y,n_4)}(y)]$$
     $$\text{and } req[(n_y, n_4), (n_4, n_3), \alpha_{(n_y,n_4)}(y)]$$
     are created since $R_{n_4}(n_y, n_1) = R_{n_4}(n_y, n_3) = 1$.

2. Resources (n_1), (n_1, n_2), (n_4, n_1) and (n_4, n_3) work concurrently to process all available requests. Links proceed as above. Activator (n_1) processes $req[(n_x, n_1), (n_1), \alpha_{(n_x,n_1)}(x)]$ as follows :
   – An acknowledgement is sent to (n_x, n_1). Local updates are : $c_{n_1} = 1$, $x_{n_1} = \theta_1 + \alpha_{(n_x,n_1)}(x)$. No request is created, since $c_{n_1} < a_{n_1}$.
3. Resources (n_1), (n_2), (n_2, n_3) and (n_3) work concurrently. Link (n_2, n_3) and activators (n_2) and (n_3) proceed as above, whereas :
   – $req[(n_4, n_1), (n_1), \alpha_{(n_4,n_1)}(\alpha_{(n_y,n_4)}(y))]$ is processed : an acknowledgement is sent to (n_4, n_1). Local updates are : $c_{n_1} = 2$, $x_{n_1} = \theta_1 + \alpha_{(n_x,n_1)}(x) + \alpha_{(n_4,n_1)}(\alpha_{(n_y,n_4)}(y))$. Now $c_{n_1} = a_{n_1}$, so that $req[(n_1), (n_1, n_4), tanh(x_{n_1})]$ is created since $S_{n_1}(n_4) = 1$. Then $c_{n_1} = 0$ and $x_{n_1} = \theta_1$.

   $req[(n_4, n_1), (n_1, n_2), \alpha_{(n_4,n_1)}(\alpha_{(n_y,n_4)}(y))]$ is another available request, but resource (n_1, n_2) still waits for the acknowledgement of the two requests that it created at step 2.
4. Resource (n_1, n_2) is now able to process its waiting request. Resources (n_1, n_4) and (n_3) also work concurrently.

   ...
8. Activator (n_4) chooses a request among the two ones it has received.
9. Activator (n_4) processes the last request, so that $c_{n_4} = a_{n_4}$. Therefore (n_4) computes its output, but no request is created, since all $S_{n_4}(.)$ binary values are '0'. This output is

$$\begin{aligned}
\tanh\big(\theta_4 &+ \alpha_{(n_1,n_4)}(\tanh(\theta_1 + \alpha_{(n_x,n_1)}(x) + \alpha_{(n_4,n_1)}(\alpha_{(n_y,n_4)}(y)))) \\
&+ \alpha_{(n_1,n_4)}(\alpha_{(n_2,n_1)}(\tanh(\theta_2 + \alpha_{(n_1,n_2)}(\alpha_{(n_x,n_1)}(x)) \\
&\qquad + \alpha_{(n_1,n_2)}(\alpha_{(n_4,n_1)}(\alpha_{(n_y,n_4)}(y)))))) \\
&+ \alpha_{(n_3,n_4)}(\tanh(\theta_3 + \alpha_{(n_2,n_3)}(\alpha_{(n_1,n_2)}(\alpha_{(n_x,n_1)}(x))) \\
&\qquad + \alpha_{(n_2,n_3)}(\alpha_{(n_1,n_2)}(\alpha_{(n_4,n_1)}(\alpha_{(n_y,n_4)}(y)))) \\
&\qquad + \alpha_{(n_4,n_3)}(\alpha_{(n_y,n_4)}(y)))))\big)
\end{aligned}$$

The above result is *exactly* the same as with the MLP in figure A.3, provided that :
– all coefficients $T_n(p)$ are equal to 0,
– the compositions of link weights (virtual synaptic weights) are equal to the weights of [the MLP].

As mentioned above, this example shows that the FPNA computation paradigm permits a given set of neural resources to behave as a standard neural network with a very different architecture. Nevertheless, this FPNN is not useful : the architecture of figure A.1 does not simplify the MLP architecture of figure A.3. Indeed, this MLP is so simple that it is neither possible nor necessary to expect any architecture simplification. Some examples of significant simplifications of larger neural networks are described [in [Gir06b]].

A.2 Computational power of FPNNs

The aim of this section is to study the computational power of FPNNs. [The conditions for the correctness and for the equivalence of all FPNN computation methods have been thoroughly studied in [Gir99].]

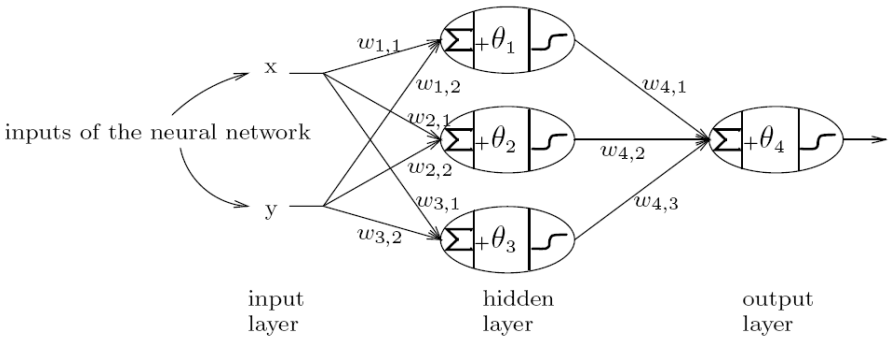


FIG. A.3 – A multilayer perceptron

As shown in [Gir99], it is *always* possible to define an FPNN that exactly computes the same function as any standard neural network. But this FPNN may be as complex as the desired neural network. Therefore, it should be known which functions may be computed by *hardware-friendly* FPNNs. This question leads to the idea of *underparameterized convolutions*.

A.2.1 Underparameterized partial convolutions

Feedforward neural models are often justified by their approximation capabilities as non-linear regression tools, since [Fun89, HSW89, Hor91]. This regression is performed by means of discrete frequential convolutions based on neuron transfer functions.

[...]

FPNNs are also defined to compute such partial convolutions for non-linear regression, though their architecture is simplified with respect to standard neural models. The FPNN definition permits to connect any link to any *local resource*. Locality permits simplified topologies, whereas connected links result in more various frequency vectors : direct connections between affine links appear, so that the FPNN may compute numerous composite affine transforms (*virtual neural connections*). Therefore different convolution terms may be obtained with a reduced number of connection weights (underparameterization).

Though various, the weights of the virtual connections are not independent of each other. The complex induced dependencies are studied in [Gir99]. They imply that the computational power of an FPNN with k neural resources and K virtual connections is lying between the computational power of a standard neural network with k neural operators and the computational power of a standard neural network with $k+K$ neural operators. Next subsection illustrates this intermediate situation through two opposite results : a first example shows that there are FPNNs for which the computational power takes advantage of all virtual connections, whereas theorem 1 shows that there are FPNNs that are strictly less powerful than the corresponding standard neural model with $k+K$ operators.

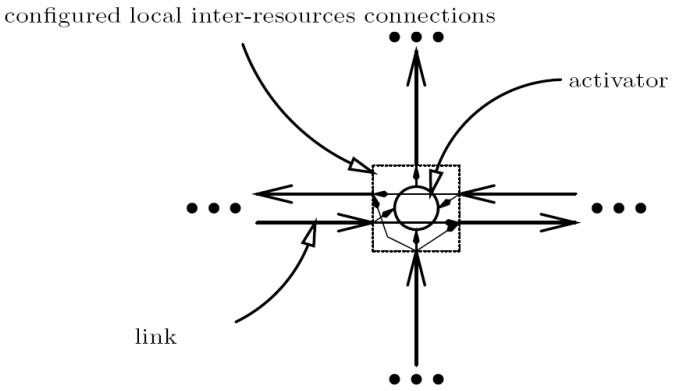


FIG. A.4 – Node of a grid-based layered FPNN

A.2.2 Underparameterized exact computing

Towards hardware-targetted simplified topologies : example

FPNNs make it possible to obtain complex neural network behaviours with simple 2D topologies. Such FPNNs have been studied for the standard parity problem : the d-dimensional parity problem consists in classifying vectors of $\{0,1\}^d$ as *odd* or *even*, according to the number of non zero values among the d coordinates.

This problem may be solved by d-input multilayer perceptron (MLP) or shortcut perceptron. The search for *optimal* two-hidden layer shortcut perceptrons in [SRK91] has solved the d-dimensional parity problem with only $\sqrt{d}(2+o(1))$ neurons[1]. This neural network uses $d(\sqrt{d}+1+o(1))$ weights. For all d, an FPNN with the same number of neurons (activators), but only $\frac{15}{2}\sqrt{d}+o(\sqrt{d})$ weights *exactly* performs the same computation.

Grid-based layered FPNNs : weight dependencies

A FPNA graph may use full-duplex links with a grid topology. Let λ_n be the index of the row of node n. A *layered* FPNN derived from such an FPNA is an FPNN where :

- $r_n(p) = 1$ *iff* $\lambda_p \leq \lambda_n$
- $R_n(p,s) = 1$ *iff* $\lambda_p \leq \lambda_n$ and $\lambda_n = \lambda_s$
- $S_n(s) = 1$ *iff* $\lambda_n < \lambda_s$

Figure A.4 shows the local structure of such a *grid-based layered FPNN*.

The virtual connections of a layered FPNN are such that consecutive rows are *virtually fully connected* as in a multilayer perceptron. But there are strong dependencies between the weights of different virtual connections.

Property 1 *Let $n_1, \ldots, n_d$ be the nodes of a single row. Let p_i be the predecessor of n_i in the previous row ($\lambda_{p_i} = \lambda_{n_i} - 1$). Let y_{n_i} (resp. y_{p_i}) be the output computed by the neuron of node n_i (resp. p_i).*

[1] For some functions f and g, $f = o(g)$ means that $\frac{f(x)}{g(x)} \overset{x \to +\infty}{\longrightarrow} 0$

Computing virtual connection weights shows that if $1 \leq i < j < k \leq d$ *then :*

$$\frac{\partial y_{n_i}}{\partial y_{p_k}} = \frac{W_{n_k}(p_k)}{W_{n_j}(p_j)} \prod_{k=j}^{k-1} W_{n_k}(n_{k+1}) \left(\frac{\partial y_{n_i}}{\partial y_{p_j}} \right)$$

Therefore, if all the FPNN weights are fixed, then for all i, $1 \leq i < d$ *and for all real values* $y_{p_1}, \ldots, y_{p_i}$, *the function* $(y_{p_{i+1}}, \ldots, y_{p_d}) \mapsto (y_{n_1}, \ldots, y_{n_i})$ *defines an arc from* $\mathbb{R}^{d-i}$ *into* $\mathbb{R}^i$.

This result is an example of the strong dependencies that may exist between the different points where an FPNN virtually estimates the terms of a partial convolution (each y_{n_i} is a convolution term computed at $(y_{p_1}, \ldots, y_{p_d})$). [Applications in [Gir06b]] show in what way these dependencies influence FPNNs, and how this unfavourable result has led to the definition of a specific way to set FPNN weights.

Shortcut FPNNs

Standard neural models may be used as FPNAs (so that each neuron and its input and output connections become freely connectable). And any standard neural network can be exactly simulated by an FPNN based on the FPNA form of this neural network. For example, one can use the multilayer topology of a one hidden-layer MLP to build an FPNA, and then set a derived FPNN that computes the same function as the original MLP. To obtain this, there must be no direct configured connection between two different links, that is $R_n(p, s) = 0$. If some $R_n(p, s)$ values are set to 1, virtual shortcut connections are obtained between the input and output neural layers. Such FPNNs may then be called *one-hidden layer shortcut FPNNs*. Examples can be given in which these FPNNs can solve specific tasks that no one-hidden layer MLP can solve with the same number of neurons. No conclusive result has been obtained when comparing a one-hidden layer shortcut FPNN with a one-hidden layer MLP having more hidden neurons. The only general result addresses the issue of the respective computational powers of such FPNNs and of real *one-hidden layer shortcut perceptrons* (MLP plus shortcut connections).

Exact learning of a set of N_p patterns in general position in $\mathbb{R}^{d_i}$ is studied in [EPM96] for standard one-hidden layer MLPs. An intermediate theorem of this study can be first extended to one-hidden layer shortcut perceptrons :

Property 2 *Let* $\mathcal{F}$ *be a vectorial function which maps neural network weights onto the values computed by the* d_o *output neurons when all input patterns are presented.*

Let $N_h = \dfrac{d_o(N_p - d_i)}{d_i + d_o}$. *If there are* N_h *hidden neurons in a one-hidden layer shortcut perceptron, then there is a vector of neural weights at which* $\mathcal{F}$ *is a local diffeomorphism.*

This property can not be reached by any one-hidden layer shortcut FPNN with the same hidden layer size :

Théorème 1 *Let the above function* $\mathcal{F}$ *be computed by a one-hidden layer shortcut FPNN, with* N_h *hidden nodes. Sard's theorem proves that* $\mathcal{F}$ *can not be locally surjective, since its input space dimension (number of weights) is lower than its output space dimension* $N_p d_o$.

In an FPNN, the weights of the virtual shortcut connections depend on the weights between its consecutive layers. This *underparameterization* phenomenon results in a computation power weaker than the one of the simulated shortcut perceptron. This unfavourable result is softened by the weak influence of exact learning capabilities in concrete neural applications.

A.2.3 Underparameterized approximate computing

Usual neural network applications only use approximation capabilities. Therefore, the most useful question is whether hardware-friendly FPNNs are able to approximately compute the same function as standard neural networks that are too large for direct hardware implementation. This problem is studied in [Gir99] through the determination of hardware-friendly FPNNs such that their virtual connections are the connections of standard neural networks used in several applications. The parameters of these FPNNs are then learned so as to achieve the application task, and the results are compared to the ones of the initial standard neural networks.

Standard neural benchmarks have been tested with different FPNNs. These experiments attest that FPNNs may learn classifications as well as standard models (gradient descent learning algorithm), although they use an almost 2D underlying topology and 4 to 10 times less resources than the equivalent standard neural networks.

A.3 Synchronous FPNNs

This section focuses on a specific class of FPNNs, for which both description and computation can be greatly simplified, so that an implementation with on-chip learning as well as an optimized pipelined implementation may be described (see section A.4).

A.3.1 Definition of synchronous FPNNs

A synchronous FPNN is an FPNN in which any resource may *simultaneously* process all its received values. In such an FPNN, any resource can "wait" for all its input values before it begins to process them, without modifying the computation of the whole FPNN.

The study of [Gir99] defines three (rather tricky) conditions for an FPNN to be considered as synchronous. Nevertheless, these conditions are satisfied by a set of easily identified FPNNs, for which all activators use simple additions to update their variable state and for which all links apply linear functions. Therefore, only this set of synchronous FPNNs will be considered from now on. This set includes the FPNNs defined such that their activators perform the computations of standard sigmoidal neurons.

More formally, a simple definition (close to standard feedforward neural networks) may be given for synchronous FPNNs. Following the FPNA/FPNN definition given in A.1, A FPNN must fulfill the following conditions to be synchronous :

- All iteration functions i_n are associative and commutative.
- Whatever computation method is chosen, the activator in node n exactly receives a_n values. In the asynchronous parallel computation method, it means that for all n, there are exactly a_n requests $req[(p,n),(n),x]$ where $p \in Pred(n)$. Therefore an activator always sends an output (except if $a_n = 0$) and it does not receive any value after having sent this output. This condition is fulfilled if the following recursive property is satisfied :

$$\begin{aligned} \forall n \qquad a_n \quad &= \sum_{p \in Pred(n)} d_{(p,n)} r_n(p) \\ \text{where} \qquad d_{(p,n)} \quad &= \chi_{\mathbb{N}^\star_+}(a_p) S_p(n) + \sum_{q \in Pred(p)} d_{(q,p)} R_p(q,n) \end{aligned}$$

- Links must be linear operators $x \mapsto W_n(p)x$ (for each (p,n) in $\mathcal{E}$), instead of affine ones.

A.3.2 Equivalent and simplified computations

In [Gir99], synchronous FPNNs have been introduced when studying the determinism of both sequential and parallel computation schemes for feedforward and recurrent FPNNs. The main result[2] is :

Théorème 2 *Feedforward synchronous FPNNs are deterministic : all neurons compute the same outputs with both sequential and parallel computations, and these outputs do not depend on any graph ordering nor request scheduling policy.*

This is the fundamental property of synchronous FPNNs : the above conditions ensure that the order in which input requests are sent to any activator does not have any influence on its computed output.

These equivalent results may be then expressed in a very simplified way. For each input node n in $\mathcal{N}_i$, a value x_n is given (outer inputs of the FPNN). Then, activators (n) and links (n, s) compute :

$$y_{(n)} = f_n \left(\sum_{r_n(p)=1} y_{(p,n)} + \theta_n \right)$$

$$y_{(n,s)} = W_s(n) \left(\sum_{R_n(p,s)=1} y_{(p,n)} + S_n(s) y_{(n)} \right)$$

Similar results are available for recurrent synchronous FPNNs.

A.3.3 Learning of synchronous FPNNs

Both learning and generalization phases of synchronous FPNNs may be efficiently implemented on configurable hardware. This key feature shows that this kind of neural models is particularly able to take advantage of the characteristics of reconfigurable computing systems : prototyping may be efficiently performed, and then the system may be reconfigured so as to implement the learned FPNN with an outstanding generalization speed, [as shown in [Gir06b]].

The learning phase of many standard neural networks (such as MLPs) is often performed thanks to what is called the back-propagation learning algorithm, where an error function Err estimates the distance between the neural network output and the expected output that corresponds to the given training pattern. A positive real value ϵ is given as learning rate. Each parameter p is updated as follows : $$p \Leftarrow p - \epsilon \frac{\partial Err}{\partial p} \quad .$$

The generalized back-propagation algorithm of [GGR95b] may be applied to synchronous FPNNs (see [Gir99]). It shows that the gradient of a synchronous FPNN may be computed by means of simple formulae. Moreover, these computations are performed by the resources with *local* data (provided by the neighbouring resources). It provides a simple hardware implementation that is based on assembled predefined blocks (section A.4). The simplified architectures of FPNAs ensure that this assembling is straightforward and hardware-friendly.

A node in a synchronous FPNN is an output node if its local activator computes an output value without sending it to any other resource (i.e. $S_n(s) = 0$ for all $s \in Succ(n)$). Let Err be

[2]Indeed, the last condition (linearity of links) is not required here.

the quadratic distance $Err = \frac{1}{2}\sum_{n \text{ output node}}(y_{(n)} - e_n)^2$, where e_n is the corresponding expected output. The following notations are also introduced :

- $\partial Err_{(p,n)}$ is the differential of Err with respect to the output of link (p, n)
- $\partial Err_{(n)}$ is the differential of Err with respect to the output of activator (n)

Then the gradient of Err with respect to the FPNN learnable parameters $W_n(p)$ and θ_n is computed thanks to the following formulae, that are similar to a standard back-propagation process in a multilayer neural network.

Three formulae express the back-propagation of differentials.

- If n is an output node : $\partial Err_{(n)} = y_{(n)} - e_n$.
- Else : $\partial Err_{(n)} = \sum_{S_n(s)=1} W_s(n)\partial Err_{(n,s)}$
- $\partial Err_{(p,n)} = r_n(p)\, f'_n \left(\sum_{r_n(p)=1} y_{(p,n)} + \theta_n \right) \partial Err_{(n)}$
  $+ \sum_{R_n(p,s)=1} W_s(n)\partial Err_{(n,s)}$

Then the gradient values are locally computed.

- Differential w.r.t. θ_n :
  $\frac{\partial Err}{\partial \theta_n} = f'_n \left(\sum_{r_n(p)=1} y_{(p,n)} + \theta_n \right) \partial Err_{(n)}$
- Differential w.r.t. $W_s(n)$:
  $\frac{\partial Err}{\partial W_s(n)} = \left(\sum_{R_n(p,s)=1} y_{(p,n)} + S_n(s) y_n \right) \partial Err_{(n,s)}$

A.4 Implementations of synchronous FPNNs

The computational simplifications of synchronous FPNNs provide two specific implementation methods : on-chip learning is possible, and a fast pipeline implementation may be used (without learning). The general topological simplifications that are made possible by FPNAs are still available, so that these implementation methods consist of an easy modular hardware mapping of basic building blocks.

A.4.1 On-chip learning

The above computations are local, so that each proposed implementation block corresponds to one neural resource in the FPNA. These basic blocks successively handle :

- the computation of the resource output
- the computations required in the back-propagation process
  - an activator (n) back-propagates

$$\partial Err_{(n)}\, f'_n \left(\sum_{r_n(p)=1} y_{(p,n)} + \theta_n \right)$$

towards its connected predecessors

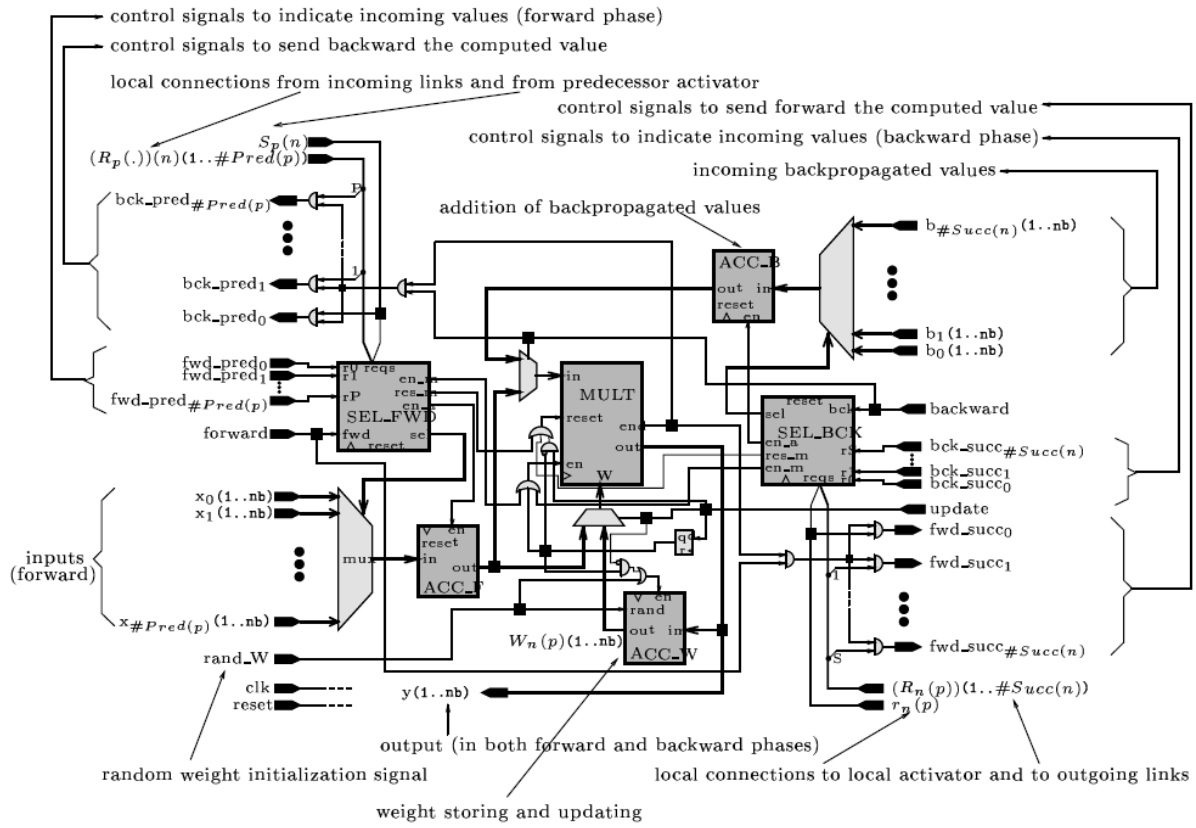


FIG. A.5 – Link architecture (learning)

– a link (n, s) back-propagates $W_s(n)\partial Err_{(n,s)}$
– the computations required to update the local learnable parameters.

Links

Figure A.5 shows a possible implementation of a link with on-chip learning. All flip-flops use asynchronous reset signals that are active high. The clock and reset signal connections are not shown in order to lighten this figure. The reset signal is active before each new iteration of the learning algorithm. There are six main sub-blocks :

– `MULT` is a multiplier. It is shared by the different computation steps. Signal `end` is active when the multiplier result is available on bus `out`.
– `ACC_F` accumulates the received values. Since the FPNA concept may handle neural architectures with reduced fan-ins, there is no need to use additional bits to store accumulation results. These results (stored in `ACC_F`) remain unchanged when back-propagation is performed (when signal `forward` is idle).
– `SEL_FWD` receives a request signal when any connected predecessor sends a value to the link. This block selects a request signal, so that `ACC_F` accumulates the corresponding value. When all expected values have been received, `SEL_FWD`, signal `en_m` is set active, so that `MULT` mul-

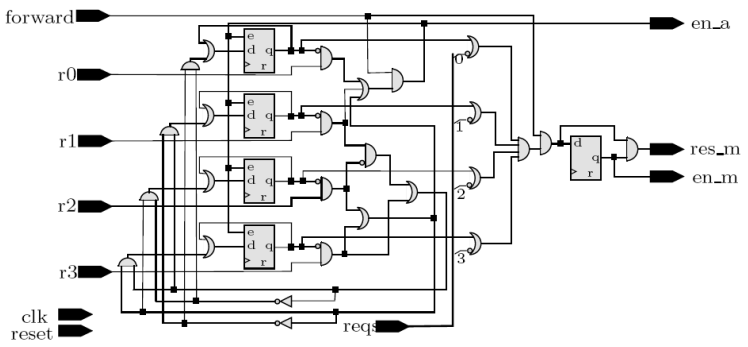


FIG. A.6 – Blocks SEL_FWD and SEL_BCK

tiplies the local weight $W_n(p)$ by the accumulated value in ACC_F. When the multiplication result is available, request signals are sent to all connected successors (links or activators). Figure A.6 shows such a SEL_FWD block, supposing that the resource fan-in is equal to 4 (as in a grid-based layered FPNA).

- ACC_B and SEL_BCK are similar to the above ACC_F and SEL_FWD. They are used for back-propagated computations.
- ACC_W stores the local weight $W_n(p)$. When signal update is active, values stored in ACC_F and ACC_B are multiplied. The result is accumulated in ACC_W : the local weight is updated.

Various implementation choices are proposed for the accumulators and the multipliers in [Gir99]. With 16-bits[3] values (fixed point, 11 fractionary bits), 2-complement accumulators, and a semi-parallel multiplier (four 8×8 multiplications sequentially handled), this link implementation uses less than 120 CLBs on a Xilinx Virtex-E FPGA. Accumulations require one clock cycle, multiplications require 5 clock cycles (including reset), at 50 MHz.

Activators

The main changes with respect to a link are related to the computation of f_n and f'_n. When sigmoid function $f_n(x) = \tanh(x)$ is used, then $f'_n(x) = 1 - (f_n(x))^2$. It leads to [an architecture] with an additional part that computes a piecewise polynomial approximation[4] of tanh : a table stores the polynomial coefficients (address given by the most significant bits stored in ACC_F), a counter (CNT) gives the degree of the expected coefficient, an adder (ADD) and MULT compute the approximation with a Hörner scheme.

With the above implementation choices, less than 145 CLBs of a Xilinx Virtex-E FPGA are required. The tanh function approximation lasts 11 clock cycles (a piecewise degree-2 polynomial approximation is sufficient to obtain the expected precision, provided that the coefficients are 20-bit values and MULT is a 16×20 semi-parallel multiplier).

[3]The required precisions are precisely studied in [Gir99]. 16 bits are sufficient for most tested synchronous FPNNs. It confirms the studies of [HH93] despite the major differences between standard neural networks and FPNNs.

[4]Preferred to a CORDIC-like algorithm : such a method better applies to a sigmoid function such as tan^{-1} than *tanh* (see [ABDG+97]), but computing tan^{-1} derivative requires a division, whereas computing *tanh* derivative only requires a multiplier that is already mandatory to handle back-propagated values. Moreover, this multiplier is sufficient in the forward phase for the polynomial approximation

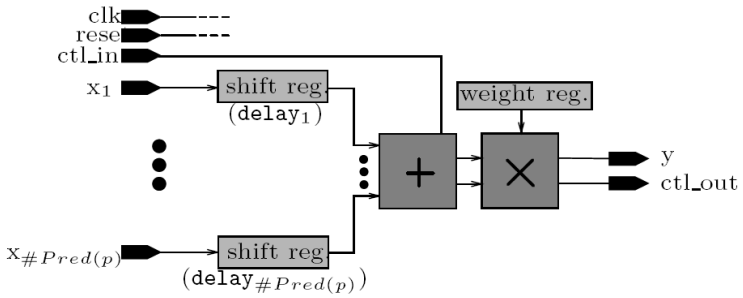


FIG. A.7 – Link (pipelined synchronous FPNN)

A.4.2 Pipelined implementation

A pipeline implementation is proposed in [Gir99] for any synchronous FPNN. It uses on-line operators (see [GT00] for the adequation of on-line arithmetic [5] to neural computations). In such an implementation, building blocks are (again) assembled according to the FPNA underlying structure. Thanks to simplified computations (§A.3.2), these building blocks do not require any protocol handling. They handle data and computation in a serial way. Thanks to the use of on-line arithmetic, both activators and links handle data in the same way. Activators use *elementary functions* ($\sin, \cos, \tan, \tanh, \exp, \log, \arctan, \ldots$). The evaluation of these elementary functions can be performed using polynomial or rational approximations, shift and add algorithms or table-based algorithms. Those algorithms are presented in [Mul97]. In this work, we have chosen the evaluation of the tanh function using polynomial approximations with a Hörner scheme[6]. See [GT00, Gir99] for more details about on-line operators and delay handling.

Figure A.7 sketches a "link" building block. It uses 37 CLBs (170 for an activator) of a Xilinx Virtex-E FPGA for a 16-bit precision. It may be clocked at 100 MHz.

A.5 Implementation performances

This section discusses the implementation performances (area and speed) of the various implementation methods that have been described through several examples.

Our implementations are based on a Celoxica RC1000PP board with a Xilinx Virtex XCV1000E-BG560 FPGA. Xilinx FPGAs have been chosen because the fine grain parallelism of neural computations requires many elementary memory devices, and such FPGAs offer a high memory/logic ratio. We use the Virtex-E FPGA family to illustrate FPNN implementation performances, indicating the smallest one that is sufficient to implement each FPNN. Each CLB in these FPGAs corresponds to 4 configurable logic cells.

It must be pointed out that current FPGAs already outperform the capacity of the Virtex XCV1000E : such an FPGA contains 27648 logic cells, to be compared with the 73008 ones of the largest Virtex-E, as well as with the 125136 logic cells of the largest current Virtex-II Pro.

[5] An on-line arithmetic is a bit-serial arithmetic that processes any operation or elementary function with the most significant digits first, thanks to the use of a redundant number representation system that avoids carry propagation within additions.

[6] Again preferred to a CORDIC-like algorithm, that would require an iterated process unfitted for a pipeline implementation

Such improvements make the scalability of the FPNA concept even more useful : larger neural networks may be implemented without routing bottlenecks, or spared CLBs may be used for other computations required by the application.

We currently use Xilinx ISE 4.2 as synthesis tool. The mapping of each building block is asked to optimize the implementation area. Modular mapping is priviledged for large FPNNs. Speeds are given according to both FPGA synthesis indications and memory bandwidth limits in our board. It should be pointed out that these performances have not been optimized by experts of FPGA implementation, and that they might be also improved by a combined use of area-saving neural operators (for example bit-stream neurons may be used in conjunction with the pipeline implementation of FPNNs).

A.5.1 Application to a benchmark problem

Several implementations are available for the FPNN [that has been defined for the Proben 1 Diabetes standard benchmark problem in [Gir99]].

- The implementation of the learning phase of this FPNN requires 13250 CLBs (16-bit precision, based on parallel 8×8 multipliers and a piecewise polynomial approximation of tanh). It may use a single Xilinx Virtex XCV3200E (largest Virtex-E), or two Xilinx Virtex XCV1600E-BG560. In this case the small fan-ins of the FPNA resources make it possible to exchange all required signals with the 404 available I/O ports of the FPGAs : at most 168 ports are required for each FPGA. The [equivalent] shortcut perceptron could not be directly mapped onto any number of FPGAs : these FPGAs would have to exchange too many signals, besides the topology and fan-in problems inside each FPGA. One learning iteration of the FPNN lasts 378 clock cycles, so that a 14 FPNN-MCUPS speed is reached : 14.10^6 links are updated by the learning process per second. It corresponds to 70 MCUPS (70.10^6 connections updated per second) with the optimal shortcut perceptron which outputs the same function as the FPNN.
- For the generalization phase, the general implementation of [the asynchronous parallel computation method of A.1.3] uses 7500 CLBs (a Xilinx XCV1600E is necessary). A 24 FPNN-MCPS speed is reached : 24.10^6 neural resources are processed per second. It corresponds to 120 MCPS with the equivalent optimal shortcut perceptron.
- An optimized pipeline implementation of the generalization phase of this FPNN requires only one Xilinx XCV1000E (3820 CLBs). A 250 FPNN-MCPS speed is reached, with a sufficient 11-bit precision (according to preliminary tests). It corresponds to 1.25 GCPS (giga connections per second) with the equivalent optimal shortcut perceptron.

Similar performances have been obtained for other Proben 1 applications (see [Gir99]).

A.5.2 Performance analysis

The above speeds must be compared with the different neural network implementations discussed in section 2.1. Nevertheless, it is difficult to make a fair comparison between the performances of the various FPGA-based methods : they implement different neural architectures, with different implementation choices (precision, on-chip learning, etc), the FPGA types vary, and a part of the performance differences may be attributed to FPGA technology improvements.

Without on-chip learning, the speeds for the implementation of standard multilayer architectures range from a few MCPS per FPGA (2 MCPS in [EH94], 4 MCPS in [BAA93]) to a few tens MCPS

(18 MCPS in [GT00], 22 MCPS in [BH94]). With on-chip learning, speeds range from 0.1 MCUPS in [EH94] to 8 MCUPS in [GT00].

Despite these heterogeneous previous results, it appears that the above FPNNs outperform the few previous FPGA-based implementations, with or without on-chip learning. The performances for the various FPNNs of [Gir99] are more similar to the ones of complex neuro-computers : 5 MCUPS to 1 GCUPS and 10 MCPS to 5 GCPS (neuro-computers CNAPS, RAP, etc). But only one or a few FPGAs are required for FPNN implementations.

Indeed, great implementation speeds are not the main advantage of the FPNA/FPNN concept : above all, this new neural framework defines neural networks that are easy to map onto some configurable hardware by means of a hardware-friendly assembling of predefined blocks, whereas such a straightforward parallel implementation is impossible for equivalent standard neural networks. As soon as a neural solution fits a chosen hardware device thanks to the FPNA paradigm, effort is put on the optimization of implementation choices (arithmetic, precision, etc) so as to reach outstanding performances.

FPNAs make it possible for current and future neural implementations to take advantage of the rapid FPGA technology improvements, whereas the previous advanced methods are limited by large neural architectures that do not fit hardware topologies.[7] Moreover the FPNA paradigm may be used in conjunction with such advanced area-saving methods.

[7] A bit-stream based method leads to very small implementation areas, but the limits of the FPGA connectivity are already reached for a rather small FPGA in [BH94].

Annexe B

Détection embarquée de l'hypovigilance

B.1 Introduction

Plusieurs études ont déjà été menées pour tenter de discriminer, à l'aide de réseaux de neurones artificiels, les différents états de vigilance d'un sujet humain à partir de signaux électroencéphalographiques (EEG). Nous avons mené une étude orientée de manière à pouvoir obtenir un système léger, utilisable sans entrave par un sujet humain, grâce à une limitation des besoins de calcul et de mémoire. Cette annexe résume ces travaux de mise en application sur circuit numérique programmable embarqué des résultats d'une démarche menée antérieurement pour dégager les paramètres électroencéphalographiques susceptibles de caractériser et de classifier les différents états de vigilance chez des sujets en situation réelle et en tenant compte des artefacts.

Le but est d'aboutir à un système portable de détection de l'hypovigilance à partir d'un nombre minimal de dérivations d'EEG en vue d'une exploitation en ambulatoire. Les contraintes d'utilisation envisagées impliquent la nécessité d'aboutir à une solution très basse-puissance, ce qui constitue le critère essentiel pour les différents choix technologiques opérés. Il faut noter que les travaux qui sont menés jusqu'à maintenant [JKL95, VRCP02], traitent essentiellement la détection elle-même sans présenter les travaux d'implantation sur des systèmes programmables.

B.2 Détection de l'hypovigilance

Nous utilisons, entre autres, des algorithmes de classification automatique à apprentissage non supervisé qui se fondent sur les cartes auto-organisatrices[1] de Kohonen. Les résultats de ces travaux sont présentés dans [BKBB+03b].

B.2.1 Prétraitement du signal EEG

Comme premier traitement spectral, une transformée de Fourier Rapide à Court Terme (TFRCT) est appliquée à la dérivation EEG pariéto-occipitale droite (P4-O2) sur des portions de 4 secondes, avec une fenêtre de pondération de type Hamming et une résolution de 512 points.

[1] Self-organizing map, SOM

Par la suite un découpage en bandes de 1 Hz du spectre du signal EEG est effectué. En effet 23 bandes de 1 Hz (allant de 1 à 24 Hz, normalisées par rapport à la puissance spectrale totale) sont extraites :
$PPS_i = \frac{PS(i \text{ à } (i+1)Hz)}{PST}$*100, i allant de 1 à 23 Hz.

Où PPS_i représente le Pourcentage de la Puissance Spectrale de la bande i Hz correspondante, PST la Puissance Spectrale Totale, et PS la Puissance Spectrale.

Après ce codage initial par bandes du signal EEG, un traitement connexionniste par les cartes auto-organisatrices de Kohonen est appliqué. Les entrées du réseau de neurones en question sont les Pourcentages de la Puissance Spectrale par bandes (PPS_i) du même signal.

B.2.2 Méthodes connexionnistes

Dans ce travail deux algorithmes d'extraction automatique de catégories à l'aide de modèles connexionnistes auto-organisés (apprentissage non supervisé ou apprentissage supervisé) ont été exploités. Ils sont présentés ci-dessous.

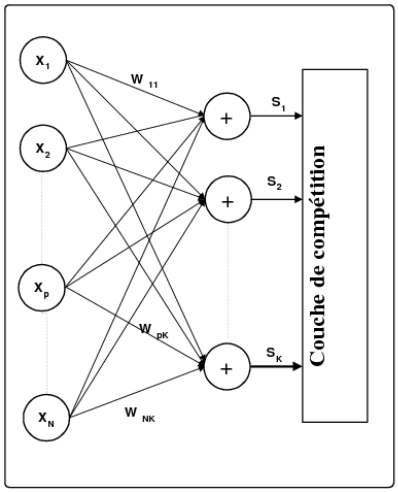


FIG. B.1 – SOM de Kohonen

les cartes auto-organisatrices de Kohonen

Le but est d'utiliser la capacité des cartes auto-organisatrices de Kohonen à séparer, d'une façon non supervisée, des états par ailleurs déjà qualifiés par l'expert, pour analyser aussi bien la répartition de ces états sur l'espace de sortie que les associations qui peuvent se dégager entre ces états. Le modèle SOM modélise le mécanisme de l'auto-organisation spatiale des perceptions opérée par le cortex sous forme d'un processus de classification topographique. Selon ce processus, les données d'entrées, représentables dans le cas général sous forme de vecteurs à N dimensions, sont ramenées à des classes qui s'auto-organisent selon une structure bidimensionnelle de nœuds sur laquelle les relations de voisinage sont prédéfinies. Le processus de classification topographique du modèle SOM combine donc une étape de classification avec une étape de projection des données. Pour la réalisation de ce modèle connexionniste, deux couches de neurones sont utilisées : la première représente les entrées, la seconde les sorties (les classes). Les deux couches

sont entièrement connectées. L'algorithme d'apprentissage du modèle SOM est présenté de manière détaillée dans [Koh01]. Il est de type compétitif, non supervisé. Il comprend principalement deux étapes : (1) Sélection d'un nœud gagnant - (2) Mise à jour du profil du nœud gagnant et de ceux des nœuds appartenant à son voisinage.

Sélection du nœud gagnant :
Soit $x(t) = \{x_1(t), x_2(t), ..., x_N(t)\}$ le vecteur d'entrée (i.e. la donnée) sélectionné au temps t, et soit

$$W_k(t) = \{W_{k1}(t), W_{k2}(t), ..., W_{kN}(t)\}$$

le vecteur de poids associé au nœud k au temps t. Une mesure de distance est choisie (par exemple, la distance euclidienne) et la distance la plus faible $\|x(t) - W_k(t)\|$ permet de définir le nœud gagnant c, soit : $\|x(t) - W_c(t)\| = \min_k \|x(t) - W_k(t)\|$

Apprentissage non supervisé et sélection du voisinage : Après la sélection du nœud gagnant c, le vecteur de poids associé à ce nœud ainsi que les vecteurs de poids associés aux nœuds se trouvant dans un voisinage donné du nœud c (c'est-à-dire les sorties situées dans un périmètre défini autour du nœud c) sont ajustés de manière à ce que leur profil se rapproche de celui de la donnée d'entrée. Cet ajustement de poids qui caractérise l'apprentissage non supervisé du modèle peut être décrit par l'équation : $W_{ki}(t+1) = W_{ki}(t) + a(t) * h(k, k_*) * [X_i(t) - W_{ki}(t)]$ pour $1 \leq i \leq$ N où $a(t)$ est un terme de gain ($0 \leq a(t) \leq 1$) décroissant en fonction du temps et convergeant vers 0, et $h(k; k_*)$ est la fonction de voisinage (ou d'interaction), qui est une fonction de la distance $d(k; k_*)$ entre les unités k et k_* sur la carte. Cette fonction vaut 1 lorsque $d(k; k_*) = 0$ et décroît quand la distance augmente.

A l'issue de l'algorithme d'apprentissage, les données en entrée peuvent être reprojetées sur la carte obtenue. Le nœud d'affectation d'une donnée représente alors celui dont le vecteur de poids est le plus proche du vecteur descriptif de la donnée. Il peut donc être sélectionné en réutilisant l'étape (1) ci-dessus.

Cette première approche exploite le pouvoir de séparation non linéaire des SOM de Kohonen. Il faut noter que peu d'informations sur les variations du signal EEG pendant les périodes d'hypovigilance sont disponibles. Ceci explique le choix de ce type d'algorithme à apprentissage non supervisé dans notre application. Les cartes auto-organisatrices de Kohonen ont permis de donner une visualisation topographique du spectre du signal EEG enregistré au moment de la phase de transition éveil-sommeil. Les résultats dégagés ont permis de trouver des associations de voisinage possibles entre différents niveaux de vigilance. Nous avons réussi à avoir des résultats comparables à ceux trouvés dans [JKL95] en réduisant le nombre de dérivations d'EEG de 20 à une seule.

Le Learning Vector Quantization : LVQ

Les cartes non supervisées de Kohonen constituent déjà un outil de prétraitement efficace pour la séparation et la redistribution des vecteurs d'entrées en différentes classes. En effet, les SOM nous permettent d'avoir une idée sur la distribution statistique des vecteurs d'entrées sur la couche de sortie. Après apprentissage chaque neurone de cette couche peut être activé par des vecteurs d'entrée qui correspondent à différentes classes, ce qui pose un problème dans le processus de décision et d'étiquetage des neurones.

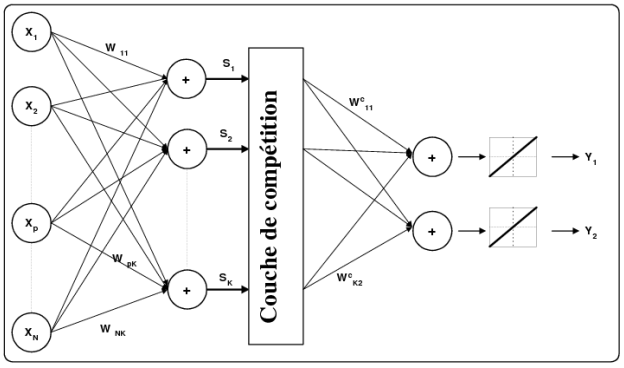


FIG. B.2 – LVQ de Kohonen

Pour remédier à cette limite, on a appliqué une deuxième phase d'apprentissage, cette fois-ci supervisée. Cette phase va permettre un réajustement des distributions de probabilités sur la carte de sortie et prendre par la suite des décisions sur l'étiquette attribuée à chaque neurone.

Notre choix s'est porté sur l'algorithme à apprentissage supervisé proposé par Kohonen : Le Learning Vector Quantization (LVQ). L'architecture du LVQ est similaire à celle de la carte de Kohonen, sans connexions latérales pour les cellules de la deuxième couche. Cet algorithme constitue, avec ses différentes variantes, une amélioration de la séparation en classes à partir de la solution proposée par l'apprentissage non supervisé.

La méthode consiste à rapprocher le prototype le plus activé de l'entrée s'il est de la bonne classe (apprentissage supervisé), et à le repousser dans le cas contraire. Les autres prototypes (c'est-à-dire les perdants) restent inchangés. Les prototypes deviennent ainsi les représentants des classes.

Cette deuxième approche fondée sur l'exploitation des LVQ a permis de faire d'une part une classification des niveaux de veille et de sommeil et d'autre part un filtrage des séquences artefactées. Les résultats dégagés par ce réseau de neurones sont similaires à ceux présentés par Vuckovic dans [VRCP02]. Le taux de reconnaissance total des niveaux de vigilance atteint sur des corpus de test 76,73% [BKBB$^+$03b].

B.2.3 Problématique de l'implantation

Le but final de ce travail est de réaliser un système embarqué pour la détection de l'hypovigilance. Pour cela, nous avons effectué une implantation des SOM de Kohonen en phase de décision sur un FPGA (en exploitant les paramètres d'apprentissage obtenus par simulation). Pour une telle réalisation il faut tenir compte de plusieurs paramètres et caractéristiques utiles comme la puissance consommée en fonction de la fréquence de l'horloge externe, le nombre d'entrées/sorties, la surface totale d'intégration, les besoins en signaux de contrôle (CLK, reset des opérateurs), l'architecture choisie et enfin le parallélisme neuronal (les différents neurones peuvent travailler de façon concurrente). Il est intéressant de signaler que certains de ces paramètres sont difficilement estimables avec précision avant la synthèse.

La vitesse d'exécution est parfois un autre critère essentiel. Dans notre cas, la prise de décision (détection d'une situation d'hypovigilance) se fait en temps réel sans difficulté grâce à une im-

plantation totalement parallèle ne nécessitant ni utilisation séquentielle des ressources de calcul, ni reconfiguration dynamique du FPGA. La vitesse d'exécution n'est donc pas une véritable contrainte pour les choix technologiques effectués. En revanche, l'hypothèse d'un circuit embarqué de manière ambulatoire implique le besoin d'aboutir à une implantation très basse-puissance. Parmi les paramètres cités ci-dessus, le nombre d'entrées/sorties, mais surtout l'obtention d'une solution qui exploite entièrement le parallélisme neuronal ont une influence directe sur la consommation de l'implantation obtenue.

B.3 Implantation directe de la détection d'hypovigilance

Différents types d'implantations de SOM sur circuits intégrés existent déjà. On peut les répartir principalement en deux catégories :

- Des implantations analogiques des SOM sur circuits intégrés dédiés ont été réalisées (par exemple [MVJL93]). Ces implantations sont particulièrement bien adaptées aux fonctionnalités simples des neurones et permettent d'atteindre par la suite un rapport entre vitesse de calcul, densité et consommation hors de portée des implantations numériques. En revanche, ces technologies présentent des limites techniques telles que leur manque de précision, de robustesse ainsi que leur sensibilité à la technologie utilisée.
- Des implantations de cartes auto-organisatrices de Kohonen sur circuits numériques ASIC (Application Specific Integrated Circuit) ont également été réalisées (neuroproceseurs). Actuellement, ces composants constituent la catégorie de circuits VLSI la plus utilisée pour l'intégration d'algorithmes neuromimétiques.

Les défauts cités pour ces deux types d'implantations peuvent être contournés par l'utilisation de circuits reprogrammables, comme par exemple les FPGA (Field Programmable Gate Arrays). Dans [FTGR96] l'auteur a réussi à implanter une SOM en phase de décision à 8 entrées (codées sur 8 bits) et 4096 neurones sur la couche de sortie (grâce à des calculs séquentialisés). L'architecture proposée nécessite beaucoup de lignes d'interconnexions parallèles entre les différents co-processeurs, ce qui rend l'exploitation de ce réseau non modulaire pour d'autres applications. Speckmann dans [STR93] a implanté sur FPGA des unités de calcul (Processing Unit : PU) qui permettent d'extraire la distance euclidienne d'un vecteur d'entrée (codé sur 16 bits) pour seulement un neurone. L'architecture est entièrement parallèle et ne permet pas de calculer la distance minimale (en effet elle se fait à l'extérieur sur un PC qui s'interface avec le FPGA via des mémoires FIFO). Dans [PKW$^+$01], les auteurs ont implantés une architecture dynamique reconfigurable de SOM (9 entrées codés sur 8 bits et 250x250 neurones sur la couche de sortie) sur une carte RAPTOR2000 architecturées autour de 4 FPGA Xilinx de la série Virtex-E et un CPLD de la famille Xilinx aussi. Comme dans [FTGR96] l'architecture proposée est entièrement parallèle donc elle va nécessiter beaucoup de lignes d'interconnexions parallèles entre les différents co-processeurs (4 FPGA).

Cette section présente notre implantation du modèle de réseaux de neurones de type Kohonen sur un circuit FPGA de type VIRTEX, à l'aide d'une arithmétique série LSBF : ce choix est motivé par la nature des opérations arithmétiques utilisées pour ce type d'ANN comme la soustraction, le carré et l'addition. Le cas des comparaisons (naturellement MSBF) introduit un traitement spécifique, par bufferisation des résultats des autres calculs.

Le parallélisme introduit par le pipeline au niveau du bit et par le traitement de plusieurs entrées et sorties (23 neurones sur la couche entrées et 5x5 neurones sur la couche de sortie) simultanément permet d'obtenir des performances (en terme de surface d'intégration et puissance

de consommation) très nettement meilleures que celles obtenues par des solutions basées entièrement sur une arithmétique parallèle.

B.3.1 Opérateurs sériels

L'arithmétique sérielle permet de réaliser des architectures de calcul où les chiffres circulent en série, chiffre après chiffre. Deux types d'architectures sont possibles :

- Architecture sérielle poids faibles en tête [LSBF].
- Architecture sérielle poids fort en tête [MSBF] :

L'intérêt de l'architecture MSBF est de pouvoir calculer toutes les fonctions (en arithmétique série poids faibles en tête, il n'est pas possible de calculer certaines opérations comme la division, la comparaison ainsi que les fonctions mathématiques élémentaires). Néanmoins, compte tenu des contraintes de taille d'implantation et de la possibilité de regrouper les opérations posant problème (comparaisons) après stockage et renversement de l'ordre des chiffres, notre choix s'est porté sur l'arithmétique sérielle classique LSBF.

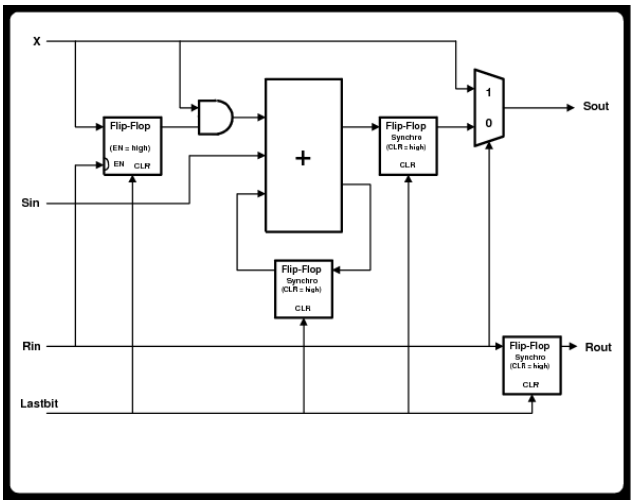


FIG. B.3 – Unité élémentaire de calcul du carré

Nous avons utilisé les additionneurs et soustracteurs sériels classiques. En revanche l'opérateur utilisé pour le calcul du carré correspond à celui étudié et réalisé par Ienne dans [IV94]. L'architecture de l'opérateur est présentée sur les figures B.3 et B.4. L'idée consiste, dans le cas plus général d'une multiplication, à recoder les entrées sur $2n + 1$ bits (n représente le nombre de bits des entrées), ce qui permet de décaler la gestion du bit de complément à 2 au-delà des $2n$ du résultat attendu. Après la présentation des deux bits d'ordres i, le résultat du produit partiel est :

$$P_i = X_i.Y_i = P_{i-1} + x_i.Y_{i-1}.2^i + y_i.X_{i-1}.2^i + x_i.y_i.2^{2.i}$$

X_i et Y_i sont les valeurs X et Y réduites aux bits 0 à i : $X_i = X \bmod 2^{i+1}$. Notons que les valeurs initiales sont : $P_{-1} = X_{-1} = Y_{-1} = 0$.

Cette équation est valable jusqu'à l'apparition du bit de complément ($i < n - 1$).

Pour ($i \geq n$) l'équation ci-dessous est exploitée :

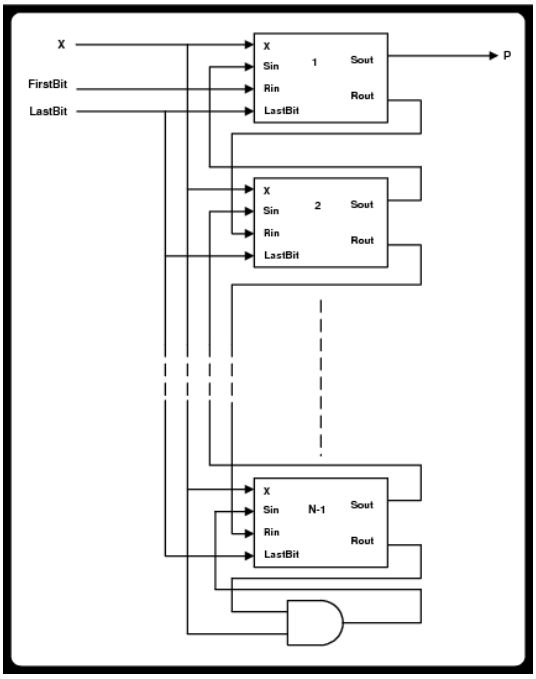


FIG. B.4 – Calcul du carré (à base d'unités élémentaires)

$$\overline{P_i} = \overline{P_{i-1}} + x_{n-1}.Y_{n-2}.2^i + y_{n-1}.X_{n-2}.2^i + x_i.y_i.2^{2.i} \tag{B.1}$$

$\overline{P_i} = P_i$ est toujours vrai pour $i < n$.
Pour $i \geq n$ l'erreur générée à la suite de l'utilisation de l'équation B.1 est la suivante :

$$E_i \quad = \quad x_{n-1}.y_{n-1}.2^{2.i}.\sum_{j=n}^{i} 2^j$$

D'après cette dernière égalité on a $E_i \bmod 2^{i+1} = 0$. On peut extraire par la suite le résultat de la multiplication (ou du carré).

Le choix de cette implantation correspond à nos besoins de minimisation de surface sur le FPGA, grâce à l'absence de registre spécifique de stockage interne des résultats intermédiaires (cf opérateurs à addition et décalage).

B.3.2 La précision du calcul

La précision requise par notre application neuronale a été étudiée par simulation logicielle. Les précisions ont été étudiées en fonction de la nature des données : poids, entrées et carrés. Huit chiffres binaires sont suffisants pour représenter les poids et les données d'entrées (1 bit de complément, 4 bits pour la partie entière et 3 bits pour la partie décimale). De plus 8 bits parmi les

16 issus de la mise au carré suffisent pour représenter les sorties des calculs de distance (6 bits pour la partie entière et 2 bits pour la partie décimale, pas de bit de signe, les valeurs étant positives).

L'implantation peut être aisément étendue à des précisions supérieures en cas de besoin, n'entraînant qu'une augmentation linéaire de surface de certains opérateurs. D'autre part un changement de précision implique une modification des intervalles de temps du contrôle de l'architecture.

B.3.3 Architecture de la carte auto-organisatrice

L'implantation du réseau de neurones SOM (Kohonen ou LVQ, puisque les deux modèles sont identiques en phase de décision) sera décrite tout d'abord dans la sous-section B.3.3 où on va détailler l'architecture d'un neurone de la couche de sortie et dans la section B.3.3 où on va détailler l'architecture globale du réseau.

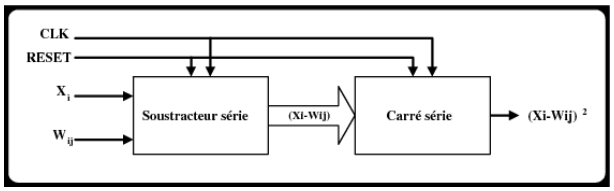


FIG. B.5 – Unité élémentaire pour le calcul série de la soustraction et du carré

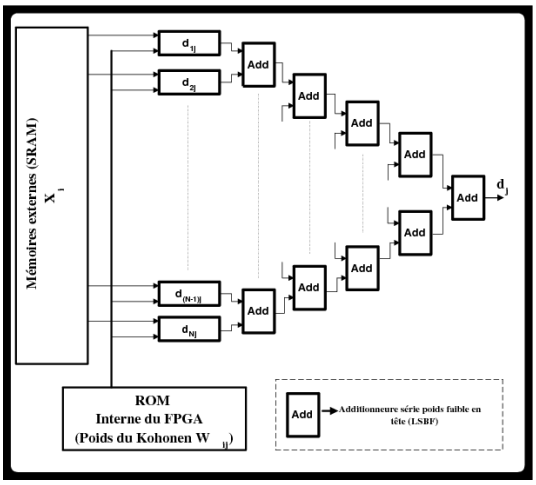


FIG. B.6 – Architecture d'un neurone

Architecture d'un neurone

Pour chaque neurone j de la couche de sortie on doit calculer la combinaison $d_j = \sum_{i=0}^{N-1}(X_i - W_{ij})^2$ (N représente le nombre des entrées de la SOM, dans notre cas $N = 23$). L'architecture de l'opérateur $(X_i - W_{ij})^2$ est présentée sur la figure B.5 : association entre un soustracteur sériel et un multiplieur carré sériel. L'architecture globale de la fonction $\sum_{i=0}^{N-1}(X_i - W_{ij})^2$ est présentée

sur la figure B.6. La taille relativement réduite des opérateurs série permet d'effectuer la totalité des opérations en parallèle, au moyen d'une colonne de N soustracteurs suivis d'une colonne de N multiplieurs carré. Les N sorties obtenues sont alors connectées aux N entrées d'un additionneur série (architecture arborescente simple). Il est intéressant de noter que les poids W_{ij} ont été figés sur la ROM interne du FPGA et les entrées X_i sont directement connectées aux blocs SRAM externes de la carte RC1000-PP.

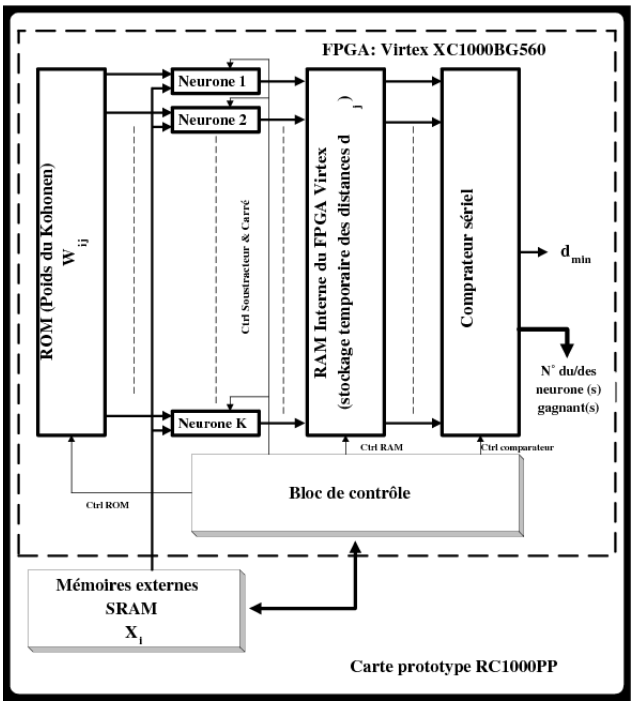


FIG. B.7 – Architecture du Réseau de neurones de Kohonen

Architecture globale de la SOM implantée sur le FPGA

L'architecture globale du réseau est formée de 4 blocs (figure B.7) :

- Une colonne de $K = 25$ neurones (détaillés dans la sous-section précédente).
- Un bloc de mémoires SRAM internes pour le stockage temporaire des distances euclidiennes de chaque neurone de la couche de sortie. Ce bloc sera nécessaire pour changer le sens de la propagation sérielle des bits (passer d'une propagation LSBF à MSBF). Ce changement est nécessaire pour le fonctionnement du comparateur.
- Un comparateur sériel pour la sélection de la distance euclidienne minimale d_i correspondant au neurone gagnant. L'architecture du comparateur est spécifique aux besoin de la SOM : elle permet d'extraire en série (en mode MSBF) simultanément la distance et le numéro correspondant au neurone gagnant (figure B.8).
- Une unité de contrôle global qui supervise et pilote le flux de signaux entre les différentes entités. Le contrôleur est basé essentiellement sur un automate fini dont l'architecture de base s'articule autour d'un compteur d'états.

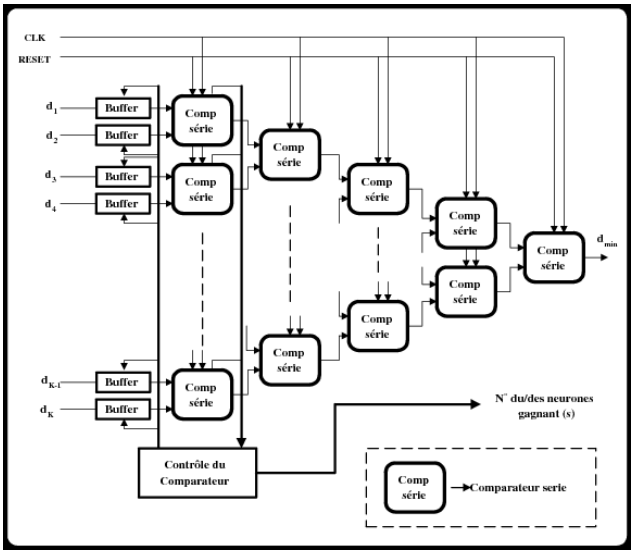


FIG. B.8 – Comparateur

B.3.4 Résultats

Pour valider nos choix matériels et algorithmiques, on a exploité une carte de test (carte de prototypage rapide) Celoxica RC1000-PP qui se base sur une électronique d'acquisition et un FPGA.

Temps d'exécution
et surface d'implantation

Le tableau B.1 regroupe les données de temps et d'espace d'implantation des différents opérateurs arithmétiques, blocs de contrôle, blocs mémoires, ainsi que du réseau complet, en fonction des solutions arithmétiques envisagées (sériel en mode LSBF ou MSBF en-ligne, ou parallèle). Les résultats sont obtenus avec une fréquence de 25 MHz, lors de la synthèse avec l'outil ISE 5.2 de Xilinx des différentes architectures étudiées dans cet article.

Le FPGA disponible sur la carte RC1000-PP s'avère trop petit pour accueillir la SOM de Kohonen constituée entièrement d'opérateurs en arithmétique parallèle, avec une précision sur 8 bits. En effet, un neurone nécessite alors 1390 slices, ce qui limite à 9 le nombre de neurones simultanément implantables. De plus, une telle solution technologique demande un nombre de ports d'entrées/sorties qui dépasse le nombre d'IOB disponibles.

La deuxième solution que nous avons étudiée est l'utilisation d'une arithmétique sérielle MSBF (ou en-ligne). Une telle arithmétique sérielle permet un calcul poids fort en tête de tous les opérateurs arithmétiques et fonctions élémentaires grâce à l'utilisation d'un système redondant de représentation des nombres, cf [Erc84, GT00]. Cette approche nécessite environ 19000 slices pour implanter tout le réseau de Kohonen (23 entrées et 5x5 neurones sur la couche de sortie), ce qui dépasse les ressources du FPGA utilisé.

L'utilisation d'une architecture sérielle LSBF, en exploitant un pipeline au niveau du bit, s'avère la plus adaptée à notre application. En effet une implantation entièrement parallèle de tout le réseau nécessite 12886 slices avec un temps de prise de décision T_{exec} de 1,37 μ s pour une fréquence

Type du bloc	T_{exec} (μ s)	densité d'intégration		
		Slices (2 par CLB)	LUT	FF
Soustracteur/additionneur [LSBF]	0,32	1	2	1
Soustracteur/additionneur en-ligne [MSBF]	0,32	3	4	1
Soustracteur/additionneur [parallèle]	0,04	4	8	8
additionneur [LSBF] 23 entrées	0,32	24	48	24
additionneur en-ligne [MSBF] 23 entrées	0,32	72	96	24
additionneur [parallèle] 23 entrées	0,04	91	179	0
Calcul du carré [LSBF]	0,51	22	21	26
Calcul du carré en-ligne [MSBF]	0,64	26	45	16
Calcul du carré [parallel]	0,04	40	60	16
Comparateur en-ligne [MSBF]	0,20	90	107	90
ROM (sauvegarde W_{ij})	-	3	4	0
SRAM temporaire (sauvegarde d_j pour le cas LSBF)	0,4	13	25	0
Contrôleur de l'architecture	-	16	27	14
$(X_i - W_{ij})^2$ [LSBF]	0,60	24	23	28
$(X_i - W_{ij})^2$ [parallel]	0,08	44	68	8
Un neurone : $\sum_{i=0}^{N-1}(X_i - W_{ij})^2$ [LSBF]	0,69	614	582	690
Un neurone : $\sum_{i=0}^{N-1}(X_i - W_{ij})^2$ [parallèle]	0,12	1390	1936	568
SOM de Kohonen 23 entrées et 5x5 Sorties [LSBF]	1,37	12286	14557	17659
SOM de Kohonen 23 entrées et 2x2 Sorties [LSBF]	1,37	4320	3269	2882

Tab. B.1 – Performances de l'implantation des cartes auto-organisatrices de Kohonen

d'horloge de 25 MHz, au prix d'une intégration d'une SRAM interne utile pour le changement de sens de propagation des bits LSBF à MSBF : un comparateur série fonctionne toujours en MSBF.

Etude de la consommation

Nous avons aussi abordé dans notre étude le problème de la consommation de la SOM de Kohonen, pour effectuer des comparaisons avec des solutions conventionnelles. Ce point est très important mais assez complexe car un grand nombre de paramètres entrent en compte dans cette étude.

A la fréquence minimale tolérée par notre carte, soit 400 KHz, le temps d'exécution est de 86 μs, ce qui est encore largement suffisant pour les contraintes temps-réel de la détection d'hypovigilance. La consommation est alors de 32 mW (soit 1000 à 2000 fois moins que les processeurs actuels). Si on tient compte de la période de veille autorisée lors d'une utilisation concrète avec prise de décision toutes les secondes, la consommation reste inférieure à 4 μW.

Annexe C

Implantations optimisées pour la détection embarquée de l'hypovigilance

Cette partie fait suite aux travaux rapportés dans l'annexe précédente, et est extraite d'un article soumis (section C.1) et de [GBK06] (section C.2).

C.1 Reduction of the implementation area

In order to reduce the area of our implementation as much as possible, we have adapted our SOM to a L_1 norm distance criterion, so as to get rid of quadratic operators. Nevertheless, some of the remaining operators can not be handled by standard area-saving serial[1] arithmetics (LSBF, Least Significant Bit First) : comparisons must be computed in a MSBF mode. Therefore, we use an on-line serial arithmetics which mode is MSBF thanks to the use of a redundant number representation system.

C.1.1 prototypical implementation

Subsections C.1.1 and C.1.1 describes the implementation of the SOM neural network. Subsection C.1.1 gives details of the architecture of an output neuron. The global architecture of the network is described in subsection C.1.1.

On-line operators

[See [Gir99] for a description of on-line arithmetics. This work uses standard implementations of on-line operators, except to extract the winner neuron.]

The winner neuron determination is achieved by a set of a comparators that compute minimum L_1 *norm* distance and the index of the corresponding input. We have used on-line Comparator

[1] Using any serial arithmetics, where numbers circulate digit by digit starting, smaller operators are designed and less communication links are required.

Elements (CE), capable of extracting the minimum of two redundant radix-2 numbers. An on-line algorithm with zero on-line delay is developed [NK97]. Based on this algorithm a finite state transducer, with 5 states and 15 transitions, is derived.

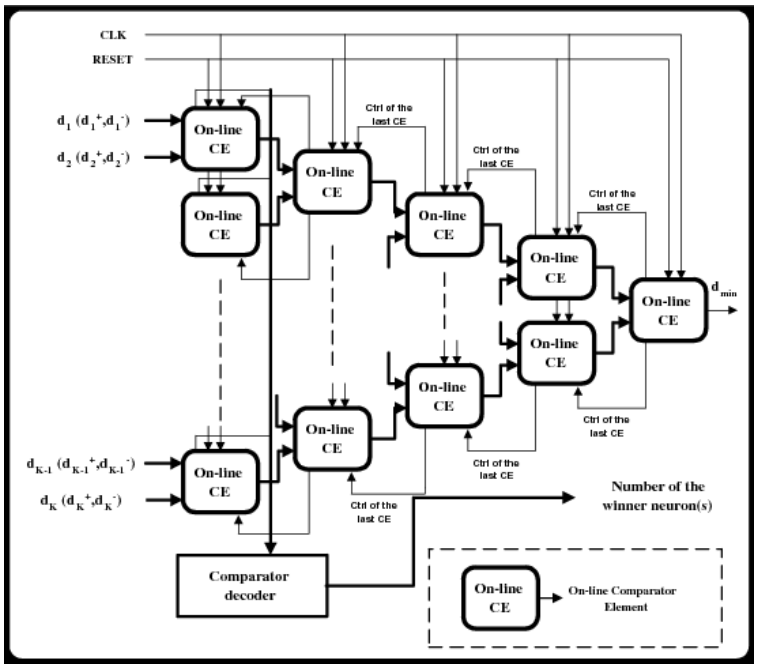


FIG. C.1 – Schematic Implementation of 25-input on-line comparator

To carry out a comparison of the 25 inputs, we use the on-line CE arranged in stages, with the output of neurons connected to the first stage on-line CEs. The output of the first stage on-line CEs are connected to the inputs of the second stage CEs and so on (figure C.1). The last stage has a single on-line CE, whose output is the minimum distance. To extract the winner neuron number,we decode the output controls of the on-line CE first stage. This extraction is performed after the minimal distance processing. The global comparator is composed of 5 stages, with delay 5.

Architecture of a neuron

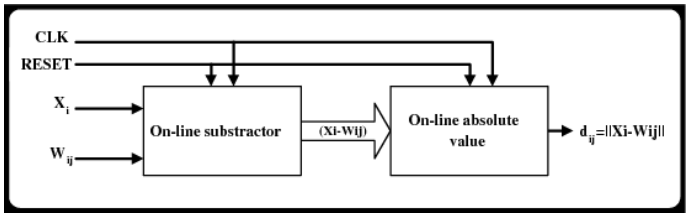


FIG. C.2 – Elementary unit for the serial computation of on-line substraction and absolute value

For each neuron j of the output layer, the L_1 *norm* distance [SSB03] $d_j = \sum_{i=0}^{N-1} |X_i - W_{ij}|$

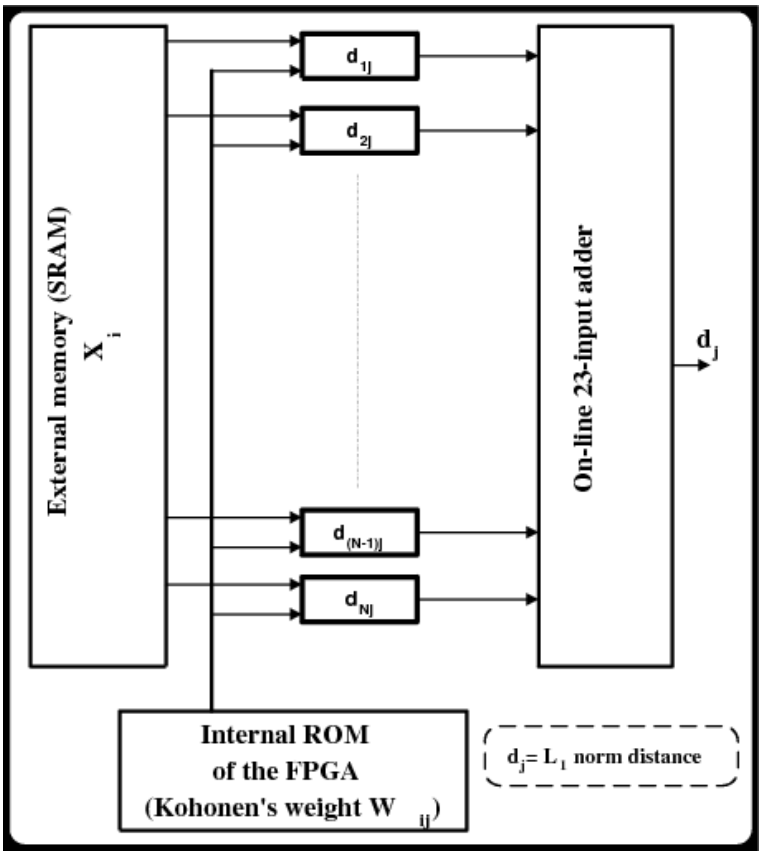


FIG. C.3 – Architecture of a neuron

must be computed (N is the number of inputs of the SOM, in our case $N = 23$). The architecture of operator $|X_i - W_{ij}|$ appears in figure C.2 : a serial on-line substracter and a serial on-line absolute value processing are associated. The global architecture of the distance computation is given in figure C.3. The rather small area of serial operators makes it possible to perform all computations in parallel, by means of a column of N substracters followed by a column of N absolute value processing. It provides N outputs that are connected to the N inputs of a simple tree of serial adders. Weights W_{ij} are fixed, and they are stored in the internal ROM of the FPGA. Inputs X_i are directly connected to the SRAM external blocks of the RC1000-PP board.

Global architecture of the SOM on the FPGA

The global architecture of the neural network consists of 3 blocks (figure C.4) :
- A block of internal ROM memories, used to stored fixed SOM Weights.
- A column of $K = 25$ neurons (detailed in the previous subsection).
- A serial on-line comparer to select minimal distance d_i and the corresponding winner neuron. The architecture of this K-inputs comparer is specific to the requirements of a SOM : it simultaneously extracts the minimal distance and the number of the winner neuron in serial on-line architecture (figure C.1).

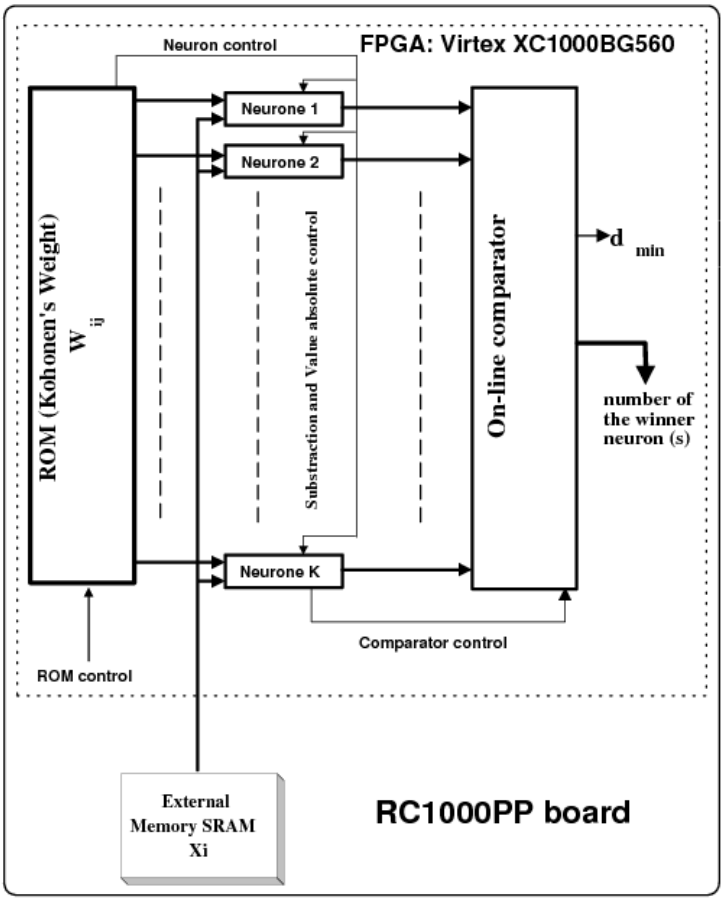


FIG. C.4 – Architecture of the Kohonen map

C.1.2 Results

We have validated our choices in terms of hardware devices and algorithms on a RC1000-PP test board (fast prototyping board) that uses a Virtex XCV1000E FPGA.

Computation time and implementation area

MSBF serial arithmetics may compute any arithmetical operator or elementary function. In our case, it makes it possible to get rid of data buffering before comparison, so that a complete pipeline parallelism is reached.

Two approaches were compared. The first is based on the use of the L_2 norm (Euclidean distance) to compute the minimum distance. A single neuron would require 660 slices, so that up to 16 neurons may be simultaneously implemented (see table C.2). The second approch is based on the use of the L_1 norm. This approach requires about 6190 slices to implement the whole Kohonen map (23 inputs, 5×5 output neurons) see table C.1.

Block type	delay δ	CLB use rate		
		Slices (2 per CLB)	LUT	FF
PPM	-	1	2	-
On-line substracter/adder	2	5	8	4
On-line 23-inputs adder	6	56	98	18
On-line absolute value	1	2	4	-
On-line Comparator Element	-	17	32	3
On-line 25-input Comparator	5	425	797	124
On-line $\|X_i - W_{ij}\|$	4	7	12	8
On-line neuron : $\sum_{i=0}^{N-1} \|X_i - W_{ij}\|$	10	241	401	214
ROM (W_{ij} storage)	-	3	4	-
Kohonen SOM : 23 inputs, 5×5 outputs [On-line]	23	6190	10701	5275
Kohonen SOM : 23 inputs, 5×5 outputs [LSBF]	34	12286	14557	17659

Tab. C.1 – On-line implementation performances of Kohonen SOMs using L_1 norm for select the winner neuron

Block type	delay δ	CLB use rate		
		Slices (2 per CLB)	LUT	FF
On-line square [GT00]	3	26	45	16
On-line $(X_i - W_{ij})^2$	5	31	23	28
On-line neuron : $\sum_{i=0}^{N-1}(X_i - W_{ij})^2$ en-ligne	10	660	582	690

Tab. C.2 – On-line implementation performances of Kohonen SOMs using L_2 norm for select the winner neuron

Consumption analysis

With the minimal frequency of the board, 400 KHz, computation time is 57,5μs, which is still largely sufficient for the real-time constraints of hypovigilance detection. Consumption is then of 27 mW (1000 to 2000 times less than with current processors). Taking into account a decision process made every second in the daytime, consumption remains lower than 1.5 μW.

C.2 Implementation of the discrimination of the states of vigilance

As shown in [BKGAB04], the use of a SOM-based decision process has led to a very satisfactory implementation on a FPGA circuit (Field Programmable Gate Array). Nevertheless, we have preferred until now this approach mainly because of its ability to be implemented in a rather straightforward way : the simplicity of the neural architecture (a 5x5 Kohonen map) may be mapped as a parallel architecture on a large FPGA, though it requires well-chosen arithmetical and technological choices.

The use of a MLP-based decision process requires a quite large neural architecture (23 inputs, at least 10 hidden neurons and 8 output neurons), which direct parallel implementation on a FPGA is not possible. Therefore, this solution has been first considered as unsuitable for our implementation requirements.

This [section] shows how a FPNA neural network has been derived from our rather large MLP,

by means of the principles of [Gir06b]. The obtained FPNA results in highly satisfactory alertness decision performance, while its simplified topology may be easily mapped on an FPGA device, using the pipeline implementation method described in [Gir06b].

Multilayer perceptron

The MLP model is the most used and well-known neural architecture. It consists of several ordered layers of neurons. Two consecutive layers are fully connected. A neuron computes a weighted sum of the outputs from the previous layer. Then it adds a threshold and it applies a transfer function to the sum. The learnable parameters of this neuron are the synaptic weights and the threshold.

MLP learning is often performed thanks to what is called the *back-propagation* learning algorithm. This well-known algorithm is indeed the combination of the back-propagation method (for gradient computing) and of the gradient descent algorithm. Back-propagation allows an efficient computation of the gradient of an error function with respect to the parameters of a neural network. As soon as they are computed, the gradient values are used to modify the parameters to make the neural network perform the chosen task.

In our study, several MLP architectures have been tested. Satisfactory results have been obtained with an MLP that uses 23 inputrs, one hidden layer of 10 neurons, and 8 output neurons (standard coding for 8-class discrimination). Every neuron applies the same transfer function, which is a sigmoidal function. The *tanh* function has been chosen. This architecture is shown in figure C.5 (see section C.6 for the thick connection).

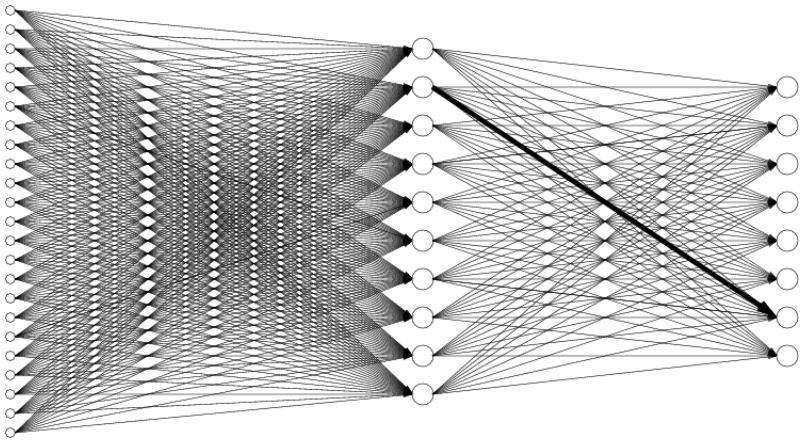


FIG. C.5 – MLP for vigilance states classification

Results

Our connectionist tools have been applied to portions of EEG signals for various subjects. Different tasks have been studied : vigilance states classification or discrimination, recognition of artefacted states, classification of sleep and awakening states (see [BKBB$^+$03a]). We report here the results for alertness decision.

For each subject, a training and a test corpus (for learning and generalization phases respectively) were built gathering states Cace and Cawoe as well as states Drow and Stg1. Therefore there are only 2 states of vigilance : Awakening and Sleep (artefacted states Art-Cawoe, Art-Cace and Mv are not taken into account). The network architecture includes 23 units in the input layer, that represent the 23 spectral bands.

With the SOM-based approach (5x5 map), a global success rate of roughly 77 % is reached (correctly classified states in the generalization phase). With the MLP-based approach (23x10x8), this global success rate reaches 88 %. More detailed results may be found in [BKBB+03a].

C.2.1 Technological choices

On-line arithmetic

Two main kinds of serial arithmetics are available : LSBF (least significant bit first), and MSBF (most significant bit first). The only existing radix-2 MSBF serial arithmetics is called *on-line* arithmetics. It uses a redundant number representations system, thanks to which any carry propagation issue may be avoided. For more information about this very specific arithmetics, see [ET77]. Our implementation uses this arithmetics, since a MLP uses sigmoidal functions that can not be computed in a LSBF mode.

Computation precision

Software simulation must be performed to study the precision that is required by a neural application before its hardware implementation. Precisions must be studied with respect to differents kinds of data : weights, inputs, neuron outputs, internal computations, ...

It must be mentioned that implementations based on serial arithmetics might be more easily extended to larger precisions than implementations based on parallel arithmetics. It would mainly induce a linear increase of the implementation area of multipliers and elementary functions, and a modification of control time intervals.

Implementation of the MLP

The straightforward parallel implementation of the 23x10x8 MLP is simply not possible. Each connection requires a multiplier, each neuron requires a global adder and a sigmoidal transfer operator (such operators only exist in MSBF mode). Therefore, more than 50000 logic cells (see [Gir06b] for the implementation requirements of on-line operators). Moreover, complete connection schemes between layers can not be handled by the routing structure of FPGAs.

C.2.2 FPNA architecture and learning for alertness decision

Based on the systematic approach of [Gir06b], a FPNA has been built to be functionally equivalent to the 23x10x8 MLP used in our application. This FPNA is shown in figure C.6. It uses as many activators as there are neurons in the MLP. The local connections between neural resources are :

- inter-layer links are connected to their input and output activators, as well as to the intra-layer links of their output node (to broadcast inputs inside layers)
- intra-layer links are connected to their output activator, and to other intra-layer links so as to forward data inside the layer

Virtual connections are illustrated in figure C.6 by the successive links that appear thick : the combination of these connected resources creates a virtual connection that corresponds to the MLP thick connection in figure C.5.

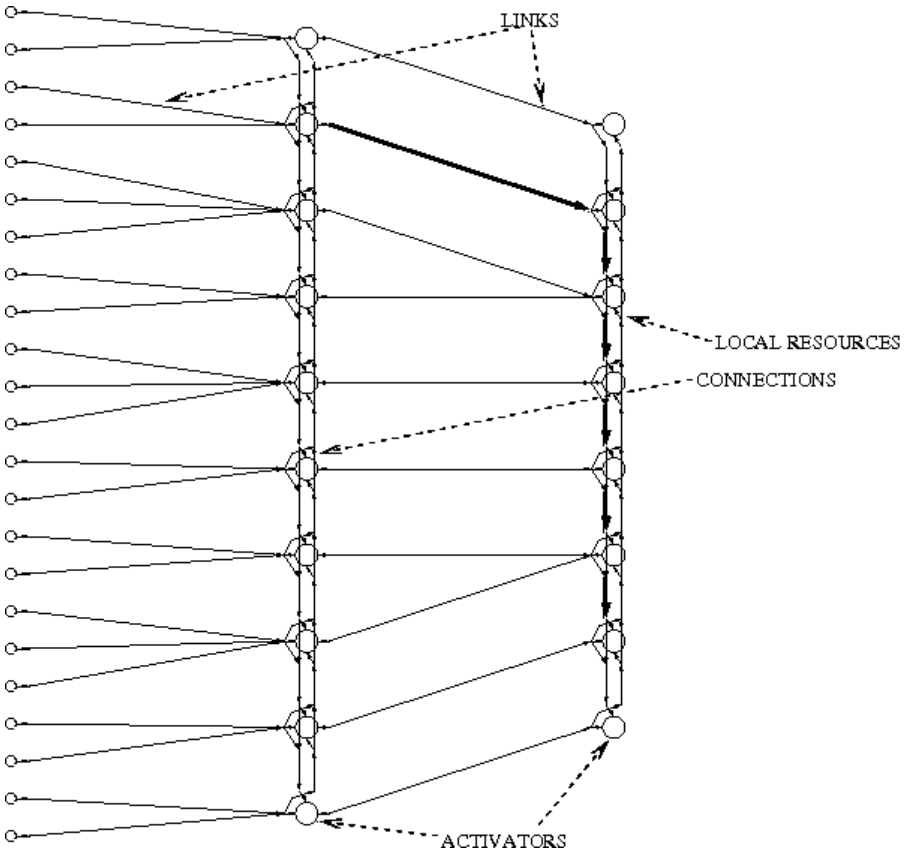


FIG. C.6 – FPNA for alertness decision

The parameters of this FPNA have been tuned by means of a gradient descent algorithm. The global success rate for alertness decision is 86 %. It is already very close to the performance of the MLP, and far better than with the Kohonen map. Further learning improvements (for example using second order learning algorithms such as BFGS, that might prove more efficient on complex models such as FPNAs) might lead to FPNAs with the performance of the initial MLP.

Implementation

The pipeline implementation method of [Gir06b] applies to the synchronous FPNA of figure C.6. On-line arithmetics is used. Each FPNA link is mapped onto a multiplier and a small adder which fan-in is equal to the number of incoming local connections (always less than 3). Each activator is mapped onto an on-line *tanh* operator and a small adder (maximum fan-in equal to 3). Flip-flops are added so as to delay the different inputs of each adder : incoming data must be synchronized according to [Gir06b].

Implementation results

Our implementation uses 12-digit precision operators. The total number of logic cells (including the storage of the weights) is 16440. The global delay is 92. With the minimal frequency of the board, 400 KHz, the computation time is 250μs, which is still largely sufficient for the real-time constraints of our application. Consumption is then of 82 mW. Taking into account a decision process made every second in the daytime, consumption remains lower than 10 μW.

It must be pointed out that these results have been obtained through simulations of a modular VHDL description of the system. An on-chip validation is still required.

Annexe D

Implantation FPGA du modèle IF-LEGION

Cette annexe est directement extraite de [GTH07]

[...]

D.1 Introduction

Computational neuroscience is a very active field of research. It aims at modelling, simulating and understanding mechanisms that underlie neural processes such as perception, action, learning, memory or cognition. Currently, one of the main lines of research of computational neuroscience studies the properties of the low-level mechanisms that are used by neural systems. In this approach, the computation mode is organized around neuronal events called spikes [GK02]. Biological arguments indicate that some vision tasks, as well as most olfactory tasks, cannot be satisfactorily performed by standard neural models that use an analog computation mode, where values stand for mean firing rates of natural spiking neurons [Wan05, HM05, Abe82, RWSB98]. The temporal coding that lies in spikes has to be fully exploited, for example through the notion of neuron synchronization [Sin03, Maa03].

On the other hand, the fundamental properties of neurons seen as processing units is a major source of inspiration to adapt processing architectures to algorithms and to embed neural processing in fine grain distributed systems. Artificial neural networks, considered as models of parallel computation, may represent an alternative to standard computation models for machine intelligence and intelligent behavior [Col05]. This claim is based on both theoretical and practical properties of neural networks. Their ability to deal with strong implementation and application constraints is a major current research domain to find ways to map neural connectivity and functionality onto hardware through architectures inspired by such massively parallel and hierarchical processing. The hardware plausibility is related to a very fine grain parallelism that fits parallel hardware devices, as well as to the emergence of very large digital reconfigurable systems, such as FPGAs (Field Programmable Gate Arrays) that become able to handle both adaptability and massive parallelism of neural networks.

These two major trends of computational neuroscience research (spiking models and embedded

neural systems) are now mixed in several works. One of the projects carried out by our team is to develop a fully neural system for autonomous robotics. In this context we mainly develop two kinds of spiking models : advanced spiking models that model the olfactory bulb, and spiking neural networks that perform a rough real-time analysis of the robot visual environment.

Our models for visual perception are based either on adapted standard spiking models or on our recent works about multimap spiking models using principles inspired by neural field theory [Tay98]. Simultaneously, we carry out several attempts to evaluate our preferred type of hardware device as a possible implementation basis for massively distributed spiking models. Following our works about adapted standard spiking models for visual perception, we have first implemented a well-known though criticized spiking model for image segmentation on FPGAs : the integrate-and-fire LEGION model ([TW95a, TW95b], as well as [WT97, CWJ99] for its adaptation to grey-level images). LEGION is criticized due to its rather simple biological plausibility and the mathematical/computational problems for hardware implementation feasibility, but the underlying 2D structure of this model fits the topological constraints of FPGA implementations.

Our work mostly focuses on low-area solutions, based on the use of a standard serial arithmetic within pipelined loops. As a result, a fully parallel implementation of the LEGION network has been mapped onto a Xilinx Virtex FPGA device, large enough to handle low-resolution robot image sequences. This work has confirmed the assertion of [NS92, Gir00c, Gir00a] : FPGA parallelism fits neural computation, so that FPGA implementations of neural networks may favor the use of neural approaches that keep fully distributed and localized processing. In our case, steady communication channels, synapses, have been preferred to event-driven systems. Section D.2 briefly discusses related works : FPGA implementations of spiking models as well as event-driven implementation methods. We will justify our choice of an embedded reconfigurable system based on FPGAs as the hardware implementation option in section D.3, and we will describe its possible limitations as an implementation device for spiking models. The theoretical foundations of the LEGION model, its derivation using integrate-and-fire neurons, and its adaptation to the segmentation of grey-level images will be described in section D.4. The global architecture of the hardware implementation of the spiking neural model and its different parts will be detailed in section D.5, whereas section D.6 will provide the main features of our hardware mapping onto the Xilinx Virtex family.

D.2 Related works

Related works are recent and still very few. A single simple neuron is considered in [WMLB00]. In [RHTF03, UPRS05] implementations are still based on sequential processing and parallel arithmetic, so that their lack of scalability makes them unable to take advantage of the strong technological improvements that are a major aspect of FPGA solutions. Our work is closer to an almost simultaneous study that is described in [SvC06]. Both works share common implementation principles : serial operators and massively distributed implementation of spiking neurons. Even results are very close. Yet both works are more complementary than competing. In [SvC06], the authors develop an implementation method for a partially generic model of integrate-and-fire spiking neuron. Their work stands as a very interesting step towards an automated design of spiking models on FPGAs, but it does not take into account several aspects of the computation when real data are considered. Our work is more specific to a given model (integrate-and-fire LEGION), which results in a more compact but less generic implementation, that is able to perform all required computations for the considered application (visual segmentation), including the complex image preprocessing that is

mandatory for such a model.

In this paper, as well as in [SvC06], a time-discretized computation has been preferred to an event-driven approach. Event-driven simulations of spiking neural networks have been proposed in [Wat93], and they have been extended in [RM03]. Instead of using fixed time steps to update neurons, the state of the whole network is computed at event (spikes) occurence times. Such methods provide a better accuracy in the simulation of the continuous phenomenons that generate spikes. But they are not always easy to implement, and they require that the differential equations of the neuron state lead to an analytical solution when determining the time of the next event. Moreover, in terms of computational efficiency, these methods become unefficient when the average spike activity increases, which might be a problem for real-time applications. Another main drawback makes event-driven methods unsuitable for our work. Despite several attempts with parallel computers, their natural lack of parallelism lies in the process of selecting the next neuron that fires so as to know the next update time for the whole network, whereas our implementation principles are based on a fine-grain parallel mapping of the neurons with local interactions.

D.3 Choosing Field Programmable Gate Arrays

D.3.1 Neural computations on FPGAs

[... cf section 1.3 ...]

D.3.2 Spiking models on FPGAs

Most works on neural network digital hardware implementations use a group of binary digits (words) to implement classical neural models where all interactions are represented by the mean firing rate of the neurons [MJH+03, ABR05]. On the other hand, recent research in neuroscience has developed spiking neural models that are much closely coupled with action potential and spikes communication and even more tightly related to the brain abilities than classical neural models. Spiking models suggest that there is significant information encoded in the relative timing between pulses and this information is partially hidden by the inherent averaging in computing firing rates. Spiking neural models are different from classical connectionist models in the sense that information is transmitted by means of pulses or spikes through time rather than average firing rates. The spike-time interrelationship enriches the dynamics of spiking models that exploit the temporal domain to encode and retrieve information in the exchanged spikes.

As shown by recent works [RHTF03, UPRS05, SvC06], the implementation of spiking neural networks on digital circuits has become an active line of research, whereas previous implementations were mostly taking advantage of the density of analog devices. This change is mainly linked to the flexibility and rapid technological improvements of FPGAs. Advanced implementation methods (such as the one we propose in this paper or the work of [SvC06]) are now necessary to make digital implementations able to handle large networks of spiking neurons. This requires low-area implementations of this kind of neuron, based on the fact that spiking neural networks are well suited to be implemented in digital logic : they are inherently binary and the time of spike can be handled through digital circuits since communication among neurons is handled by spikes, so that single bits might be used for inter-neuron communication, instead of floating point words. That is, a single wire is sufficient for the connection between two neurons and the routing requirements of neural interconnection on a 2D chip can be simplified. Besides, spiking neurons are weighted

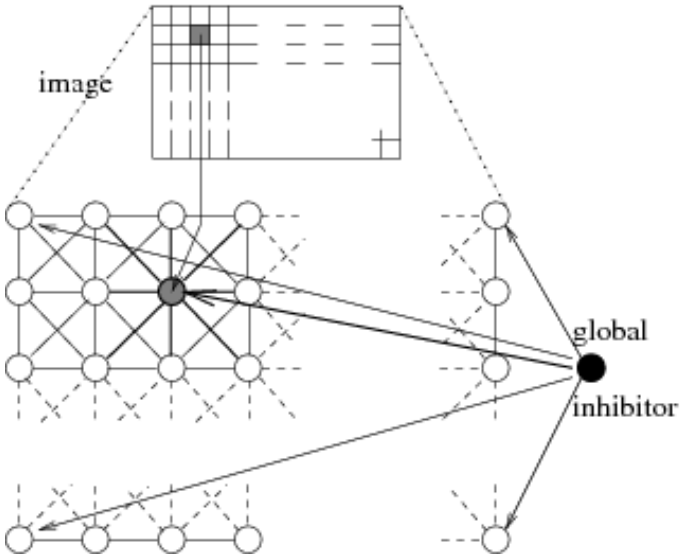


FIG. D.1 – 2D architecture of the LEGION network : a neural oscillator is connected to its eight immediate neighbors and to a global inhibitor. Each oscillator receives visual stimuli from pixels of the input image.

integrators that can fire and be reset as soon as a threshold value is reached, which avoids area greedy operators in the hardware implementation. Combining spiking neural models with specific hardware implementations on FPGA architectures opens new possibilities to build intelligent autonomous systems.

D.4 LEGION Model Description

The choice of the integrate-and-fire LEGION model follows several preliminary works in our team to study how spiking models may be used for visual perception. Moreover, the 2D architecture of this model makes it particularly well-fitted for a straightforward parallel implementation on FPGA.

D.4.1 Principle

Oscillations of neurons in the cortex are now considered as a important mechanism that is involved in several cognitive functions of the brain. Our research team is more particularly interested in the role that this mechanism plays in perceptual tasks such as vision and olfaction [vdMS86, vdMB92, Abe82, LK98, HM05].

Several studies have shown that oscillations of neurons take part in several perceptual functions of the cortex. These oscillations are used in visual perception to segment features in a visual scene [LK98, TW95a, CWJ99]. Neurons in cortical areas such as primary visual area V1 are known to "see" only a small localized part of the visual field (such areas are retinotopically organized). And yet synchronized behaviors have been detected for groups of neurons which visual fields are strictly separated. These neurons bind together thanks to the synchronization of their firing activities.

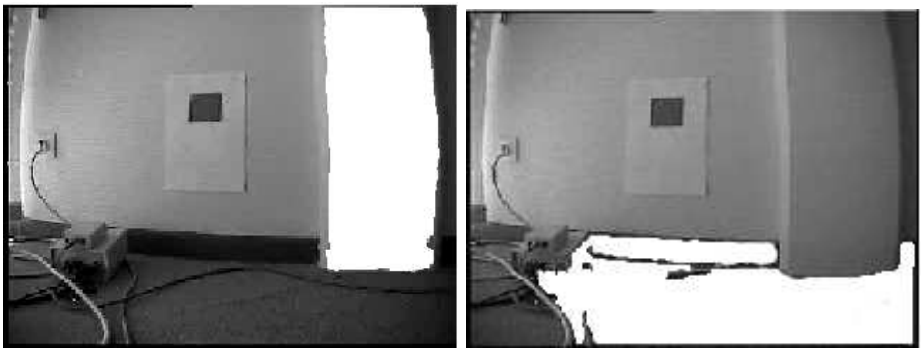


FIG. D.2 – Image segmentation by an integrate-and-fire LEGION network : simultaneously spiking neurons correspond to the white pixels

Such behaviors cannot be studied by means of standard neural network models that only take into account the mean firing rate of the neurons. This is why one of our research goals deals with neural elementary mechanisms.

The LEGION network (Local Excitatory Global Inhibitory Oscillator Network) has been proposed in [TW95b] based on biological considerations. This simple model consists of a 2D grid of oscillators (see figure D.1 where an oscillator is connected to its eight immediate neighbors) that receive visual stimuli. Neighboring oscillators are coupled by excitatory connections, and a global inhibitor gets active when any oscillator jumps up. The excitatory/inhibitory connections result in synchronizations (through excitation) of distinct (through inhibition) groups of oscillators so as to segment the input image. This model has been applied to the segmentation of binary and grey-level images [WT97].

The original LEGION model uses relaxation oscillators [TW95b] as basic computational units. Each of them is modeled by two coupled units, an excitatory and an inhibitory one. The excitatory unit is connected to both input stimulus and neighboring oscillators, whereas the inhibitory unit interacts with the excitatory one to produce oscillations. Exact equations may be found in [TW95b].

In order to segment images, the LEGION model groups oscillators that receive their input from similar features in an image. Oscillators group together by synchronization of their phase thanks to excitatory connections, and they get desynchronized from other groups of oscillators by means of global inhibition (figure D.2 shows a similar phenomenon with the integrate-and-fire LEGION network described below : in this image taken by our robot camera, pixels in white correspond to a group of simultaneously firing neurons, thus segmenting a distinct part of the wall, resp. floor). When an oscillator jumps up, it excites its neighbors, which immediately jump up if they were about to. Excitation propagates and groups get formed. Global inhibition ensures that only one group can get active at a time.

D.4.2 Integrate-and-fire LEGION

After having been applied to binary image segmentation, and then to grey-level image segmentation [TW95b, WT97], the LEGION architecture has been chosen in [CWJ99] as an interesting model to study synchronization and desynchronization of integrate-and-fire oscillators (or spiking neurons). Beside the interest of this approach so as to study the complex properties of assemblies

of such oscillators, it provides us with a model that keeps synchronization properties while being suited for digital VLSI implementation.

There are four computational components of LEGION that describe its behavior : group participation, group formation, group segregation, and group suppression. Each computational component is implemented by the following network components : input stimulus, excitatory coupling, global inhibitor, and neuron potential, respectively. The dynamics of a LEGION network of integrate-and-fire neurons is defined according to the following equation :

$$\frac{dx_i}{dt} = -x_i + I_i + \sum_{j \in N(i)} \frac{\alpha_{ij}}{Z_i} P_j - G \tag{D.1}$$

where the sum is over the neurons in a neighborhood, $N(i)$, around neuron i, x_i is the potential of neuron i, Z_i is the number of nearest neighbors that neuron i has (to take into account boundaries problems). α_{ij} is the coupling strength. P_j is a neighbor neuron that fires at a given time producing an excitation pulse to neuron i. G is an instantaneous inhibitory pulse to the entire network when any neuron in the network fires. When $x_i = 1$ the neuron is said to fire ; its potential is instantly reset to 0, and it sends excitation to its neighbors.

The parameter, I_i is the external stimulus given to neuron i and when applied to image segmentation it depends on the input image. In order to segment grey level images, the parameter I_i is computed as follows [WT97]. Let p_i be the intensity of pixel i. If $|p_i - p_j|$ is less that a given threshold, then the two pixels are said to satisfy the pixel difference test. Two neurons have a nonzero coupling strength only if they are neighbors and if their pixels satisfy the pixel difference test. The weights of the connection strengths α_{ij} are determined by the number n_i of neighboring pixels of i that pass the pixel difference test, $\alpha_{ij} = \frac{\alpha}{n_i}$ if neurons i and j satisfy the pixel difference test, where α is a constant. If neurons i and j do not satisfy the pixel difference test, α_{ij} is set to zero. If half of the pixels in $N(i)$ satisfy the pixel difference test, then I_i is set to a value I_L greater than 1 (leader value). If no neighboring pixel satisfies the pixel difference test then I_i is set to zero. Otherwise, I_i is given a stimulus I_N (near-threshold value), which is less than but near 1.

It has been demonstrated that integrate-and-fire LEGION maintains its segmentation capabilities since their synchronizing properties are similar to relaxation oscillators [CWJ99]. Due to the oscillatory characteristic in LEGION, a large amount of differential equations need to be solved, which induces a high computational load and power consuming task in conventional processors. The use of integrate-and-fire neurons are attractive for digital VLSI implementation for perceptual organization in embedded systems.

D.5 LEGION Hardware Implementation

D.5.1 General architecture

Due to the high computational requirements of LEGION, the highly dense interconnectivity and the area greedy operators for a fully parallel implementation, we have chosen a serial hardware implementation on massively fine grain parallel structures as a suitable architectural option for embedded connectionist processing for visual perception.

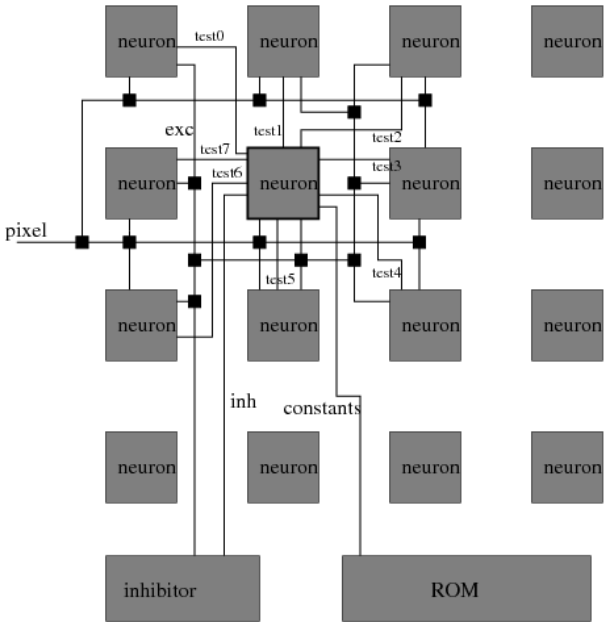


FIG. D.3 – Simplified architecture of a whole 4x4 integrate-and-fire LEGION network

Arithmetic and precision

The fine grain pipelined internal structure of the proposed digital architecture was implemented using fixed point arithmetic. The pixels of the input image were coded in 8 bits for gray level images. Analysis and experimental results were carried out to determine a suitable wordlength in terms of performance and cost for the coding of the values involved in the computation of neuron potentials. Considering the various ranges of handled data (internal states, update values, etc.), a minimum precision of 10 bits (with two bits for the integer part) is required. We used time-discretized software simulations of the LEGION network using fixed-point computations to estimate the precision required to perform a segmentation similar to our event-driven software implementation using floating-point computations (same groups of neurons simultaneously firing for small images).

For this FPGA LEGION implementation, it appeared that a serial arithmetic with a 12-bit fixed point representation should be chosen in order to favor a high density of neurons per silicon area while preserving accuracy and performance of the model. A 2-complement representation with 2 bits for the integer part and 10 bits for the fractional part is used for the internal representation of neuron potentials. Larger wordlengths can result in a more accurate synchronization and convergence at the expense of a slightly different control and a slower implementation. It must be pointed out that our architecture may handle up to a 16-bit fixed point representation without any architectural change (only control signals must be updated). Larger precisions are also possible at the cost of a more complex handling and storage of the neuron potential and constant values.

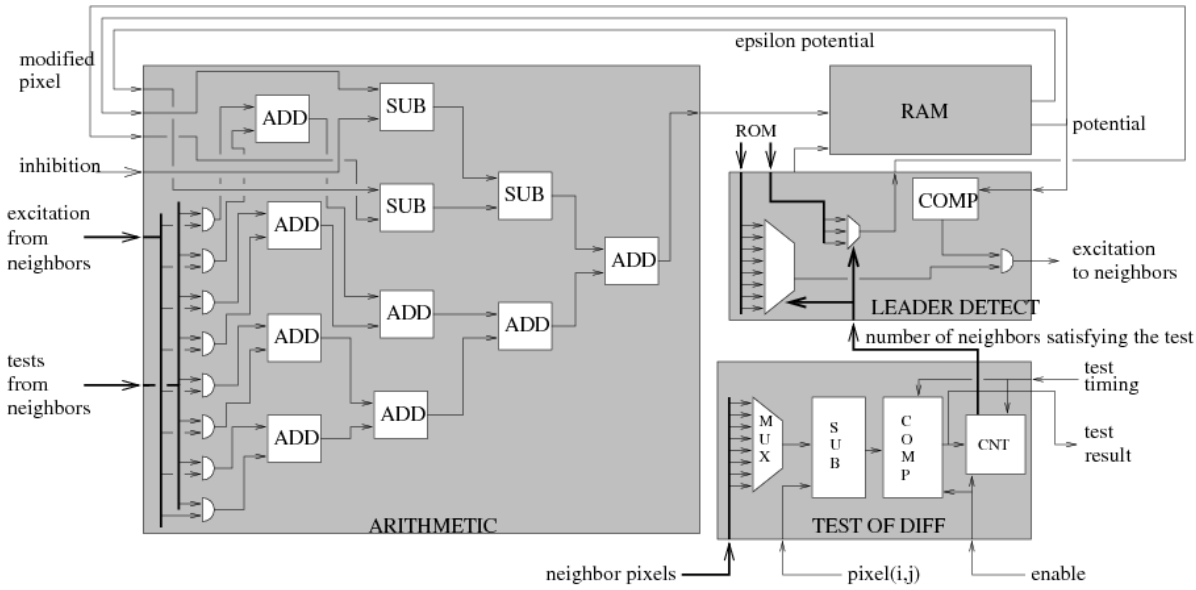


FIG. D.4 – Architecture of a single neuron

Architecture of the network

Figure D.3 illustrates the general architecture of the implementation for a 4x4 LEGION network. It consists of a grid of 4x4 identical neuron modules, a global inhibitor module, and a ROM module to store several constants. All neurons use the same implementation, but at the border each neuron module adapts to the number of neighbors as required by the equation in D.4.2.

Each neuron uses loop computations within an internal pipeline scheme. The control of these computations are synchronized in the whole network so that neurons may serially communicate their excitations to their neighbors. In order to simplify the block-diagram of figure D.3, only few buses and wires are shown : the excitation computed by the neuron in position (2,2) are sent to all its 8 neighbors and to the global inhibitor module, the input pixel signal received by neuron (2,2) is also received by its 8 neighbors, the tests of difference computed (and sequentially stored during the pixel difference test phase) by neuron (2,2) are sent to the corresponding neighbors, and finally neuron (2,2) receives the global inhibition signal and some constants from the ROM module. In the following subsections, the general hardware architecture for the LEGION neuron model and its main components will be described in some detail.

Architecture of a neuron

The general architecture for the LEGION hardware neuron model is shown in the simplified block diagram of figure D.4. The hardware neuron model has 1-bit inputs and outputs to exchange data among neurons and to the external world. Inputs are mainly used to receive image pixels and excitation spikes from neighboring neurons and outputs are used for outgoing spikes. All integrate-and-fire neurons used in the network are architecturally identical although they may be functionally different on the image boundaries since some inputs are nonexistent and no wrap-around is used. All the required logic for controlling the behavior of a neuron is built inside each neuron, though it

is based on received external signals, such as reset and clock, for synchronizing and data exchange. The dynamics of every integrate-and-fire neuron in the LEGION network is tuned by constant parameters, such as the α_{ij} and I_i parameters in equation D.1. These parameters are received from global registers and memories through a lookup table approach, that avoids the hardware implementation of area greedy operators, multiplication and division.

The proposed hardware neuron model is constituted by four main modules : a test of difference, an arithmetic, a RAM and a leader-excitation detector module. The key idea for the conception of the proposed neuron architecture relies on the association of basic operators to specific arithmetical computations of equation 1, the use of some previously stored constants to avoid hardware complexity of nonlinear operators and in the transformation of the differential equation into an equation of differences using the Euler's method. The functionality of the main modules and their implementation details are described below.

Pixel Difference Test Module : The difference test module determines the number of neighbors that fulfill the requirements of difference test. It is essentially constituted by a multiplexer, a subtractor, a counter and a comparator optimized for serial arithmetic.

The difference test is applied between the pixels of two neighbor neurons, the central pixel and the current pixel selected by the multiplexer. The absolute difference of these values is compared to a constant threshold, 16 for the current implementation (any other power of 2 value would lead to a similar architecture). If the absolute difference is below the threshold, then pixels satisfy the test of difference. The main goal of applying the test of difference is to define groups of neurons with non-zero coupling connections. A neuron that satisfies the difference test with most of its neighbors is part of a homogeneous region and tends to be synchronized through time with neurons of this region. The data flow is controlled by signal `Enable` that starts the module functionality and by signal `Test timing` that provides the timing slot to perform the difference test on each neighbor pixel. The module operates sequentially with all its neighbors and then it sends the number of neighbors that satisfy the test to the leader detector module for further utilization.

Arithmetic Module : This module performs the arithmetical operations that define the dynamics of the neuron model, as stated in equation D.1. The module is composed of a tree of serial adders and subtractors, and AND gates. The arithmetic module receives excitation signals from its neighbors, the global inhibition, and the previous internal potential stored in the RAM module. An integration step $\epsilon = \frac{1}{4}$ is used to solve the difference equation through the Euler's method. To avoid multipliers, the potential and ϵ times the potential are simultaneously provided by the RAM module using an appropriated addressing scheme. A set of AND gates verifies if a neuron has produced a spike and satisfies the difference test and labels this neuron to compute its contribution to the potential. The arithmetic module outputs the current neural potential to the leader detector module.

RAM Module : This module is basically constituted by a 16x1 dual port memory RAM that stores the neuron internal potential produced by the arithmetic module on each clock cycle. The two bus addresses are driven by two global counters, delayed 4 clock cycles one of each other, to provide the current neuron potential and the potential scaled with the integration step ϵ.

Leader Detector and Excitation Module : The leader detector module determines the modified value of a pixel associated to a neuron, and generates the excitation potential contribution at a given time, and the excitation spike. Three possible values for the modified pixel may

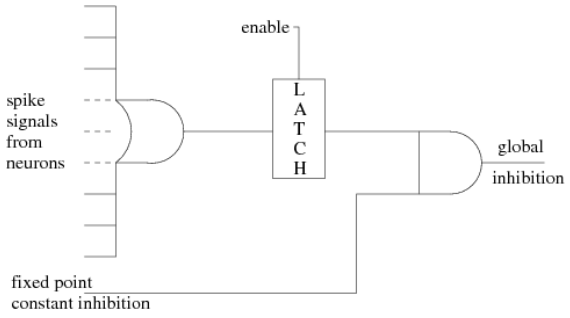


FIG. D.5 – Block diagram of the global inhibitor which produces a global inhibition for the neurons if any excitation/spike is generated

be assigned (see section D.4.2) : $I_L = 1.25$, $I_N = 0.95$, and 0. Also, the excitation potential contribution depends on the number of neighbors satisfying the difference test. The module computes this strength, selecting values previously stored in an internal memory in order to avoid multipliers and divisions. The module generates an excitation pulse and a reset signal for the internal state of the neuron, produced by the comparator component, if the internal potential is greater than the unit.

Complementary modules

Figure D.5 shows a block diagram of the global inhibitor used in the LEGION network. The inhibitor receives the excitation signals from neurons as inputs. If at least one excitation is present, then the inhibitor outputs an inhibition signal which is applied to all the neurons. As the detection of excitations must be done in a proper time, an enable signal stores the output of the OR gate into a latch. The latch output is sent to an AND gate which is enabled for the time required to transmit, serially, the fixed point value of the global inhibition value stored in a global memory. If no excitation is generated, then the inhibition is null.

Several LEGION parameters are stored in a global ROM memory module organized in several banks as shown in figure D.6. RAM memories of 16x1 are used to store the fixed point representation of parameters such as inhibition, leader and near-threshold values. A set of eight RAMs, enclosed by the broken line rectangle, are used to store the values of α times the reciprocal of the number of neighbors which are required to compute the excitation contribution to the neuron potential. The use of this approach allows reducing hardware complexity avoiding division in the neural oscillator model at the cost of storage and the use of global communication. All the RAMs use the same address bus which is generated by a counter in the global control module in LEGION (the constant values must be sent serially).

D.6 FPGA Implementation Results

The proposed hardware model for the neural oscillator and LEGION has been successfully modeled in Very High Speed Integrated Circuit Hardware Description Language (VHDL) and implemented in a FPGA device. The VHDL model for the LEGION is configurable and it may be built up for different gray level image sizes.

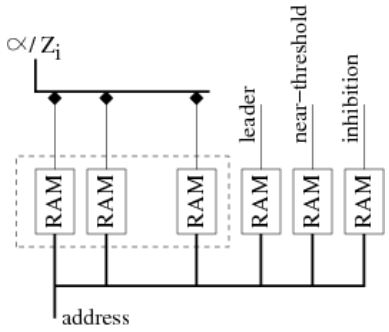


FIG. D.6 – Block diagram of the ROM module to store constant parameters of LEGION and the values of α times the reciprocal of the number of neighbors to avoid divisions

Parameter	1 neuron	8x8 LEGION	16x16 LEGION	26x26 LEGION
Slices	44/33792 (1%)	2615/33792 (7%)	10700/33792 (31%)	25760/33792 (76%)
4-input LUTs	67/67584 (1%)	4163/67584 (6%)	17019/67584 (25%)	46710/67584 (69%)
Flip-Flops	22/67584(1%)	1361/675840 (2%)	5617/67584 (8%)	15549/67584 (23%)
Max Clock Freq.	112.38 MHz	100.26 MHz	77.22 MHz	64.45 MHz

TAB. D.1 – Virtex Post-Place and Route Synthesis Results for a VIRTEX-II Device Implementation (XC2V6000-4FF1517)

D.6.1 Synthesis results

The synthesis results for the neuron hardware model and for different configurations of LEGION targeted to a Xilinx Virtex XC2V6000-4FF1517 device are summarized in table D.1. For each kind of FPGA resource and for each implemented model, the global usage ratio is given as a percentage. The design was synthesized, placed and routed automatically in Xilinx Foundation ISE 7.1i. It is important to point out that when the LEGION size increases then the maximum clock frequency is reduced considerably since more routing resources are required which limits the performance.

The hardware resource utilization summary highlights that the massively parallel and pipelined implementation is highly attractive for FPGA implementation. According to the post layout timing information in table D.1, a 26x26 LEGION can reach a maximum clock frequency of 65 MHz for segmentation times in the order of microseconds. This is a consequence of the use of pipelining registers that shorten the interconnect delay in the critical path. Due to the interconnect centric nature of LEGION, the hardware performance depends on the FPGA family chosen since routing is expensive in terms of interconnect delay. In this sense, a segmented routing FPGA architecture is beneficial [OW05, TWC05]. As can be inferred from the presented results, the proposed digital approach for LEGION implementation is particularly efficient from the device utilization point of view.

D.6.2 Simulation environment

To validate the functionality of the proposed digital hardware architecture for LEGION, both functional and timing simulations were performed. The timing simulation data produced in the

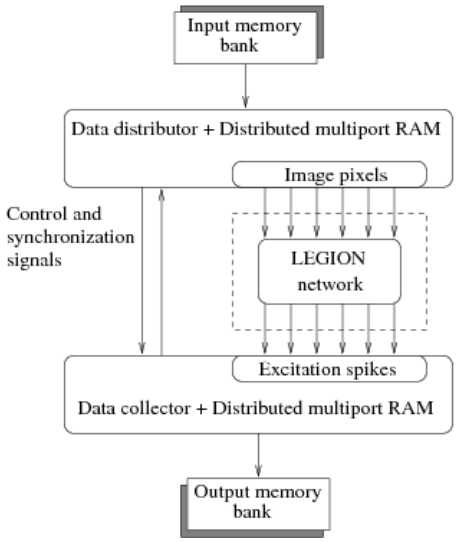


FIG. D.7 – Generic view of the simulation/hardware environment for LEGION network

place and route phase was used for the timing simulation of the design. A set of testbenches were developed to validate each component in the neuron model and the LEGION network. A simulated hardware environment for the complete LEGION network model was used. This environment is shown in figure D.7. An input image file used as stimulus is read on simulation time and data are loaded into a RAM VHDL model of an input memory bank. Then, pixels are sent by a data distributor to a distributed multiport RAM model to provide parallel access to all pixels of the input image required by LEGION network. After processing, excitation spikes are collected by the data collector module and stored in a distributed multiport RAM and then sent to an output memory bank for further analysis.

This simulation environment was developed keeping in mind the common hardware platform organizations that are used for physical implementations where the two main components are an FPGA device and external memory banks for data storage. For on chip testing, the major modification of the simulation environment is the acquisition and transfer of real image data from a camera to the memory banks. In this sense, some modules should be developed according to hardware platform constraints to acquire images from an attached camera. Data transferring to the input memory bank and result transferring to a visualization module or to a host should be developed as well. These aspects are currently being addressed for an experimental setup based on a vision-oriented Celoxica RC340 board (using a Virtex XC4VLX160) we are about to purchase. Additional results and considerations will be available with this board.

D.6.3 Experimental results

The produced results were compared to the software implementation of LEGION providing satisfactory results in the tests performed on the hardware model. The values used for LEGION parameters in this experiment are : 0.2, 0.25, and 0.01 for α, the integration step and inhibition, respectively (see equation D.1). Following section D.5.1, we recall here that this hardware implementation exactly computes the same results as the fixed-point simulation software using the same

0	1	120	249	250
2	122	121	125	251
3	4	116	253	245

P_0	P_1	P_2	P_3	P_4
P_5	P_6	P_7	P_8	P_9
P_{10}	P_{11}	P_{12}	P_{13}	P_{14}

FIG. D.8 – 3x5 input image to test the LEGION architecture : gray-level values on the left, associated pixel numbers on the right

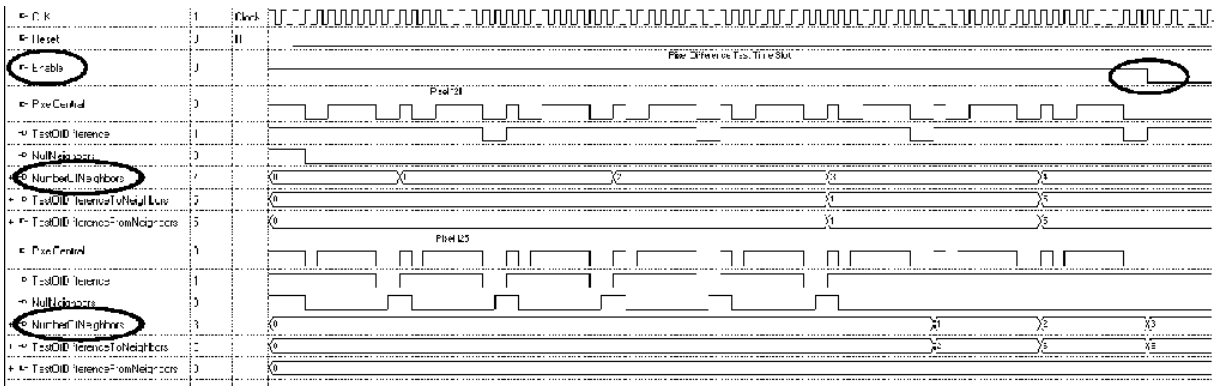


FIG. D.9 – Difference test timing for two neurons

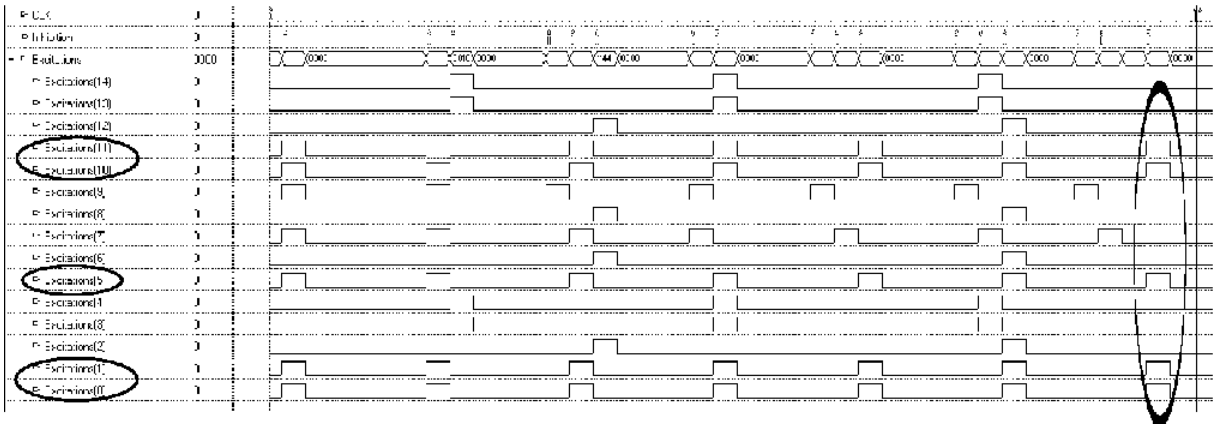


FIG. D.10 – Time evolution of the oscillatory and synchronizing activity in the network

FIG. D.11 – 16x16 input image to test the LEGION architecture : initial image on the left, an example of segmentation on the right

precision, but these results may be different from the floating-point version.

Figure D.8 shows a small 3x5 input image used to exemplify the LEGION architecture operation (top-left) and the pixel-neuron number convention as well (top-right). Figure D.9 shows the difference test timing for two neurons associated to two pixels in the input image, P7 and P8. At the end of the difference test, indicated by a low value in the signal Enable as shown in the figure, the number of neighbors satisfying the test for P7 and P8 are 4 and 3, respectively. Figure D.10 shows a snapshot of the time evolution of the oscillatory and synchronizing activity in the network. For explanation purposes and space considerations, only the excitation signals from neurons, the global inhibition signal and the main clock are shown for a short interval of time. The number associated to the Excitations corresponds to the same pixel number in the input image as shown in figure D.8. The clock signal appears grayed out due to the scale of time is large to show the transitions details. The neuron synchronization can be observed for example if the time between the vertical lines on the left of the time diagram is considered. In this time interval, excitations from neurons 11, 10, 5, 1 and 0, are generated which implies that pixels P11, P10, P5, P1, and P0 are part of the same object in the input image.

Another example is based on the 16x16 image depicted in figure D.11 (left). As shown in figure D.11 (right), this example image is particularly difficult to segment, and the LEGION network mostly succeeds in segmenting the borders of the spiral. This is an expected result because the structure of the image produces no "leader" pixel in the body of the spiral. Therefore the network correctly segments the image by separating the leader pixels from the others.

Regarding performance, the image segmentation time is in the order of microseconds for the current implementation and it is mainly related to two main processing stages : the image preprocessing stage to perform the test of difference and leader detection, and the time to reach synchrony. Due to serial processing the difference test between two neighboring pixels requires nine clock cycles. As neighbors are processed sequentially, eight difference test cycles are required. On the other hand, synchrony is easily achieved after a few oscillation cycles (10 to 10^2). For the current implementation, each oscillation cycle requires fifteen clock cycles to update the neuron potential and to produce an excitation spike if needed.

According to the implementation results in table D.1, a maximum clock frequency of 77 MHz, the segmentation time for the 16×16 test image, is about $3\,\mu s$, which is similar to the processing times reported for analog neuromorphic implementations. This time is based on the assumption that the input image is already in the input memory bank and no overhead for image acquisition and transferring is considered. On the other hand, the software implementation on a microprocessor based computer, Pentium 4, 2 GHz and 512 MB, requires around ten milliseconds to segment the image. Thus, the architecture provides a speed factor up to 1000×, that would even increase with the number of neurons (sequential vs parallel implementation). A more fair comparison is difficult to

establish since some other factors should be taken into account such as the semiconductor technology used for both platforms. The results demonstrate that the architecture performs segmentation at a higher speed while providing additional advantages as area and power consumption in comparison to a simulation on microprocessor based computer.

D.6.4 Modeling and architecture scaling

The experiments described above show good results in the test image segmentation with the specified values for the LEGION parameters. However, more experiments have to be done both for test and real images to analyze the effects of parameter tuning in the image segmentation, as well as the effects of image resolution on the segmentation quality. This problem appears for example when we try to apply a 26x26 integrate-and-fire LEGION network to the image sequence from which figure D.2 has been extracted. It requires first that the images are reduced to a 26x26 resolution. Segmentation results are then far from being as good as in figure D.2. The problem appears to be that a lot of details are lost when the image resolution is reduced. It should be pointed out that the 26x26 limit of table D.1 mostly corresponds to synthesis tool limits (the VHDL code is generic but the synthesis is limited to that configuration for most current devices). However, the size of the reported LEGION implementation is in the average for neuromorphic implementations (analog computations).

If we want to target larger images, the scalability of our architecture must be discussed. The HDL model of the architecture is generic for different LEGION sizes and it could be synthesized for larger neuron arrays if enough hardware resources are available. On the largest current Virtex FPGAs, approximately up to 50x50 neurons might be implemented. The main problems in scaling the architecture in a single FPGA device are : internal distributed memory resources for stimulus and neuron potential storage and routing congestion which slows down the architecture performance. For the current implementation, a set of 8×1 RAM memories are required inside the FPGA to store the 8-bit image pixels, as well as an equal number of 16×1 dual port RAMs to store the potentials of neurons, which rapidly becomes a limiting factor for neuron density due to the limited distributed and block RAM resources in the FPGA. This drawback is similar to that found on its analog neuromorphic counterparts. Moreover, as the number of neurons in the network increases, the routing resources demand increases as well. However, due to the local nature of most connections it is not a limiting factor for neuron density but it slows down performance due to global connections (mainly required by the inhibitor module).

One possible approach to cope with architecture scaling is the partition and the implementation of the architecture in multiple FPGA devices. The main problem to overcome under this approach is the I/O bottleneck of FPGA chip packing technology to physically connect the boundary signals among chips (for example the excitation spikes from neighboring neurons among different chips). With a multi-chip implementation where $n \times n$ neurons would be implemented on each FPGA, an FPGA would need at least $18n$ I/O pads for each side having an adjacent FPGA, and such data exchanges would significantly slow down performance.

A possible hardware solution for inter-chip communication is the use of the Address Event Representation (AER) protocol which uses a time-multiplexing technique on high speed inter-chip digital buses with a reduced number of pins ensuring that only information carrying neurons consume communication bandwidth. However, high rates of spikes produced at a time interval could compromise performance. A detailed analysis of architecture scaling using a multichip approach is currently being addressed, using both an AER protocol and the principle of duplicate excitation

values on each FPGA for the boundary neurons of adjacent FPGAs, so as to reduce performance losses.

[. . .]

Annexe E

Implantation FPGA de l'extraction du flux optique cohérent

Cette annexe est extraite de [THGCS05b]

[...]

The real-time processing of visual motion is a high compute-intensive task and other implementation factors such as power consumption and physical device dimensions limit further real applications development. Several approaches have been proposed to estimate motion from an image sequence, each approach having advantages and drawbacks in terms of computational cost and functional reliability [MNCG01]. Bio-inspired models have been proposed to mimic the computational abilities of the brain for motion perception and understanding. They provide good accuracy but low performance, since conventional processors are not architecturally well matched to these models. Thus, integrated hardware systems are being designed and developed that are strongly inspired by biological neural processing for visual perception of motion [Sto04]. In this work, a bio-inspired connectionist model for motion perception on FPGA is investigated.

[...]

E.1 Bio-inspired model for motion estimation

The approach we use for motion estimation is based on a bio-inspired connectionist model involving highly local and distributed computations. A graphical overview of the main steps of the bio-inspired approach is shown in figure E.1. It involves three kinds of processing : spatial, temporal and excitatory/inhibitory connectionist processing. The spatio-temporal processing is used to mimic the function of the magnocellular cells.

Spatial processing

The first stage of the proposed motion estimation model involves a spatial Gabor-like filtering due to its close relationship with the neurophysiological processing performed by the brain for visual perception [SB02, CSB04]. In the proposed model, Gabor filters are implemented as image convolution kernels with coefficient values selected to emphasize responses in specific orientations

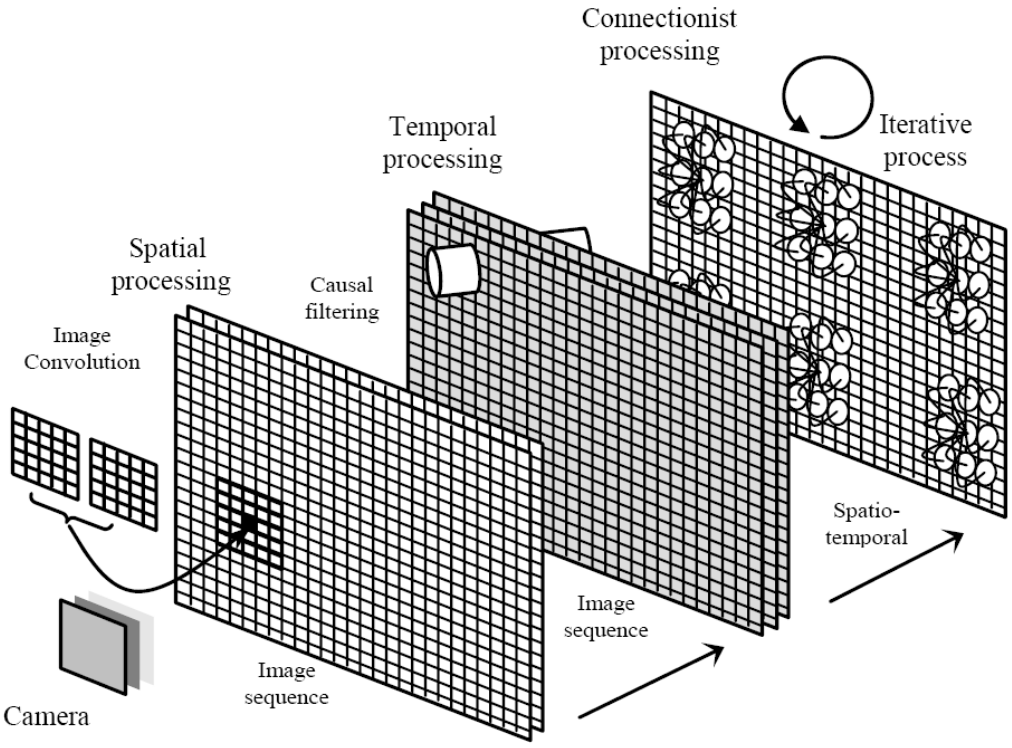


FIG. E.1 – Conceptual view of the main processing steps of the bio-inspired approach for visual perception of motion : spatial, temporal, and connectionist processing.

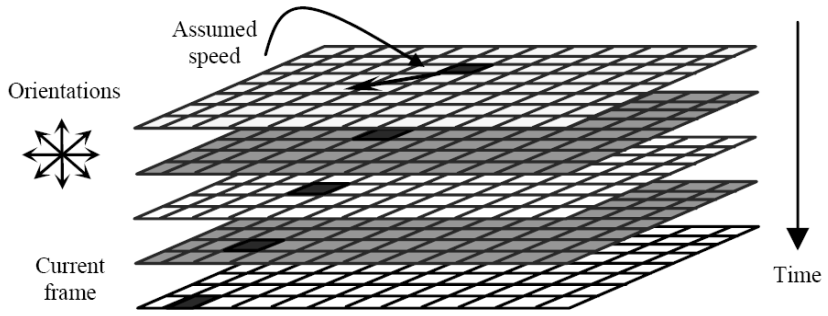


FIG. E.2 – Graphical view of a moving point following a predefined orientation and an assumed speed. The magnitudes of the spatial filters at positions indicated by the black rectangle are used to compute the average.

to mimic the columnar organization in V1. A set of discrete 7x7 Gabor-like filters are applied for eight different orientations to resemble neurons tuned to the main 8 orientations. The convolution operation leads to the potential exploitation of the inherent data parallelism to speed up the computations.

Temporal processing

As described before, neurons in MT respond strongly to a visual stimulus in a preferred direction and with a preferred speed. MT neurons are directly connected to V1 and present a similar organization. Essentially, the function of a speed selective neuron is constructed by summing up the responses of a set of V1 neurons, both over orientation and time. MT neurons sum the responses of V1 neurons with receptive fields positions inside a local spatial neighborhood defined through time with respect to a considered speed of the visual stimulus. Basically, the temporal processing involves the computation of a temporal average using the pixel values of the images in a sequence considering different orientations and for each orientation a set of search areas according to assumed speeds of pixels [CSGA04]. On the assumption of a high sampling rate, it is possible to consider a constant speed of moving pixels between consecutive frames of a sequence without abrupt changes in orientation. As a consequence, for a given assumed orientation and speed, it is possible to emphasize local motion through the averaging of intensity levels at expected positions in the image sequence as indicated by the assumed speed. The basics mechanisms of temporal processing is shown schematically in figure E.2, where the position of a moving point, following a given orientation, is shown in different images at different instant times. In the proposed connectionist approach a set of five speeds for each orientation is considered.

Excitatory/inhibitory processing

When neural networks are applied in image processing related tasks, excitation-inhibition is used as a mechanism to control the network activity and to generate coherent emergent responses based on mechanisms similar to those of winner-take-all neural networks. In such networks, a neuron receives both excitation-inhibition signals from neurons in a neighborhood or influence range, to regulate its activity according to a predefined interaction mechanism. Usually, the interconnections

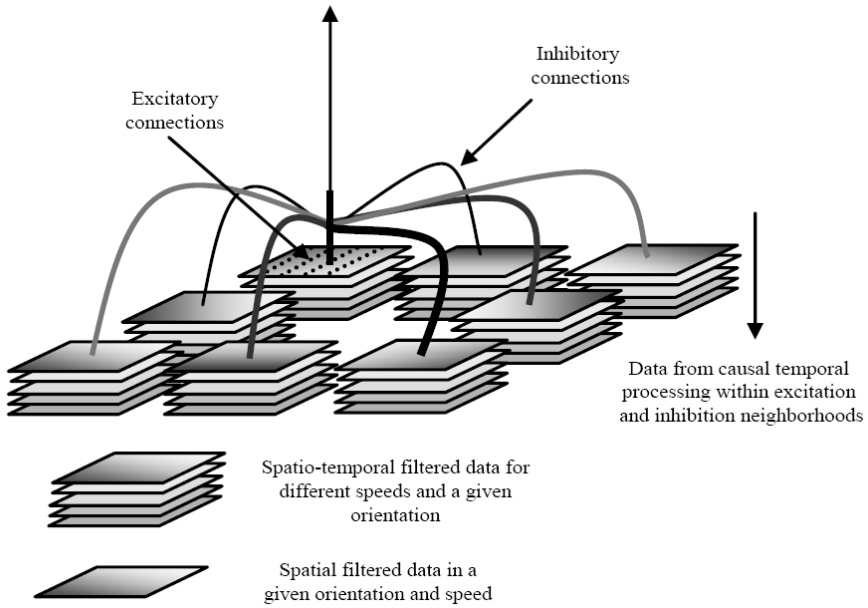


FIG. E.3 – Graphical view of the underlying excitatory-inhibitory local distributed interactions of a bio-inspired connectionist processing.

to and from neurons have a modulated strength according to the distance one of each other. Figure E.3 shows a graphical view of the underlying excitatory-inhibitory local interactions.

The connectionist processing used in our bio-inspired approach for visual perception of motion is based on a similar neural organization as briefly described above. Nevertheless, excitatory-inhibitory connections are defined not only by distance properties but also by the semantics of the different neurons. The inhibitory interconnections among neurons regulate downwards the activity of opposing or antagonist neurons, i.e. neurons that do not share a common or similar orientation and speed. On the other hand, excitatory interconnections increase the neuron activity toward the emergence of coherent responses, i.e., grouping neuron responses to similar orientations and speeds through an iterative process.

Let $n = (x, y, T, s, \theta)$ denote a neuron at position (x, y), time T, speed s, and orientation θ. The local interaction among neurons determines the updating of their internal state, $F(n)$, according to :

$$\Delta F(n) = -A.F(n) + (B - F(n)).g_e(n) - (C + F(n)).g_i(n) \tag{E.1}$$

where A, B, C are constants that specify parameters of the model, $g_e(n)$ and $g_i(n)$ are excitation and inhibition from other neurons, computed according to :

$$g_e(n) = f(x, y, T) + \Sigma_{NE}(w(n, n').F(n')) \tag{E.2}$$

$$g_i(n) = \Sigma_{NI}(v(n, n').F(n'))) \tag{E.3}$$

where $f(x, y, T)$ is the spatio-temporal filtered data at time T and position (x, y), NE and NI are the excitation and inhibition neighborhoods around neuron n, and w and v are the weights of excitation and inhibition interconnections among neuron n and neurons n' that belong to NE

or NI. The summation on NE includes all data in a given orientation and all speeds, and the summation on NI includes data of all speeds and all orientations different from the selected one for excitation.

E.2 Model hardware implementation

This section presents the hardware implementation of the bio-inspired connectionist model for visual perception of motion. For the design and implementation of the corresponding hardware architecture, three main computational modules were considered according to the three kind of processing involved in the model [THGCS05a]. The key aspects guiding the digital implementation of the bio-inspired connectionist model were : a suitable parallel processing scheme to support the underlying computational nature of the main processing stages, an adequate memory addressing scheme and partitioning to efficiently use the memory bandwidth through data reuse, and the scalability and reprogrammability of the architecture.

E.2.1 Spatial processing hardware implementation

In order to mimic the orientation selective property of neurons in the early processing stages of visual stimuli in the visual cortex of mammalian brain, a set of hardware components were designed to process concurrently the same input image but tuned to different orientations. The proposed spatial hardware module resembles the so-called hypercolumnar organization of neurons [CSB04] through a set of 2D systolic arrays that process image data according to a convolution kernel to emphasize responses in a given orientation. Figure E.4 shows a simplified block diagram of the general architecture of the module used for fast computation of Gabor-like filtering. The architecture consists of eight identical 2D systolic arrays of processing elements to resemble arrays of neurons with the same orientation tuning but with different receptive fields.

The architectural approach of 2D systolic processing is based on the design proposed in [THAE04], where a compact systolic architecture for real-time window-based image processing is presented. In this work, an extension has been proposed to allow the computation of eight image convolutions concurrently. Two advantages of the proposed architecture are its regularity and modularity which make it suitable to be scalable and to accommodate Gabor filters of different sizes if required. The architecture exploits data parallelism and data reuse through a column-based memory addressing mode and a set of processing elements working in parallel. According to the six levels of parallelism of [NS92], this architecture succeeds in exploiting both connection and neuron parallelism, i.e. most aspects of the so-called natural neural parallelism [Gir00b]. For the sake of simplicity and explanation purposes, the following paragraphs describe the organization of a single module for Gabor-like filtering in a given orientation.

Essentially, the hardware architecture is based on a 2D systolic array of processing elements which receives data from external image memories where the input image pixels and the Gabor coefficients are stored, denoted as P and W in figure E.4, respectively. Each block denoted by GWP, Gabor Window Processor, represents an orientation-tuned neuron, the block labeled by D stands for a delay line, and the LDC block represents a Local Data Collector. The LDC modules collect results of GWPs located in the same column of the array and send them to the global data collector. From a functional point of view, data flows from top to bottom and left to right in the 2D array and the GWPs work progressively in a systolic pipeline. The current read pixel is broadcast to all the GWPs and the coefficients are transmitted between adjacent GWPs.

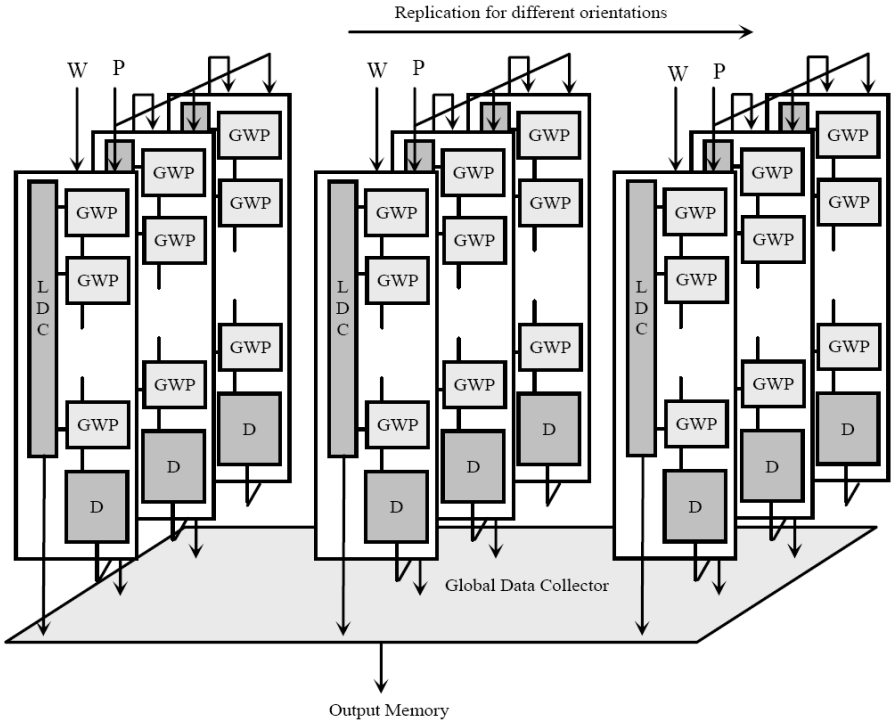


FIG. E.4 – Block diagram of the systolic array organization for spatial processing. Each 2D systolic array computes an oriented Gabor filter and a set of eight arrays computes in parallel results for the main orientations.

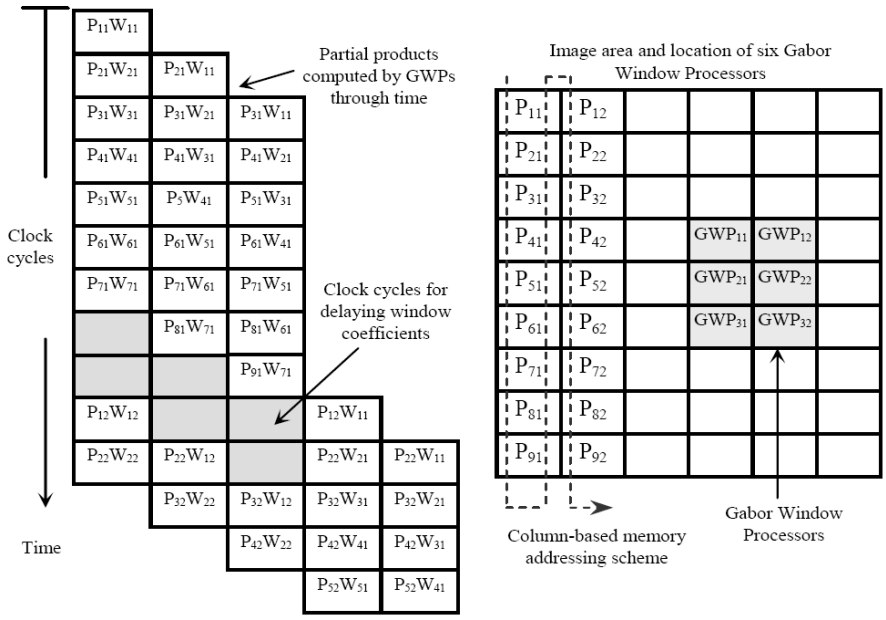


FIG. E.5 – Data flow and computation steps for a simplified array of GWPs. Internal computations through time for GWPs are shown on the left. The GWPs are closely localized one of each other and work on common data.

The GWPs are constituted by a multiplier and an accumulator using fixed-point parallel arithmetic. Though area-saving serial operators would not be possible to reach real-time performance. Each processing element processes data restricted to a local image window and computes the convolution with the assigned Gabor kernel. On each clock cycle, each GWP executes in parallel three operations : computes a multiplication between the current pixel with the corresponding Gabor coefficient, accumulates this result with previous computed values and reads a new coefficient and stores it into the internal register and transmits the previous coefficient to the next GWP.

A detail of the internal data flow and computation steps for a simplified systolic array of six GWPs organized in two columns is shown in figure E.5. The GWPs in the same column start processing progressively with the same input pixel but with different Gabor coefficients. For this simplified example, a Gabor window mask of 7x7 is considered. After a column of Gabor coefficients has been read, seven values for this example, the coefficients must be delayed two clock cycles and then transmitted to the next column of GWPs as shown by the data flow on the left side of figure E.5.

E.2.2 Temporal processing hardware implementation

The hardware module for causal temporal processing is based on a single processing element that computes the average of previous results using data at different positions of the spatial filtered images of a sequence. Data contributing to the average computation is determined by the current assumed speed and orientation at a given position according to the explanation provided in section II. A key aspect to efficiently implement the temporal processing in hardware is the memory

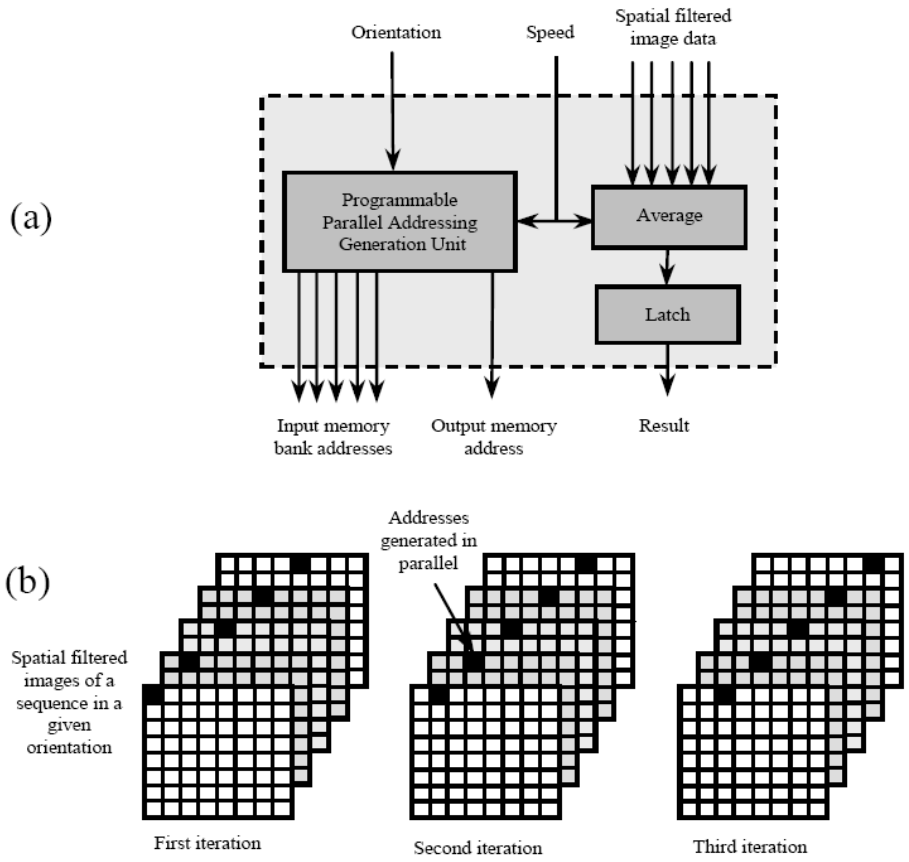


FIG. E.6 – (a) A simplified block diagram of temporal processing hardware module which computes the average of a set of spatial filtered values at different positions of the image sequence, and (b) a memory scheduling example for three pixels with an assumed speed through time.

partitioning and addressing scheme. In the proposed module architecture, it is assumed that the spatial filtered images of a sequence required to temporal processing for a specified speed and orientation are stored in independent memory banks. This hypothesis which is valid for most FPGA-based prototyping platforms, as the one used in our experiments, allows to access in parallel the spatial filtered data to compute in one clock cycle its average. Under these considerations, the temporal processing hardware module is basically composed of a programmable parallel addressing unit and an average computation module. Figure E.6(a) shows a simplified block diagram of the internal structure of the hardware module for temporal processing and its interface.

The addressing unit generates in parallel several addresses to access different pixel locations on spatial filtered images of the sequence according to the requirements established by an orientation and a speed as shown in figure E.6(b). The spatial filtered values obtained from the memory banks are sent to the average module to compute their average using parallel arithmetic. The result is sent to an output memory location which is generated by the addressing module.

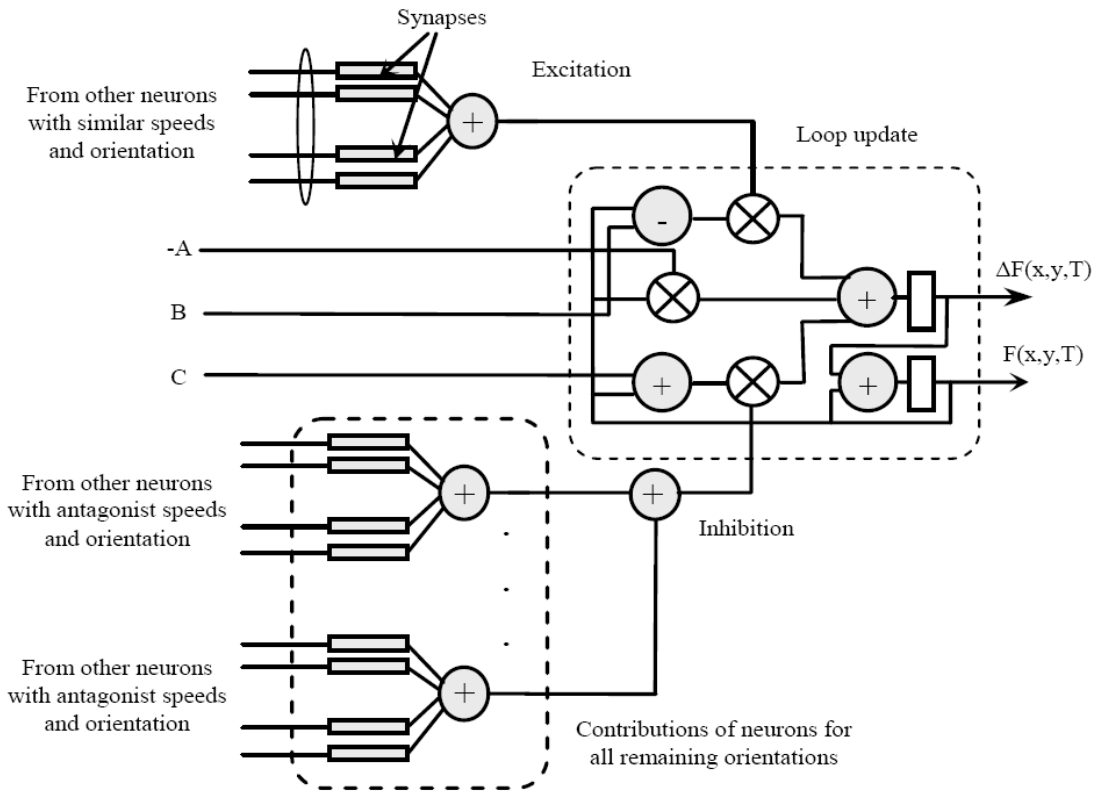


FIG. E.7 – A simplified block diagram of the neuron hardware model used for motion estimation.

E.2.3 Connectionist processing hardware implementation

The connectionist module is based on an excitatory-inhibitory network where a neuron interacts with others inside a neighborhood or influence range. The influence exerted on a neuron by others depends on three factors : the internal state of neurons, the distance one to each other and their level of antagonism as explained in section E.1. Figure E.7 shows a simplified block diagram of the neuron hardware model used in the proposed bio-inspired approach. The neuron model takes up a key property of neurons in the visual cortex : the excitatory-inhibitory mechanism for coherent emergent responses [CSGA04]. The neuron model is essentially composed of a soma, shaded rectangle in figure E.7, and a set of sub-modules or synapses to compute excitation-inhibition. The change in the neuron activity level is computed by the elements inside the gray rectangle in figure E.7, as stated in equation (E.1).

A block diagram of the hardware structure of a synapse is shown in figure E.8. The synapse takes data from neurons in a neighborhood around a specified position. Data is added and weighted by a coefficient that depends on the distance between two neurons and an antagonism proportional weight computed by the antagonism module. A look-up-table, LUT, efficiently stores coefficients at the cost of internal storage hardware resources.

The distance computation component uses the spatial coordinates to compute the distance. If the influence range of the neuron associated to the synapse, $R_{\text{inf}}(x, y)$ in figure E.8, is larger

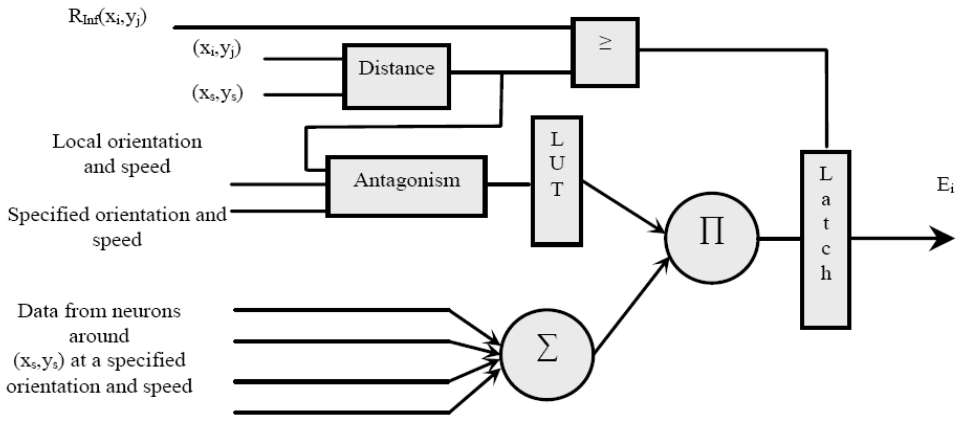


FIG. E.8 – A block diagram of the internal structure of the synapse used in the neuron hardware model.

than the computed distance, then it contributes to change the activity level of the central neuron, otherwise the central neuron is not affected by the linked neuron. The overall influence over a neuron is computed by adding all the contributions of neurons in a neighborhood within a specified influence radius. In the proposed bio-inspired approach for visual perception of motion, neurons in a 15x15 area, centered around a neuron, are allowed to potentially contribute or modify the neuron internal state.

The hardware requirements by a full parallel architecture make its implementation impractical with standard programmable logic devices. A neuron hardware module was designed with a reduced number of synapses and to compute incrementally the total contribution of neurons. The module includes eight synapses, each one computing and accumulating the contribution of neurons in local vicinities that share the same orientation but different speeds. Using time-multiplexed synapses in neurons provides some opportunities to avoid memory bottlenecks through systolic arrays of neurons with pipeline processing.

Since the neural computations of the connectionist processing are local, they can be reformulated as window-based operation, as the spatial Gabor-like filtering, but with operations of higher complexity. Under these considerations, a linear systolic array of fifteen neurons was used for the connectionist processing in the whole image.

The functional operation of the array is fairly similar to the one described in section E.1 for a column of GWPs due to the window-like nature of operations preserving the systolic and pipeline processing. A simplified conceptual view of the pipeline operation of the systolic array of neurons is shown in figure E.9. For illustration purposes a set of five neurons are shown with a 5x5 neighborhood through four iterations. The black rectangles represent the neurons already processed in the current iteration.

It is important to point out that the proposed architectural organization is regular and modular and can be extended to employ a higher number of neurons using a 2D systolic organization if hardware resources are available.

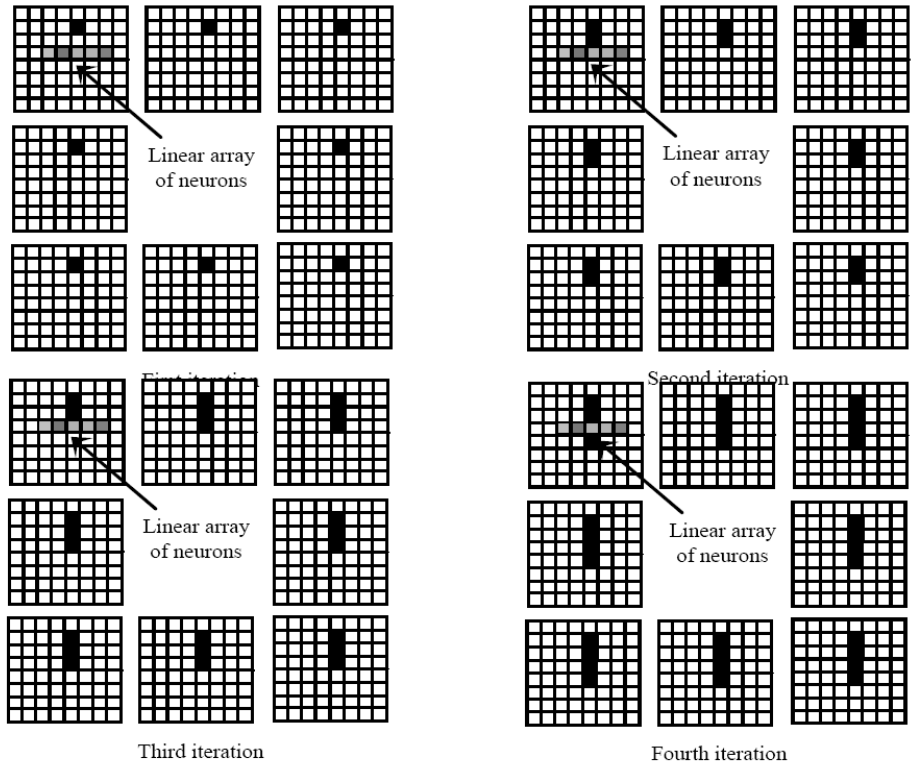


FIG. E.9 – Conceptual view of the data flow of pipeline processing for a systolic array of neurons through window-based operation in different orientations.

FPGA hardware resources	Spatial processing module	Temporal processing module	Connectionist processing module
Number of Slices	12286/12288	178/12288	9876/12288
Number of 4-input LUTs	18545/24576	323/24576	19286/24576
Number of Flip-flops	8020/24576	96/24576	2684/24576
FPGA percentage	99	1	80
Maximum clock frequency	46 MHz	84.94 MHz	40 MHz

Tab. E.1 – Synthesis results

E.3 Results and discussion

The hardware modules for the bio-inspired motion estimation approach have been modeled using the VHDL hardware description language following a structural description according to the spatial, temporal and connectionist processing modules. The VHDL model for each module has been simulated and validated through Foundation ISE 5.1 from Xilinx.

The spatial and temporal processing modules have been optimized at some extent. The three hardware modules have been independently synthesized for a Virtex FPGA device through Xilinx Synthesis Technology, XST, with good results in terms of hardware resource utilization and performance. The synthesis results and timings for these modules when synthesized independently for XCV1000-4BGA560 Virtex device are summarized in table E.1.

The spatial processing module was synthesized for a configuration of eight 2D systolic arrays, each one with 7x7 processing elements. Thus, eight different orientations were considered and computed in parallel involving 392 orientation selective neurons. This module uses fixed point arithmetic with an 8-bit word-length both for image pixels and Gabor coefficients ; internally, the module uses a 16-bit word for multiplication and accumulation operations. The 16-bit internal representation is then normalized to an 8-bit representation, neglecting the least significant bits.

According to the performance analysis in [THAE04] and the maximum clock frequency reported in the synthesis, the time required for spatial filtering of a 128x128 sized image in 8 different orientations is around 0.56 milliseconds. The time required for temporal processing is around 6 milliseconds. The spatial processing software implementation on a personal computer with a Pentium IV processor and 512 Mbytes of main memory requires around 40 milliseconds for the same image resolution and window sizes. Thus, a performance improvement of about 100x is obtained which opens an opportunity to achieve real time performance of the whole model implementation. The high computational requirements and memory bandwidth of the bio-inspired motion detection model pose hard implementation constraints even to be solved with off the shelf digital signal processors. Because of their internal structure, FPGAs are well suited to avoid memory bottlenecks by using internal registers and multi-port on-chip memory.

The connectionist module was synthesized for a configuration of fifteen neurons, each with eight synapses. A remaining time of around 20 milliseconds is available for connectionist processing which is not enough for the current configuration, since several iterations are required requiring about ten seconds to complete the processing. According to the FPGA synthesis a larger device will be required to achieve a real-time implementation of the whole model.

Bibliographie

[AB85] E. H. Adelson and J. R. Bergen. Spatiotemporal energy models for the perception of motion. *J. of the Optical Society of America A*, 2(2), 1985.

[ABDG+97] D. Anguita, S. Bencetti, A. De Gloria, G. Parodi, D. Ricci, and S. Ridella. FPGA implementation of high precision feedforward networks. In *Proc. MicroNeuro*, pages 240–243, 1997.

[Abe82] M. Abeles. *Local cortical circuits*. Springer, 1982.

[ABR05] D. Anguita, S. Boni, and S. Rindella. A digital architecture for support vector machines : Theory, algorithm, and fpga implementation. *IEEE Trans. on Neural Networks*, 14(5), 2005.

[AFD+02] M. Anguita, F.J. Fernandez, A.F. Diaz, A. Canas, and F.J. Pelayo. Parameter configurations for hole extraction in cellular neural networks. *Analog Integrated Circuits and Signal Processing*, 32(2), 2002.

[Ama77] S. Amari. Dynamics of pattern formation in lateral inhibition type neural fields. *Biological Cybernetics*, 27, 1977.

[APFP97] M. Anguita, F.J. Pelayo, F.J. Fernandez, and A. Prieto. A low-power cmos implementation of programmable CNNs with embedded photosensors. *IEEE Trans. on Circuits Systems I : Fundamental Theory and Applications*, 44(2), 1997.

[BAA93] N.M. Botros and M. Abdul-Aziz. Hardware implementation of an artificial neural network. In *Proc. ICNN*, volume 3, pages 1252–1257, 1993.

[BAJS05] H. Boumeridja, M. Atencia, G. Joya, and F. Sandoval. FPGA implementation of hopfield networks for systems identification. In *Proc. IWANN*, volume 3512 of *LNCS*, 2005.

[Beu98] J.-L. Beuchat. Conception d'un neuroprocesseur reconfigurable proposant des algorithmes d'apprentissage et d'élagage : une première étude. In *Proc. NSI Neurosciences et Sciences de l'Ingénieur*, 1998.

[BH94] S.L. Bade and B.L. Hutchings. FPGA-based stochastic neural networks - implementation. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 189–198, 1994.

[BKBB+03a] K. Ben Khalifa, M.H. Bedoui, L. Bougrain, R. Raychev, M. Dogui, and F. Alexandre. Analyse et classification des états de vigilance par réseaux de neurones. Technical Report RR-4714, INRIA, 2003.

[BKBB+03b] K. Ben Khalifa, M.H. Bedoui, L. Bougrain, R. Raytchev, M. Dogui, and F. Alexandre. Analyse et classification des états de vigilance par réseaux de neurones. Technical report, INRIA, 2003. RR-4714.

[BKGAB04] K. Ben Khalifa, B. Girau, F. Alexandre, and M.H. Bedoui. Parallel FPGA implementation of self-organizing maps. In *International Conference on Microelectronics - ICM'04, Tunis, Tunisia*, 2004.

[BL98] M. Bolduc and M. Levine. A review of biologically motivated space-variant data reduction models for robotic vision. *Computer Vision and Image Understanding*, 69(2), 1998.

[BN04] P. Bayerl and H. Neumann. Disambiguating visual motion through contextual feedback modulation. *Neural Computation*, 16(10), 2004.

[BP02] E. Bruno and D. Pellerin. Robust motion estimation using spatial gabor filters. *Signal Processing archive*, 82(2), 2002.

[BQ06] H. Berry and M. Quoy. Structure and dynamics of random recurrent neural networks. *Adaptive Behavior*, 14, 2006.

[CAL96] Y.K. Choi, K.H. Ahn, and S.-Y. Lee. Effects of multiplier output offsets on on-chip learning for analog neuro-chips. *Neural Processing Letters*, 4 :1–8, 1996.

[Cas05] Claudio Castellanos Sanchez. *Modèle connexionniste neuromimétique pour la perception visuelle embarquée du mouvement*. PhD thesis, Université Henri Poincaré Nancy 1, 2005.

[Col05] B. Colwell. Machine intelligence meets neuroscience. *IEE Computer*, 38(1) :12–15, 2005.

[CSB04] T. Choi, B. Shi, and K. Boahen. An ON-OFF orientation selective address representation image transceiver chip. *IEEE Transactions on Circuits and Systems I*, 51(2), 2004.

[CSG05] C. Castellanos Sánchez and B. Girau. Dynamic pursuit with a bio-inspired neural model. In *Advanced Concepts for Intelligent Vision Systems - ACIVS 2005, Anwertp, Belgium*, volume 3708 of *Lecture Notes in Computer Science*, pages 284–291, 2005.

[CSGA04] C. Castellanos Sánchez, B. Girau, and F. Alexandre. A connectionist approach for visual perception of motion. In *Brain Inspired Cognitive Systems - BICS 2004, Stirling, United Kingdom*, 2004.

[CWJ99] S. Campbell, D. Wang, and C. Jayaprakash. Synchrony and desynchrony in integrate-and-fire oscillators. *Neural Computation*, 11, 1999.

[CY88] L.O. Chua and L. Yang. Cellular neural networks : theory. *IEEE Trans. on Circuits and Systems*, CAS-35 :1257–1272, 1988.

[DC95] B. Dolenko and H. Card. Tolerance to analog hardware of on-chip learning in backpropagation networks. *IEEE Trans. on Neural Networks*, 6(5) :1045–1052, 1995.

[Deh02] A. Dehon. Very large scale spatial computing. In *Proc. Unconventional Models of Computation*, 2002.

[Dra00] S. Draghici. Neural networks in analog hardware–design and implementation issues. *Int. J. of Neural Systems*, 10(1), 2000.

[Dra02] S. Draghici. On the capabilities of neural networks using limited precision weights. *Neural Networks*, 15, 2002.

[DU86] Robert Desimone and Leslie G. Ungerleider. Multiple visualareas in the caudal superior temporal sulcus of the macaque. *The Journal of comparative Neurology*, 248(2) :164–189, 1986.

[EFG+97] W. Eppler, T. Fisher, H. Gemmeke, T. Becher, and G. Kock. High speed neural network chip on PCI-board. In *Proc. MicroNeuro*, pages 9–17, 1997.

[EH94] J.G. Eldredge and B.L. Hutchings. RRANN : a hardware implementation of the backpropagation algorithm using reconfigurable FPGAs. In *Proceedings of the IEEE World Conference on Computational Intelligence*, 1994.

[Eli97] F. Elie. *Conception et réalisation d'un système utilisant des réseaux de neurones pour l'identification et la caractérisation, à bord de satellites, de signaux transitoires de type sifflement*. PhD thesis, LPCE, Université d'Orléans, 1997.

[EPM96] A. Elisseeff and H. Paugam-Moisy. Size of multilayer networks for exact learning : analytic approach. Technical Report 96-16, LIP-ENSL, 1996.

[Erc84] M.D. Ercegovac. On-line arithmetic : an overview. In SPIE, editor, *SPIE, Real Time Signal Processing VII*, pages pp 86–93, 1984.

[ET77] M.D. Ercegovac and K.S. Trivedi. On-line algorithms for division and multiplication. *IEEE Trans. Comp.*, C-26(7) :pp 681–687, 1977.

[FNH97] B. Friebe, S. Neusser, and B. Höfflinger. SIOP : application-specific neural hardware. In *Proc. MicroNeuro*, pages 18–24, 1997.

[FSN02] D. Fernandes, J. Stedile, and P. Navaux. Architecture of oscillatory neural network for image segmentation. In *proc. 14th Symposium on Computer Architecture and High Performance Computing*, 2002.

[FSS97] S.K. Foo, P. Saratchandran, and N. Sundararajan. Parallel implementation of back-propagation neural networks on a heterogeneous array of transputers. *IEEE Trans. on Systems, Man, and CyberneticsPart B : Cybernetics*, 27(1) :118–126, 1997.

[FTGR96] X. Fang, P. Thole, J. Goppert, and W. Rosenstiel. A hardware supported system for a special online application of self-organizing map. In *Proceedings of ICNN96*, pages 1040–1045, 1996.

[Fun89] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2 :183–192, 1989.

[GB07] B. Girau and A. Boumaza. Embedded harmonic control for dynamic trajectory planning on FPGA. In *Conf. on Artificial Intelligence and Applications*, 2007.

[GBK06] B. Girau and K. Ben Khalifa. FPGA-targeted neural architecture for embedded alertness detection. In *Conf. on Artificial Intelligence and Applications*, 2006.

[GCBM00] R. Gadea, J. Cerda, F. Ballester, and A. Mocholi. Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation. In *Proc. ISSS*, 2000.

[GGR95a] C. Gégout, B. Girau, and F. Rossi. A general feedforward neural network model. Rapport technique NC-TR-95-041, NeuroCOLT, Royal Holloway, University of London, 1995.

[GGR95b] C. Gégout, B. Girau, and F. Rossi. Generic back-propagation in arbitrary feedforward neural networks. In *Artificial Neural Nets and Genetic Algorithms – Proc. of ICANNGA*, pages 168–171. Springer-Verlag, 1995.

[GHSM00] R. Gadea, V. Herrero, A. Sebastia, and A. Mocholi. The role of the embedded memories in the implementation of artificial neural networks. In *Proc. FPL*, volume 1896 of *LNCS*, 2000.

[Gir99] B. Girau. *Du parallélisme des modèles connexionnistes à leur implantation parallèle*. PhD thesis n° 99ENSL0116, ENS Lyon, 1999.

[Gir00a] B. Girau. Conciliating connectionism and parallel digital hardware. *Parallel and Distributed Computing Practices*, 3(2) :145–162, 2000.

[Gir00b] B. Girau. FPNA : interaction between FPGA and neural computation. *International Journal of Neural Systems*, 10(3) :243–259, 2000.

[Gir00c] B. Girau. Neural networks on FPGAs : a survey. In *Second ICSC Symposium on Neural Computation - NC'2000, Berlin, Germany*, 2000.

[Gir06a] B. Girau. *FPGA Implementations of Neural Networks*, chapter FPNA : Concepts and properties. Springer, 2006.

[Gir06b] B. Girau. *FPGA Implementations of Neural Networks*, chapter FPNA : Applications and implementations. Springer, 2006.

[GK02] W. Gerstner and W. Kistler. *Spiking neuron models - single neurons, populations, plasticity*. Cambridge University Press, 2002.

[GP07] G. Grossi and F. Pedersini. FPGA implementation of an adaptive stochastic neural model. In *Artificial neural networks - ICANN*, volume 4668 of *LNCS*. Springer, 2007.

[GPM94] D. Girard and H. Paugam-Moisy. Strategies of weight updating for parallel back-propagation. In C. Girault, editor, *Proc. Applications in parallel and distributed computing*, pages 335–336. IFIP, North-Holland, 1994.

[GSMb96] M. Gschwind, V. Salapura, and O. Maisch berger. A generic building block for Hopfield neural networks with on-chip learning. In *Proc. ISCAS*, 1996.

[GT00] B. Girau and A. Tisserand. MLP computing and learning on FPGA using on-line arithmetic. *Int. Journal on System Research and Information Science,* special issue on *Parallel and Distributed Systems for Neural Computing*, 9(2-4), 2000.

[GTH06] B. Girau and C. Torres-Huitzil. FPGA implementation of an integrate-and-fire legion model for image segmentation. In *European Symp. on Artificial Neural Networks*, 2006.

[GTH07] B. Girau and C. Torres-Huitzil. Massively distributed digital implementation of an integrate-and-fire LEGION network for visual scene segmentation. *Neurocomputing*, 70 :1186–1197, 2007.

[HA94] S.L. Hung and H. Adeli. A parallel genetic/neural network learning algorithm for MIMD shared memory machines. *IEEE Transactions on neural networks*, 5(6) :900–909, 1994.

[Ham95] D. Hammerstrom. Digital VLSI for neural networks. In *The handbook of Brain Theory and Neural Networks*, pages 304–309. MIT Press, 1995.

[HH93] J.L. Holt and J.-N. Hwang. Finite precision error analysis of neural network hardware implementations. *IEEE Transactions on Computers*, 42(3) :281–290, March 1993.

[HHP90] P.W. Hollis, J.S. Harper, and J.J. Paulos. The effects of precision constraints in a backpropagation learning algorithm. *Neural Computation*, 2 :363–373, 1990.

[Hik05] H. Hikawa. FPGA implementation of self organizing map with digital phase locked loops. *Neural Networks*, 18, 2005.

[HJP+98] J.M. Hupe, A.C. James, B.R. Payne, S.G. Lomber, P. Girard, and J. Bullier. Cortical feedback improves discrimination between figure and background by v1,v2 and v3 neurons. *Nature*, 394, 1998.

[HM05] H. Hugues and D. Martinez. Encoding in a network of sparsely connected spiking neurons : application to locust olfaction. *Neurocomputing*, 65-66, 2005.

[Hor91] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4 :251–257, 1991.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2 :359–366, 1989.

[HT94] T.-P. Hong and S.-S. Tseng. An optimal parallel perceptron learning algorithm for a large training set. *Parallel Computing*, 20 :347–352, 1994.

[Int06] Intel. Evolution of parallel computing. http ://www.intel.com/platforms/parallel.htm, 2006.

[IV94] P. Ienne and M.A. Viredaz. Bit-serial multipliers and squarers. *IEEE Transactions on Computers*, 43(12) :1445–1450, 1994.

[JBO87] R. Jain, S.L. Bartlett, and N. O'Brien. Motion stereo using ego-motion complex logarithmic mapping. *IEEE Trans. on Pattern Analysis and machine intelligence*, 9(3), 1987.

[JKL95] S.L. Joutsiniemi, S. Kaski, and T.A. Larsen. Self-organizing map in recognition of topographic patterns of EEG spectra. *IEEE Transactions on Biomedical Engineering*, 42 :1062–1068, 1995.

[JPD+92] A. Johannet, L. Personnaz, G. Dreyfus, J.D. Gascuel, and M. Weinfeld. Specification and implementation of a digital Hopfield-type associative memory with on-chip training. *IEEE Trans. on Neural Networks*, 3, 1992.

[KA97] J. Kennedy and J. Austin. A parallel architecture for binary neural networks. In *Proc. MicroNeuro*, pages 225–231, 1997.

[Kis02] L.B. Kish. End of moore's law : thermal (noise) death of integration in micro and nano electronics. *Physics Letters A*, 305, 2002.

[Koh01] T. Kohonen. *Self-Organization Maps*. Springer, 2001.

[LK98] T. Lindblad and J.M. Kinser. *Image Processing Using Pulse-coupled Neural Networks*. Springer Verlag, 1998.

[LTYS06] S. Lam, E. Tsang, Meng Y., and B. Shi. Neuromorphic translational ego-motion estimation using log-polar motion energy. In *Proc. IJCNN*, 2006.

[LW06] J.W. Lawson and D.H. Wolpert. Adaptive programming of unconventional nano-architectures. *ArXiv Condensed Matter e-prints*, 2006.

[Maa03] W. Maas. Computation with spiking neurons. In Michel A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 2003.

[MGP97] A. Montalvo, R. Gyurcsik, and J. Paulos. Towards a general-purpose analog VLSI neural network with on-chip learning. *IEEE Trans. on Neural Networks*, 8(2) :413–423, 1997.

[MI97] S. McLoone and G.W. Irwin. Fast parallel off-line training of multilayer perceptrons. *IEEE Trans. on Neural Networks*, 8(3) :646–653, 1997.

[Mic97] MicroNeuro. *Proceedings of the 6th int. conf. on Microelectronics for neural networks, evolutionary and fuzzy systems*, 1997.

[MJH+03] N. Mehrtasch, D. Jung, H.H. Hellmich, T. Schonauer, V.T. Lu, and H. Klar. Synaptic plasticity in spiking neural networks (sp2inn) : A system approach. *IEEE Trans. on Neural Networks*, 14(5) :980–992, 2003.

[MNCG01] B. MacCane, K. Novins, D. Crannitch, and B. Galvin. On benchmarking optical flow. *Computer Vision and Image Understanding*, 84(1), 2001.

[Mog00] S. Moga. *Apprendre par imitation : une nouvelle voie d'apprentissage pour les robots autonomes*. PhD thesis, Univ. Cergy-Pontoise, 2000.

[Mul97] J.-M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhauser, Boston, 1997.

[Mur92] A. Murray. Multilayer perceptron learning optimized for on-chip implementations : a noise-robust system. *Neural Computation*, 4 :366–381, 1992.

[MVJL93] D. Macq, M. Verleysen, P. Jespers, and J.D. Legat. Analog implementation of a Kohonen map with on-chip learning. *IEEE Trans. on Neural Networks*, 4(3) :456–461, 1993.

[NdMM03] N. Nedjah and L. de Macedo Mourelle. FPGA-based hardware architecture for neural networks : binary radix vs. stochastic. In *Proc. SBCCI*, 2003.

[NGT98] P. Nussbaum, B. Girau, and A. Tisserand. Field Programmable Processor Arrays. In *Proc. ICES*, volume 1478 of *LNCS*, pages 311–322. Springer, 1998.

[NK97] A.M. Nielsen and P. Kornerup. On radix representation of rings. In *IEEE Symposium on Computer Arithmetic*, 1997.

[NL95] R. Newcomb and J. Lohn. Analog VLSI for neural networks. In *The Handbook of Brain Theory and Neural Networks*, pages 86–90. MIT Press, 1995.

[NS92] T. Nordström and B. Svensson. Using and designing massively parallel computers for artificial neural networks. *Journal of Parallel and Distributed Computing*, 14 :260–285, 1992.

[OFB05] Ménard O. and H. Frezza-Buet. Model of multi-modal cortical processing : Coherent learning in self-organizing modules. *Neural Networks*, 18(5-6), 2005.

[OR05] A.R. Omondi and J.C. Rajapakse, editors. *FPGA implementations of Neural Networks*. Springer, 2005.

[OR06] A.R. Omondi and J.C. Rajapakse, editors. *FPGA implementations of neural networks*. Springer, 2006.

[OW05] S.W. Olridge and S.J.E. Wilton. A novel fpga architecture supporting wide, shallow memories. *IEEE Trans. on VLSI Systems*, 13(6), 2005.

[PGMS07] S. Philipp, A. Grubl, K. Meier, and J. Schemmel. Interconnecting VLSI spiking neural networks using isochronous connections. In *Proc. IWANN*, 2007.

[PK97] N. Petkov and P. Kruizinga. Computational models of visual neurons specialised in the detection of periodic and aperiodic oriented visual stimuli : bar and grating cells. *Biological Cybernetics*, 76, 1997.

[PKW+01] M. Porrmann, H. Kalte, U. Witkowski, J.-C. Niemann, and U. Rückert. A dynamically reconfigurable hardware accelerator for self-organizing feature maps. In *Proc. of the 5th World Multi-Conference on Systems, Cybernetics and Informatics, SCI 2001*, volume 3, pages 242–247, Orlando, Florida, USA, July 2001.

[PMPK98] U. Polat, K. Mizobe, M.W. Pettet, and T. Kasamarsu. Collinear stimuli regulate visual responses depending on cell's constrast threshold. *Nature*, 391, 1998.

[PUS96] A. Pérez-Uribe and E. Sanchez. FPGA implementation of an adaptable-size neural network. In *Proc. ICANN*. Springer-Verlag, 1996.

[RHG+96] M. Rossmann, B. Hesse, K. Goser, A. Bühlmeier, and G. Manteuffel. Implementation of a biologically inspired neuron-model in FPGA. In *Proc. MicroNeuro*, pages 322–329, 1996.

[RHTF03] D. Roggen, S. Hofmann, Y. Thoma, and D. Floreano. Hardware spiking neural networks with run-time reconfigurable connectivity in an autonomous robot. In *NASA/DoD Conf. on Evolvable Hardware*, 2003.

[RM03] O. Rochel and M. Martinez. An event-driven framework for the simulation of networks of spiking neurons. In *Proc. ESANN*, 2003.

[Rou06] N. Rougier. Dynamic neural field with local inhibition. *Biological Cybernetics*, 94(3), 2006.

[RTB+07] S. Renaud, J. Tomas, Y. Bornat, A. Daouzli, and S. Saighi. Neuromimetic ICs with analog cores : an alternative for simulating spiking neural networks. In *Proc. ISCAS*, 2007.

[RV06] N. Rougier and J. Vitay. Emergence of attention within a neural population. *Neural Networks*, 19(5), 2006.

[RVT07] K.L. Rice, C.N. Vutsinas, and T.M. Taha. A preliminary investigation of a neocortex model implementation on the cray XD1. In *Proc. SC - SuperComputing*, 2007.

[RWSB98] F. Rieke, D. Warland, R. Steveninck, and W. Bialek. *SPIKES : Exploring the Neural Code*. MIT Press, 1998.

[SAMA06] F. Smach, M. Atri, J. Miteran, and M. Abid. Design of a neural networks classifier for face detection. *J. of Computer Science*, 2(3), 2006.

[SB02] B. Shi and K. Boahen. Competitively coupled orientation selective cellular neural networks. *IEEE Transactions on Circuits and Systems*, 49(3), 2002.

[SdS06] T.J Sullivan and V.R. de Sa. A model of surround suppression through cortical feedback. *Neural Networks*, 19(5), 2006.

[SG93] S. Shams and J.-L. Gaudiot. Parallel implementations of neural networks. *Int. J. on Artificial Intelligence*, 2(4) :557–581, 1993.

[SGMb94] V. Salapura, M. Gschwind, and O. Maisch berger. A fast FPGA implementation of a general purpose neuron. In *Proc. FPL*, 1994.

[SGMM06] J. Schemmel, A. Grubl, K. Meier, and E. Muller. Implementing synaptic plasticity in a VLSI spiking neural network model. In *Proc. IJCNN*, 2006.

[SH98] E.P. Simoncelli and D.J. Heeger. A model of neuronal responses in visual area MT. *Vision research*, 38(5), 1998.

[Sin90] A. Singer. Implementations of artificial neural networks on the Connection Machine. *Parallel Computing*, 14 :305–316, 1990.

[Sin03] W. Singer. Synchronization, binding and expectancy. In Michel A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 2003.

[SJ95] K.M. Sammut and S.R. Jones. Arithmetic unit design for neural accelerators : cost performance issues. *IEEE Trans. on Computers*, 44(10), 1995.

[SMA07] A.W. Savich, M. Moussa, and S. Areibi. The impact of arithmetic representation on implementing MLP-BP on FPGAs : a study. *IEEE Trans. on Neural Networks*, 18(1), 2007.

[SRK91] K. Siu, V. Roychowdhury, and T. Kailath. Depth-size tradeoffs for neural computation. *IEEE Trans. on Computers*, 40(12) :1402–1412, 1991.

[SSB03] N. Sudha, T. Srikanthan, and Mailachalam B. A massively parallel architecture for self-organizing feature maps. *IEEE Transactions on Neural Networks, Special Issue on Hardware Implementations*, 48(11-12) :337–352, 2003.

[Sto04] A. Stocker. Analog VLSI focal-plane array with dynamic connections for the estimation of a piecewise-smooth optical flow. *IEEE Transaction on Circuits and Systems I*, 51(5), 2004.

[STR93] H. Speckmann, P. Thole, and W. Rosenstiel. COKOS : A coprocessor for Kohonen's self-organizing map. In *Proceedings of ICNN93*, pages 1040–1045, 1993.

[SvC06] B. Schrauwen and J. van Campenhout. Parallel hardware implementation of a broad class of spiking neurons using serial arithmetic. In *Proc. ESANN*, 2006.

[Tay98] J.G. Taylor. Neural bubble dynamics in two dimensions : foundations. *Biological cybernetics*, 80 :5167–5174, 1998.

[TCIF95] A. Torralba, F. Colodro, E. Ibanez, and L.G. Franquelo. Two digital circuits for a fully parallel stochastic neural network. *IEEE Trans. on Neural Networks*, 6(5) :1264–1268, 1995.

[THAE04] C. Torres-Huitzil and M. Arias-Estrada. Real-time image processing with a compact FPGA-based systolic architecture. *Journal on Real-time Imaging*, 10, 2004.

[THG05] C. Torres-Huitzil and B. Girau. FPGA implementation of an excitatory and inhibitory connectionist model for motion perception. In *IEEE 2005 Conference on Field-Programmable Technology - FPT'05, Singapore*, 2005.

[THGCS05a] C. Torres-Huitzil, B. Girau, and C. Castellanos Sánchez. Digital implementation of a bio-inspired neural model for motion estimation. In *International Joint Conference on Neural Networks, Montréal, Québec, Canada*, 2005.

[THGCS05b] C. Torres-Huitzil, B. Girau, and C. Castellanos Sánchez. On-chip visual perception of motion : A bio-inspired connectionist model on FPGA. *Neural networks*, 18(5-6) :557–565, 2005.

[THGG07] C. Torres-Huitzil, B. Girau, and A. Gauffriau. Hardware-software codesign for embedded implementations of neural networks. In *Applied Reconfigurable Computing, ARC*, 2007.

[TS93] M. Tistarelli and G. Sandini. On the advantages of polar and log-polar mapping for direct estimation of time-to-impact from optical flow. *IEEE Trans. on Pattern Analysis and machine intelligence*, 15(4), 1993.

[TW95a] D. Terman and D.L. Wang. Global competition and local cooperation in a network of neural oscillators. *Physica D*, 81, 1995.

[TW95b] D. Terman and D.L. Wang. Locally excitatory globally inhibitory oscillator networks. *IEEE Trans. Neural Networks*, 6(1), 1995.

[TWC05] L. Ting, R. Woods, and C. Cowan. Virtex FPGA implementation of a pipelined adaptive LMS predictor for electronic support measures receivers. *IEEE Trans. on VLSI Systems*, 13(1), 2005.

[UPRS05] A. Upegui, C.A. Pena Reyes, and E. Sanchez. An fpga platform for on-line topology ecploration of spiking neural networks. *Microprocessors and microsystems*, 29 :211–223, 2005.

[vDJST93] M. van Daalen, P. Jeavons, and J. Shawe-Taylor. A stochastic neural architecture that exploits dynamically reconfigurable FPGAs. In *Proc. of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 202–211, 1993.

[vdMB92] C. von der Malsburg and J. Buhmann. Sensory segmentation with coupled neural oscillators. *Biol. Cybern.*, 67, 1992.

[vdMS86] C. von der Malsburg and W. Schneider. A neural cocktail-party processor. *Biol. Cybern.*, 54, 1986.

[VRCP02] A. Vuckovic, V. Radivojevicb, A.C.N. Chena, and D. Popovica. Automatic recognition of alertness and drowsiness from EEG by an artificial neural network. *Med Eng Phys*, 24(5) :349–360, 2002.

[WAK+96] J. Wawrzynek, K. Asanović, B. Kingsbury, J. Beck, D. Johnson, and N. Morgan. SPERT-II : a vector microprocessor system and its application to large problems in backpropagation training. In *Proc. MicroNeuro*, pages 227–231, 1996.

[Wan05] D.L. Wang. The time dimension for scene analysis. *IEEE Trans. on Neural Networks*, 14(6), 2005.

[Wat93] Watts. Event-driven simulation of networks ofd spiking neurons. In *NIPS*, volume 6, pages 927–934, 1993.

[WMLB00] J. Waldemark, M. Millberg, T. Lindblad, and V. Becanovic. Implementation of a pulse coupled neural network in fpga. *Int. J. of neural Systems*, 10(3) :171–177, 2000.

[WT97] D.L. Wang and D. Terman. Image segmentation based on oscillatory correlation. *Neural Computation*, 9, 1997.

[XH01] J. Xing and D.J. Heeger. Measurement and modeling of center-surround supresion and enhancement. *Vision Research*, 41, 2001.

[Zam04] F. Zambonelli. Spatial computing and self-organization. In *Proc. WET ICE*, 2004.

[ZBT+06] Q. Zou, Y. Bornat, J. Tomas, S. Renaud, and A. Destexhe. Real-time simulations of networks of hodgkin-huxley neurons using analog circuits. *Neurocomputing*, 69, 2006.

[ZS03] J. Zhu and P. Sutton. FPGA implementations of neural networks - a survey of a decade of progress. In *Proc. FPL*, volume 2778 of *LNCS*, 2003.