Bares Victor

Tuteurs : Bresson Guillaume & Cuperlier Nicolas
Tuteur École : Jaulin Luc

Étude d'une architecture sensorimotrice pour le contrôle d'un véhicule autonome.

26 août 2019

Table des matières

Table des figures

Résumé

Les animaux peuplant la terre arrivent très bien à se déplacer dans leu environnement, à retrouver leur habitat tout en évitant les obstacles. La navigation robotisée possède les mêmes challenges que ceux auxquels les animaux ont été confrontés lors de leur évolution. Il est donc très intéressant d'étudier les techniques mises en place pour surmonter ces défis liés à la navigation.

L'étude du cerveau des animaux a permis la découverte de cellules spécialisées dans la navigation. Il est possible de reproduire le fonctionnement de certaines cellules des animaux en les simulant sur ordinateur. Ces cellules permettent de faire de la navigation en intérieur mais très peu d'implémentations ont été faites à grande échelle. Une architecture sensorimotrice impliquant ces cellules a été développée au laboratoire ETIS. De plus la navigation autonome d'une voiture est un défi complexe. Il est donc intéressant d'étudier les architectures Bio-inspirée.

L'objectif de ce projet est l'adaptation de cette architecture sur un véhicule de l'institut VEDECOM afin de le contrôler. J'ai tout d'abord avoir pris en main l'architecture, le simulateur de réseau de neurone et remonté l'application de navigation sur plateforme robotisée. Puis j'ai effectué la communication entre le véhicule et l'ordinateur que l'on embarque. Enfin j'ai adapté l'architecture pour le véhicule. De premiers résultats sont présentés, de nombreux autres sont à venir.

Abstract

Earthlings show good capability to move in their environnements, finding back their habitat by avoiding obstacles. Robotic navigation have the same challenges than those whose animals overcame during their evolution. That's why studying their methods is interesting for roboticists.

The study of animals's brain enable the discovery of specialized cells for navigation. Scientists imitate those cells by simulating them on computers. These simulations allow indoor navigation, but few implementations were done at huge scale. A sensorimotor architecture using these cells have been developed in ETIS laboratory. Moreover car's navigation is a complex challenge, that's why it's interesting to study Bio-inspired architecture.

This project aims the adaptation of this architecture on one of VEDECOM's cars in order to control it. After handling the architecture, the simulator of neural networks and reassemble the application of navigation on robotic plateform. The adaptation to make the vehicle and the computer we embed is presented as well as the adaptation on the architecture. First results are presented and several others are coming.

Introduction

La navigation autonome pour les véhicule est un réel défi. Faire de la navigation autonome est très complexe. En effet, l'environnement est changeant et de grande taille. Pour la navigation en robotiques, de nombreux travaux faits sur la navigation autonome utilise,t les méthodes de SLAM (Simultaneous Localization And Mapping). Ces méthodes ont montré de bons résultats en navigation en intérieur ainsi qu'en extérieur. Les capteurs les plus souvent utilisés dans ces implémentations sont des télémètres lasers, des caméras et des sonars ont aussi été utilisés. Cependant, ces méthodes possèdent leurs limites et ont des difficultés à s'adapter à des environnement différents et à des conditions variées, comme le jour, la nuit ou l'été, l'hiver.

Afin de faire de la navigation autonomes sur véhicule il peut être intéressant d'étudier les stratégies mises en places par les animaux pour résoudre le défi qu'est la navigation. En effet, les animaux sont capables de se déplacer en évitant des obstacles et en empruntant des chemins déjà parcourus pour retrouver leur habitat, c'est donc une source de techniques fonctionnelles intéressante et ceux dans des environnements changeant et de taille variables.

Les cerveaux des animaux sont étudiés afin de comprendre le fonctionnement de leurs neurones. Ainsi de nombreuses cellules spécialisées dans la navigation ont été découvertes. Celles-ci simulées par ordinateur permettent de reproduire les mécanismes de navigation des animaux sur des plateformes robotisées. Ce type d'approche neuro-inspirée a déjà fait ses preuves pour de la navigation en intérieur ainsi que sur quelques tests en extérieur. Ces différents tests en extérieur sont toujours restés dans des échelles de distances relativement petites.

Une architecture modélisant ces cellules à été développée au laboratoire ETIS. Cette l'architecture sensorimotrice que nous allons adapter utilise principalement la vision, donc une caméra, pour se localiser. De bons résultats ont été obtenus par celle-ci en localisation sur des bases de données véhicule. L'objectif de mon stage est d'adapter ce type d'approche neuro-inspirée dans le cadre d'un véhicule de VEDECOM afin de faire de la conduite déléguée.

Dans la première partie de ce mémoire, je vais vous présenter les deux établissements qui m'ont accueilli lors de mon stage : Le laboratoire ETIS et l'institut de recherche VEDECOM. Le premier travail que j'ai effectué lors de mon stage était de faire une étude bibliographique de l'architecture que j'utilise afin de bien la comprendre. Je vous l'exposerais en deuxième partie. Ensuite, j'ai mis en oeuvre cette architecture sur une plateforme robotisée afin de la prendre en main dans des expérimentations. Ce que je vous présenterais dans la troisième partie. Puis j'ai adapter l'architecture pour qu'elle soit fonctionnelle sur le véhicule et ainsi j'ai obtenu les premiers résultats. C'est que ce que je vous présente en quatrième partie.

Chapitre 1

Établissements d'accueil

Mon projet de fin d'études, s'est déroulé en temps partagé entre le laboratoire ETIS et l'institut de recherche VEDECOM. En effet, il s'agit d'un stage en collaboration entre deux institutions. J'ai commencé par effectuer la première partie de mon stage dans le laboratoire ETIS.

1.1 Laboratoire ETIS

Le laboratoire ETIS (Équipes Traitement de l'Information et Système), est une unité de recherche commune au CNRS (UMR 8051), à l'ENSEA Cergy et à l'Université de Cergy-Pontoise. Le laboratoire ETIS est rattaché principalement à l'Institut des sciences informatiques et leurs interactions (INS2I). Il est structuré en quatre équipes de recherche :

- Indexation Multimédia et Intégration de données (MIDI)
- Information, Communications, Imagerie (ICI)
- Architectures, Systèmes, Technologies pour les unités Reconfigurables Embarquées (ASTRE)
- Neurocybernétique

Neurocybernétique est l'équipe avec laquelle j'ai eu l'occasion de travailler, afin d'étudier et de prendre en main l'architecture sensorimotrice dévellopée au sein du laboratoire sur une plateforme robotique en intérieur (Robulab). Plusieurs plateformes robotisées sont disponibles dans cette équipe de recherche : Tinos, un robot hydraulique, Berenson,Figure 1.1 un robot qui apprend à reconnaître la beauté de l'art dans les musées, Pinobo une tête robotique, RobotSoc un robot prévu pour de la navigation en extérieur.

L'équipe Neurocybernétique a pour but de faire de la robotique développementale et bio-inspirée. Le projet de l'équipe est de modéliser les mécanismes neuronaux et les structures cérébrales importantes lors du développement infantile. Dans le but de développer des robots autonomes qui puissent percevoir le monde extérieur, apprendre par eux-mêmes et interagir avec celui-ci. Les recherches en neurorobotique permettent de vérifier certaines théories de fonctionnement du cerveau issues des neurosciences.

1.2 Institut VEDECOM

L'institut de recherche VEDECOM est un hub de recherche coopérative créé en 2014. Certifié institut pour la transition énergétique (ITE) en 2014 par l'agence de recherche national (ANR), mis en place dans le cadre du programme d'Investissements d'Avenir (PIA) du gou-

FIGURE 1.1 – Le Robot Berenson

vernement Français,VEDECOM est dédié à la mobilité individuelle, décarbonée et durable. En 2018 VEDECOM emploie environ 200 personnes essentiellement ingénieurs ou docteurs ainsi que 45 doctorants.

En 2016, l'IVM (L'Institut pour la ville en mouvement) est intégré à VEDECOM. Cet institut étant un organisme à vocation internationale qui mène des recherches et actions innovantes dans le champ de la mobilité urbaine. L'IVM complète le programme d'innovation technologique de VEDECOM par une approche sociale et organisationnelle.

FIGURE 1.2 – Logo de l'Institut pour la Ville en Mouvement

L'institut est formé d'un écosystème d'environ 50 membres, acteurs publics et privés. Il conçoit et anime des modules de formations. En 2017, l'institut proposé 14 modules de formation.

En 2017 est né VEDECOM Tech, qui a pour but de valoriser les biens et savoirs de l'institut en les transformant en produits ou services commercialisables et aussi de consolider sa stratégie

d'innovation vis-à-vis des retours clients.

La mission de l'institut est de développer des technologies sûres, abordables, efficaces, et robustes pour la mobilité du futur. Pour ce faire VEDECOM est au cœur de trois principaux challenges :

— L'électrification des véhicules, afin d'améliorer la qualité de l'air dans les zones urbaines, ainsi que de réduire considérablement les émissions de CO2.
— La conduite déléguée et connectivité des véhicules, afin d'apporter de la sécurité au volant, et améliorer l'accessibilité à la mobilité.
— Pôle Mobilité et énergie partagées, en collaborant avec les autorités locales pour concevoir les infrastructures de demain.

La Figure 1.3 suivante synthétise les domaines de recherche de l'institut.

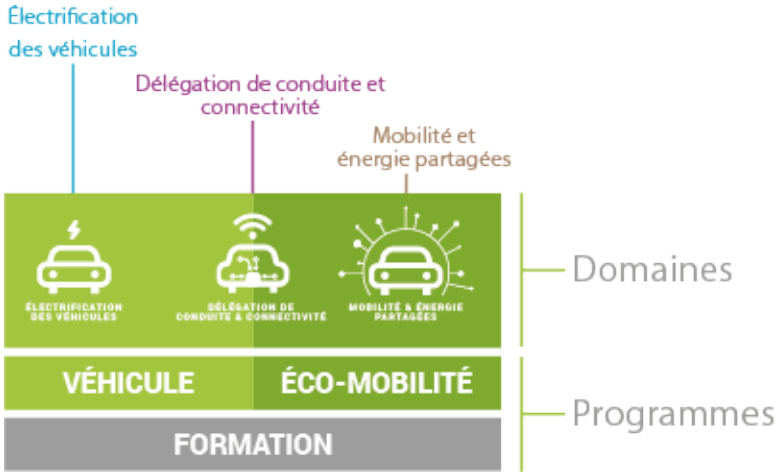

FIGURE 1.3 – Domaines de recherches de VEDECOM

Ces trois Domaines (Electrification des véhicules, délégation de conduite et connectivité, mobilité et énergie partagées) sont ensuite divisés en 15 projets. La 1.4 indique la liste des 15 projets actuellement étudiés par l'institut.

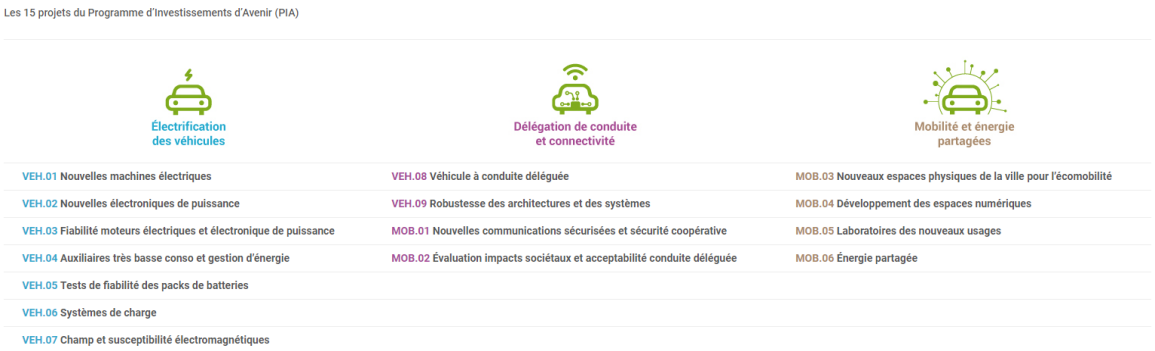

FIGURE 1.4 – Les 15 Projets du Programme d'Investissements d'Avenir (PIA)

L'objectif du domaine Délégation de conduite et connectivité dans lequel j'ai été intégré est de faire émerger des applications de véhicule autonome de niveau 4 et 5 pour proposer des solutions de mobilités plus sécurisées et plus efficaces. En effet, il existe plusieurs niveaux d'autonomie pour un véhicule, comme présenté en Figure 1.5. Ces niveaux d'autonomie dépendent de la responsabilité qu'il reste au conducteur à reprendre la main dans certains cas. Par exemple les véhicules incluant un régulateur de vitesse adaptatif sont au niveau 1 d'autonomie.

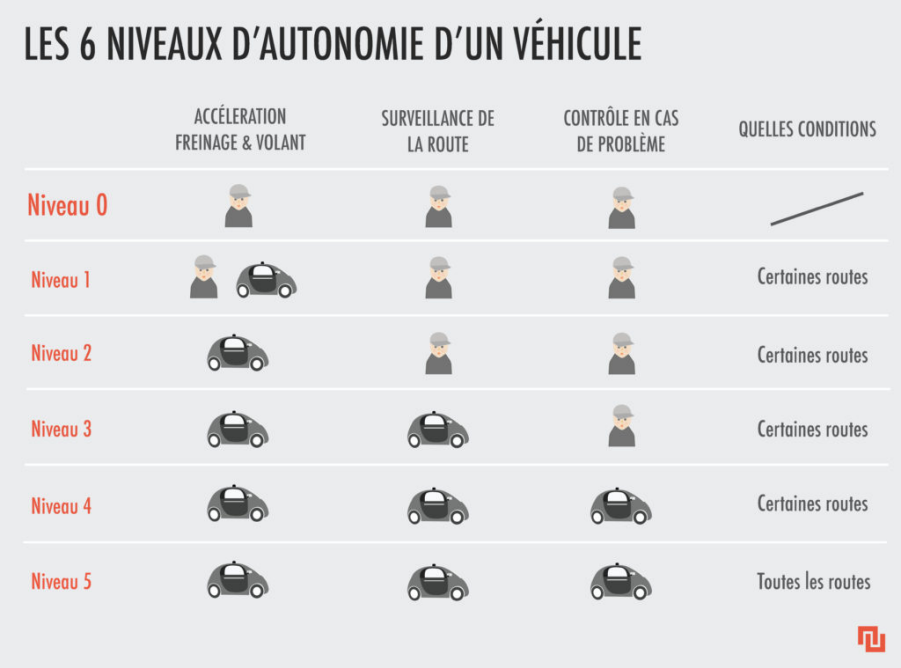

FIGURE 1.5 – Les 6 niveaux d'autonomie des véhicule autonome

Lors de mon stage, j'ai eu l'occasion d'intégrer le projet VEH.08, véhicule à conduite déléguée le projet véhicule à conduite délégué. L'équipe VEH.08 est composée de 25 personnes : 7 doctorants, 9 docteurs, 9 ingénieurs. L'équipe dispose de 4 véhicules pour faire leurs expérimentations, un véhicule de perception et 3 véhicules automatisés (2 Renault ZOE, 1 Renault C4).

Les objectifs du projet véhicule à conduite déléguée sont de développer des briques innovantes pour le véhicule autonome, ainsi que de les valider via des expérimentations sur véhicule. Les sujets développés dans l'équipe sont centrés sur la localisation et la cartographie, la perception de l'environnement et la détection des intentions des autres usagers (vulnérables ou non).

L'institut VEDECOM a la chance d'avoir des pistes d'essais en face de son établissement. Il s'agit d'une route fermée sur laquelle on peut faire des tests maîtrisés. Les différents tests que je vais avoir l'occasion de mener lors de mon stage s'effectueront principalement sur ses pistes d'essais. Elles possèdent l'avantage d'être relativement grande (5km pour la grande boucle) et de ne pas avoir d'autres usagers imprévisibles lors de nos tests. La vue satellites des pistes est donnée en Figure 1.6.

Les premiers tests s'effectueront sur la zone d'évolution, une zone d'environ 120*60m, vide d'obstacles avec des tracés au sol. Par la suite les tests s'effectueront sur de plus grandes parties des pistes. L'objectif étant de faire une boucle en utilisant uniquement l'architecture.

J'ai tout d'abord travaillé au laboratoire ETIS afin de comprendre et faire fonctionner l'architecture sensorimotrice sur une plateforme robotique mobile, avant de l'adapter sur un véhicule ZOE de l'institut VEDECOM.

FIGURE 1.6 – Vue satellites des pistes disponibles à Versailles Satory

Chapitre 2

Architecture sensorimotrice

Afin de prendre en main l'architecture sensorimotrice j'ai effectué la première partie de mon stage au laboratoire ETIS, ce qui m'a permis de tester l'architecture sensorimotrice sur un robot mobile et dans le même temps comprendre Promethe le simulateur de réseau de neurones créé, utilisé et maintenu par le laboratoire ETIS.

2.1 Introduction à la navigation bio-inspirée

L'hippocampe est une zone cérébrale située profondément dans le cerveau, voir Figure 2.1. C'est une structure qui se retrouve dans la majorité des mammifères. Cette région est impliquée dans la mémoire ainsi que dans la cognition spatiale. C'est principalement dans cette zone du cerveau que l'on retrouve des cellules spécialisées liées à la navigation. Un dessin de l'hippocampe par Santiago Ramón y Cajal (1911) est visible en Figure 2.2.

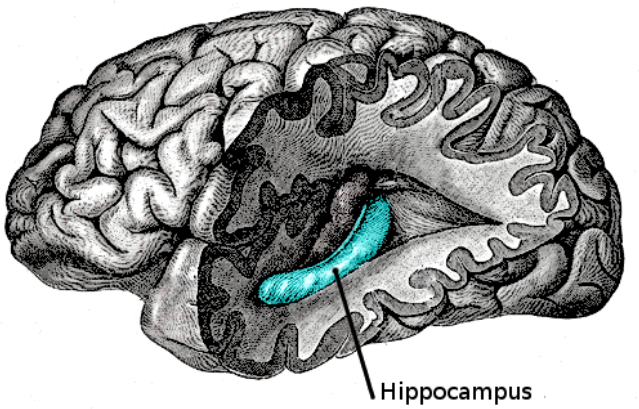

FIGURE 2.1 – Emplacement de l'hippocampe dans le cerveau humain

C'est dans l'hippocampe que l'on observe plusieurs types de cellules qui nous intéressent particulièrement dans le cadre de notre architecture sensorimotrice :

— Les cellules de lieu
— Les cellules de direction de tête
— Les cellules de grille.

Les cellules de lieu ont été découvertes dans le cerveau des rongeurs [17]. Elles permettent d'encoder des informations caractéristiques d'un endroit. Plus l'animal sera proche du lieu dans

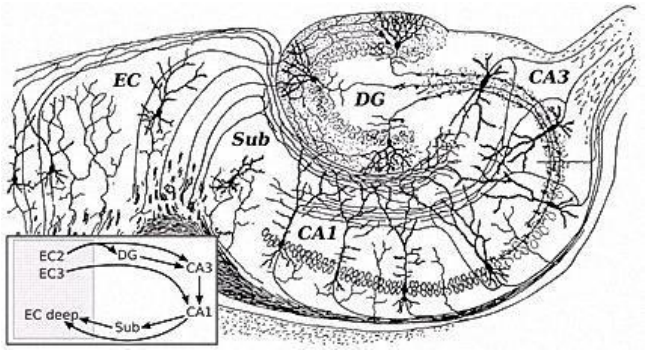

FIGURE 2.2 – Dessin de l'hippocampe par Santiago Ramón y Cajal (1911)

lequel a été créé la cellule, plus la réponse neuronale sera forte, permettant ainsi de reconnaître un endroit préalablement visité. Ces cellules existent dans plusieurs parties de l'hippocampe. Leur champ de lieu, distance pendant laquelle une cellule a la plus forte activité par rapport aux autres, est plus ou moins grand en fonction de leurs emplacements dans le cerveau [5]. Elles ont plus récemment été observées dans le cerveau humain [8]. Chez les primates, des cellules de vue ont été mises en évidence[19]. Contrairement aux cellules de lieu qui sont omni-directionnelles, ces cellules s'activent dans une unique direction. Les cellules de direction de tête sont des neurones qui décrivent la direction absolue de la tête de l'animal [21]. Les cellules de grille ,quant à elles, s'activent périodiquement en fonction d'une distance parcourue [15]. L'ensemble des positions où la cellule s'active forme une grille hexagonal.

Des modèles de ces différentes cellules ont été implémentés en simulation et sur plateformes robotisées, afin de valider leurs fonctionnement [2]. Les cellules de lieu associées avec les cellules de directions de tête ont permi de de poser les briques de base d'une architecture de navigation appelée PerAc [11]. De nombreux travaux utilisent cette architecture en y ajoutant de nouvelles cellules (des cellules de transition afin de rejoindre une zone d'intérêt [6], des cellules de grille [10]).

Nous allons maintenant présenter plus dans le détail le fonctionnement des principales cellules qui entrent en jeu dans l'architecture sensorimotrice utilisée au cours de ce stage.

2.2 Cellule de direction de tête

Les premières cellules qui vont nous intéresser dans l'implémentation de notre architecture sont les cellules de direction de tête. Ces cellules agissent comme une boussole interne en encodant une direction absolue. En effet elles s'activent en fonction de la direction dans laquelle la tête de l'animal pointe. Ces cellules de direction de tête peuvent être créées chez les animaux à partir d'informations visuelles, ou d'informations idiothétiques. Les informations idiothétiques sont des informations proprioceptives. Les informations allothéthiques quand à elles sont celles externes à l'animal.

Dans nos expérimentations nos cellules de direction de tête seront générées à partir des informations d'un compas magnétique associé à l'odométrie en utilisant le modèle [7]. Il est aussi possible d'utiliser un modèle de boussole visuelle, qui utilise les informations idiothétiques (odométrie) et allothétiques (vision) [12]. Ces modèles de boussole permettent d'améliorer la précision et la robustesse, par rapport à un compas magnétique par exemple. En effet, une boussole magnétique peut subir des perturbations magnétiques et ainsi renvoyer une valeur erronée. Pour la voiture nous utiliserons l'information de lacet provenant d'une centrale inertielle fusionnée avec un GNSS (Global Navigation Satellite System).

Après avoir récupéré la valeur de direction absolue du robot, nous l'encoderons sur $N_{azimuth}$ neurones. De plus, nous calculerons le produit de convolution de cette valeur avec une gaussienne, afin d'obtenir une bulle d'activité. Ces cellules seront utilisées lors de la création des cellules de lieu afin d'associer des informations visuelles à une orientation spécifique.

2.3 Modèle de cellule de lieu

Les cellules de lieu sont formées à partir de plusieurs informations : les informations visuelles ainsi que les informations venant des cellules de direction de tête qui correspondent à la direction absolue dans laquelle l'image est acquise. Ces cellules ont été modélisées par de plusieurs chercheurs [3] [20].

Les cellules de lieu visuelles ont été observées chez les primates, les cellules de lieu ont été observées chez les rongeurs. La diversité de ces cellules pourrait venir de la différence de champ de vision. En effet les rongeurs ont un grand champ de vision par rapport aux primates qui ont les deux yeux dans la même direction. Cette particularité permet d'expliquer la différence entre les cellules de lieu et les cellules de lieu visuelles.

Le modèle de cellule de lieu que l'on utilise permet de simuler ces deux types de cellules en fonction du type de caméra utilisé (fisheye ou faible champ de vision) et de l'ajout ou non d'une monture panoramique.

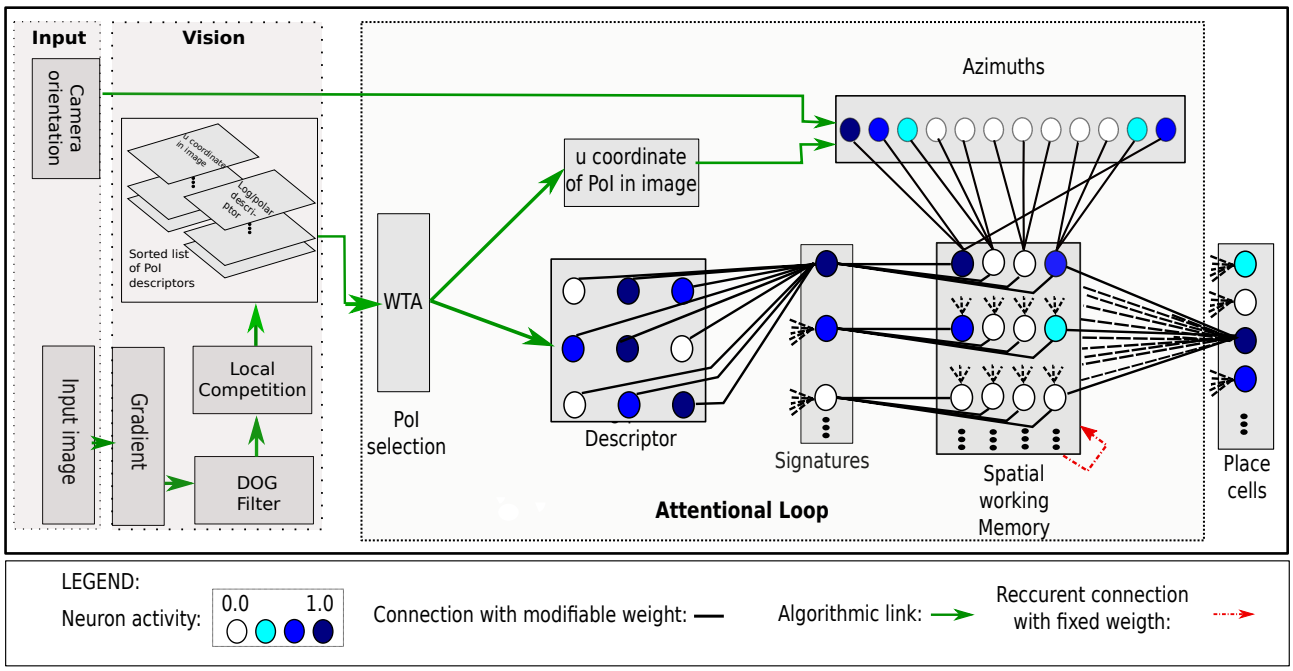

FIGURE 2.3 – Architecture du modèle de cellule de lieu [9]

Voici en Figure 2.3 le modèle de cellule de lieu que l'on utilise. Ce modèle est constitué de deux sous ensembles, une chaîne de traitement visuel, qui alimente une partie neuronale.

2.3.1 Chaîne visuelle

La première étape de notre modèle de cellule de lieu est le traitement visuel de l'image récupérée afin d'en extraire des descripteurs qui vont servir à caractériser le lieu dans lequel nous nous trouvons. L'image acquise par la caméra est tout d'abord transformée en niveaux

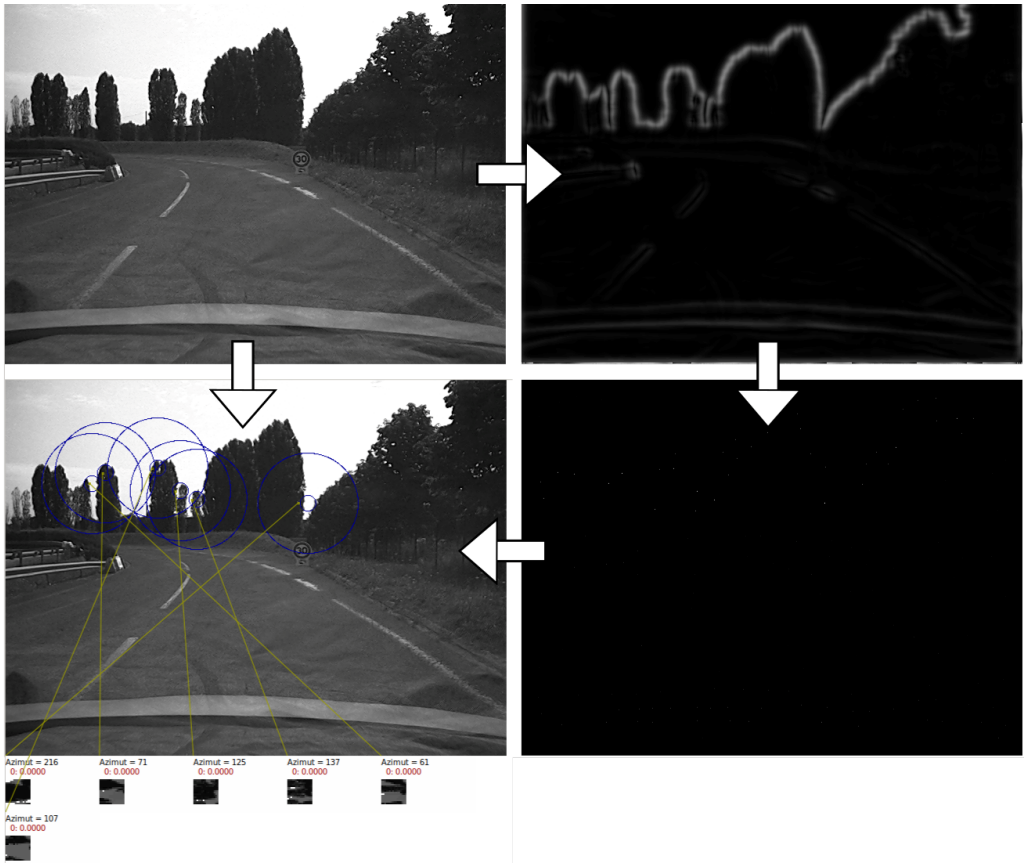

FIGURE 2.4 – Chaîne visuelle du modèle.

de gris. On applique ensuite un filtre qui se base sur les gradients afin d'extraire les contours de l'image. Ce contour est ensuite convolué avec un filtre de différence de gaussiennes (DoG) d'écart-types respectif σ_{DoG_1} et σ_{DoG_2}, ce qui permet de trouver les potentiels points d'intérêts. Pour finir, une compétition locale est faite sur ces potentiels points d'intérêts, afin de récupérer les maxima locaux.permet également de s'assurer d'une répartition spatiale des points d'intérêt dans l'image en excluant ceux situés dans un rayon r_{compet} des maxima déjà sélectionnés. Au final, N_{PoI} points d'intérêt sont conservés.

Pour chaque point d'intérêt on extrait une portion de l'image autour de celui-ci, comprise entre deux disques de rayon r_{small} et r_{big} , que l'on appelle vue locale. Celle-ci subie finalement une transformation en log polaire. Cette transformation mime la structure de la rétine [1]. Elle a l'avantage d'être invariante aux petites rotations et petits changements d'échelle. Pour éviter une trop forte dilatation d'informations au centre de la vue locale, on exclut un rayon r_{small} autour du point d'intérêt. La liste des points d'intérêt et des vues locales projetées en log polaire sont envoyées à la partie neuronale du système.

2.3.2 Partie neuronale

La partie neuronale commence par une couche *descripteur* qui correspond à la projection de la vue locale sur une matrice de taille $N_d \times N_d$. Chaque neurone de cette couche est connecté à l'ensemble des neurones de la couche *signature* par des liens avec des poids modifiables, modélisant les poids synaptiques entre les neurones. La couche *signature* encode la description des points d'intérêt faite par le descripteur. Pour ce faire chaque neurone de signature s_i mémorise une configuration du descripteur sur ses poids synaptiques. A chaque temps t de la simulation,

l'activité des neurones de la couche *signature* est exprimée comme suit :

$$s_i(t) = 1 - \frac{1}{N_d^2} \sum_{j=1}^{N_d} \sum_{k=1}^{N_d} |w_{jk,i} - d_{jk}(t)| \tag{2.1}$$

— d_{jk} est le niveau d'activité du neurone du descripteur présent à la j^{eme} ligne et à la k^{eme} colonne au temps t.

— $w_{jk,i}$ est le poids synaptique reliant le i^{eme} neurone de la couche *signature* (s_i) au neurone appartenant au descripteur à la j^{eme} ligne et à la k^{eme} colonne.

Cette équation calcule l'erreur entre les poids synaptiques et les activités des neurones du descripteur. Si l'erreur est faible alors l'activité $s_i(t)$ du neurone i de la couche signature, sera élevée (proche de 1). Lors du recrutement du PoI, cette activité est maximale et vaut 1. Concrètement, cette équation, évalue la ressemblance entre le point d'intérêt étudié et les signatures enregistrées.

Ensuite, la couche *azimuth* permet de savoir où la direction (encodée de façon absolue) dans laquelle se situe le point d'intérêt étudié. Cette couche est générée à partir des cellules de direction de tête qui donnent l'orientation absolue de la caméra. Afin d'exprimer la direction du point d'intérêt dans un repère global, la coordonnée horizontale (en pixels) de celui-ci ainsi que l'angle d'ouverture de la caméra sont utilisées afin de calculer sa direction dans le repère caméra. Cette direction locale est ensuite sommée à l'orientation absolue de la caméra afin d'obtenir la direction du point d'intérêt observé vis-à-vis d'une référence absolue. Cela a l'avantage de pouvoir discriminer plus facilement les points d'intérêt dans l'environnement (des descripteurs similaires peuvent exister dans l'environnement dans des directions similaires) et ainsi d'être localement invariant à l'orientation du robot.

L'orientation absolue du point d'intérêt est encodée sur une population de $N_{azimuth}$ neurones, appelé couche *azimuth*. L'encodage sur ces neurones se fait via un profil Gaussien non-normalisé défini par un écart type $\sigma_{azimuth}$, comme cela est exprimé par l'équation 2.2.

$$a_i(t) = exp(-\frac{((\theta^l - \frac{2\pi}{N_a} i) \ mod \ 2\pi)^2}{2\sigma_{azim}^2}) \tag{2.2}$$

— θ_k^l est l'azimuth selon lequel le l^{eme} PoI est observé.

— $\frac{2\pi}{N_a} i$ est la direction encodée par le neurone i.

En utilisant cette bulle d'activation, il n'est pas nécessaire que le point d'intérêt soit retrouvé avec précision à la même orientation, et permet une meilleur généralisation des cellules de lieu.

Nous avons maintenant, d'un côté l'information de signature des points d'intérêts (la couche *signature*) et l'information d'emplacement, leurs directions absolues (la couche *azimuth*). Ces deux couches sont connectées avec des liens dont les poids synaptiques sont modifiables à un champ neuronal mémoire de travail ou Spatial Working Memory (SWM). Chaque neurone de la mémoire de travail encode un point d'intérêt selon une orientation.

Les neurones de la couche SWM sont organisés selon un champ en deux dimensions, de taille $N_{SWM} = N_{signature} \times N_l$. Le nombre de lignes est le même que le nombre de signatures. Le nombre de colonnes N_l spécifie le nombre d'orientations différentes où il est possible de reconnaître un même PoI. Chacun des N_l neurones est relié à un sous-ensemble de la couche *azimuth*, de taille $l_\theta = \frac{N_a}{N_l}$.

Pour chaque nouvelle image, cette mémoire de travail se remplit via les activités des différents PoI extraits (signature et azimuth). Afin d'obtenir des cellules de lieu plus générale, il est possible d'utiliser une caméra panoramique, dans ce cas la mémoire de travail conserve

les activités le temps du traitement des points d'intérêt sur toutes les images du panorama. L'équation 2.3 régit l'activité des neurones de la mémoire de travail.

$$SWM_{il}(t) = f(SWM_{il}(t-1) + I_{il}(t) - SWM_{il}(t-1).In(t)) \tag{2.3}$$

In est un signal d'inhibition permettant d'effacer la mémoire de travail lorsque l'on passe d'une image à une autre. I_{sj} est le signal d'entrée qui active le neurone SWM_{sj}. Il est calculé à l'aide de l'équation 2.4.

$$I_{sj}(t) = (s_i(t).w_{s,il}(t)).(\max_{j \in N_{al}}(a_j(t).w_{j,sj}(t))) \tag{2.4}$$

Le sj_{eme} neurone de la mémoire de travail est connecté à un sous-ensemble de neurones appartenant à la couche *azimuth*. La liste des indices de chaque neurone de ce sous-ensemble est représentée par N_{al} dans l'équation précédente 2.4. N_{al} est la liste des indices des neurones de la couche *azimuth* qui font partie du sous-ensemble de neurones liés au sj^{eme} neurone de la SWM.

$f(x)$ est donnée en équation 2.5.

$$f(x) = \begin{cases} 1, & \text{si } x > 1 \\ 0, & \text{si } x < 0 \\ x, & \text{sinon} \end{cases} \tag{2.5}$$

Une fois l'image ou le panorama entièrement traité, la SWM est remplie des informations visuelles et de leur localisation, elle regroupe toutes les informations du lieu observé. C'est donc à partir de cette couche que sont modélisées les cellules de lieu.

On effectue le même raisonnement qu'entre la couche de *descripteur* et la couche *signature*, avec la couche SWM et la couche *Lieu*. En effet chaque neurone de la couche SWM est connecté aux neurones de la couche lieu.

Chaque neurone de la couche *Lieu* a une activité définie par l'équation 2.6.

$$p_m(t) = 1 - \frac{1}{N_s * N_l} \sum_{j=1}^{N_s} \sum_{k=1}^{N_l} |w_{jk,m}^{SWM}(t) - SWM_{il}(t)| \tag{2.6}$$

— SWM_jk est le j^{eme} ligne et k^{eme} élément appartenant à la mémoire de travail.
— $w_{jk,m}^{SWM}$ est le poids synaptique reliant le neurone p_m au neurone x_{jk} de la couche SWM.

C'est dans cette dernière couche *Lieu* que l'on obtient les activités des différents lieux appris, par rapport au lieu où se situe actuellement le robot.

2.3.3 Apprentissage

Il faut bien discerner deux étapes dans notre modèle :
— L'étape d'apprentissage lorsque l'image ou le panorama en cours va être utilisé pour apprendre un nouveau lieu et donc de nouveaux points d'intérêt.
— L'étape d'exploration où l'on utilise les lieux déjà appris pour se localiser.

L'ensemble de ce modèle est activé par un signal binaire qui active le recrutement d'une nouvelle cellule de lieu ainsi que des points d'intérêt qui lui sont associés. Ce signal est appelé vigilance. Lorsque celui-ci est à 1 alors le modèle apprend la signature de tous les points d'intérêt trouvés dans l'image et apprend l'emplacement des points d'intérêt dans la couche SWM et assimile l'état de la SWM dans la couche *Lieu*.

Dans notre simulateur Promethe, présenté dans la partie 3.1 chaque neurone possède un indicateur qui permet de déterminer s'il est en mesure d'apprendre ou s'il a déjà appris.

L'apprentissage débute avec la couche *signature* et les poids synaptiques reliant tous les neurones de la couche *descripteur* à ceux de la couche *signature*. Pour apprendre faut avoir l'ordre d'apprendre et ne pas avoir déjà appris. Seul le prochain neurone à ne pas avoir appris le fait.

De cette vigilance viens l'équation d'apprentissage de la couche *signature*.

$$w_{jk,i}(t) = (1 - V_{igi}).w_{jk,i}(t-1) + d_{jk}.V_{igi} \tag{2.7}$$

— $w_{i,jk}$ étant le lien reliant le i^{eme} neurone de la couche *signature* au neurone de la j^{eme} ligne et de la k^{eme} colonne de la couche *descripteur*.

— d_{jk} activité du neurone présent à la j^{eme} ligne et à la k^{eme} colonne de la couche *descripteur*.

Pour la SWM les équations 2.8 et 2.9 régissent l'apprentissage des poids synaptiques $w_{s_i,x_{il}}$ et $w_{a_j,x_{il}}$ liant le neurone SWM_{il} de la mémoire de travail respectivement au neurone actif s_i de la couche *signature* et au neurone actif a_j de la couche *azimuth*.

$$w_{s_i,x_{il}}(t) = (1 - V_{igi}(t)).w_{s_i,x_{il}}(t-1) + V_{igi}(t) \tag{2.8}$$

$$w_{a_j,x_{il}l}(t) = (1 - V_{igi}(t)).w_{a_j,x_{il}} + V_{igi}(t) \tag{2.9}$$

Uniquement les poids reliant le neurone apprenant aux couches précédentes sont ainsi mis à 1. L'apprentissage des cellules de lieu est régi par la même équation que pour la couche *signature* 2.7. Voici son équation 2.10.

$$w_{m,il}(t) = (1 - V_{igi}).w_{m,il}(t-1) + SWM_{il}.V_{igi} \tag{2.10}$$

Les cellules de lieu sont apprises d'un seul coup, il peut donc y avoir des points d'intérêt appris pour le lieu qui ne seront pas retrouvés par la suite et donc non pertinents (par exemple, un oiseau dans le ciel). Pour éviter d'avoir une mauvaise reconnaissance d'un lieu à cause de l'occlusion de points d'intérêt, il est possible d'améliorer le modèle. Pour ce faire le principe est d'utiliser, pour le calcul de l'activité des cellules de lieu, uniquement les points d'intérêt les plus reconnus (25%), ce qui permet, lorsqu'une partie des points d'intérêt appris ne sont pas visibles sur l'image, de ne pas complètement faire chuter l'activité des cellules de lieu et de toujours reconnaître convenablement le lieu [14]. En contrepartie, cela implique une plus forte concurrence entre les cellules de lieu.

Plusieurs méthodes seront utilisées pour envoyer un signal de vigilance à 1 lors de nos expérimentations. Sur la plateforme robotisée, l'utilisateur peut manuellement indiquer quand apprendre un lieu, voir 3.2. Pour le véhicule les cellules de lieu seront apprises automatiquement lors du tour d'apprentissage. Pour ce faire, on implémentera un seuil S_v en dessous duquel on considère qu'un lieu n'est pas reconnu. Si aucune cellule de lieu ne possède d'activité supérieure à cette valeur, alors à la prochaine image sera utilisé pour apprendre un nouveau lieu.

Nous avons décrit le modèle de cellule de lieu que nous allons utiliser. Afin d'améliorer la qualité de la localisation, nous nous proposons également d'utiliser des cellules de grille.

2.4 Intégration de chemin et cellule de grille

Les cellules de grilles ont, elles aussi été mises en évidence dans le cerveau, plus récemment que les cellules de lieu. Elles s'activent de façon périodique par rapport à une distance parcourue.

Il existe des cellules de grille à plusieurs endroits dans le cerveau. elles ont la propriété d'avoir des tailles différentes. La taille d'une cellule de grille est la distance entre deux activations successives. Le fait d'avoir des cellules de grille de différentes tailles permet de discrétiser plus finement l'environnement exploré.

Le modèle de cellule de grille que nous utilisons les génère à partir de l'intégration de chemin [16][10], à la différence d'autres modèles qui utilisent les cellules de grille dans le but de faire de l'intégration de chemin. L'intégration de chemin est utilisé en navigation à l'estime pour présumer la position du véhicule.

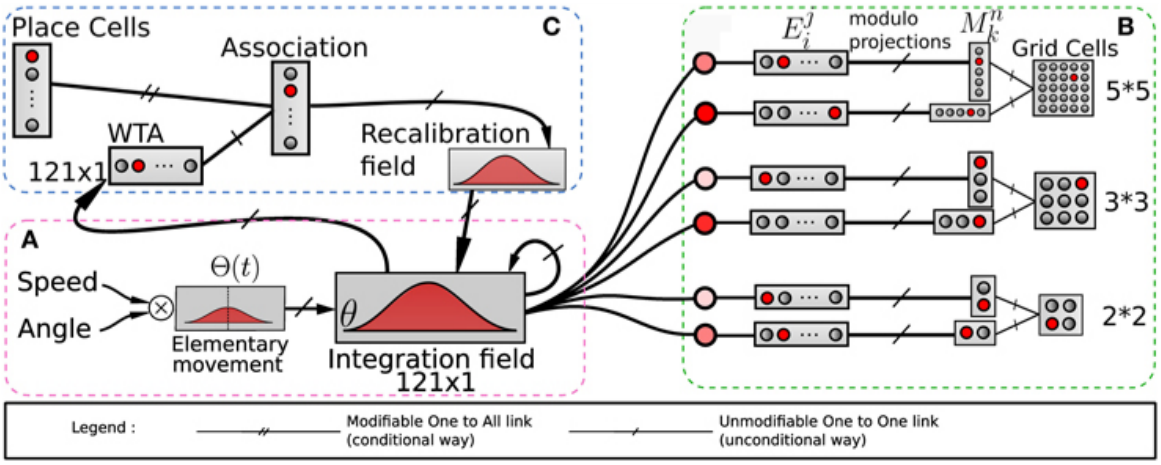

FIGURE 2.5 – Architecture de l'intégration de chemin, de son mécanisme de recalibration, ainsi que des cellules de grille [16].

La Figure 2.5 montre le fonctionnement global du modèle que nous allons maintenant détailler.

2.4.1 Intégration de chemin

La partie intégration de chemin est schématisée dans la Partie A de la Figure 2.5. Afin de générer notre intégration de chemin, nous utilisons les odomètres du robot ou du véhicule, qui nous permet d'obtenir la distance parcourue depuis la dernière itération. Nous utilisons également l'orientation actuelle du robot obtenue avec un compas magnétique ou une boussole visuelle.Ces informations sont utilisées afin de calculer ce que l'on appelle mouvement élémentaire effectué et qui s'exprime sur N_{me} neurones, en calculant son produit de convolution avec un cosinus. Le champ de mouvement élémentaire est régi par l'équation 2.11.

$$ME_i(\theta_{robot}(t)) = d_{iter} * \cos(\theta_{robot}(t) - \theta_i) \tag{2.11}$$

$$\theta_i = -2\pi \frac{i}{N_{me}} \tag{2.12}$$

Avec :

— d_{iter} étant la distance parcourue depuis la dernière itération.

— $\theta_{robot}(t)$ l'orientation actuelle du robot, donnée par les cellules de direction de tête.

Cela nous génère le mouvement élémentaire actuel. Le Champ d'intégration que l'on peut voir dans la partie A de la Figure 2.5 somme les mouvements élémentaires.

$$CI_i(t) = [CI_i(t-1) + ME_i(\theta_{robot}(t))]^+ \tag{2.13}$$

$$[x]^+ = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases} \tag{2.14}$$

Le champ d'intégration a une valeur maximale dans la direction globale du mouvement. L'activité de la direction globale est proportionnelle à la distance pour retourner à la position initiale.

2.4.2 Recalibration

Le système de recalibration est schématisé dans la partie C de la Figure 2.5. Le champ d'intégration est créé à partir d'informations idiothétiques. Cependant, de façon analogue à n'importe quel modèle se basant sur des données odométriques, les erreurs d'intégration se cumulent. Pour éviter cela, un système de recalibration à l'aide des cellules de lieu est mis en place. Pour ce faire, à chaque cellule de lieu se retrouve associé l'état courant de l'intégration de chemin (une direction θ ainsi qu'une amplitude, qui correspond à la distance à parcourir pour revenir au point de départ). A partir de cet apprentissage, lorsqu'une cellule de lieu possède une activité suffisante par rapport à un seuil absolu, et suffisante par rapport aux activités des cellules de lieu concurrentes, alors on recalibrera l'intégration de chemin en utilisant le champ enregistrer pour la cellule de lieu gagnante.

2.4.3 Cellule de grille

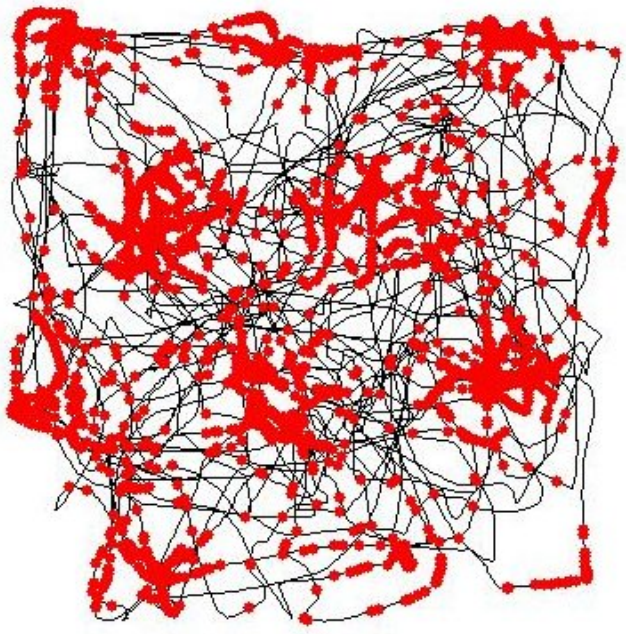

FIGURE 2.6 – Les lignes noires représentent la trajectoire d'un rat dans un environnement carré. Les points rouges montrent les activations d'une unique cellule de grille.

La Figure 2.6 montre le fonctionnent des cellules de grille. On y observe ainsi une grille triangulaire. On distingue aussi une distance d'activation entre deux activations. Notre modèle

de cellule de grille utilisé est indiqué dans la partie C de la Figure 2.5. Le modèle utilisé se base sur celui présenté dans [16]. On récupère la valeur du champs d'intégration selon 2 orientations. Ensuite on discrétise cette valeur. L'équation 2.15 régit la discrétisation pour le neurone j.

$$D_i^j(t_s) = \begin{cases} 1, & \text{if } j = floor(\frac{PI_i(t_s) \times N_E}{D_{max}}) \\ 0, & \text{otherwise} \end{cases} \tag{2.15}$$

— D_{max} distance maximale pouvant être encodée par le champ d'intégration PI.
— N_E nombre de neurones utilisé pour discrétiser.

A partir de cette valeur discrétisée, on en prend ensuite le modulo afin d'avoir une période spatiale d'activation. En projetant le champ de neurones D_i^j sur un champ M^n de plus petite taille. L'équation 2.16 décrit l'opération effectuée.

$$M_k^n(t_s) = \begin{cases} 1, & \text{if } k = argmax_j(D_i^j(ts)) \mod MG^n \\ 0, & \text{otherwise} \end{cases} \tag{2.16}$$

MG^n étant le modulo utilisé pour générer notre cellule de grille. Cette valeur représente une distance dans l'unité abstraite $\frac{D_{max}}{N_E}$ et représente la taille des cellules de grille générées. Ensuite, le produit matriciel entre ces deux vecteurs, ayant chacun un seul neurone actif, est calculé pour générer des cellules de grille.

La différence absolue entre θ_1 et θ_2 choisie dans le champ d'intégration donne l'orientation de la grille. Chaque neurone de la couche Grid Cells de la partie B de Figure 2.5 d'un modulo donné correspond à une cellule de grille avec la même orientation et taille, mais à une phase différentes. La taille des cellules de grille ainsi créées, dépendent de la valeur MG^n [16].

2.5 Cellule de lieu multimodale

Dans cette partie nous allons combiner ces deux types de cellules (Lieu et grille) afin de créer des cellules de lieu multimodales.

Voici en Figure 2.7 notre implémentation des cellules de lieu multimodales.

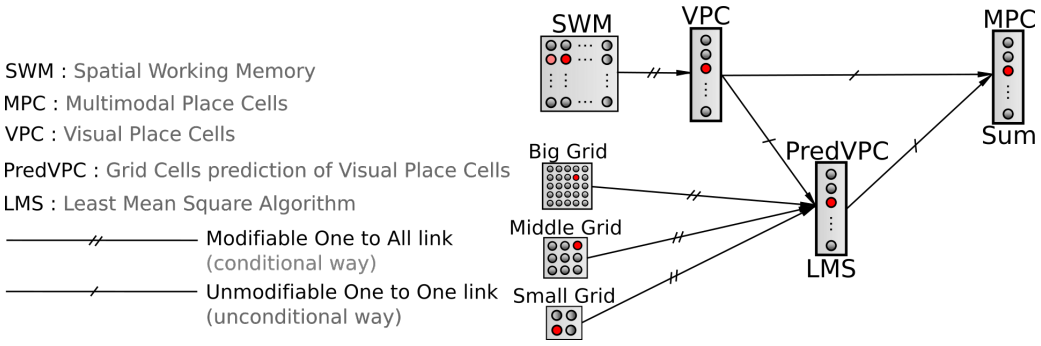

FIGURE 2.7 – Modèle de cellule de lieu multimodale [16]

Il a été montré qu'il était possible de générer des cellules de lieu visuelle à partir des cellules de grille [10]. Ce sont les Cellule de lieu visuelle prédites par les cellules de grille (PREDVPC). Dans la Figure 2.7, on utilise un lien conditionnel, indiquant le neurone qui doit apprendre, provenant des cellules de lieu lors de leur apprentissage, pour associer la disposition des cellules de grille à une cellule de lieu. Cependant les auteurs de [10] ont montré que ces cellules prédites possèdent un champ de lieu restreint. Afin de l'augmenter, et ainsi d'améliorer

la généralisation, nous utiliserons des connexions latérales afin de ne plus avoir une activité binaire, mais décroissante de la distance à la cellule de grille qui est activée. En pratique nous calculerons le produit de convolution de notre cellule de grille par un champ gaussien.

Nous avons désormais modélisé des cellules de lieu visuelles à partir d'informations visuelles (points d'intérêt), et des cellules de lieu prédites à partir de cellules de grille. A partir de celles-ci deux dernières nous modélisons les cellules de lieu multimodale MPC, en sommant l'activité des cellules de lieu, et les cellules de lieu prédites. L'activité de la i^{eme} cellule multimodale est donnée par l'équation 2.17

$$MPC_i(t) = \alpha VPC_i(t) + (1 - \alpha)P_{red}VPC_i(t) \tag{2.17}$$

Avec alpha un facteur de pondération. Dans nos expériementations nous prendrons une valeur de 0.5. $P_{red}VPC$ est le champ d'activité des cellules de lieu prédites par les cellules de grille.

2.6 Champs de neurones dynamiques

Depuis le début de la partie deux, sont présentées des cellules utilisées dans le cerveau afin de faire de la localisation. Ici nous allons exposer la méthode que nous allons utiliser pour apprendre des actions, à savoir les commandes à appliquer au robot pour qu'il rejoigne un point donné.

Afin de faire de la navigation pour suivre une trajectoire, nous associons à chaque cellule de lieu la direction actuelle du robot, comme présenté dans la partie A de la Figure 2.8. Le champ Orientation, correspond au même champ *Azimuth* de la partie cellule de lieu. L'association a pour effet de sortir l'orientation apprise pour la cellule de lieu entrante. Afin d'associer l'orientation à une cellule de lieu, une couche WTA (Winner Take All) est ajouté. Cette couche permet de ne garder actif que le neurone avec la plus grande activité. Il est possible dans ce genre de mécanisme de choisir le nombre de gagnants. Dans la Figure 2.8 l'entrée du WTA est la réponse des cellules de lieu visuelles. Le même mécanisme peut aussi s'appliquer aux champs d'activation des cellules de lieu multimodales, ou aux cellules de lieu prédites par les cellules de grilles.

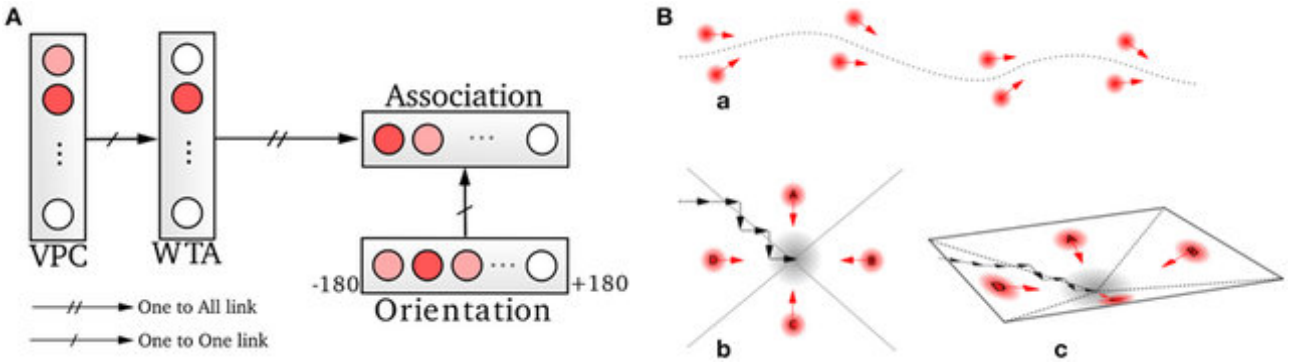

FIGURE 2.8 – Modèle du mécanisme d'association entre les cellules de lieu et l'orientation et exemple d'utilisation de celle-ci.

La partie B de la Figure 2.8 montre quelques exemples de cellules de lieu et d'orientation apprises. En effet chaque point rouge associé à une flèche rouge, correspond une cellule de lieu apprise et sa direction associée.

Ce modèle nous donne donc une orientation cible. Celle-ci peut être calculée à partir de plusieurs cellules (les cellules de lieu et les cellules multimodales par exemple). Ensuite on calcule

le produit de convolution de la direction cible encodée sur N_d neurones avec une gaussienne afin d'utiliser les champs de neurones dynamiques. Nous avons alors une gaussienne centrée sur la direction cible. Il est alors possible d'inhiber certaines parties de ce champ. Cette particularité est utilisée sur le robot (voir section 3.2) afin d'inhiber les directions où des obstacles ont été observés par les capteurs laser.

Ensuite pour envoyer des commandes on utilise la notion de champ de neurones dynamique ou DNF (Dynamical Neural Networks). Le principe est de dériver la gaussienne obtenue précédemment qui renseigne sur la direction cible. La valeur de la dérivée correspondant à l'orientation actuelle renseigne directement sur la vitesse de rotation à envoyer aux actuateurs du robot. Ainsi nous avons un attracteur vers la direction cible, comme illustré dans la Figure 2.9 [18].

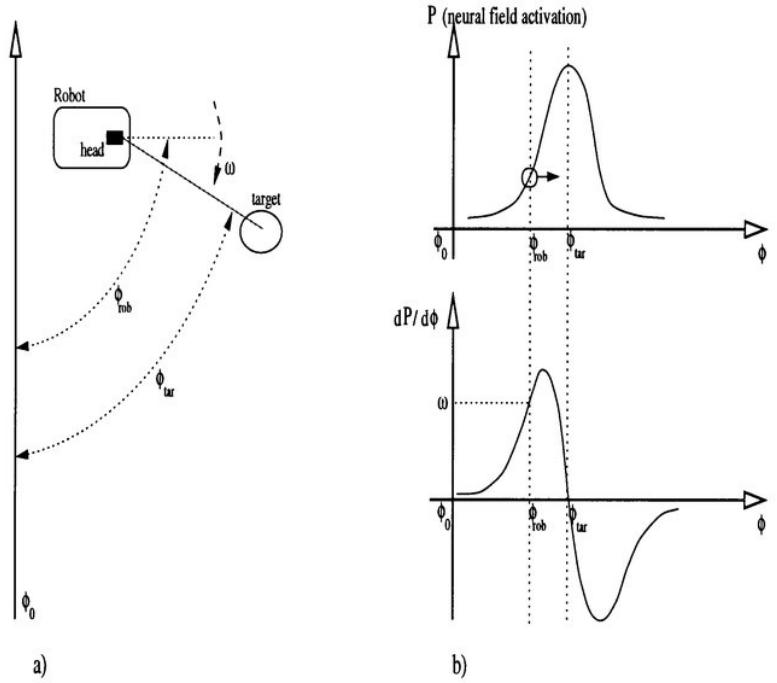

FIGURE 2.9 – Exemple fonctionnement champ de neurones dynamique [18]

Chapitre 3

Validation préliminaire sur plateforme mobile Robosoft

Après avoir fait la recherche bibliographique l'architecture sensorimotrice et afin de bien prendre en main et de bien comprendre l'architecture développée au sein du laboratoire ETIS, j'ai continué mon stage dans le laboratoire ETIS avec pour objectif de faires fonctionner le modèle sur une plateforme robotisée à l'aide du simulateur de neurones Promethe.

3.1 Présentation du simulateur de neurones Promethe

Ce modèle sensorimoteur développé dans l'équipe Neurocybernétique, a été testé dans de nombreuses simulations et implémentations sur plateformes robotisées à l'aide de Promethe, un simulateur de réseau de neurones créé et maintenu par le laboratoire ETIS. Il est écrit dans le langage de programmation C et fonctionne sur la distribution Ubuntu 18.04 LTS. Il permet de faire de la simulation de réseau de neurones en temps réel. On peut ainsi créer des applications de toutes sortes : navigation, apprentissage des expressions faciales, etc. Il existe Promethe et bPromethe (blind Promethe). Avec Promethe il est possible d'observer les activités des neurones de nos scripts en temps réel de façon synchrone, ce qui n'est pas le cas avec bPromethe. L'affichage de l'activité des neurones ayant un fort coût computationnel, Promethe fonctionne à une vitesse réduite par rapport à bPromethe.

On appellera script sur Promethe un programme décrivant un réseau de neurone. Chaque script est une suite de fonction, boîte, groupe de neurones. Les scripts possèdent différentes échelles de temps. Chaque échelle de temps s'exécutent un certain nombre de fois (définis dans un fichier de configuration). Chaque boîte s'exécute l'une après l'autre en attendant la boîte d'avant si le lien la reliant n'est pas secondaire. Il existe de très nombreuses fonctions, il est aussi possible de développer les siennes. En Figure 3.1 on peut voir l'interface graphique Coeos sur un script qui permet de tester différentes manières de transformer une image couleur en niveau de gris, tester l'effet de la normalisation sur l'image, et sur le gradient de l'image.

Promethe utilise plusieurs fichiers pour fonctionner. Certains sont facultatifs et dépendent de la complexité de l'application. Il est possible d'exécuter Promethe avec seulement le .script, .config et .res.

- Le .var répertorie les valeurs des variables.
- Le .res la valeur des poids synaptiques des reseaux de neurones.
- Le .symb décrit le réseau de neurones, en utilisant au besoin des variables.
- Le .script utilise le .symb en remplaçant la valeur des variables par celles écrites dans le .var

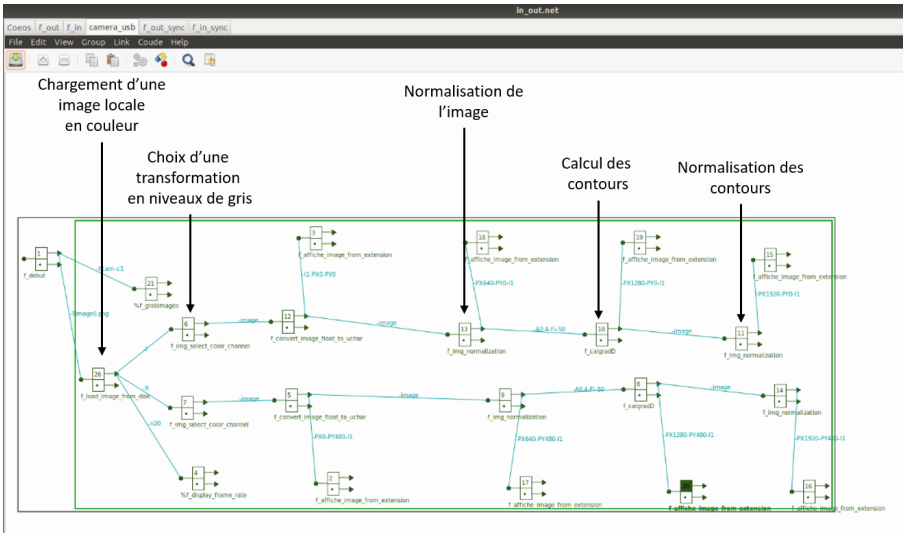

FIGURE 3.1 – Exemple d'un script permettant de tester les différents niveaux de gris, ainsi que la normalisation des images et du gradient.

— Le .config contient les paramètres de la simulation comme le nombre d'itérations par échelle de temps par exemple.
— Le .dev liste le matériel utilisé.
— Le .prt précise les liens qui existent entre les scripts lorsque l'application en possède plusieurs.

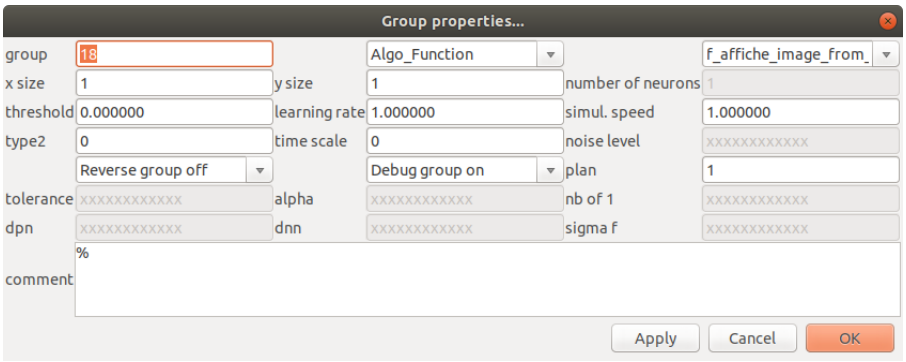

FIGURE 3.2 – Options disponibles pour les groupes

Chaque boîte Promethe est paramétrable. Tous les paramètres ne sont pas utilisés dans toutes les boîtes. La Figure 3.7 montre un exemple de paramètres qu'il est possible de trouver pour configurer une boîte :

— Le nombre de neurones contenue par le groupe (x size et y size)
— L'échelle de temps d'exécution du groupe (time scale)
— Le seuil dans le cas d'une boîte Hebb par exemple (treshold)

Les liens entre les boîtes sont eux aussi paramétrables (voir Figure 3.3), les paramètres sont inscrits dans le nom du lien dans le cas des liens algorithmiques.

Pour illustrer le fonctionnement des boîtes, nous allons nous focaliser sur celle intitulée *f_multiply*. Dans cette boîte la sortie est égale au produit des activités des neurones entrants et de leurs poids synaptiques respectifs. La figure 3.4 présente cette fonction sous Leto.

Ce simulateur comporte plusieurs logiciels permettant d'éditer des scripts qui sont ensuite exécutés par Promethe :

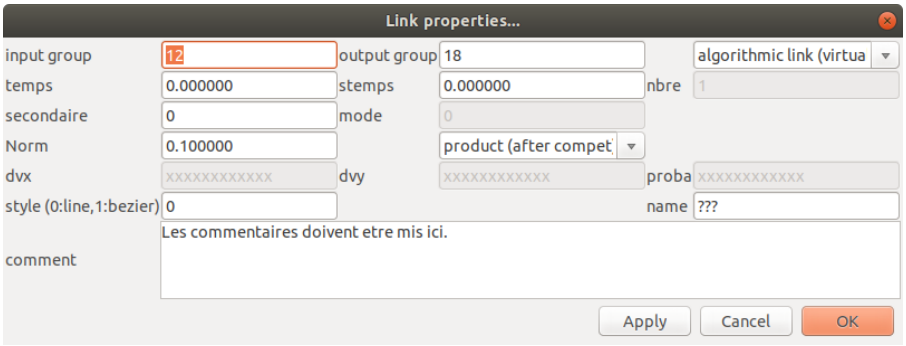

FIGURE 3.3 – Options disponibles pour les liens

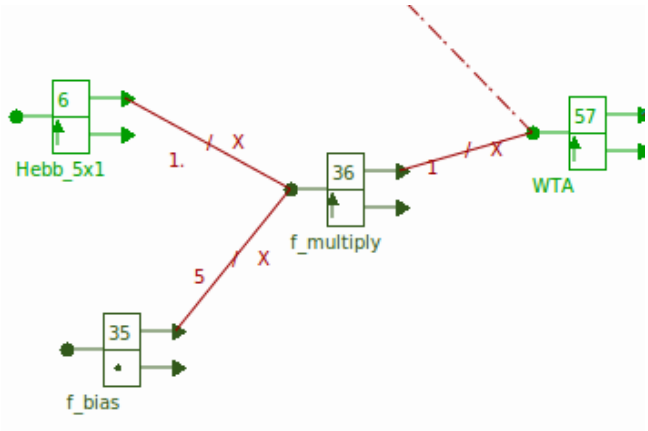

FIGURE 3.4 – Présentation de la fonction *f_multiply* dans un script.

— Promethe, interface graphique qui permet d'exécuter les scripts décrivant les réseaux de neurones. Sur la Figure 3.5 on observe le champ d'intégration de chemin ainsi que les cellules de grille. Le carré de gauche représente l'activité des neurones encodant le champ d'intégrant. Ici c'est un champ vertical. Les neurones à 1 sont blanc, ceux négatifs sont rouge. Les 3 carrés du haut sont les activités binaire des cellules de grille. Ceux du bas sont les activités des cellules de grilles avec les connexions latérales.
— Leto, interface graphique permettant d'éditer les scripts symboliques (.symb) décrivant les réseaux de neurones qui seront compilés en .scipt avec le .var contenant les variables. La Figure 3.6 montre le script utilisé pour créer des cellules de grille. Les différentes boites correspondent aux fonctions Promethe. Les liens caractérisent les échanges de données entre les boîtes. Ceux en bleu correspondent à des liens algorithmiques (ils arrivent dans des fonctions algorithmique qui ne sont pas purement biologique.) Au dessus de chaque lien sont écrit les options du lien.
— Coeos qui permet d'éditer plusieurs scripts en même temps, très utile pour des applications utilisant plusieurs scripts communicants les uns avec les autres. Les rectangles représentent les échelles de temps. La Figure 3.7 montre l'architecture des fichiers et leurs connexions sous Coeos. Sur la figure on observe les scripts en haut. Au milieu les deux machines configurées (Robulab et pc). Et enfin en bas les liens correspondant aux communications entre les scripts.
— Themis est une interface graphique qui simplifie le lancement de plusieurs scripts en même temps. En le configurant il est possible de diviser les scripts entre plusieurs machines. Les scripts communiqueront alors avec le protocole TPC/IP. La Figure 3.8 présente l'architecture de scripts utilisés sur Themis. C'est sur Themis, que l'on peut upload

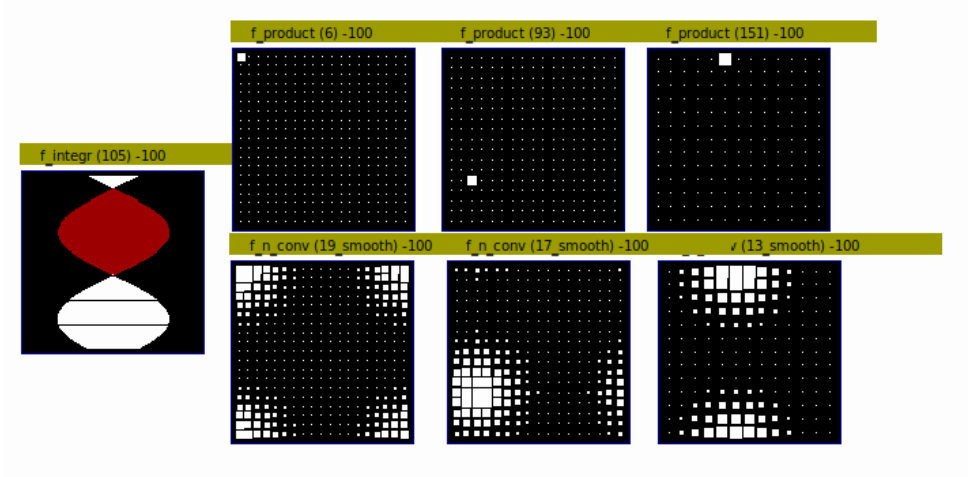

FIGURE 3.5 – Activité des neurones du champ d'intégration et des cellules de grilles sous Promethe

les scripts sur une machine distante. On peut aussi exécutés les scripts en mode debug ou en utilisant valgrind (outil de débogage de la mémoire).

— Pandora permet d'observer les activités des neurones des scripts en cours d'exécution sur la machine. Ce logiciel est bien plus ergonomique que la version graphique de Promethe, il permet aussi d'afficher les temps d'exécution de chaque boîte. Cela permet d'identifier les points de l'architecture qui prennent beaucoup de temps d'exécution. Afin de recevoir les informations de Promethe, Pandora utilise le bus Ivy. La Figure 3.9 montre l'activité des cellules de grille et du champ d'intégration de chemin sous Pandora, les boîtes observés ici sont les même que celle de la Figure 3.5.

C'est à l'aide de ce simulateur de réseau de neurones que les modèles de cellules vont être modélisés. Dans l'implémentation de notre modèle, le nombre maximun de cellules de lieu et de points d'intéret que l'architecture peut apprendre est spécifié dans les paramètres des fonctions Promethe et sont écrits dans le fichier contenant les variables. Cela permet d'uniformiser les temps de calcul.

3.2 Présentation de la plateforme mobile RobotSoc

RobotSoc est le nom donné à la plateforme mobile utilisée pour faire de la navigation au sein du laboratoire ETIS. Elle est composée d'une base principale Robulab de Robosoft. La base du robot est commandable à l'aide d'une manette. À cette base est ajouté un coffre dans lequel est installé un ordinateur et sa batterie et sur lequel sont installés des capteurs et un routeur. Veuillez trouver en Figure 3.10 une photo de RobotSoc.

Voici les différents capteurs installés :

— Une boussole Magnétique.
— Une Caméra.
— Une monture panoramique pour la caméra.
— Des lasers.
— Un joystick.

La boussole est d'abord connectée à une carte Arduino, elle-même connectée à l'ordinateur. De même le moteur de la monture Panoramique est piloté par une carte Pololu. Tous ces capteurs sont reliés à l'ordinateur, lui-même connecté à un routeur auquel on communique

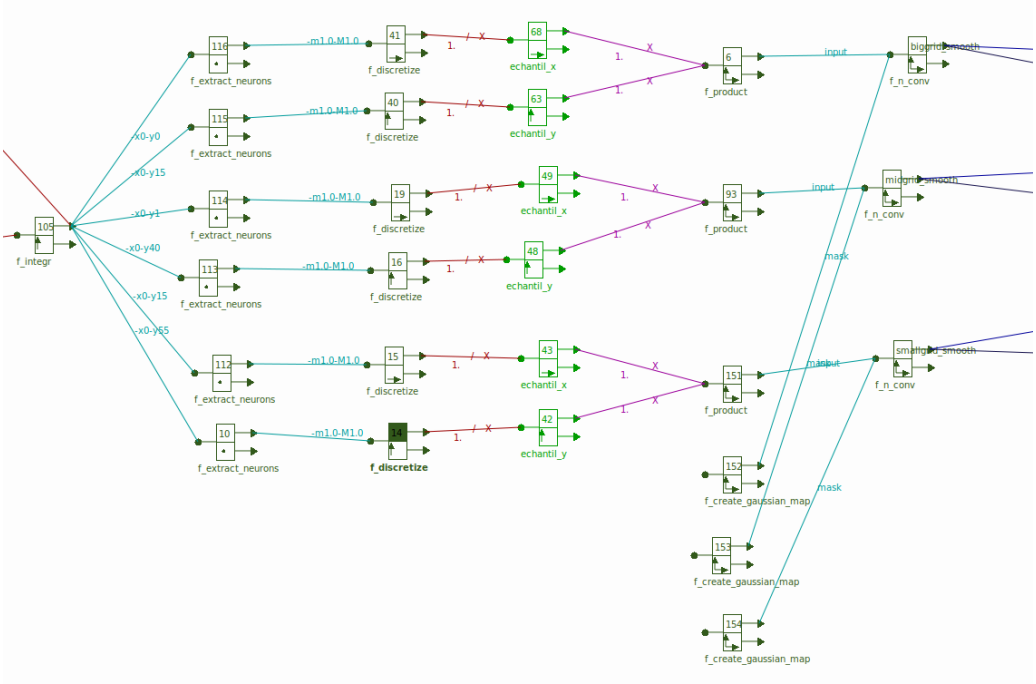

FIGURE 3.6 – Script de création des cellules de Grille.

en WIFI. On peut alors se connecter à l'ordinateur du RobotSoc depuis notre ordinateur via SSH. Une laisse est attachée au joystick. Celle-ci peut être tirée par l'opérateur du robot pour déclencher l'apprentissage de cellules de lieu et ainsi lui apprendre un certain chemin à parcourir. Veuillez trouver dans le tableau 3.1 les caractéristiques de l'ordinateur présent dans le coffre de RobotSoc qui sert à exécuter les scripts Promethe.

Caractéristiques techniques de l'ordinateur	
Processeur	i7 7700 3.6GHz
Stockage	128Go SSD
Système d'exploitation	Ubuntu 18.04 LTS
RAM	32Go
Carte mère	MSI (B2501 Gaming Pro AC)

TABLE 3.1 – Spécifications de l'ordinateur du RobotSoc

Après avoir étudié l'application sous forme théorique, l'étape suivante de mon stage était de remonter l'application de navigation Bio-inspirée sur RobotSoc, en utilisant les cellules de lieu et les cellules de grille. En effet une application similaire a déjà été faite, cependant avec les changements de matériel, et les mises à jour de Promethe elle n'était plus fonctionnelle.

En repartant des scripts déjà existants pour la navigation, il fallait vérifier le bon fonctionnent de chaque script, en vérifiant fonction par fonction que tout se déroulait comme attendu. La première partie concernait la vérification de la partie "vision" : la caméra et sa configuration. Ensuite, il était nécessaire de s'assurer du bon calcul de l'image de contour à partir de celle en niveaux de gris, puis les points d'intérêt récupérés par la différence de gaussienne. Nous avons vérifié que les points d'intérêt étaient similaires lorsque sur deux panoramas distincts sans mouvement du robot.

L'étape suivant était la calibration des lasers embarqués sur la plate-forme, afin qu'ils permettent au robot de passer une porte, tout en l'empêchant d'avancer lorsqu'il risque de toucher

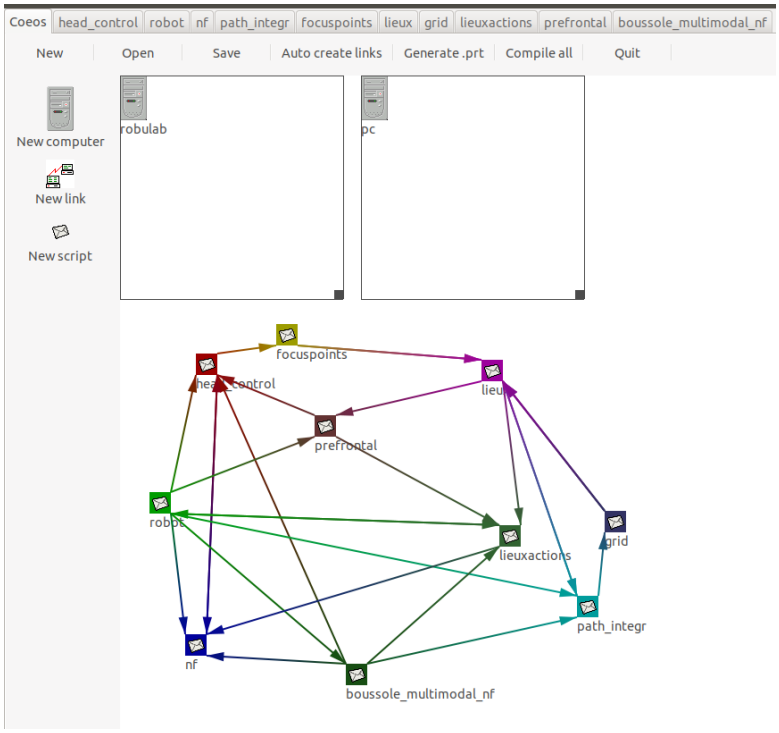

FIGURE 3.7 – Architecture de l'application sous Coeos.

un élément. Une fois les lasers calibrés, le dernier capteur à vérifier était la boussole. La première vérification concernait la cohérence des valeurs renvoyées par cette dernière afin de s'assurer qu'il n'y avait pas de perturbation électromagnétique. Enfin, il fallait s'assurer que les neurones encodaient les informations issues du laser et de la boussole dans le même sens afin de pouvoir utiliser ces deux capteurs en cohérence ensemble pour l'évitement d'obstacles. Une fois le fonctionnement de tous les capteurs vérifiés, il devenait alors possible d'évaluer la dynamique des cellules de lieu.

Paramètre	Valeur
σ_{DoG_1}	5 pixels
σ_{DoG_2}	2 pixels
r_{small}	5 pixels
r_{big}	21 pixels
r_{compet}	21 pixels
N_{PoI}	6
N_d	16
$Image\ width$	640
$Image\ height$	480
$N_{azimuth}$	361
σ_{azim}	0.5
N_l	5

TABLE 3.2 – Paramètres utilisés dans les expérimentations sur RobotSoc.

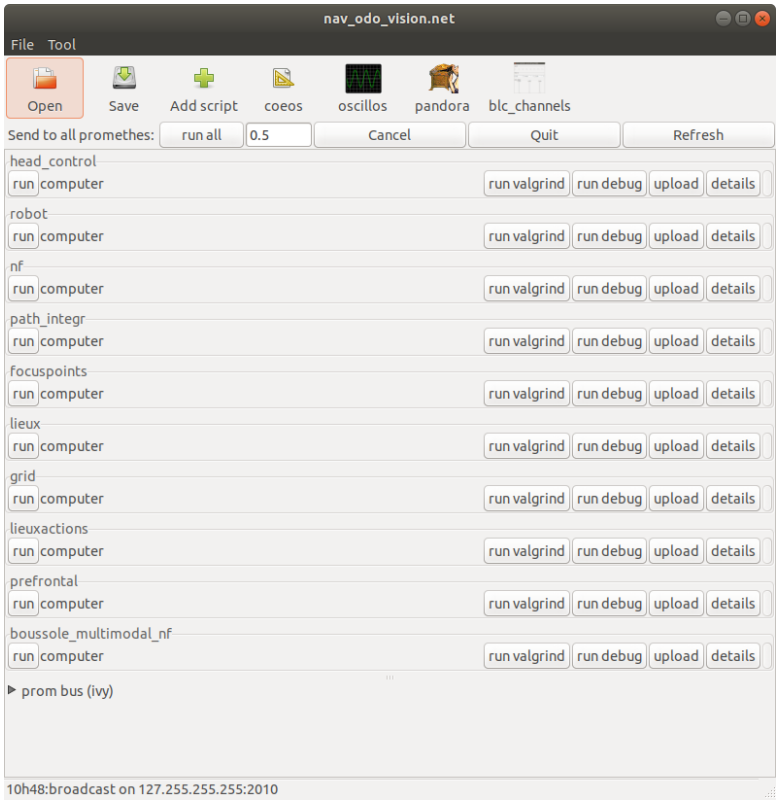

FIGURE 3.8 – Application sous Themis avec possibilité d'upload sur d'autres machines.

3.3 Expérimentation sur plateforme mobile

La plupart des tests pour prendre en main l'architecture ont pu être effectués dans la salle de TP du laboratoire ETIS, que cela soit la calibration des lasers, la vérification de la dynamique des cellules de lieu et des cellules de grille ou la recalibration de l'intégration de chemin. Pour vérifier la dynamique des différentes cellules nous avons fait 3 marques au sol, afin d'apprendre les cellules à ces endroits précis, et nous avons vérifié que lorsque nous retournions à ces endroits les activités des cellules de lieu étaient cohérentes.

Le tableau 3.2 présente la valeur des différents paramètres de notre modèle, lors des expérimentations sur RobotSoc. $N_{signature}$ et N_{lieu} sont modifiés en fonction du nombre de cellules de lieu que l'on prévoit d'apprendre. Apres avoir vérifié la dynamique des cellules de lieu, la prochaine étape a été le test de la recalibration de l'intégration de chemin. Nous avons fait faire un parcour au robot, et nous avons vérifié que le champ d'intégration était proche de zéro lorsqu'il revenait sur son point de départ. Ensuite nous avons testé la recalibration de cette intégration de chemin. Pour ce faire, nous avons réutilisé l'apprentissage des cellules de lieu avec les marques au sol en associant à chaque fois l'intégration de chemin actuel. Nous avons ensuite pu vérifier lors de la reconnaissance des cellules de lieu que l'intégration de chemin associée était bien rechargée, recalibrant ainsi la valeur courante.

La procédure pour utiliser l'architecture et reproduire une trajectoire voulue avec RobotSoc est la suivante :

— Commencer par uploader puis exécuter les différents scripts à l'aide de Themis.
— Tirer sur la laisse pour créer une cellule de lieu à la position du robot (Un signal Vigilance est envoyé).
— Relâcher la laisse lorsque le robot à effectué l'acquisition du panorama avec la caméra.

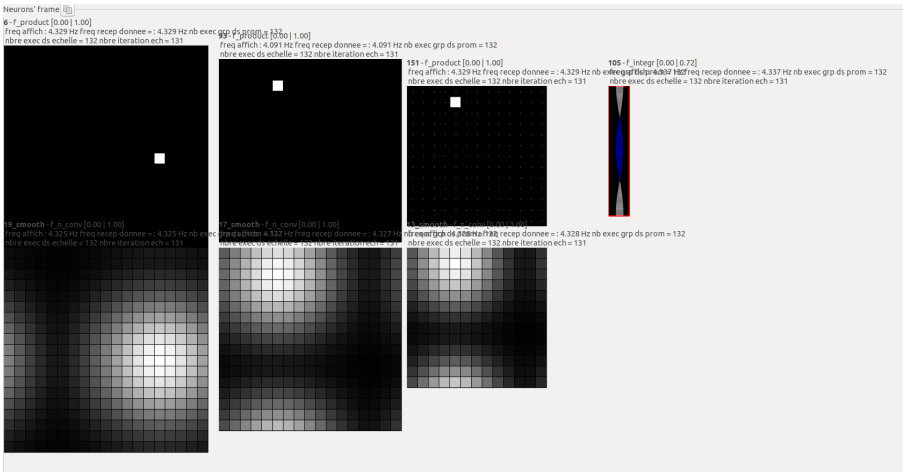

FIGURE 3.9 – Activité des neurones du champ d'intégration et des cellules de grilles sous Pandora

Le robot se mettra alors à avancer jusqu'à ce que la laisse soit de nouveau tirée.

Le robot va alors se réorienter avec l'orientation apprise vis-à-vis de la cellule de lieu qui est reconnue. Si l'utilisateur considère que le robot s'éloigne trop de la trajectoire voulue, il peut forcer l'apprentissage d'une nouvelle cellule de lieu. Au bon d'un certain nombre de cellules de lieu ajoutées, le robot devrait être capable de reproduire la trajectoire désirée sans aide extérieure de l'utilisateur [13].

Cette expérimentation se base sur l'interaction Homme-Machine [13]. En effet, il est demander à l'opérateur de faire apprendre des cellules de lieu lorsque le robot est jugé trop éloigné de la trajectoire désirée, afin de créer un bassin attracteur.

Il pouvait aussi être intéressant de voir l'architecture entière (incluant les cellules multimodales) tourné sur une trajectoire prédéfinie. Cependant, la salle de travaux pratiques du laboratoire n'est pas de taille suffisante pour discriminer correctement les différentes cellules de lieu afin de faire un cercle. Ces tests ont été effectués dans le hall de l'université, afin d'avoir un espace suffisamment grand.

3.4 Validation de l'architecture complète

Et nous avons essayé d'apprendre au robot à suivre une trajectoire circulaire autour d'un poteau. Cependant les cellules de lieu avaient une mauvaise dynamique lorsque le robot se trouvait proche du poteau. La concurrence était trop forte entre les cellules de part et d'autre du poteau.

C'est pourquoi nous avons essayé de lui faire faire un tour autour de deux poteaux, afin de limiter les cellules de lieu et les cellules de grille trop similaires. Lors de ce test, nous avons observé une dynamique correcte pour les cellules de lieu, avec moins de fausses reconnaissances, ainsi que pour les cellules de grille. Cependant l'intégration de chemin ne se recalibrait pas car l'activité de la cellule de lieu gagnante n'était pas suffisante pour le permettre. Cela peut être dû à une différence de luminosité entre le moment de l'apprentissage de la cellule et le moment de la reconnaissance de cette dernière. Un autre point qui posait problème est la sensibilité des lasers. En effet ceux-ci avaient tendance à trop limiter les déplacements du robot. Le robot était arrêté à un emplacement et une direction où il pouvait continuer à avancer. Il faudrait donc reprendre la calibration des lasers afin de limiter leur inhibition dans le champ d'action.

FIGURE 3.10 – Photo de RobotSoc.

Ces tests avaient pour but de faire tourner l'architecture complète et m'ont fait prendre conscience de l'importance des différents paramètres, qu'il s'agisse de la vigilance générale, de la calibration des lasers, ou encore des différents seuils de recalibration. Cela m'a aussi donné des idées d'amélioration quant à la direction finale envoyée en consigne. En effet lors de ces tests, une seule cellule multimodale gagnante était gardée pour donner la direction sur le champ de neurones dynamique. Cependant il est possible de garder plusieurs cellules gagnantes. La consigne de direction est alors une moyenne des directions apprises pour chaque cellule gagnante. Cette moyenne est pondérée par la valeur d'activité de chaque cellule. Cela permet d'éviter les changements potentiellement bruts de consigne de direction lors du changement de la cellule de lieu gagnante. Cela m'a aussi interrogé sur la normalisation de l'activité des différentes cellules à faire. Faut-il normaliser toutes les cellules, celles de lieu, celles de grille, les multimodales ? La normalisation des cellules est importante dans le cas on l'on choisit plusieurs gagnants, car elle ne change le gagnant. Il semble intéressait de normaliser les cellules de lieu afin d'augmenter. Pour les cellules de lieu prédites par les cellules de grille la normalisation ne semble pas intéressante car il y a peu de concurrence entre celle ci, cela pourrait potentiellement apporter plus d'erreur. Pour les cellules multimodale, si l'on normalise celle de lieu, il ne semble pas nécessaire de les normaliser. À la suite de ces différents tests, j'ai commencé à travailler sur la portage de cette architecture sur la base véhicule de VEDECOM.

Chapitre 4

Adaptation de l'architecture sur véhicule

4.1 Présentation du Véhicule

FIGURE 4.1 – Photos de la voiture ZOE VEDECOM.

Le véhicule sur lequel j'adapte l'architecture sensorimotrice est une Renault Zoe ,voir Figure 4.1, robotisée afin de pouvoir piloter les actionneurs (accélération, freinage et angle volant). De nombreux capteurs ont été ajoutés. Entre autres, des LIDAR aux quatre coins de la voiture afin de détecter les obstacles, un odomètre, une centrale inertielle, un GNSS avec correction RTK et des caméras. Les informations de l'odomètre et du GNSS sont fusionnées dans la centrale inertielle. Le coffre de la ZOE est visible en Figure 4.2a, les différentes unités de traitement sont installées. Les caméras sont positionnées à l'avant du véhicule comme le montre la Figure 4.2b.

Le moteur du véhicule est commandé en couple, la direction du véhicule est commandée via un angle sur le volant, celui-ci donnant ensuite une direction aux roues.

L'ordinateur de la voiture possède le système d'exploitation Windows 7 Embedded en 32 bits. Promethe ne fonctionnant que sous Linux et plus particulièrement sur la distribution Ubuntu 18.04 LTS, il était nécessaire d'embarquer un ordinateur sur lequel seront exécutés les scripts Promethe.

4.2 Logiciel RTMaps et connexion à Promethe

Les informations du véhicule sont récupérées et traitées par le logiciel RTMaps. C'est sur ce logiciel, que l'on récupère les informations du bus CAN véhicule, des caméras et de la centrale

(a) Photo du coffre de la ZOE.

(b) Photo des caméras de la ZOE.

inertielle à l'aide de sockets. Les commandes moteur et volant sont envoyées par celui-ci.

RTMaps est un environnement de développement logiciel et d'exécution orienté composant. Sur ce logiciel il est possible d'enregistrer des données et de les rejouer en dehors du véhicule. Sur RTMaps un diagramme est une suite de briques avec des flux de données s'échangeant entre les briques. Voici en Figure 4.3 le diagramme RTMaps utilisé pour tester la communication entre RTMaps et l'interface Python que nous avons développé afin de connecter Promethe et RTMaps. Il est possible de créer ses propres composants en C++ ou en Python.

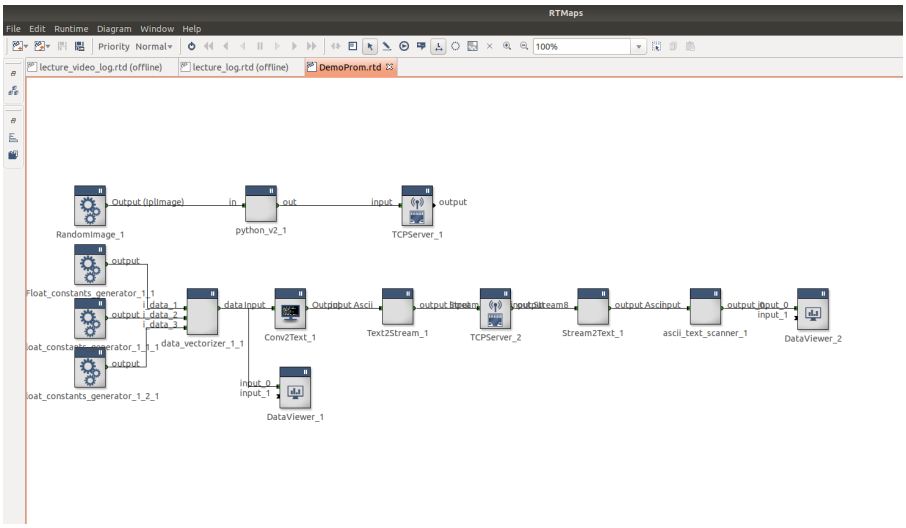

FIGURE 4.3 – Diagramme RTMaps de test de la communication avec l'interface Python.

Nous avons besoin dans notre architecture de récupérer les informations de lacet du véhicule, d'odométrie et les images prises par les caméras de la voiture. Dans la voiture, la caméra est installée dans l'axe du véhicule. Ces informations sont récupérées par un diagramme RTMaps qui est exécuté sur l'ordinateur de la voiture. Il est nécessaire de faire communiquer RTMaps avec la machine embarquée sur lequel seront exécutés les scripts Promethe. Pour ce faire nous avons développé une interface Python qui se connecte avec le protocole TCP aux sockets créées par RTMaps. Puis cette interface Python écrit les informations reçues dans une mémoire partagée de l'ordinateur embarqué. Ensuite, Promethe vient lire les valeurs présentes dans celle-ci. Cette solution utilise le Framework BLAAR développé par Blanchard Arnaud au laboratoire ETIS qui créé des *Blc_channels* qui permettent de lire et écrire facilement dans la mémoire partagée. Il était aussi possible de créer des sockets directement sur Promethe en interprétant

RTMaps comme étant un périphérique de Promethe, cependant cela avait le désavantage d'être plus compliqué à implémenter, et moins robuste à tout changement. En effet, si un jour le véhicule intègre ROS à la place de RTMaps par exemple, il sera alors possible il sera possible de mettre un nœud Ros sur l'ordinateur où sont exécutés les scripts Promethe. Ensuite, il serait possible de récupérer les images et les informations des capteurs via des *rostopic* ou *rosmsgs* puis de les écrire dans la mémoire partagée à l'aide de BLAAR. Voici en Figure 4.4 un schéma globale de l'architecture entre les machines.

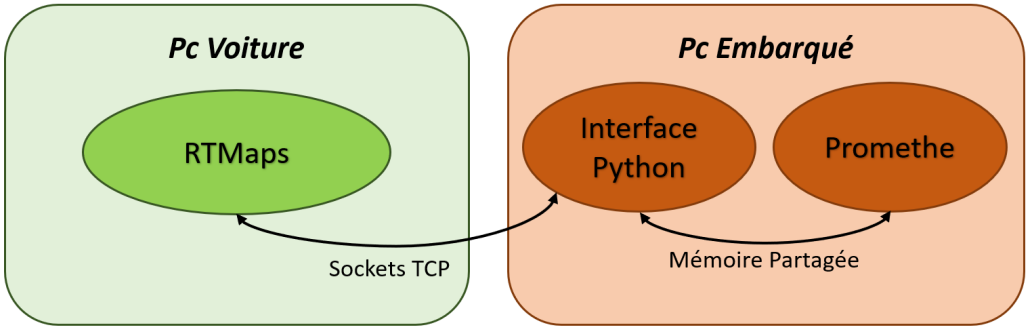

FIGURE 4.4 – Schéma communication entre les machines

L'architecture sous Promethe a un temps d'exécution assez long pour chaque image. Ce dernier dépend directement du nombre de cellules de lieu et de points d'intérêt prévus. Avec 200 lieux et 6000 points d'intérêt, Promethe peut traiter 2 à 3 images par seconde avec notre configuration. La caméra du véhicule ainsi que les informations issues de la centrale inertielle ont une fréquence de traitement bien plus grande. La caméra du véhicule est capable d'envoyer 10 images par seconde. La centrale inertielle est cadencée à 25Hz. Dans les diagrammes RTMaps utilisés pour par les équipes de VEDECOM, les commandes sont envoyées à une fréquence de 25Hz. Afin de limiter l'envoi de données non traitées par Promethe nous avons choisi d'envoyer une image par le socket depuis RTmaps uniquement lorsque Promethe a fini de traiter la précédente. Pour ce faire nous utilisons la commande envoyée depuis Promethe à l'issue du traitement de l'image comme déclencheur de l'envoi d'une nouvelle image et du lacet du véhicule côté RTMaps. En effet lorsqu'une commande est envoyée cela signifie que le script Promethe a terminé tous les traitements sur l'image. Voici en Figure 4.5, un diagramme RTMaps permettant de rejouer un log enregistré. On y retrouve les informations de la centrale inertielle décodées par le bloc ixsea, du Bus CAN du véhicule, et des images de la caméra. Il y a aussi les sockets de communication avec l'interface Python. Lorsque le composant *TCP_server_1* reçoit une commande, celle-ci est envoyée dans les composants *data_vectorizer_2* et *Vectorizer_1* qui envoie respectivement l'image et un vecteur contenant l'orientation du véhicule, l'odométrie, et l'angle des roues).

Une fois cette communication mise en place, il est désormais possible de tester la localisation de partir à cellules de lieu apprise depuis un log préalable.

4.3 Cellules de lieu à l'échelle du véhicule

Afin de vérifier le fonctionnement des cellules de lieu, nous avons enregistré des logs sur la piste d'essais et sur la zone d'évolution. Ceux-ci permettront de tester le fonctionnement des cellules de lieu, l'intégration de chemin et les cellules de grilles sur table. En effet en rejouant les logs créés avec RTMaps avec un premier ordinateur et en exécutant les scripts Promethe et

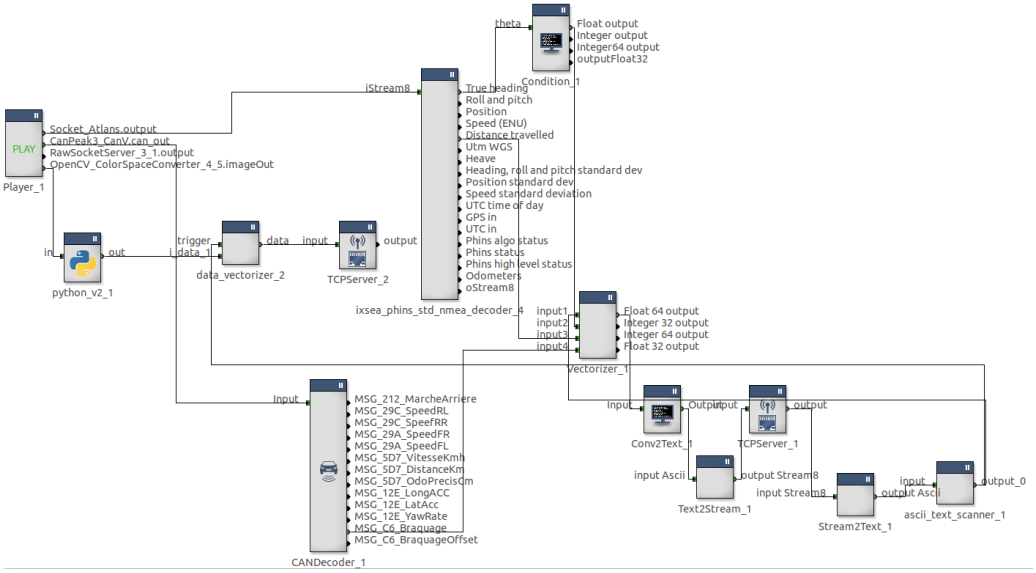

FIGURE 4.5 – Digramme RTMaps rejouant un log et l'envoyant a l'interface Python.

l'interface Python sur un second, nous nous retrouvons dans la même configuration que sur la voiture. La figure 4.6 présente la trajectoire d'un des logs enregistrés où le véhicule fait le tour de la zone d'évolution et la Figure 4.7 la trajectoire d'un log enregistré sur les pistes d'essais comportant deux virages.

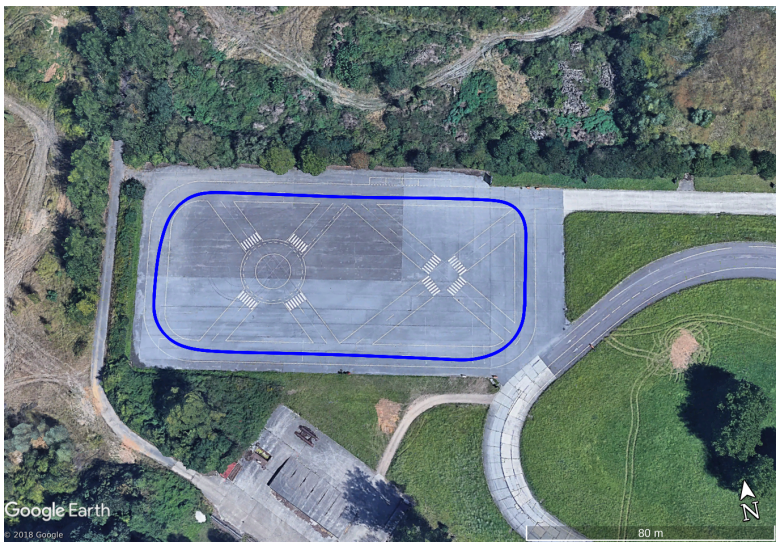

FIGURE 4.6 – Log enregistré sur la zone d'évolution.

La première étape de l'intégration de l'architecture sur véhicule est de vérifier que les cellules de lieu sont reconnues de façon suffisante dans un environnement où les points d'intérêt pertinents peuvent être limité en nombre et fortement concurrents entre les différentes étapes du parcours. Il faut alors chercher les meilleurs paramètres pour avoir le minimum d'erreurs de reconnaissance.

Pour le test des cellules de lieu nous avons commencé à nous intéresser aux travaux faits en localisation sur la base de données KITTI [9]. Il a été montré qu'avec un même seuil de vigilance S_{Vigi} le champ de lieu des cellules de lieu était plus ou moins grand en fonction de l'environnement d'évolution du véhicule. Il a ainsi été observé des champs de lieu plus grands

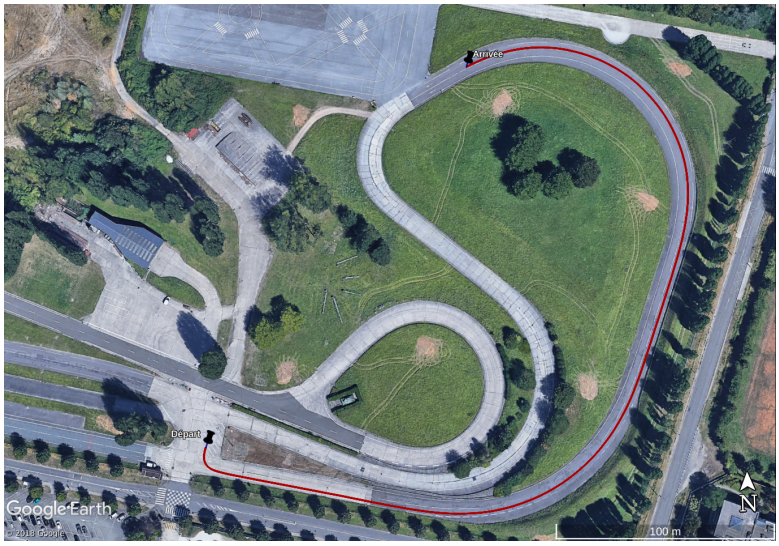

FIGURE 4.7 – Log enregistré sur les pistes d'essais avec deux virages.

sur autoroute où il y a peu de détails à récupérer sur les images, par rapport à un environnement urbain où les champs de lieu sont alors de plus petite taille.

Par rapport à ce qui a été fait, nos tests sont différents car nous allons récupérer les images et les orientations du véhicule en temps réel. De fait en faisant du contrôle nous avons besoin de temps réel. Notre modèle sera alors très dépendant du temps de traitement par image. C'est pourquoi les logs que nous avons enregistrés sont à relativement basse vitesse (15-20km/h).

Afin de tester la reconnaissance des cellules de lieu, on commence par rejouer un log, en l'occurrence le log avec les deux virages, qui fait un total de 494 mètres, présenté en Figure 4.7 sur lequel on apprend des cellules de lieu en fonction d'un seuil de vigilance. Ce seuil conditionnant le recrutement d'une nouvelle cellule de lieu lorsque aucune cellule de lieu n'est au dessus de ce seuil. Voici en Figure 4.8 l'activité des cellules de lieu lors de l'apprentissage, un trait pointillé est mis au niveau de seuil de vigilence. Dans cette figure, comme dans les suivantes, chaque couleur représente l'activité d'une cellule de lieu.Puis dans un second temps on rejoue le log en utilisant les cellules de lieu apprises lors de la première étape. Dans cette seconde étape le mécanisme d'apprentissage est désactivé. Les figures 4.9 et 4.10 affichent l'activité des cellules de lieu pour un seuil de vigilance respectif de 0.42 et 0.75 (ce seuil étant compris entre 0 et 1). On peut ainsi voir qu'il y a plus de cellules de lieu recrutées avec un seuil de vigilance plus élevé. Car l'activité de la cellule gagnante descend d'abord en dessous du seuil le plus élevé. 27 cellules de lieu ont été recrutées avec le seuil de 0.42 et 107 avec celui de 0.75. L'activité de certaines cellules est à 1, cela signifie que lorsque le log est rejoué, la même image qui a été utilisée pour l'apprentissage est traitée par Promethe. Pour générer ces cellules de lieu nous avons utilisé les paramètres donnés dans le tableau 4.1. Aussi dans le véhicule, le tiers inférieur de l'image est dégradé par les reflets du pare-brise. De ce fait nous choisissons de ne pas prendre de points d'intérêt dans cette partie de l'image.

L'un des inconvénients de ces tests est leurs non répétabilité. En effet, les images sont récupérées lorsque le script à terminé le traitement de celle d'avant. Donc d'un test à l'autre les images récuperées sur le log peuvent être différentes, et donc les cellules de lieu créées peuvent être différentes. A partir d'un même log, on peut obtenir des résultats différents.

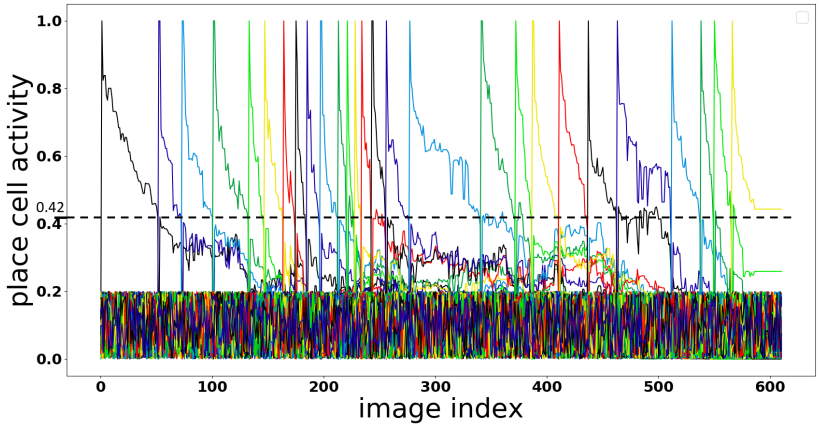

FIGURE 4.8 – Activité des cellules de lieu lors de l'apprentissage avec un seuil de vigilance de 0.42.

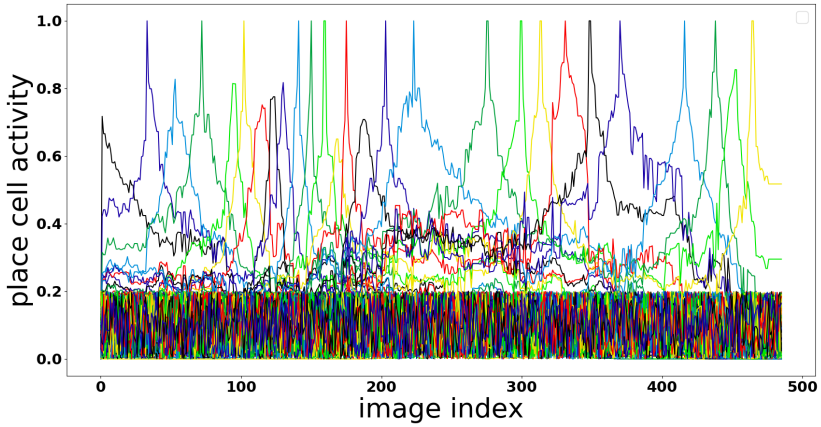

FIGURE 4.9 – Activité des cellules de lieu apprises avec un seuil de vigilence de 0.42.

4.4 Apprentissage de l'orientation du véhicule

Le script Promethe que nous avons créé, envoie une vitesse de lacet comme commande. La brique de commande de la Renault ZOE prend en entrée un angle volant. Nous avons donc développé une brique RTmaps afin de convertir la valeur de vitesse de lacet en angle volant désirée. La vitesse est récupérée à partir du Bus CAN du véhicule.

En partant du modèle du tricycle en Figure 4.11 on obtient les équations d'état 4.1

$$\begin{cases} \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ \dot{\theta} = \frac{v}{D}\tan(\delta) \\ \dot{v} = u_1 \\ \dot{\delta} = u_2 \end{cases} \tag{4.1}$$

Avec D l'empattement de la voiture, δ désigne l'angle des roues, u_1 est la commande sur la vitesse linéaire et u_2 la commande sur l'angle volant. Alors on a l'équation 4.2 pour l'angle volant.

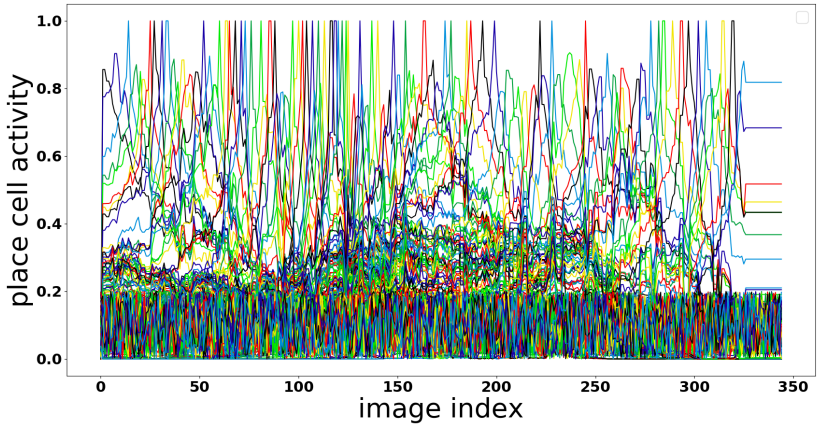

FIGURE 4.10 – Activité des cellules de lieu apprises avec un seuil de vigilence de 0.75.

Parameter	Value
σ_{DoG_1}	5 pixels
σ_{DoG_2}	2 pixels
r_{small}	10 pixels
r_{big}	61 pixels
r_{compet}	30 pixels
N_{PoI}	15
N_d	16
$Image\ width$	640
$Image\ height$	480
$N_{azimuth}$	361
σ_{azim}	0.5
N_l	5

TABLE 4.1 – Paramètres utilisés pour les cellules de lieu apprise lors des expérimentation sur la voiture.

$$\delta_{vol} = D_{emul} \times \phi \tag{4.2}$$

D_{emul} étant le coefficient de démultiplication entre l'angle du volant et l'angle des roues. Ce calcul est effectué par une brique RTMaps paramétrable pour s'adapter à toutes les voitures.

En utilisant un log d'un parcours identique à celui que l'architecture serait potentiellement capable de reproduire au niveau de la vitesse, nous avons étudié la vitesse de lacet que le véhicule effectué lors des différents virages. Nous avons aussi fait des virages avec un angle volant maximum pour se rendre compte des valeurs maximal de vitesse de lacet. Cela a permis d'accorder les valeurs envoyées par Promethe en matière de vitesse de lacet avec celle du véhicule lorsqu'il est piloté par une personne.

Il a par exemple été nécessaire inverser les valeurs négatives et positives, car la voiture tourne vers la gauche lorsqu'elle reçoit une valeur positive de vitesse de lacet alors que le RobotSoc tourne vers la droite.

Avant de pouvoir faire des tests sur le véhicule, nous avons testé la cohérence des commandes envoyées par Promethe puis transformées par la brique RTMaps. Pour ce faire, nous avons utilisé

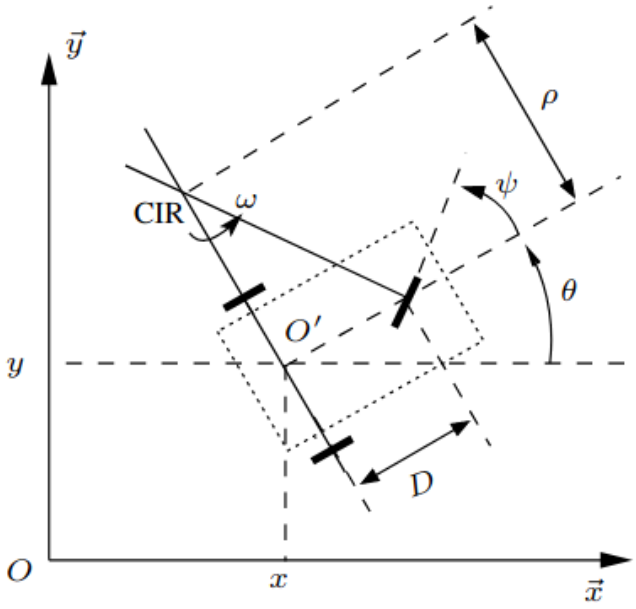

FIGURE 4.11 – Modèle du tricycle, équivalent au modèle de la voiture.

un log et vérifié, en ajoutant un valeur de biais sur le lacet de la voiture, la cohérence des valeurs envoyées.

Un des points importants de la navigation de la plateforme robotisée Robotsoc consiste en l'interaction homme-machine afin de créer un champ attracteur pour le robot. De la même façon, il est nécessaire d'ajouter cette interaction sur le véhicule, afin de permettre au conducteur de forcer l'apprentissage d'une nouvelle cellule de lieu lorsque cela semble nécessaire. Tout d'abord lors de nos tests, il est possible de reproduire plusieurs fois la trajectoire désirée, l'orientation du véhicule sera différente dans chacune des trajectoires car elle ne sera pas parfaitement reproduite par le pilote. Cela permet de recruter autant de cellules de lieu que nécessaire. Dans les véhicules utilisés par VEDECOM, le mode autonome se désactive automatique à la moindre action du conducteur sur les pédales ou sur le volant. Au moment de la reprise du contrôle, à partir de RTMaps nous pouvons envoyer un signal à Promethe lui indiquant qu'il faut apprendre une cellule de lieu. Cependant lorsque le véhicule se situe dans une situation nécessitant un apprentissage de cellule de lieu il va mettre un certain temps avant d'avoir une orientation satisfaisante pour retourner dans le bassin attracteur. C'est pourquoi à cette cellule de lieu nous associons une orientation future (de quelques mètres en aval) afin d'anticiper le retard de direction.

4.5 Cellules de grille

Ensuite nous avons testé l'intégration de chemin afin d'adapter les cellules de grille au véhicule. Pour ce faire, nous avons utilisé le log faisant le tour de la zone d'évolution car celui-ci boucle. Nous avons vérifié que le champ d'intégration diminuait et avait des valeurs proches de 0 lorsque le tour de la zone s'achevait. Le champ d'intégration ayant une valeur maximal pour la direction globale du mouvement, valeur proportionnelle à la distance pour retourner à l'emplacement initiale. Avoir des valeurs proche de 0 signifie que l'on est très proche de l'emplacement de départ. Ce qui est cohérent. Malgré le faible nombre de données odométriques

utilisées (autant que le nombre d'images traitées) le champ d'intégration est fonctionnel.

Ensuite, nous avons testé l'intégration des cellules de grille à la suite de ce champ d'intégration. Lors de l'écriture de ce mémoire, la dynamique des cellules de grille créées lors du tour de la zone d'évolution est mauvaise, et ne permet pas de discriminer l'endroit ou l'on se trouve dans la zone. Cela fait partie des futurs travaux à effectuer lors de ce projet. Les cellules de grille possèdent plusieurs paramètres qu'il faudra adapter correctement, tel que le la distance maximal encodable, la discretisation et le modulo.

4.6 Premiers résultat du contrôle sur pistes d'essais

Plusieurs étapes de validation de l'architecture sensorimotrice peuvent être effectuées sur les pistes. Une de ces étapes était de vérifier que la dynamique de contrôle venant de l'architecture était correcte sur la voiture. Ensuite l'un des premiers objectifs était de réussir à faire plusieurs fois le tour de la zone d'évolution en utilisant uniquement l'architecture sensorimotrice. Les premiers tests effectués avaient pour but de valider la communication entre les deux ordinateurs, celui de la voiture et celui embarqué.

Notre architecture ne gère pas la vitesse linéaire du véhicule. Sur la plateforme RobotSoc celle-ci est fixée à une certaine valeur. Il est possible de choisir comme pour RobotSoc une vitesse consigne. Pour simplifier dans la voiture nous utilisons le mode coopératif, qui permet de commander l'angle volant en laissant le contrôle de la vitesse linéaire (accélérateur et frein) au conducteur. Par la suite il est envisageable d'associer une vitesse à chaque cellule de lieu ce qui pourrait permettre de ralentir avant les virages et d'accélérer lors des lignes droites.

Nous avons ensuite testé l'apprentissage d'une trajectoire sur un virage dans la zone d'évolution. Voici en Figure 4.12 les trajectoires effectuées lors de l'apprentissage en bleu et lors de reproduction par l'architecture en rouge. Lors de ce test nous avons fixé le seuil de vigilance à 0.70. Dans ce premier test nous n'avons pas ajouté l'interaction homme machine, car nous ne faisions pas une trajectoire bouclée. Ce premier test a donné des résultats très encourageants vis-à-vis des commandes envoyées, qui étaient cohérentes.

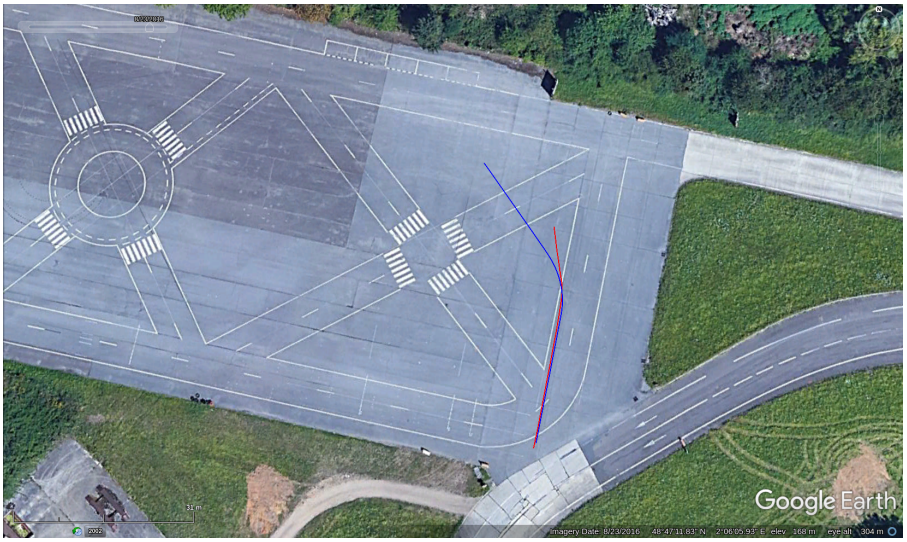

FIGURE 4.12 – Trajectoire lors du test du contrôle sur la voiture sur la zone d'évolution. En bleu, la trajectoire faite durant l'apprentissage, en rouge celle reproduite.

Lors de ce premier test le véhicule ne tournait pas suffisamment par rapport à la trajectoire apprise. Plusieurs paramètres entrent en compte dans ce retard. Une possibilité peut être lié

au fait que l'on apprenne le lacet actuel du véhicule sans tenir compte de l'orientation du volant, l'orientation des roues ou bien de la vitesse de lacet. Cela signifie que dans un début de virage, la voiture commence à avoir les roues tournées mais reste dans la direction de l'itération précédente. A l'itération suivante la voiture commence à tourner alors que le modèle pense être dans la bonne direction.

Une des possibilités permettant de résoudre ce problème est d'apprendre non pas l'orientation courante du véhicule mais celle dans le futur, ce futur peut être un temps en secondes ou un nombre d'images traitées (implémentions facile sous Promethe). Cependant, avec ces deux valeurs il est possible que le véhicule soit à vitesse très faible voire nulle. Une autre information qui semble plus pertinente est le déplacement depuis l'apprentissage de la cellule. L'idée est donc lors de l'apprentissage d'une cellule de lieu d'associer non pas l'orientation actuelle, mais le lacet que le véhicule aura dans quelques mètres. De plus, cet apprentissage dans le futur de l'orientation rend efficace le mécanisme d'apprentissage de cellule de lieu lorsque le conducteur reprend la main. En effet, à ce moment-là, le véhicule retourne dans une direction convenable avant que l'architecture apprenne son orientation.

4.7 Futurs tests

Maintenant, l'un des objectifs majeurs est de faire le tour de la zone d'évolution de façon autonome. Il était intéressant dans un premier temps, de valider la reconnaissance et le contrôle du véhicule sur un virage en implémentant uniquement les cellules de lieu. Cependant il semble difficile de faire un tour complet uniquement avec la vision sachant qu'autour de la zone d'évolution il y a relativement peu de bâtiment, ou autres points distinct. Néanmoins, en ajoutant l'intégration de chemin et les cellules de grilles le tour de la zone d'évolution parait parfaitement faisable car les cellules de grilles départageront les cellules de lieu lorsque celles-ci sont trop concurrentes.

Un point d'amélioration de notre implémentation sur véhicule est le nombre de caméras utilisées. Dans la voiture Zoé, il y a deux caméras installées. Jusqu'à présent nous ne servons que d'une seule sur les deux. Cependant au vue de la faible valeur d'angle d'ouverture de ces caméras (50,3°horizontalement et 30,6°verticalement) il peut être intéressant d'utiliser les deux à la façon d'un panorama afin d'augmenter l'angle de prise de vue. Cela fait partie des améliorations à intégrer dans les semaines à venir.

Ensuite il est alors possible d'essayer de faire un tour des pistes. L'un des challenges à surmonter pour réussir à faire le tour complet des pistes est de diminuer le coût computationnel. En effet si l'on veut continuer à avoir des champs de lieu suffisamment petits, le coût computationnel augmente considérablement. Il est possible de changer certains paramètres afin de réduire ce coût computationnel, comme par exemple le nombre de points d'intérêt appris à chaque image. Diminuer la vitesse traitement des images signifie diminuer la vitesse du véhicule, afin de récupérer des images avec écart correct en terme de distance.

Dans un premier temps, l'architecture sera testée avec le véhicule à vitesse limitée afin d'avoir suffisamment d'images par mètre. Dans le but d'avoir une vitesse plus élevée il faut augmenter la vitesse de traitement des images par Promethe. L'ordinateur que nous utilisons lors des expérimentations est un ordinateur bureautique et n'est donc pas forcement très puissant. Il serait donc possible d'utiliser un ordinateur avec un meilleur processeur. Pour estimer l'amélioration, nous avons exécuté l'application de localisation sur table, sur une base de données de 617 images d'abord l'ordinateur bureautique actuellement utilisé intégrant un processeur Intel i5 6200U 2Cores 4Threads @2.3GHz 2.8Ghz en Turbo, avec 8Go de RAM pusis sur un ordinateur plus performant possédant un processeur Intel i7-6950X 10 Coeurs 20 threads @3Ghz

3.5Ghz Turbo, avec 128Go de Ram voir Tableau 4.2. Dans ces tests on récupére un maximun de 15 points d'intérêt par image. On observe une amélioration par rapport à l'ordinateur actuellement utilisé, ce qui pourrait être intéressant dans le cadre de notre application. Cependant, l'augmentation reste anecdotique considérant la fréquence de traitement que nous avons lorsque nous avons beaucoup de cellules de lieu. Il faut bien ce rendre compte que lorsque l'application a une fréquence d'images de 2 images par seconde, cela signifie à 30km/h on traite une image tous les 4 mètres, ce qui peut être considérable dans les virages.

Ordinateur / Valeur	ordinateur bureautique	ordinateur 2
Temps d'exécution 200 lieu et 6000 PoI	287.66 s	199.58 s
Images par secondes	2.14 Ips	3.09 Ips
Temps d'exécution 100 lieu et 1500 PoI	103.0 s	67.9 s
Images par secondes	6.0 Ips	9.1 Ips

TABLE 4.2 – Comparaison performances ordinateurs

A l'aide du logiciel Pandora et de l'oscilloscope (une fonctionnalité de Promethe qui permet de voir les temps d'exécution pris par chaque boîte/fonction), on se rend compte que ce sont les calculs dans la SWM (mémoire de travail) qui prennent le plus de temps. Il n'est pas évident de réduire sa complexité, car c'est ici que l'on retrouve toutes les informations du lieu. Des travaux ont été menés sur la contextualisation du lieu [4] dans le but de caractériser le lieu globalement. L'idée de contexte est d'avoir un descripteur global par image qui permet de reconnaître globalement un type d'environnement. Actuellement ces travaux améliorent la localisation, mais augmentent le coût de calcul. Dans notre architecture nous nous intéressons aux vues locales des points d'intérêt, nous n'avons pas de mécanisme globale sur l'image. Il pourrait par exemple être possible de diviser la mémoire de travail en fonction du contexte des images que l'on perçoit. Par exemple, entre un environnement urbain et péri urbain, l'architecture pourrait faire réagir une seule partie spécifique de la mémoire de travail, celle qui a apprise pour l'environnement présent.

Un point a modifier lorsque l'on teste l'architecture dans des environnements de plus grande taille est la façon dont on calcule la commande à envoyer. En effet, actuellement une commande est envoyée après le traitement de chaque image. C'est une fréquence qui peut être assez faible lorsque l'on a beaucoup de cellules de lieu. La commande ne nécessite pas un long temps de calcul, il est donc possible de séparer l'architecture en deux scripts localisation et commande, afin d'envoyer des commandes plus régulièrement par rapport à l'orientation du véhicule.En effet, le script commande récupère les activités des cellules de lieu calculées par le script lieu ainsi que l'orientation du véhicule. Ainsi pour un même image, plusieurs commandes peuvent être envoyées, ce qui pourrait permettre une meilleure régulation de la direction.

Conclusion

Ce stage m'a permis d'intégrer deux institutions différentes, au fonctionnement et aux objectifs distincts. J'ai ainsi intégré l'équipe neurocybernétique du laboratoire ETIS et une équipe de développement de nouvelles technologies pour la voiture à conduite déléguée chez VEDECOM. Nous avons commencé par décrire de façon explicite l'architecture sensorimotrice qui est utilisée en présentant tous les variétés de cellules qui sont utilisées. De nombreux mécanismes neuronaux sont mis en place.

De nombreux tests préliminaires ont été nécessaires afin de remonter l'application de navigation sur le robot. Lors de ces tests j'ai pris conscience de la difficulté de développer un logiciel de façon collaborative, tout en le rendant ergonomique pour les futurs utilisateurs. En effet, Promethe est un logiciel très complet et très complexe, mais qui n'est pas facile à appréhender.

Nous avons ensuite exposé les résultats de localisation à l'aide de cellules de lieu. Nous avons montré l'influence du seuil de vigilance pour avoir des champs de lieu de tailles différentes donc une plus ou moins grande généralisation des cellules de lieu. Nous avons aussi exposé l'interface développée dans le cadre de la connexion entre RTMaps et Promethe. Celle-ci est amenée à être modifiée en fonction des données que l'on souhaite échanger.

Après avoir vérifié la cohérence des commandes calculée par l'architecture sensorimotrice, nous avons pu effectuer nos premiers tests sur un virage. La suite des tests vont être effectués dans la suite de mon stage. Il serait très intéressant d'arriver à faire une trajectoire bouclée de relativement petite taille, puis d'essayer de faire des parcours plus longs.

Des axes d'améliorations existent sur l'architecture afin de l'adapter aux grands environnements. Il faut trouver un compromis entre précision de la localisation et vitesse d'exécution de l'architecture. En effet, avec des environnements plus grands, il faut plus de cellules de lieu donc l'application a un temps d'exécution considérablement plus long par image.

Pour finir, l'état actuel de l'architecture vis-à-vis de la conduite délégué est prometteur. En ajustant quelques paramètres il semble possible de faire de la conduite de véhicule autonome, avec une faible vitesse et sur de petites distance dans la suite de mon stage.

Bibliographie

[1] Araujo, H., and Dias, J. An introduction to the log-polar mapping. *Proc. of Second Workshop on Cybernetic Vision* (01 1996).

[2] Arleo, A., and Gerstner, W. Modeling rodent head-direction cells and place cells for spatial learning in bio-mimetic robotics.

[3] Arleo, A., and Gerstner, W. Spatial cognition and neuro-mimetic navigation : A model of hippocampal place cell activity. *Biological Cybernetics 83* (06 2000).

[4] Belkaid, M., Cuperlier, N., and Gaussier, P. Combining local and global visual information in context-based neurorobotic navigation. pp. 4947–4954.

[5] Brun Kjelstrup, K., Solstad, T., Heimly Brun, V., Hafting, T., Leutgeb, S., P Witter, M., Moser, E., and Moser, M.-B. Finite scale of spatial representation in the hippocampus. *Science (New York, N.Y.) 321* (07 2008), 140–3.

[6] Cuperlier, N., Quoy, M., and Gaussier, P. Neurobiologically inspired mobile robot navigation and planning. *Frontiers in neurorobotics 1* (02 2007), 3.

[7] Delarboulas, P., Gaussier, P., Caussy, R., and Quoy, M. Robustness study of a multimodal compass inspired from hd-cells and dynamic neural fields.

[8] Ekstrom, A., Kahana, M., B Caplan, J., Fields, T., Isham, E., Newman, E., and Fried, I. Cellular networks underlying human spatial navigation. *Nature 425* (10 2003), 184–8.

[9] Espada, Y., Cuperlier, N., Bresson, G., and Romain, O. Application of a bio-inspired localization model to autonomous vehicles. pp. 7–14.

[10] Gaussier, P., Banquet, J., Sargolini, F., Giovannangeli, C., Save, E., and Poucet, B. A model of grid cells involving extra hippocampal path integration, and the hippocampal loop. *Journal of integrative neuroscience 6* (10 2007), 447–76.

[11] Gaussier, P., and Zrehen, S. Perac : A neural architecture to control artificial animals. *Robotics and Autonomous Systems 16* (12 1995), 291–320.

[12] Giovannangeli, C., and Gaussier, P. Orientation system in robots : Merging allothetic and idiothetic estimations. *13th International Conference on Advanced Robotics (ICAR07)* (01 2007).

[13] Giovannangeli, C., and Gaussier, P. Interactive teaching for vision-based mobile robots : A sensory-motor approach. *Systems, Man and Cybernetics, Part A : Systems and Humans, IEEE Transactions on 40* (02 2010), 13 – 28.

[14] Giovannangeli, C., Gaussier, P., and Banquet, J. Robustness of visual place cells in dynamic indoor and outdoor environment. *International Journal of Advanced Robotic Systems 3* (06 2006).

[15] Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., and Moser, E. Microstructure of a spatial map in the entorhinal cortex. *Nature 436* (09 2005), 801–6.

[16] Jauffret, A., Cuperlier, N., and Gaussier, P. From grid cells and visual place cells to multimodal place cell : A new robotic architecture. *Frontiers in Neurorobotics 9* (04 2015).

[17] O'Keefe, J. Place units in the hippocampus of the freely moving rat. *Experimental neurology 51* (05 1976), 78–109.

[18] Quoy, M., Moga, S., and Gaussier, P. Dynamical neural networks for planning and low-level robot control. *Systems, Man and Cybernetics, Part A : Systems and Humans, IEEE Transactions on 33* (08 2003), 523 – 532.

[19] Rolls, E., and Z Xiang, J. Spatial view cells in the primate hippocampus and memory recall. *Reviews in the neurosciences 17* (02 2006), 175–200.

[20] Sheynikhovich, D., Chavarriaga, R., Strösslin, T., and Gerstner, W. Spatial representation and navigation in a bio-inspired robot. pp. 245–264.

[21] Taube, J., Muller, R., and Ranck, Jr, J. Head-direction cells recorded from the postsubiculum in freely moving rats. i. description and quantitative analysis. *The Journal of neuroscience : the official journal of the Society for Neuroscience 10* (03 1990), 420–35.