



24 avril 2017

BERTHIER Eloïse
BOUCHAFAA Amina
BUGNET Auriane
LANGEVIN Denis
LAURENS Jean

REMERCIEMENTS

Nous tenons tout d'abord à remercier nos deux tuteurs de l'entreprise Thales, M. Philippe Mouttou, directeur du développement des études amont de SICS (Systèmes d'Information et de Communication Sécurisés) et M. Jean-François Goudou, responsable des projets collaboratifs, pour leur aide, leur disponibilité et pour le temps et l'attention qu'ils ont consacrés à notre projet. Nous associons à ces remerciements M. Kevin Hoang pour son support technique et pour la clarté de ses explications.

Nous remercions également le coordinateur des projets PSC de Mathématiques Appliquées, M. Lucas Gérin ainsi que notre cadre référent, le capitaine Paul-Henri Gauzence de Lastours pour leur disponibilité et leurs conseils.

Enfin, nous remercions M. Charles-Henri Pin, membre de la Fondation de l'X, pour le temps qu'il accordera à notre travail.

TABLE DES MATIÈRES

Introduction	4
1 Démarche de conceptualisation	5
1.1 Objectifs de notre travail	5
1.2 État de l'art	6
1.2.1 Segmentation d'images	6
1.2.2 Reconnaissance d'objets	6
1.2.3 Reconnaissance de contextes	7
1.2.4 Réseaux de neurones	8
1.3 Réflexions sur les notions d'objet, de contexte, et élaboration d'un lexique	8
2 Exploitation des images	11
2.1 Choix et récupération des données utilisées	11
2.2 Créer des associations pour reconnaître les entités	12
2.2.1 Théorie de Gestalt	12
2.2.2 Fonctions de similarité construites	13
2.2.3 Plateforme de tests	14
2.3 Mise en place technique de ces associations	16
2.3.1 Caractéristiques souhaitées du système de reconnaissance de contextes	16
2.3.2 Solutions techniques	16
3 Déduction du contexte	20
3.1 De la carte de Kohonen au contexte	20
3.1.1 Caractéristiques d'une carte de Kohonen	20
3.1.2 Fonctionnement d'une carte de Kohonen	21
3.1.3 Reconnaissance du contexte	23
3.2 Structure finale du projet et communication avec le robot	25
3.2.1 Cahier des charges du livrable final	25
3.2.2 Décomposition technique du livrable	25
3.3 Résultats obtenus	29
3.4 Réflexions sur les apports du projet et ses éventuels prolongements	31
Conclusion	33
Références	34
Annexes	35
Annexe A : Plateforme de tests de fonctions de similarité	35
Annexe B : Librairie psc_utils	43
Annexe C : Nœuds ROS	48

INTRODUCTION

Notre projet s'intéresse à la **vision artificielle**, c'est-à-dire à la façon dont un système informatique comprend ce qu'il voit. Il s'agit actuellement d'un domaine très actif de l'intelligence artificielle, car la création d'un système intelligent nécessite la compréhension du monde qui l'entoure. Les recherches récentes tentent non seulement d'améliorer la précision et la rapidité des systèmes, mais aussi de les rendre performants pour des utilisations plus variées.

Dans notre cas, suite à la proposition d'une équipe de **Thales Research & Technology**, nous avons cherché à **améliorer la précision d'un système de reconnaissance visuelle existant**. En effet, nous avons constaté que les systèmes de vision artificielle commettent souvent des erreurs qui pourraient être évitées facilement par un humain, notamment s'ils prenaient en compte le contexte de ce qu'ils voient. Par exemple, si le système est à l'intérieur d'une maison, il devrait exclure la possibilité de "voir" une voiture, et en inférer qu'il s'agit plutôt d'une maquette. Pourtant, et c'est ce qui a motivé notre démarche, l'état de l'art que nous avons dressé au début de notre travail a révélé qu'il n'existe que **peu de systèmes où le contexte influence la reconnaissance d'objets**.

Nous avons donc décidé de nous concentrer sur la **reconnaissance des contextes** et le **lien entre contextes et objets**. De plus, nous avons tenté de nous inspirer du fonctionnement de la vision biologique, en particulier la théorie de Gestalt.

Ce projet a vocation à être mis en application sur la plateforme BioVision développée par Thales. Il s'agit d'un robot construit sur une base TurtleBot comportant deux caméras pour l'acquisition d'images, qui contient déjà une partie des mécanismes permettant l'implémentation de nos travaux.



FIGURE 1 – La base de robot OpenSource TurtleBot

1 DÉMARCHE DE CONCEPTUALISATION

1.1 OBJECTIFS DE NOTRE TRAVAIL

Des algorithmes de deep learning pour la détection et pour la classification d'objets sont déjà implémentés sur la plateforme robotique de Thales. Il nous a été demandé de travailler sur l'aspect contextuel et sur l'aspect temporel pour les améliorer. Pour démontrer la pertinence de notre travail, nous avons décidé de mettre en œuvre un système qui serait **capable de réagir face à trois scénarios différents** :

1. Arriver dans un lieu inconnu et être capable d'**annoncer quel est le "contexte"**, c'est-à-dire quel est le type de lieu. Nous avons décidé de nous concentrer sur les espaces clos (bureau, salle de classe, cuisine, etc.) pour notre projet. Cependant il est possible d'envisager le même type de reconnaissance pour des lieux extérieurs.
2. **Déetecter les "anomalies"** présentes dans une salle, c'est-à-dire les objets qui ne sont pas cohérents au vu du contexte (on pourra penser à l'exemple d'un couteau dans une chambre d'enfants).
3. **Chercher dans une salle un objet précis**, suite à une demande de l'utilisateur, en identifiant les zones de recherche prioritaires, c'est-à-dire les zones dans lesquelles il est le plus probable de trouver cet objet.

Dans ces trois scénarios, les notions d'**objets** et de **contextes** sont interdépendantes. Un des objectifs de notre projet était de rendre cette influence **dynamique**, notamment en évitant de créer une hiérarchie entre la notion de contexte et la notion d'objet. En effet, nous ne voulions pas nous contenter d'un système de reconnaissance d'objets dans lequel un groupe d'objets donné renverrait à un contexte donné. Nous souhaitions au contraire que la relation entre objets et contextes soit capable d'évoluer.

Au cours de notre projet, nous avons décidé de nous concentrer sur le **premier scénario**, qui nécessite déjà à lui seul la réalisation de l'objectif principal du projet. A la fin de ce rapport, nous expliquerons en quoi la réalisation du premier scénario sert de support aux deux autres scénarios, et quels sont les prolongements éventuels du projet.

1.2 ÉTAT DE L'ART

Pour pouvoir décider de ce que nous allions faire au cours du projet, il a d'abord fallu évaluer les technologies existantes. Nous avons étudié les différents systèmes actuels pour savoir où il était intéressant de fournir du travail.

1.2.1 • SEGMENTATION D'IMAGES

Un type d'algorithme que l'on peut utiliser pour la reconnaissance d'objets est **la segmentation d'images** [2]. Le but est, à partir d'une image, de définir une partition telle que chaque sous-partie corresponde à un objet. Il faut donc éviter que la segmentation soit incomplète (sous-segmentation), c'est-à-dire que plusieurs objets soient reconnus dans la même sous-partie. Il faut aussi éviter que la segmentation soit trop détaillée (sur-segmentation), c'est-à-dire que les objets soient découpés en plusieurs sous-parties. L'équilibre parfait étant rarement atteignable, pour la reconnaissance d'objets la sur-segmentation est préférable, puisque les algorithmes de classification les plus récents peuvent identifier un objet même avec seulement une de ses parties.

Il existe également des algorithmes faisant une segmentation de l'image volontairement grossière, en distinguant simplement des zones plus "saillantes" [1]. Cette **notion de saillance**¹ dépend fortement du type d'objet et d'image étudiés, mais correspond assez bien au fonctionnement de la vision humaine. En effet, notre œil se tourne de lui-même vers des zones plus saillantes de notre champ de vision, les couleurs vives par exemple. Cette division des images entre zones saillantes et zones peu significantes permet de définir un ordre de priorité des "zones à explorer" pour les autres algorithmes de reconnaissance d'objet.

1.2.2 • RECONNAISSANCE D'OBJETS

La **reconnaissance d'objets** est un des enjeux majeurs de la vision artificielle depuis plusieurs années. Le but est de pouvoir concevoir un système qui, à partir d'une image, donne la liste des objets présents [5]. Cela nécessite deux capacités distinctes, à savoir la détection des objets (par exemple en se basant sur une segmentation), puis leur classification.

On peut distinguer **deux classes de détection d'objets**.

D'une part, certains algorithmes sont spécialisés dans **la reconnaissance d'un type d'objet** [7, 13] (déTECTeurs de Viola-Jones, indexing local features, algorithmes RANSAC ou transformée de Hough pour la détection d'objets à partir de points d'intérêt, ou plus récemment

1. Issue d'observations biologiques, la notion de saillance modélise et cherche à reproduire artificiellement des phénomènes perceptifs du système réflexe.

algorithmes de deep learning). Ceux-ci vont permettre de distinguer les images où ce type d'objet est présent de celles où il ne l'est pas. Une des façons de se servir de ces algorithmes est de les faire travailler sur des sous-parties de l'image globale, afin de savoir dans quelles zones se situe le type d'objet recherché.

D'autre part, il existe des algorithmes qui sont capables de dire **si l'image qu'on leur fournit contient une frontière entre deux objets différents**, par exemple par la détection de changements d'intensité lumineuse [14]. Comme précédemment, en faisant passer ces algorithmes sur des sous-parties de l'image globale, il est possible d'avoir une approximation des endroits où se situent les différents objets de l'image.

Une fois les objets détectés, il faut les identifier. Pour cela, on peut faire passer les différentes zones de l'image par un classifieur . Ces algorithmes travaillent surtout sur des images ne contenant qu'un seul objet, et donnent ensuite la catégorie de l'objet figurant sur l'image. Pour être plus exact, la majorité de ces algorithmes **donne une liste des objets pouvant figurer sur l'image**, et associe à chacun un nombre qui représente la probabilité que ce soit cet objet plutôt que les autres de la liste.

1.2.3 • RECONNAISSANCE DE CONTEXTES

Un autre enjeu de la vision artificielle est la reconnaissance de contextes, c'est-à-dire la capacité d'**associer à une image une description globale** de ce qu'elle représente. Par exemple, et c'est ce qui nous aura intéressé dans ce projet, cette description peut être simplement le nom de la pièce observée. Par ailleurs, pour des applications sécuritaires, cette description peut être un indicateur de la présence d'actes inhabituels ou violents sur les images.

En fonction du type de caractérisation souhaitée, les algorithmes récents [6, 3] ont deux fonctionnements principaux :

D'une part, certains systèmes se servent des **données brutes de l'image** et les font passer par un système classifieur appelé réseau de neurone, dont le principe sera rapidement détaillé ci-après. Comme pour les classifieurs en reconnaissance d'objet, ils donnent comme résultat **la liste des contextes** que représente probablement l'image, souvent associés à une probabilité de chaque contexte.

D'autre part, de nombreux systèmes de reconnaissance de contexte **travaillent à partir des résultats d'une reconnaissance d'objet** sur l'image considérée. Ceux-ci permettent en général de donner le type de lieu de l'image, avec l'idée sous-jacente que les objets caractérisent le lieu. Ils se basent notamment sur des **règles logiques** telles que "s'il y a une chaise et une table mais pas de frigo ni de lit, cette pièce est un bureau", qui dépendent toutefois fortement de l'efficacité de la reconnaissance d'objet qui précède.

1.2.4 • RÉSEAUX DE NEURONES

Les réseaux de neurones sont des systèmes d'intelligence artificielle permettant de faire de nombreuses tâches différentes, allant de la reconnaissance d'écriture manuscrite à la conduite autonome. L'idée centrale est de copier le fonctionnement du cerveau humain et des réseaux de neurones biologiques qu'il contient, mais pour une seule tâche [4, 8]. Ces systèmes ont donc une architecture comportant plusieurs couches successives de neurones artificiels, où **l'information donnée en entrée (une image par exemple) se propage pour traverser le réseau**. Cette propagation a lieu en activant chaque couche successive en fonction de l'activation de la couche précédente et des liens reliant les neurones entre eux. En effet, l'activation d'un neurone cérébral dépend de l'activation des neurones environnants et des liens (en biologie, des axones), qui les relient au neurone donné.

Les réseaux de neurones les plus simples ont donc pour résultat **l'activation des neurones de leur dernière couche**. C'est par exemple le cas des classifieurs, où chaque neurone de la dernière couche est associé à une des catégories que le réseau peut reconnaître. C'est pourquoi un classifieur est capable de donner la liste des objets qu'une image donnée pourrait contenir, associés à des probabilités : cette liste provient de la liste des neurones de la dernière couche du réseau qui reçoivent des activations plus ou moins intenses.

Il existe des réseaux de neurones avec des architectures plus élaborées pour améliorer leurs performances dans des environnements plus complexes. C'est le cas des **réseaux de neurones convolutionnels**, qui ont eu une grande importance pour notre travail [11]. Leur rôle n'est plus de classifier, mais d'extraire depuis les images les caractéristiques les plus pertinentes pour la classification. En somme, ce sont des systèmes qui viennent se placer entre l'image et le réseau de neurones "simple" qui classe, afin d'optimiser les données sur lesquelles travaille le classifieur.

Ainsi, comme nous l'avons signalé en introduction de ce rapport, il existe **peu d'algorithmes se servant d'informations sur le contexte pour améliorer la reconnaissance d'images** [6, 3]. Un autre constat possible est qu'ici, nous n'avons parlé que d'images, jamais de vidéos, alors que beaucoup des systèmes de vision artificielle se basent sur des flux vidéos. La dimension temporelle n'est donc jamais prise en compte, alors qu'elle fournit beaucoup d'informations à la vision humaine. Cela fait partie des idées nouvelles sur lesquelles nous nous sommes basés pour effectuer ce projet.

1.3 RÉFLEXIONS SUR LES NOTIONS D'OBJET, DE CONTEXTE, ET ÉLABORATION D'UN LEXIQUE

Tout au long de notre réflexion sur le projet, nous avons été amenés à **préciser la définition des différents concepts** avec lesquels nous travaillons afin d'éviter tout quiproquo sur les notions manipulées. La difficulté initiale était que, dans le cadre de la vision artificielle, nous

étions très souvent amenés à parler de concepts liés à la vie de tous les jours (objets, contextes), alors que leurs définitions renvoient souvent à des notions subjectives (fonctions des objets par exemple).

Nous avons commencé par réfléchir à la notion d'objet, car que notre première idée était d'utiliser la reconnaissance des objets présents dans une salle pour en déduire le contexte. Mais la difficulté majeure soulevée par cette méthode est directement **la reconnaissance d'objets**. En effet, celle-ci est encore un champ très ouvert de la vision artificielle, car elle met en jeu plusieurs aspects complexes :

- **Identification des contours** : Pour reconnaître des objets sur une image, il faut avant tout être capable de savoir où ils sont. Cela nécessite d'identifier les limites d'un objet, ou au moins dans quelles sous-parties de l'image il apparaît.
- **Classification** : Les algorithmes de reconnaissance d'objets les plus performants se basent sur des méthodes de classification. Cela signifie qu'ils classent les entités qu'ils voient parmi une liste d'objets déjà rencontrés, et ne sont donc pas capables de traiter le cas de nouveaux objets.
- **Sémantique** : Ce qui donne le sens d'un objet pour un humain n'est ni sa forme, ni sa localisation, mais sa fonction. D'une part, pour certains objets dont la fonction définit fortement la forme, tels qu'une fourchette, un train ou une antenne radio, cela permet tout de même de se servir de la forme de l'objet pour le reconnaître. D'autre part, pour des objets comme les chaises, ou les tables, qui n'ont pour prérequis qu'un plan horizontal et une forme d'appui au sol, les variations de formes sont extrêmement diverses. Ce qui définit une chaise est le fait que l'on puisse s'asseoir dessus, tout comme ce qui définit une table est le fait que l'on puisse poser des choses dessus. Or ce type d'information sémantique est très difficile à représenter dans un système informatique.

Nous avons donc décidé par la suite de **ne pas utiliser de reconnaissance d'objets pour reconnaître le contexte**. Ainsi, l'apport principal de notre projet a été de chercher à reconnaître des "entités" de manière plus générale, pouvant correspondre dans la réalité à des parties d'objets ou à des ensembles d'objets, et de s'intéresser à leur **présence simultanée** pour avoir des informations sur le contexte.

Afin de définir à quoi correspondaient ces "entités" dans la pratique, nous avons commencé par définir le **type de données** que nous avons utilisées, notamment en sortie du système de transformation des images. Comme nous le détaillerons plus loin, nous avons décidé de travailler avec un **réseau de neurones convolutionnel**, qui transforme chaque image perçue par la caméra en un tableau de vecteurs de caractéristiques. Nos données sont donc localisées dans un espace à 4 dimensions, à savoir :

- **2 dimensions d'espace** (x, y), pour localiser les informations sur l'image ou dans les

données de sortie du réseau de neurones.

- **1 dimension de temps**, renvoyant à l'instant où l'image a été perçue par la caméra, et permettant d'ordonner les images les unes par rapport aux autres.
- **1 dimension de caractéristiques**, avec un nombre fini d'éléments, fixé par le nombre de caractéristiques dans la sortie du réseau de neurones : 512 dans notre cas.

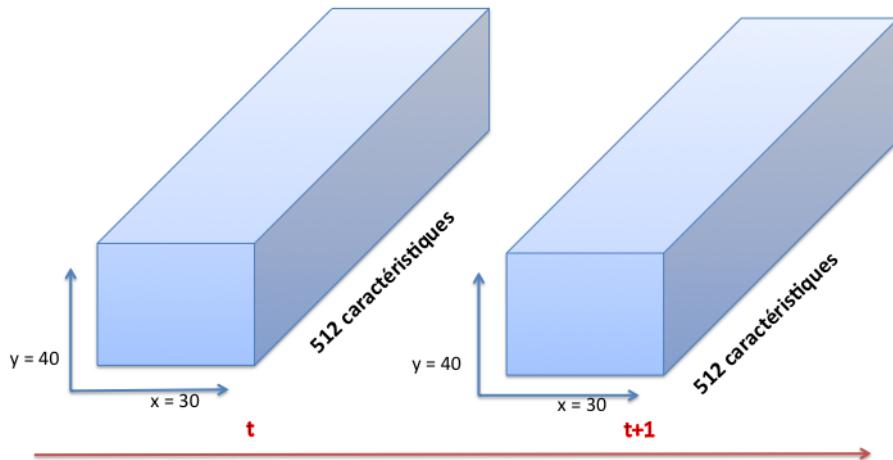


FIGURE 2 – Schéma des différentes dimensions des données

Nous avons ainsi pu donner des **définitions plus précises** aux notions utilisées dans notre projet :

- **Vecteur de caractéristiques** : Donnée de sortie du réseau de neurone, dont on a pris un élément dans l'espace et le temps. On a donc l'ensemble des caractéristiques venant de cet instant et de cette position.
- **Entité** : Ensemble de vecteurs de caractéristiques qui sont proches pour une mesure donnée (en terme de caractéristiques, d'espace ou de temps). Dans la réalité, cela ne renvoie pas nécessairement à un objet mais par exemple à une partie d'un objet, ou à un ensemble d'objets. Il n'y a pas non plus dans la notion d'entité de lien avec la fonction de l'objet pour un humain. Néanmoins, pour des raisons de simplicité, nous utiliserons indifféremment les termes d'entité ou d'objet dans la suite du rapport.
- **Contexte** : Concept général qui regroupe plusieurs lieux similaires. Par exemple, une salle de petite classe donnée est l'instance du concept de "salle de classe". Nous détaillerons dans la partie III de ce rapport la façon dont le contexte peut être déduit des entités reconnues par le robot.

2 **EXPLOITATION DES IMAGES**

2.1 CHOIX ET RÉCUPÉRATION DES DONNÉES UTILISÉES

Le robot détecte son environnement grâce à plusieurs capteurs :

- Une **caméra "Time Of Flight"**, qui permet sa navigation en évitant les obstacles.
Nous n'exploitons pas ces images directement.
- Une **caméra champ large**, qui produit des images avec une résolution moyenne de l'ensemble d'une scène.
- Une **caméra fovéale**, qui permet d'avoir une meilleure résolution au centre du champ de vision. Dans une approche biomimétique, cette caméra imite le fonctionnement de l'œil humain, qui voit mieux au centre que sur les côtés. Associée avec la caméra grand champ, elle permet de créer une asymétrie entre les informations qui proviennent du centre, plus précises, et celles des côtés, plus vagues.

Pour choisir ce que doit regarder le robot, nous avions au départ décidé d'exploiter **une carte de saillance** déjà établie par le groupe de travail de Thales, qui nous permettrait d'orienter le centre de la vision du robot vers les régions les plus saillantes de l'image. Cela permet au robot d'orienter sa caméra fovéale vers des zones d'intérêt, définies par la présence de lignes perpendiculaires (par exemple des objets posés sur une table).

Pour les tests que nous présentons dans ce rapport, nous avons décidé de n'exploiter que les images issues de la caméra champ large, et non celles de la caméra fovéale. Cela implique en particulier que **nous n'utilisons pas la carte de saillance**. En effet, le développement de cette fonction ne fait pas partie des objectifs principaux du projet, mais cette fonction peut être facilement combinée avec la reconnaissance de contextes (l'utilisation de nœuds ROS permet d'utiliser plusieurs processus en parallèle). En première approche, le robot est dirigé manuellement dans la salle, sans attention particulière à l'endroit où pointe la caméra fovéale.

Nous avons choisi en parallèle la caméra que nous allions utiliser et la façon dont on exploiterait les images. Nous avons exploré et discuté plusieurs pistes avant de prendre notre décision finale. Nous nous sommes d'abord demandé **comment distinguer les objets présents sur l'image**.

Pour cela, les deux solutions envisagées présentaient des inconvénients :

- Utiliser le **champ de la caméra fovéale**, qui est suffisamment réduit pour se concentrer sur un ou deux objets, tandis que le reste de l'image serait flou. Cela reste cependant très dépendant de la configuration de la scène, des tailles relatives des objets et de la distance du robot aux objets.
- Segmenter l'**image** grâce à des algorithmes existants. Cela pose la question du réglage de la précision de la segmentation, qui serait plutôt empirique. Nous avons abandonné cette solution car la segmentation n'est a priori pas stable au cours du temps (à des temps rapprochés et séparés par un petit mouvement de la caméra, les deux segmentations peuvent être très différentes).

Ces difficultés nous ont conduits à exploiter un autre type de données issues de l'image de la caméra champ large. Nous avons utilisé **exclusivement les vecteurs de caractéristiques** de l'image, qui sont issus du traitement de l'image par un réseau de neurones convolutionnel (RNC).

En quelque sorte, nous avons délégué le travail de **pré-traitement de l'image à un RNC**. Celui-ci a été pré-entraîné sur un grand nombre d'images, et permet d'extraire de l'image les caractéristiques les plus pertinentes pour la reconnaissance d'objets. Nous avons donc choisi de nous intéresser à des informations à un plus haut niveau d'abstraction.

Notre idée était que **l'exploitation de ces vecteurs de caractéristiques puisse servir de base pour la reconnaissance de contextes**.

2.2 CRÉER DES ASSOCIATIONS POUR RECONNAÎTRE LES ENTITÉS

2.2.1 • THÉORIE DE GESTALT

Afin de distinguer les entités les unes des autres, nous avons décidé de nous inspirer du fonctionnement de la reconnaissance visuelle humaine. Nous nous sommes essentiellement basés sur **la théorie de Gestalt**, une théorie psychologique et biologique établie au XXème siècle sur la façon dont le cerveau humain interprète les observations de l'œil pour reconnaître des objets [9, 12]. Selon cette théorie, pour reconnaître un objet, le cerveau procède par **regroupement** : à partir d'une multiplicité de petites formes distinctes, le cerveau reconnaît une forme globale qui ressemble le plus à ce qu'il connaît déjà. Autrement dit, le tout est perçu différemment que la somme des parties.

Six "lois de la forme" régissent ce regroupement inconscient :

- **La loi de proximité**, selon laquelle les zones de l'image les plus proches ont plus de chance d'appartenir au même objet.
- **La loi de similitude**, selon laquelle les zones qui se ressemblent le plus ont également plus de chance d'appartenir au même objet.
- **La loi de continuité**, selon laquelle le cerveau perçoit une continuité entre des zones proches, même si elles ne sont pas en contact directement.
- **La loi de clôture**, selon laquelle une forme avec un contour fermé est plus facilement identifiée comme telle par le cerveau.
- **La loi de symétrie**, selon laquelle une entité qui présente des axes de symétrie sera plus facilement perçue comme une forme globale.
- **La loi de destin commun**, selon laquelle des zones en mouvement qui ont la même trajectoire seront reconnues comme appartenant au même objet.

2.2.2 • FONCTIONS DE SIMILARITÉ CONSTRUITES

Nous avons eu l'idée de créer des **fonctions de similarité à partir des lois psychologiques et biologiques de Gestalt**. Cette idée a été très peu explorée jusqu'à présent dans le domaine de l'intelligence artificielle. C'est pourquoi nous mettons l'accent ici sur les tests informatiques des fonctions que nous avons créées. Voici les deux fonctions que nous avons considérées :

- Une distance spatio-temporelle entre points de l'espace 4D :

$$D(X_1, X_2) = \begin{cases} \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}, & \text{si } t_1 = t_2 \\ \alpha|t_1 - t_2|, & \text{si } 0 < |t_1 - t_2| < \tau \\ \alpha\tau, & \text{sinon} \end{cases}$$

Où $X = (x, y, t, U)$ avec (x, y) les coordonnées du pixel sur l'image, la date d'enregistrement de l'image, U le vecteur de caractéristiques et α un facteur de proportionnalité.

Pour fabriquer cette fonction, nous nous sommes inspirés de la « **loi de proximité** » de Gestalt. Mais il était difficile de calculer la distance entre deux points de deux images différentes car le robot est en mouvement et prend des images à partir d'endroits différents de la pièce. Nous avons donc décidé de prendre en compte **l'information temporelle** : plus les dates d'enregistrement des deux images sont proches, et plus leurs points sont proches physiquement. Néanmoins, nous avons remarqué que cette information n'était pertinente que si les deux observations étaient très proches dans le temps par exemple.

En effet, sur des temps plus longs, il est possible que le robot revienne plusieurs fois au même endroit, donc des observations éloignées dans le temps ne correspondent pas nécessairement à des objets éloignés. Nous avons donc décidé de fixer une durée limite (τ) entre les prises des deux images, à partir de laquelle la distance temporelle considérée n'augmente plus.

— Des fonctions de similarité sur les caractéristiques :

Une autre loi de Gestalt dont nous nous sommes inspirés est la "**loi de similarité**". En effet, nous pensons que la similarité (de couleurs, d'aspects, de forme,...) est un critère majeur pour distinguer les objets les uns des autres. Nous avons donc essayé de construire une fonction de similarité entre vecteurs de caractéristiques, permettant de rapprocher les vecteurs correspondants à des zones de l'image similaires. Pour comparer 2 vecteurs entre-eux, nous avons d'abord pensé à la **norme 2**. Nous avons également pensé à utiliser la **similarité cosinus** :

$$S(X_1, X_2) = \cos(\theta) = \frac{X_1 \cdot X_2}{\| X_1 \| \| X_2 \|}$$

La valeur de cette fonction est comprise entre -1 et 1, et les vecteurs sont d'autant plus similaires que $S(X_1, X_2)$ est proche de 1. Bien que l'opposé de cette similarité ne soit pas une distance au sens mathématique du terme, nous parlerons de distance pour des raisons de simplification.

2.2.3 • PLATEFORME DE TESTS

Nous avons réalisé en janvier une **plateforme de test en langage Python** afin de tester la norme 2 et la similarité cosinus sur les caractéristiques des vecteurs obtenus à partir de plusieurs images, afin de savoir si elles reflétaient bien une similarité sur les objets réels.

La plateforme de tests devait remplir un **cahier des charges** succinct :

- permettre de tester efficacement un nouvel algorithme, ou une nouvelle fonction de similarité, etc., sur un ensemble de données (fournies par Thales) en un temps de développement relativement court. Cela suppose une certaine abstraction et une certaine facilité d'utilisation de sorte à ce que les premières difficultés techniques (gestion d'erreur, parsing des fichiers, etc.) puissent être écartées pour tous les membres du groupe.
- permettre la visualisation des résultats des tests sous une forme synthétique et adaptée à leur nature.
- permettre une gestion documentée des erreurs et des processus pendant l'exécution.
- être aussi modulaire que possible de sorte à pouvoir être exploité pendant tout le projet et après.

- être exécutable sur plusieurs plateformes (Windows, MacOs, Ubuntu).

Pour satisfaire à ces exigences, un certain nombre de **choix techniques** ont été faits, notamment pour l'utilisation :

- de Python 3.5. de la norme PEP-8 pour le code, de sorte à maximiser la lisibilité et la réutilisation du code.
- du module "multiprocessing" de Python, chaque fois que cela est possible sans nuire à la compréhension du code.
- du module "logging" permettant une gestion simple et automatisée des interactions avec l'utilisateur pendant et après l'exécution du code.
- du module PyYAML plutôt que du module OpenCV pour Python, pour des raisons de simplicité d'utilisation du premier. Les données qu'exploite notre plateforme de tests sont en effet écrites en YAML, un format de sérialisation de données courant qui pouvait donc être exploité soit nativement par OpenCV soit par YAML.

La **plateforme de tests** (voir code en **annexe A**) comprend un module nommé "**psc_utils**" comprenant une classe "Tensor" capable de parser les données et de représenter en mémoire un tenseur de caractéristiques issu d'un vecteur de caractéristiques, et d'effectuer sur celui-ci les tests des fonctions de similarité. Pour ce faire, cette classe exploite les ressources du module PyYaml et de la librairie PIL (Python Image Library) pour fournir à l'utilisateur la possibilité d'exploiter les données fournies par Thales sans se soucier des aspects techniques de l'obtention des données. Le module "psc_utils" a permis d'**apprécier la pertinence des différentes fonctions de similarités que nous voulions exploiter** et de pouvoir discriminer vis-à-vis de nos objectifs.

Nous avons utilisé trois images prises par le robot, aux instants $t = 1$, $t = 6$ et $t = 99$. Pour toute position (i, j) sur l'image, nous avons calculé la **distance entre les vecteurs de caractéristiques** à cette position **provenant de deux images**. Nous nous attendions donc à obtenir des similarités plus fortes entre les deux premières images qu'entre la première et la dernière. Les résultats sont reproduits en **annexe A**.

Les résultats sont **conformes à nos attentes**, en ordre de grandeur. En effet, la distance norme 2 est plus grande entre les caractéristiques issues des images éloignées, et la similarité cosinus est plus grande entre les caractéristiques issues des images proches. Cela n'était pas totalement évident, car nous ne comparons pas directement des images, mais des vecteurs en sortie d'un réseau de neurones convolutionnel.

Nous utilisons dans le livrable les fonctions de similarités sous deux formes distinctes au sein d'une carte de Kohonen. Nous détaillerons l'utilisation de ces fonctions dans la partie suivante.

2.3 MISE EN PLACE TECHNIQUE DE CES ASSOCIATIONS

2.3.1 • CARACTÉRISTIQUES SOUHAITÉES DU SYSTÈME DE RECONNAISSANCE DE CONTEXTES

Une fois les informations visuelles transformées par le système fourni par Thales, il faut les utiliser pour interpréter l'image et en tirer l'information souhaitée : son contexte. Pour cela, nous avons listé les différentes **fonctionnalités nécessaires au système de reconnaissance de contexte** :

- *Mémoire et reconnaissance* : Il s'agit d'associer chaque sortie du réseau de neurones convolutionnel aux contextes connus. Cela permet donc d'associer les images prises par le robot aux contextes dans lesquels elles se trouvent. Il faut bien sûr que cette association ait lieu même si l'image n'a jamais été vue auparavant.
- *Apprentissage* : Si le contexte est connu, alors le système doit pouvoir évoluer lors de la réception des images, pour associer de manière autonome les vecteurs de caractéristiques et contextes.
- *Propagation de l'information* : Faire en sorte que l'information de chaque étape influence les étapes suivantes. Cela permet de prendre en compte la dimension temporelle des vidéos, où chaque image s'inscrit dans la continuité des précédentes.

Nous avions également eu l'idée de créer un système qui puisse utiliser l'**inférence contextuelle**, c'est-à-dire que la connaissance du contexte puisse donner des informations utiles pour la reconnaissance d'objets. Par exemple, le fait d'être dans une cuisine nous indiquerait qu'il y a une forte probabilité que l'on observe des chaises. Nous avons finalement **abandonné cette propriété** à partir du moment où nous avons abandonné la notion d'objet.

2.3.2 • SOLUTIONS TECHNIQUES

La première idée que nous avons envisagée était d'utiliser un réseau de neurones habituel, c'est-à-dire un **classifieur**. En effet, dans le cadre de l'intelligence artificielle, les réseaux de neurones sont omniprésents car efficaces pour des tâches différentes. Nous avons donc tenté de voir si ces systèmes répondaient à tous nos critères.

- *Mémoire et reconnaissance* : La **capacité de classification** des données est une des caractéristiques de base d'un réseau de neurones. De plus, si l'apprentissage a été fait avec suffisamment de données, ils ont aussi une bonne **capacité de généralisation**, c'est-à-dire de reconnaissance de contextes sur des images qui n'ont jamais été vues auparavant. Toutefois, le fonctionnement d'un tel réseau nécessite d'avoir entièrement figé son architecture, en particulier le nombre de neurones de la dernière couche. Or,

dans notre cas, chacun de ces neurones serait associé à un contexte que l'on souhaite reconnaître. Cela signifie que l'ajout d'un nouveau contexte à reconnaître nécessiterait de **reconstruire un nouveau système à partir de zéro**.

- *Apprentissage* : Comme la classification, l'apprentissage est une des caractéristiques fondamentales pour lesquelles les réseaux de neurones sont utilisés. Il subsiste toutefois un bémol : le fonctionnement classique de l'apprentissage d'un réseau de neurones **se déroule en une fois**, sur l'intégralité des données. Or nous aimerais pouvoir analyser les images une par une au fur et à mesure que le robot se déplace dans la pièce. Par ailleurs, l'apprentissage de ces systèmes est dit "**supervisé**", c'est-à-dire que pour le faire apprendre, il faut commencer par lui fournir des données dont on connaît le contexte. Cela constitue donc une difficulté supplémentaire.
- *Propagation de l'information* : Ceci est le point faible majeur des réseaux de neurones. En effet, les réseaux de neurones **n'intègrent pas d'information temporelle**, et ont été justement pensés pour que l'ordre dans lequel les données arrivent lors de l'apprentissage ne modifie pas le résultat. Parvenir à propager l'information d'une étape sur l'autre nécessiterait donc de repenser entièrement leur fonctionnement.

En somme, bien qu'un réseau de neurones semble être la réponse évidente, notre volonté de prendre en compte la dimension temporelle des vidéos étudiées nous a amené à **ne pas les utiliser**.

C'est pourquoi nous avons réfléchi à d'autres solutions techniques. Nous avons notamment eu l'idée de construire un **graphe** dont les sommets seraient des objets et des contextes. Nous avions pensé que la force d'une arête reliant deux sommets donnés serait influencée par la **probabilité de coprésence** des objets correspondant aux sommets. Les contextes auraient simplement été des sommets de nature différente, reliés aux objets par la probabilité qu'un tel objet se trouve dans ce contexte (cf schéma). Par exemple, le contexte cuisine aurait eu un lien plus fort avec les objets chaises et table qu'avec l'objet ordinateur. Le but était de faire apprendre au système quelles étaient les forces des liens entre les sommets du graphe, pour pouvoir ensuite propager l'information le long de ces arêtes. Ceci aurait permis de répondre de manière efficace à toutes les fonctions listées ci-dessus, à l'exception de l'évolution lors de l'apprentissage.

- *Propagation de l'information* : L'information présente dans le graphe serait dénotée par l'activation de chaque sommet, c'est-à-dire la probabilité de présence de l'objet ou du contexte associé. Ainsi, cela permettrait de propager l'information en faisant en sorte que chaque sommet active ses voisins en fonction de sa propre activation et de la force des liens qui les lient à lui. L'arrivée d'une information provenant des images perçues par le système aurait permis d'activer spécifiquement un sommet.
- *Mémoire et reconnaissance* : Le but de ce graphe est qu'à chaque instant, le sommet-contexte ayant la plus grande activation soit le contexte le plus probable . L'hypothèse

sous-jacente est que la relation entre les objets figurant sur une image définit le contexte de cette image. Dans ce cas, un tel graphe aurait une bonne capacité de reconnaissance.

- *Apprentissage* : Pour faire apprendre le graphe, il faudrait placer le robot dans un contexte connu et le faire reconnaître les objets présents dans la salle pour qu'il puisse, par lui-même, modifier la force des liens entre les objets présents et le contexte. Ceci nécessite déjà de reconnaître des objets, ou plus généralement, des entités.

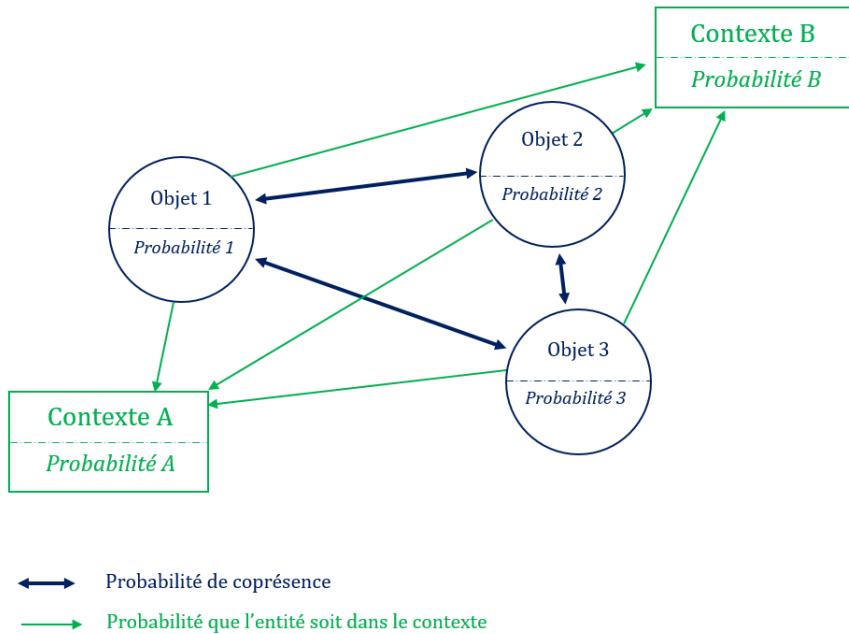


FIGURE 3 – Schéma de la structure du graphe d'objets et de contextes

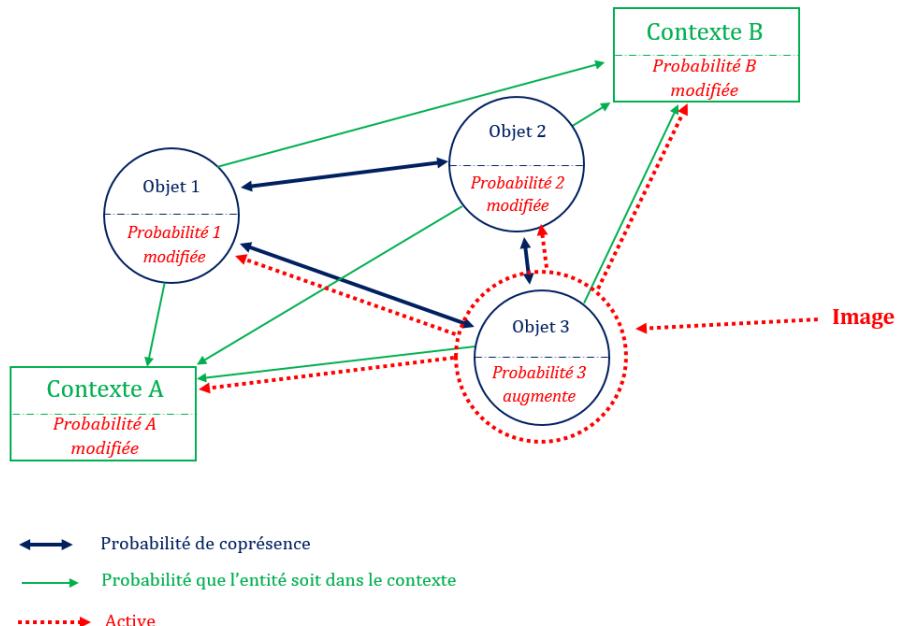


FIGURE 4 – Schéma de l'évolution du graphe lors de l'activation par une image

Pour cela, nous avons d'abord pensé à utiliser un **algorithme de "clustering"**. Le clustering est une opération mathématique qui permet de définir des sous-catégories (appelés clusters) parmi un ensemble de données. Ici, il s'agirait de définir des sous-catégories de vecteurs de caractéristiques, qui correspondent dans la réalité à des entités. Pour faire cela, il fallait répondre à deux questions :

1. **Quelle mesure de distance** utiliser pour comparer les vecteurs de caractéristiques ? En effet, la façon courante de définir l'appartenance à un cluster est de définir une distance entre les objets à classifier, et de décider que chaque objet appartient au cluster dont il est le plus proche. Or si les clusters doivent correspondre à des entités, il faut que la distance soit pertinente dans cette optique.
2. **Comment faire évoluer le graphe** de manière continue ? Par exemple, comment faire évoluer le nombre d'entités présentes dans le graphe lorsque le système en rencontre des nouvelles ? La difficulté est de savoir dans quelle situation il est nécessaire de rajouter un sommet au graphe, au lieu de l'associer à un sommet existant.

Nous avons pu répondre à la première question à partir de notre réflexion sur la théorie de Gestalt. Toutefois, nous n'avons pas réussi à élaborer le modèle mathématique permettant de répondre à la seconde. Cette difficulté, ainsi que l'abandon de la notion d'objet nous ont conduits à **repenser la structure de données** permettant de remplir les fonctionnalités souhaitées.

Finalement, nous avons opté pour la **classification des vecteurs de features grâce à une carte de Kohonen**. Nous précisons leur fonctionnement plus en détail dans la partie ci-après.

3**DÉDUCTION DU CONTEXTE****3.1 DE LA CARTE DE KOHONEN AU CONTEXTE****3.1.1 • CARACTÉRISTIQUES D'UNE CARTE DE KOHONEN**

Les cartes de Kohonen [10] sont un type particulier de réseau de neurones. Elles sont constituées d'une **carte en deux dimensions**, quadrillée, où chaque point du quadrillage est associé à un "neurone". Ces neurones sont en réalité des vecteurs de la même taille que ceux qui doivent être classifiés. Cela permet de comparer les neurones de la carte aux données qu'elle doit représenter. Cette comparaison se fait avec les fonctions de similarité décrites précédemment.

- *Mémoire et reconnaissance* : Le but de la carte de Kohonen est d'ajouter de la cohérence aux données qu'elle reçoit en entrée. En effet, après l'apprentissage, **les neurones présents dans une même région de la carte sont proches au sens des fonctions de similarité** considérées. Ainsi, deux vecteurs semblables en entrée seront proches des neurones situés dans la même région, qui sera alors activée. Pour reconnaître un contexte à partir d'une carte de Kohonen, il faut passer par une **carte de chaleur** qui lui est associée. Lorsque des vecteurs de caractéristiques à classifier sont donnés en entrée, ils **activent la région de la carte de Kohonen** dont ils sont les plus **similaires**. Cela signifie que la température de la région correspondante de la carte de chaleur est augmentée. Des vecteurs similaires vont donc activer des régions semblables, et conduire à la même carte de chaleur. En conclusion, cela permet de visualiser les informations contenues dans les vecteurs à 512 dimensions à l'aide d'une **carte des températures en deux dimensions**. Cette visualisation devrait permettre de **distinguer des régions** correspondant à des contextes et donc de les reconnaître.
- *Apprentissage* : L'apprentissage d'une carte de Kohonen est **non supervisé**, c'est-à-dire qu'il ne nécessite pas la connaissance du contexte d'où les données proviennent. En effet, l'apprentissage fait en sorte que les différentes régions de la carte correspondent aux différents contextes déjà rencontrés. Pour cela, il faut que l'apprentissage rende la carte **cohérente localement**, c'est-à-dire que les neurones proches sur la carte soient également proches selon les fonctions de similarité. La première étape est de comparer le vecteur de caractéristiques d'entrée aux neurones de la carte, pour **trouver celui qui en est le plus proche**. Les valeurs du neurone trouvé et de ses voisins sont ensuite **modifiées** pour qu'elles se rapprochent de celle du vecteur d'entrée. Cela permet de modifier les régions progressivement, tout en veillant à ce que les neurones voisins sur la carte se ressemblent. De plus, le fait de rapprocher les neurones des vecteurs donnés en entrée permet de modifier la carte de Kohonen pour avoir des régions correspondant

aux vecteurs observés. Cela est très utile pour traiter **le cas des nouveaux contextes**, par exemple, car certains neurones de la carte de Kohonen seront simplement modifiés pour se rapprocher des vecteurs d'entrée (cf figure 5).

- *Propagation de l'information* : Dans le cadre de ce système, c'est la carte de chaleur qui permet de **donner une dimension temporelle à l'information**. En effet, plutôt que de récupérer l'activation suite à un seul vecteur, l'idée est de laisser plusieurs vecteurs réchauffer différentes zones de la carte, et d'en prendre la synthèse. De plus, afin d'éviter qu'un seul vecteur puisse à lui seul avoir une grande influence sur le résultat, il est possible de rajouter un **refroidissement progressif de la carte**. Ainsi, les régions directement liées au contexte vont rester chaudes car elles seront activées à plusieurs reprises, tandis que les régions correspondant à des anomalies se refroidissent après avoir été activées une seule fois.

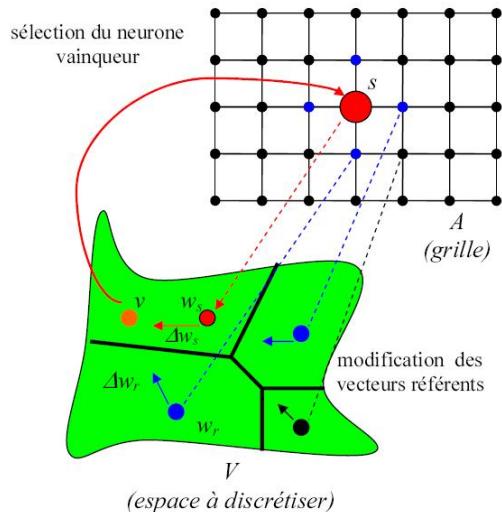


FIGURE 5 – Évolution d'une carte de Kohonen - source : Wikipedia

3.1.2 • FONCTIONNEMENT D'UNE CARTE DE KOHONEN

Nous allons décrire l'**algorithme** précis de l'utilisation de la carte de Kohonen, qui est découpé en deux étapes. Pour le détail des opérations mathématiques, nous introduisons les notations suivantes :

- $w_{i,j}$ désigne le neurone de la carte aux coordonnées (i, j) .
- v désigne le vecteur que l'on donne en entrée à la carte pour son apprentissage.
- α désigne la vitesse d'apprentissage de la carte de Kohonen, β l'incrément de température de la carte de chaleur lors de l'activation, γ le décrément de température de la carte

de chaleur lors du refroidissement général.

- $d_{Gestalt}$ est la distance de similarité utilisée par la carte de Kohonen et définie à l'aide de la théorie de Gestalt, elle travaille avec des vecteurs de dimension 512. Dans le livrable, nous avons implémenté les distances de la partie 2.2, et nous laissons le choix de la distance utilisée à l'utilisateur.
- d_{Carte} est une fonction permettant de trouver les voisins au sein de la carte, qui travaille simplement avec les coordonnées des points. R est le rayon qui définit la relation de voisinage sur la carte.

La première étape est celle de **l'apprentissage**, où la carte de Kohonen apprend à mieux représenter les données. A partir de chaque nouveau vecteur de caractéristiques donné à la carte de Kohonen, l'apprentissage se décompose en 3 temps.

1. Trouver le neurone le plus proche du vecteur donné en entrée :

$$(i_0, j_0) = \operatorname{argmin}(d_{Gestalt}(v, w_{i,j}), (i, j) \in Carte)$$

2. Trouver les voisins de ce neurone :

$$U = \text{voisins}(i_0, j_0) = \{(i, j) \in Carte, d_{carte}((i, j), (i_0, j_0)) \leq R\}$$

3. Modifier les vecteurs sélectionnés :

$$\forall (i, j) \in U, w_{i,j} = w_{i,j} + \alpha(v - w_{i,j})$$

Une fois cette étape effectuée avec des vecteurs provenant de nombreuses images différentes, il existe dans la carte des neurones ressemblant à chacun des vecteurs rencontrés. Cela permet ensuite d'utiliser la carte de Kohonen, qui associe à chaque vecteur entrant une région de la carte.

La deuxième étape est donc celle de **l'activation**, c'est-à-dire de l'utilisation de la carte de Kohonen pour créer une carte de chaleur dans laquelle les zones les plus activées seront plus "chaudes" que les autres.

1. Trouver le neurone le plus proche du vecteur donné en entrée :

$$(i_0, j_0) = \operatorname{argmin}(d_{Gestalt}(v, w_{i,j}), (i, j) \in Carte)$$

2. Trouver les voisins de ce neurone :

$$U = \text{voisins}(i_0, j_0) = \{(i, j) \in Carte, d_{carte}((i, j), (i_0, j_0)) \leq R\}$$

3. Augmenter la température (i.e. activer) les neurones sélectionnés :

$$\forall (i, j) \in U, h_{i,j} = h_{i,j} + \beta$$

4. Refroidir l'ensemble de la carte :

$$\forall (i, j) \in Carte, h_{i,j} = h_{i,j} - \gamma$$

Une fois cette étape effectuée avec les vecteurs correspondant au contexte dans lequel le robot est présent actuellement, la carte de chaleur est une représentation bidimensionnelle de l'information visuelle reçue.

3.1.3 • RECONNAISSANCE DU CONTEXTE

En résumé, pour classifier les contextes, nous procédons de manière **partiellement supervisée**.

En effet, la **carte de Kohonen** est établie par apprentissage **non supervisé**, sur des données issues de différents contextes. Nous fixons alors en mémoire la carte de Kohonen.

Cette carte est ensuite activée avec des données nouvelles, issues du contexte que l'on veut reconnaître. Nous obtenons alors une **carte de chaleur** correspondant à ce contexte. Deux cartes de chaleur correspondant à deux occurrences d'un même contexte seront proches.

Nous utilisons enfin un **classifieur** (réseau de neurones par exemple) qui associe à une carte de chaleur le contexte le plus probable. Pour que ce classifieur soit efficace, il doit au préalable apprendre de façon **supervisée**. Cela signifie qu'on lui fournit en entrée des cartes de chaleur associées à leurs contextes respectifs, et qu'il adapte lui-même son algorithme pour que les cartes de chaleurs soient effectivement associées aux contextes donnés.



Apprentissage



Cheminement préexistant à notre projet



L'apport du projet

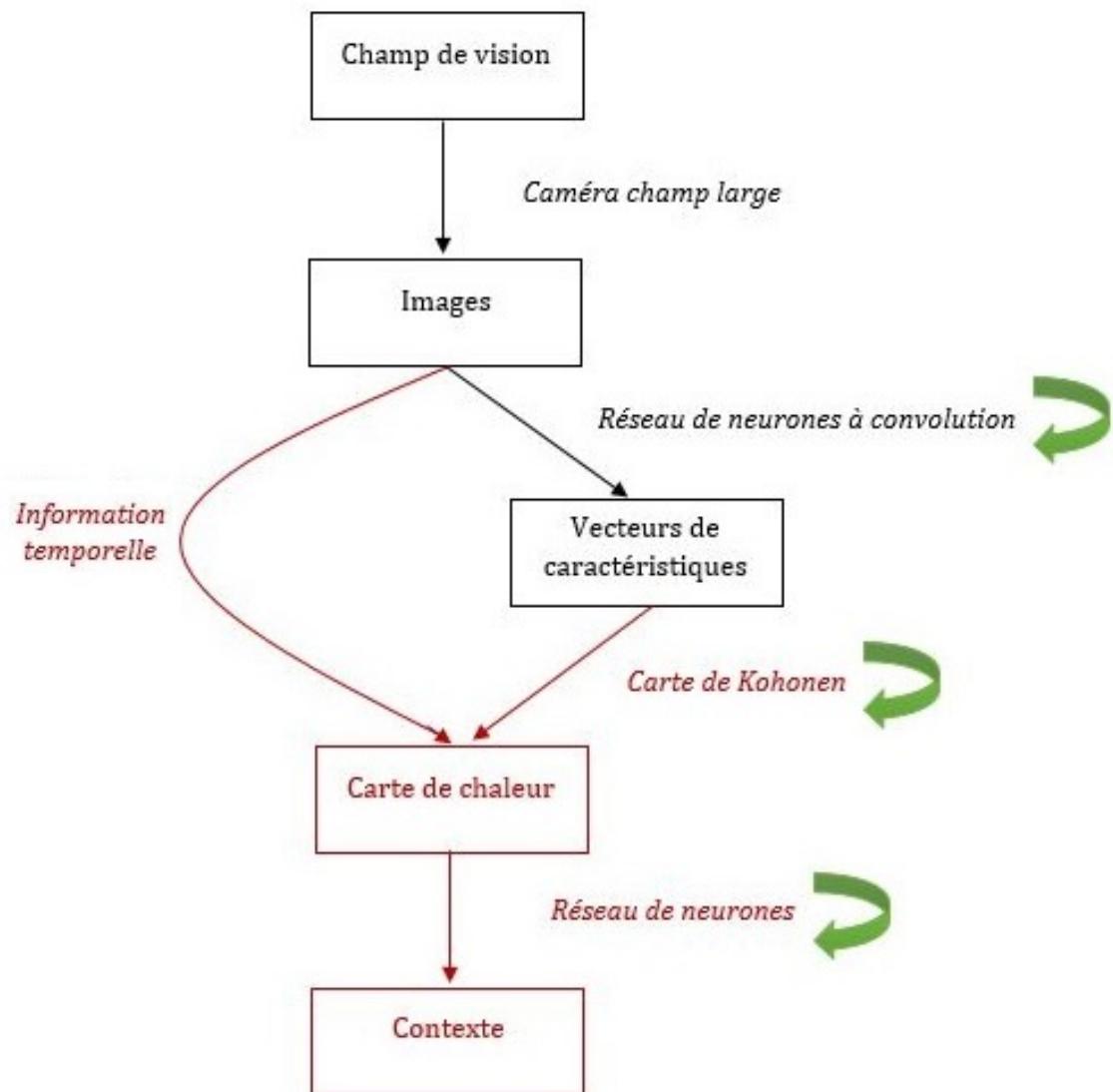


FIGURE 6 – Évolution de l'information

3.2 STRUCTURE FINALE DU PROJET ET COMMUNICATION AVEC LE ROBOT

3.2.1 • CAHIER DES CHARGES DU LIVRABLE FINAL

Le livrable de ce PSC devait répondre à un certain nombre de **contraintes techniques** :

- être compatible avec les éléments déjà implémentés dans le projet BioVision de Thales.
- être en mesure d'exécuter les algorithmes d'apprentissage et d'exploitation des données que nous avons choisi de façon efficace et dans un temps raisonnable.
- être modulaire et facile à relire via une interface claire.

Pour satisfaire à ces exigences, nous avons pris **plusieurs décisions importantes** :

- utilisation de **C++11** pour bénéficier au maximum des fonctionnalités offertes par ce langage (facilité de typage et de génération de l'aléatoire notamment)
- utilisation limitée des fonctionnalités offertes par la librairie **OpenCV** au profit d'une solution plus lisible fabriquée spécifiquement pour ce projet
- utilisation du firmware **ROS** pour structurer le projet et permettre l'interaction avec le robot TurtleBot
- décomposition du projet en deux composantes : une librairie implémentant les différents algorithmes et structures de données spécifiques au projet, et son exploitation dans un projet ROS exécutable sur le robot.

3.2.2 • DÉCOMPOSITION TECHNIQUE DU LIVRABLE

Le code du livrable final peut se décomposer en **deux parties** bien distinctes : la **librairie "psc_utils"** implémentant les différentes structures en mémoire ainsi que les différentes opérations pouvant être effectuées sur celle-ci, et le **livrable sous la forme d'un projet ROS**.

La librairie psc_utils

La librairie "psc_utils" implémente un template **raw_matrix<T>** pouvant contenir des tableaux multidimensionnels de types arbitraires et de tailles arbitraires affecté dynamiquement. Toutes les classes du projet héritent de **raw_matrix<float>** :

- la classe **Vecteur**, unidimensionnelle dont toute instance est destinée à contenir un vecteur de caractéristiques pouvant être aisément exploité par les différents algorithmes du projet.
- la classe **Matrice**, tridimensionnelle et de profondeur fixe (le nombre de couches du réseau de neurones convolutionnel) permettant le stockage en mémoire des données entrantes et celui de la carte de Kohonen.

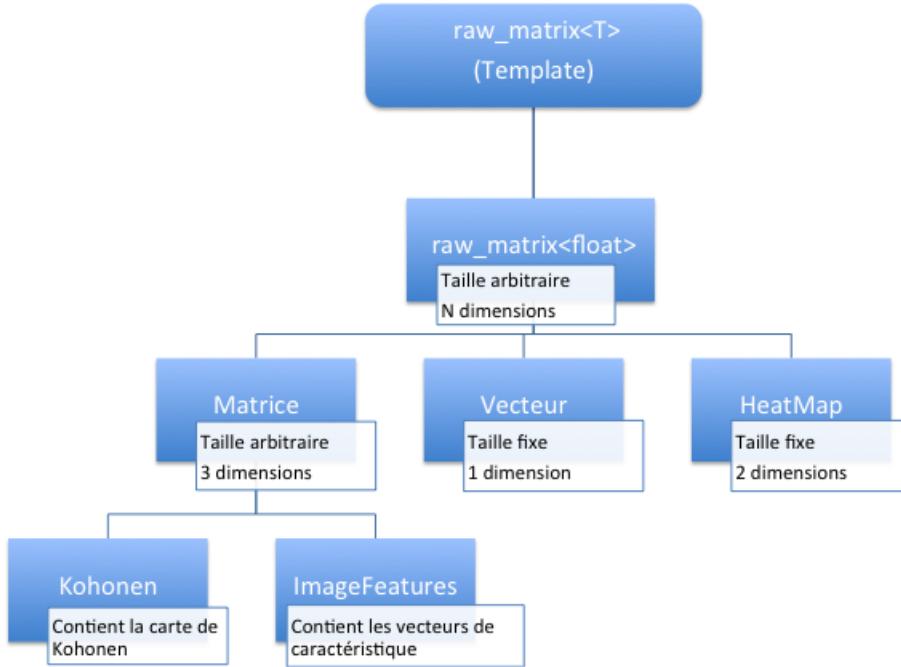


FIGURE 7 – Les différentes classes de la librairie

— La classe **HeatMap**, bidimensionnelle, permettant le stockage des cartes de chaleur.

De la classe **Matrice** héritent deux classes visant à implémenter spécifiquement ces deux usages :

- la classe **ImageFeatures** dans laquelle les caractéristiques (features) sont stockées et munies d'une interface facile.
- la classe **Kohonen** contenant les vecteurs composant la carte de Kohonen, munie de méthodes permettant notamment de faire évoluer la carte au cours du temps.

Nous avons encapsulé l'intégralité de la librairie "psc_utils" dans un namespace C++ pour plus de lisibilité et de modularité.

L'implémentation de ces différentes classes a été freinée par des problèmes liés à l'allocation dynamique de la mémoire. Les choix d'architecture que nous avons fait, même s'ils paraissent techniquement plus complexes que certaines solutions fournies de base dans C++, ont l'avantage d'une **complète modularité** ainsi que de permettre une plus grande clarté dans le code. Le détail des méthodes de chaque classe est précisé dans chacun des headers.

Les nœuds ROS

Afin de pouvoir interagir avec le robot TurtleBot, nous avons décidé d'utiliser le firmware **ROS**. Le fonctionnement de ce firmware fournit un cadre naturel au découpage du code correspondant à l'architecture conceptuelle du projet.

Le firmware ROS, avec lequel le livrable final est implémenté, est construit sur un principe de modularité stricte. Un programme ROS se décomposant en plusieurs sous-programmes que l'on nomme des **nœuds** (ou "node") lesquels interagissent entre-eux par l'intermédiaire de **flux de données** (que l'on nomme ordinairement des "topics"). Lorsqu'un nœud lit les données d'un "topic", on dit qu'il **souscrit** ; réciproquement, lorsqu'il produit un tel flux de données, on dit qu'il **publie** un "topic". Un même topic est accessible sans interférence par tous les nœuds du projet, ce qui permet l'exécution parallèle et interdépendante de multiples tâches de façon claire et aisée.

Le livrable final, conçu pour être exécuté en temps réel sur le robot se décompose en **6 nœuds** :

- **Kohonen_Learn_Node** : ce nœud souscrit au topic "deep_features_extractor", qui correspond au flux de vecteurs de caractéristiques obtenu après le passage par un réseau de neurones convolutionnel des images fournies par la caméra. Il traite les images (sous forme de vecteurs de caractéristiques) successivement pour faire évoluer la carte de Kohonen, laquelle est régulièrement sauvegardée dans un fichier sous forme de binaire(mémoire à long terme). Ce stockage permettant la permanence de la carte tout au long de l'apprentissage et de l'exploitation du robot.
- **Kohonen_Heat_Node** : ce nœud souscrit au topic "deep_features_extractor" pour activer les différents points de la carte de Kohonen (après avoir chargé celle-ci depuis un fichier binaire) à mesure que les vecteurs de caractéristiques se présentent. Il publie après avoir exploité chaque image une carte de chaleur actualisée via le topic "heatMap". Celle-ci est régulièrement sauvegardée de sorte à pouvoir être ré-exploitée (mémoire à court-terme).
- **Publish_Data_Node** : ce nœud publie à intervalle régulier des images sous forme de vecteurs de caractéristiques via le topic "deep_features_extractor" pour des finalités de test. Il permet de faire fonctionner le projet sur n'importe quel ordinateur en simulant l'envoi de données depuis les capteurs du robot.
- **Displayer_Node** : ce nœud permet l'affichage sous forme graphique des données publiées via le topic "heatMap".
- **Neural_Learner_Node** : ce nœud reçoit des cartes de chaleur via le topic "heatMap", associées à un label de contexte. Il fait évoluer les poids d'un réseau de neurones.
- **Neural_Classifier_Node** : ce nœud reçoit des cartes de chaleur via le topic "heatMap", sans label de contexte. Il renvoie la sortie (un label de contexte) associée à la carte de chaleur par le réseau de neurones.

Les deux schémas ci-après résument les implémentations réalisées, dans le mode apprentissage et dans le mode reconnaissance d'un contexte.

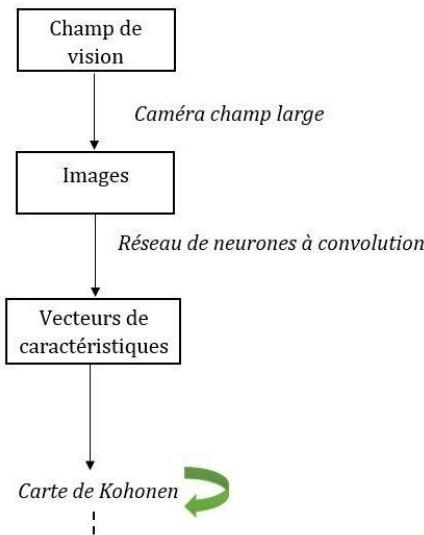
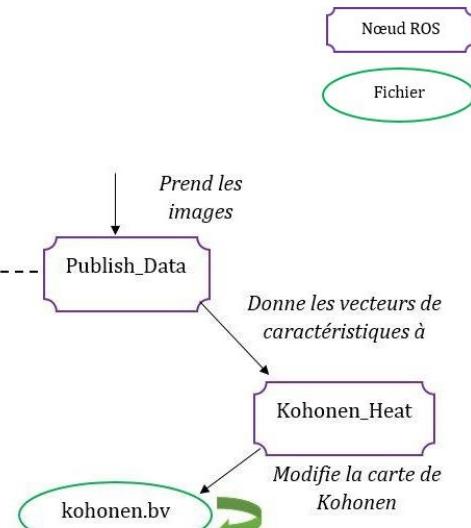
THEORIE**IMPLEMENTATION**

FIGURE 8 – Implémentation en mode apprentissage

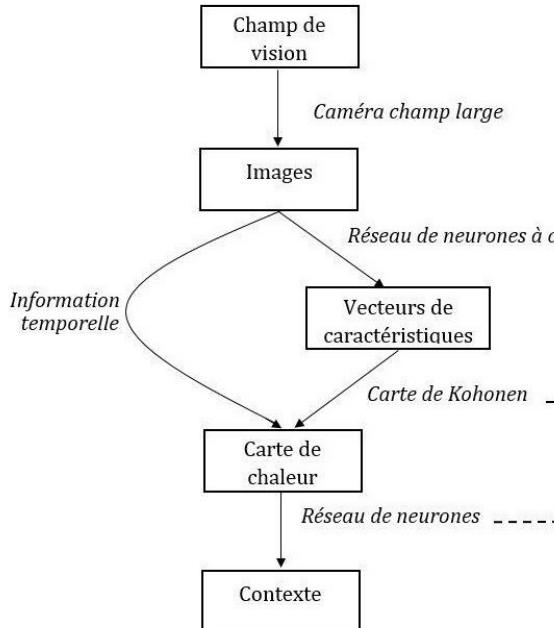
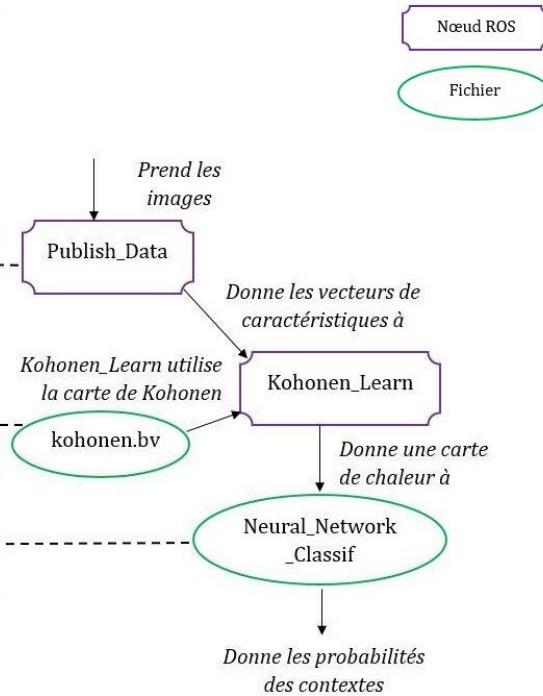
THEORIE**IMPLEMENTATION**

FIGURE 9 – Implémentation en mode reconnaissance

3.3 RÉSULTATS OBTENUS

Pour tester la pertinence de notre modèle, nous avons essayé notre plateforme sur des données fournies par Thales, correspondant à un **contexte de bureau** observé à l'ENSTA.

Nous avons choisi d'utiliser la **norme 2** comme distance $d_{Gestalt}$ dans l'espace des vecteurs de caractéristiques.

Dans un premier temps, la carte de Kohonen a été entraînée sur **200 images** différentes issues d'un même contexte.

Ensuite, **la carte est "chauffée"** sur chaque image que l'on veut analyser, à trois reprises pour la rendre plus contrastée.

Voici les **cartes de chaleur obtenues pour chacune des trois images** suivantes :

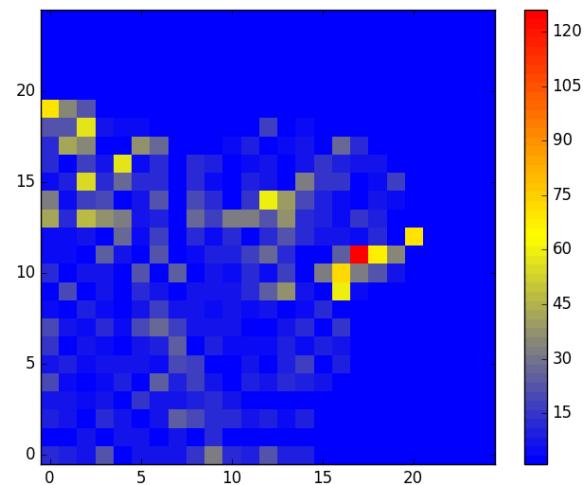


FIGURE 10 – Image et carte de chaleur correspondante, à $t = 1$

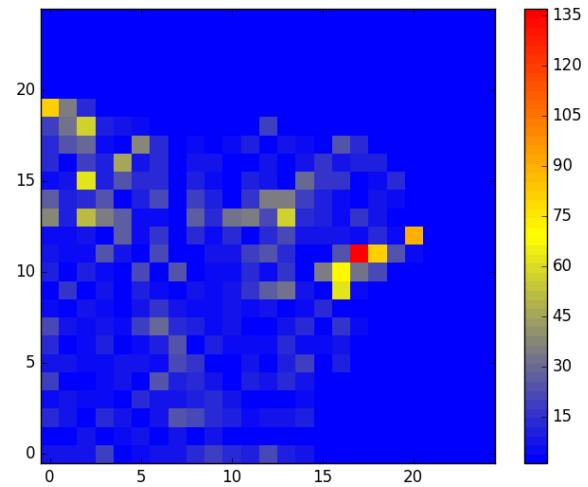


FIGURE 11 – Image et carte de chaleur correspondante, à $t = 6$

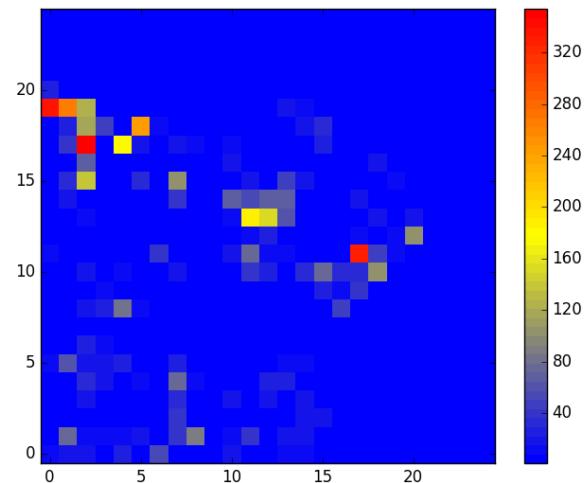


FIGURE 12 – Image et carte de chaleur correspondante, à $t = 250$

Nous pouvons constater que :

- Certaines zones des cartes sont plus activées que d'autres, les activations des différents points de la carte ne sont donc pas indépendants. Cela illustre la **cohérence locale** que nous voulions de la carte de Kohonen : des neurones proches de la carte se ressemblent, et sont donc activés dans des circonstances similaires.
- deux cartes issues d'images très similaires sont similaires,
- et la carte issue d'une image différente est différente.

Cela confirme que les cartes de chaleur telles que nous les avons implémentées permettent de représenter des images de façon cohérente, et que nos choix de fonctions de similarité sont également cohérents pour faire le lien entre des entités qui se ressemblent.

Néanmoins, comme nous ne disposons que des données sur un seul contexte, nous n'avons pas du faire apprendre notre système sur une multitude de contextes différents. Nous ne pouvons donc rien conclure pour le moment sur la pertinence du modèle des cartes de Kohonen pour la reconnaissance de contextes.

3.4 RÉFLEXIONS SUR LES APPORTS DU PROJET ET SES ÉVENTUELS PROLONGEMENTS

Ce projet nous a permis de produire un système théoriquement capable d'effectuer le premier des scénarios établis à l'origine. En effet, nous avons un système qui **classifie le contexte dans lequel il se trouve**, en s'appuyant sur des données visuelles.

Cela permet également de se rapprocher d'une solution pour le **deuxième scénario** (la détection d'anomalie). Une idée pour le résoudre avec ce qui a déjà été fait est de laisser le système évoluer un certain temps au sein d'une pièce, afin de récupérer la carte de chaleur "moyenne" qui en découle. Ensuite, il s'agit de comparer les vecteurs de caractéristiques entrant aux neurones de la carte de Kohonen, pour savoir quelle région de la carte de chaleur ils activent. Ils peuvent donc être classifiés comme normaux s'ils activent une zone chaude de la carte de chaleur, car cela signifie qu'ils sont similaires aux vecteurs rencontrés précédemment. A l'inverse, les vecteurs peuvent être classifiés comme étant des anomalies s'ils activent une zone froide de la carte de chaleur, car cela signifie qu'aucun vecteur similaire n'a été rencontré précédemment.

Le **troisième scénario**, néanmoins, est plus distant de ce que nous avons fait jusque là, bien qu'il utilise la reconnaissance de contexte. En particulier, celui-ci nécessite de conserver la notion d'objet, puisqu'il se base sur la recherche d'objets. Par ailleurs, la notion de contexte utilisée pour ce scénario est beaucoup plus précise, car il s'agit de déterminer des zones de recherche au sein même d'un lieu. Si nous voulions nous baser sur un système semblable à celui de ce projet, il faudrait qu'il soit capable de déterminer des contextes "locaux", tels que la surface d'une table, ou un placard.

De manière plus générale, une fois le système de reconnaissance de contexte opérationnel, il est possible d'**influencer la reconnaissance d'objets**. D'une part la connaissance du contexte dans lequel se trouvent les données permet de mieux les classifier. D'autre part, il est aussi possible de se servir directement de la carte de Kohonen, puisqu'elle est basée sur ces mêmes vecteurs de caractéristiques que l'on souhaite classifier. La localisation du vecteur dans

la carte est une information supplémentaire pour la reconnaissance d'objets.

Enfin, une idée importante de notre projet qui serait exploitable plus généralement est le lien que nous avons fait avec la **théorie de Gestalt**. Nous n'avons rencontré que très peu de références à cette théorie dans le domaine de la vision artificielle, alors qu'elle semble prometteuse. Il pourrait être pertinent d'implémenter des fonctions de similarité se basant sur chacune des 6 lois de la forme, afin de mieux distinguer les entités présentes sur une image.

CONCLUSION

Le projet que nous avons mené a consisté à **reproduire une partie des caractéristiques de la vision humaine sur une plateforme robotique**. Nous nous sommes inspirés pour cela de la théorie de Gestalt afin d'améliorer l'efficacité des dispositifs de reconnaissance d'images. Au cours du projet, nos objectifs se sont précisés pour se concentrer autour d'une problématique explicite : nous avons entrepris la **conception d'un système capable de reconnaître des contextes**. Nous avons proposé une méthode inédite de reconnaissance de contextes fondée sur l'utilisation d'une carte de Kohonen pour représenter les caractéristiques d'une image issues d'un réseau de neurones convolutionnel.

Au-delà de l'apport purement scientifique, ce projet aura été pour nous une **première expérience de gestion d'un projet en groupe sur le long terme**, aux objectifs ambitieux. Nous avons été confrontés à des problématiques de répartition efficace des tâches, de révisions des objectifs initiaux et de réarrangements du planning. Finalement, ce projet nous aura montré l'importance et la complexité des relations humaines dans le cadre d'un travail de groupe. Nous avons utilisé des méthodes de travail et de valorisation de la créativité qui se sont révélées très utiles, en particulier la méthode « Process Quality Management », développée par IBM, pour la prise de décisions en groupe.

Nos premiers résultats sont encourageants et la plateforme que nous avons développée permettra de juger la pertinence de notre modèle. Ce projet a vocation à être mis en application sur la plateforme BioVision développée par Thales Research & Technology.

RÉFÉRENCES

- [1] C. Craye, D. Filliat, and J. F. Goudou. Exploration strategies for incremental learning of object-based visual saliency. In *2015 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 13–18, August 2015.
- [2] Michael Van den Bergh, Xavier Boix, Gemma Roig, and Luc Van Gool. SEEDS : Superpixels Extracted via Energy-Driven Sampling. *arXiv :1309.3848 [cs]*, September 2013. arXiv : 1309.3848.
- [3] A. Torralba, K. P. Murphy, W. T. Freeman, and M. A. Rubin. Context-based vision system for place and object recognition. In *Ninth IEEE International Conference on Computer Vision, 2003. Proceedings*, pages 273–280 vol.1, October 2003.
- [4] Charles F. Cadieu, Ha Hong, Daniel L. K. Yamins, Nicolas Pinto, Diego Ardila, Ethan A. Solomon, Najib J. Majaj, and James J. DiCarlo. Deep Neural Networks Rival the Representation of Primate IT Cortex for Core Visual Object Recognition. *PLOS Comput Biol*, 10(12) :e1003963, December 2014.
- [5] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple Object Recognition with Visual Attention. *arXiv :1412.7755 [cs]*, December 2014. arXiv : 1412.7755.
- [6] Aude Oliva and Antonio Torralba. The role of context in object recognition. *Trends in Cognitive Sciences*, 11(12) :520–527, December 2007.
- [7] M. J. Choi, A. Torralba, and A. S. Willsky. A Tree-Based Context Model for Object Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(2) :240–252, February 2012.
- [8] Jurgen Schmidhuber. Deep learning in neural networks : An overview. *Neural Networks*, 61 :85–117, January 2015.
- [9] Song-Chun Zhu. Embedding Gestalt laws in Markov random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11) :1170–1187, November 1999.
- [10] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9) :1464–1480, September 1990.
- [11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.
- [12] Elizabeth S. Spelke. Principles of Object Perception. *Cognitive Science*, 14(1) :29–56, January 1990.
- [13] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.
- [14] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6) :679–698, 1986.

ANNEXES

ANNEXE A : PLATEFORME DE TESTS DE FONCTIONS DE SIMILARITÉ

Cette annexe présente le code Python utilisé pour les tests des fonctions de similarité, suivi des résultats des tests effectués sur deux mesures.

Ce code utilise le module "psc_utils" que nous avons développé pour ce projet afin de lancer des tests sur un certain nombre de fichiers.

```
# -- coding : utf-8 --
#File run_test.py : launches tests on Thales files

import json
from multiprocessing import Pool
import logging
import math

import psc_utils.tensor as tensor
import psc_utils.psc_yaml as psc_yaml

#global parameters:

path_to_yaml = "small_tensor/yaml/"
output_path = "output_test/json/"

def export_to_json(matrix, pathfile):
    try:
        logging.info("Dumping through JSON to {}".format(pathfile))
        with open(pathfile, 'w+') as fd:
            fd.write(json.dumps(matrix))
    except IOError:
        logging.info("The turtle failed to write it down in {}".format(pathfile))

def make_input_name(i):
    l = path_to_yaml + "sequence_0000"
    for x in range(int(2 - math.floor(math.log(i+1, 10))) :
```

```

        l += "0"
l += str(i) + ".png.yml"
return l

def make_output_name(d_name, i, j):
    return (output_path + "test_" + d_name + "{}x{}".format(i, j))

def cross_test(t, d, i, j):
    logging.debug('{}x{}'.format(i, j))
    return [[t.distance(d, i, x, y, j, x, y) for y in range(30)]
            for x in range(40)]

def init_num_test(n):
    input_paths = [make_input_name(i) for i in range(n)]
    t = tensor.Tensor()
    t.fillTensor(input_paths)
    return t

def run_test(t, d, d_name):
    n = t.t_size
    logging.info("Running test {}".format(d_name))
    for i in range(n):
        for j in range(n):
            m = cross_test(t, d, i, j)
            print(m)
            export_to_json(m, make_output_name(d_name, i, j))

def L2(x, y):
    res = 0
    for a, b in zip(x, y):
        res += (a - b)*(a - b)
    return math.sqrt(res)

def cosinus(x, y):
    res = 0
    nx = 0

```

```

ny = 0
for a, b in zip(x, y):
    res += a*b
    nx += a*a
    ny += b*b
nx = math.sqrt(nx)
ny = math.sqrt(ny)
res = res / (nx * ny)
return res

def aux_multiprocess(d):
    run_test(d[ 'tensor' ], d[ 'distance' ], d[ 'd_name' ])

if __name__ == "__main__":
    psc_yaml.init_test()

    input_paths = [make_input_name(i) for i in [1, 2, 97, 98]]
    t = tensor.Tensor()
    t.fillTensor(input_paths)
    # print(t.tensor)

    # t = init_num_test(48)

    logging.info("Tensor was successfully loaded")

    run_test(t, L2, 'L2')
    run_test(t, cosinus, 'cosinus')

p = Pool(2)
l = [{ 'tensor' : t, 'distance' : cosinus, 'd_name' : "cosinus" },
      { 'tensor' : t, 'distance' : L2, 'd_name' : "L2" }]
p.map(aux_multiprocess, l)

```

Nous reproduisons ci-après le code de la classe "Tensor" de "psc_utils". Celle-ci gère l'ouverture, la lecture et le parsing des fichiers.

```

import os
import logging
import math

```

```

from . import psc_yaml as psc_yaml

class Tensor:

    def __init__(self):
        self.tensor = []
        self.t_size = 0

    def fillTensor(self, list_path):
        self.tensor = [psc_yaml.open_file(x) for x in
                      list_path]
        self.t_size = len(self.tensor)

    def getMatrix(self, t):
        if (t < self.t_size) and (0 <= t):
            return self.tensor[t]
        else:
            logging.info("The turtle cannot recall that moment in time")
            return None

    def isEmpty(self):
        return (self.t_size == 0)

    def integrityCheck(self):
        """
        check if the content of the tensor is "coherent" :
        every matrix has the same x length
        """
        if self.isEmpty():
            return True
        else:
            res = True
            p = len(self.tensor[0])
            for x in self.tensor:
                res = res and (p == len(x))
                p = len(x)
            if not res:
                logging.info("The tensor is no good for mainstream use, be careful it may upset the turtle")
        return res

```

```

def distance(self, d, t_1, x_1, y_1, t_2, x_2, y_2):
    if self.integrityCheck():
        try:
            matrix_t_1 = self.getMatrix(t_1)
            matrix_t_2 = self.getMatrix(t_2)
            return d(matrix_t_1[x_1][y_1],
                     matrix_t_2[x_2][y_2])
        except IndexError:
            logging.info("Distance won't be computed, "
                         "inconsistencies detected")
            return None
    else:
        logging.info("Distance won't be computed, "
                     "inconsistencies detected")

if __name__ == "__main__":
    """
    tests
    """
    my_tensor = Tensor()
    my_tensor.fillTensor(1)
    my_tensor.distance(d, 1, 12, 12, 1, 13, 13)

```

Nous avons testé notre plateforme avec deux fonctions de similarité (la norme 2 et la similarité cosinus) sur plusieurs images. Nous calculons la distance entre tout vecteur de caractéristiques provenant d'une image en position (i, j) et le vecteur provenant de l'autre image à la même position. Nous utilisons trois images prises par le robot, aux instants $t = 1$, $t = 6$ et $t = 99$. Nous nous attendons donc à obtenir des similarités plus fortes entre les deux premières images qu'entre la première et la dernière.

Nos tests produisent des tableaux de floats de taille conséquente, ainsi nous ne reproduisons qu'un extrait des résultats. On ne s'intéresse dans un premier temps qu'aux ordres de grandeurs des valeurs obtenues.

Les résultats sont conformes à nos attentes. En effet, la distance norme 2 est plus grande entre les caractéristiques issues des images éloignées, et la similarité cosinus est plus grande entre les caractéristiques issues des images proches. Cela n'était pas totalement évident, car nous ne comparons pas directement des images, mais des vecteurs en sortie d'un réseau de neurones convolutionnel. Nous n'avions donc *a priori* aucune assurance que nos distances soient pertinentes et correspondent à l'intuition.



FIGURE 13 – Image prise à $t = 1$



FIGURE 14 – Image prise à $t = 6$



FIGURE 15 – Image prise à $t = 99$

```
[[18.394235242092275, 22.82849160415594, 17.25958517270923, 10.332315988531176, 5.2955525290833 ^  
562879, 13.507705208186144, 22.31194097927143, 32.129310792077106, 35.352992821157834, 32.28568  
766453438684, 56.552003152694596, 65.02393064472136, 48.12158573768043, 23.024611081527134, 14.  
51111424, 23.28200620013354, 20.55607738091788, 12.917581470439512, 43.46874183150119, 53.48964  
93, 21.744293478531596], [27.59779399193388, 30.27880506777764, 33.27498677768153, 34.33858594  
47.68852136366887, 44.70604151574973, 28.064803155248473, 14.8426547503539, 16.065216575069115,  
9296378, 14.223029384225592, 17.666360882763115, 18.405565726245854, 17.032083133215274, 12.632  
6392, 11.086472354320902, 9.403684587022454, 9.67685802259354, 11.706980028988802, 12.199189476  
08, 24.052180261666134, 26.81472783094568], [34.75284520836405, 45.783209436643084, 46.97367538  
2, 19.309375992865558, 19.984703264473115, 17.252092500485944, 18.64858275766166, 26.6834846591  
.35029453242551, 36.314793819776945, 31.357560920274622, 19.75673792515277, 10.935125398163244,  
.952952892045387, 16.335621877000495, 16.779394268525767, 18.434997091053244, 19.24733293222304  
10.827995146362657, 10.10414784015139, 10.458822237590153], [10.61873608369515, 13.096906146457  
34064, 7.78809825782129, 11.16903610677945, 11.429635208362827, 9.910348862101387, 8.0502569131  
, 33.07591517384838, 29.389826481198988, 19.29784034682899, 14.797179073537812, 14.539148946090  
51617496396974, 12.324985824510664, 11.747534339401811, 11.111167498939453, 12.817883950552442,  
48862138, 43.98107847932429, 42.052159160068065, 32.333461881057985], [19.945420549158992, 28.0  
32117, 20.773834145344374, 18.754980866833357, 18.065331668861404, 18.351918702744115, 18.60542  
2317, 17.163770128108602, 19.123627486129006, 25.46825095725115, 29.37991205733541, 27.99278215  
07, 49.739711671817105, 50.12775571146319, 62.00062967522278, 78.07251048133031, 87.63960727185  
94516072, 11.899875565565695, 11.197690167908245, 11.921012856622417, 11.817947844346962], [10.  
187826888, 12.173119511290812, 12.259228541585813, 19.555917550742194, 38.18761115383863, 59.38
```

< [] >

FIGURE 16 – Comparaison des caractéristiques issues des images à $t = 1$ et $t = 6$ pour la norme 2

```
[[84.3252534848412, 99.53241245323794, 62.72662168492992, 36.12735130714299, 32.77658417980039, ^  
88.39844945576716, 137.53593491153936, 210.53046965690348, 216.75786015048686, 167.931702186648  
6452217955, 378.0884058452676, 304.9215780901652, 205.43811961801387, 132.70447816529128, 81.16  
86, 128.35441851461118, 75.20322488997193, 177.30292161000213, 219.429563332812, 157.2725486275  
08.8993835592053, 231.89353424863376, 248.66985004895272, 234.00042314170162, 229.5851094055975  
5412803263, 190.39612070676867, 103.72581812448503, 128.68360652659487, 190.91131026961835, 279  
693590023, 195.0199960977367, 221.2139996035894, 222.17489412294873, 179.67213908823487, 128.79  
356, 109.62965462484338, 126.05273683824169, 122.7138051940478, 147.9136592489626, 176.69232432  
178.7492990632767], [265.5322632057118, 336.7435012223348, 295.3843069389312, 174.3159418491251  
1.6032530435397, 139.67079147419796, 162.8652822321686, 197.02011200516742, 229.8544980334167,  
93378151, 229.00239766581635, 201.44915758256477, 164.34780839138674, 156.19384712295417, 162.9  
465280304065, 171.5895831115131, 185.7022653040922, 196.58492949169766, 229.48818812966707, 263  
.60470950687582, 128.59200190938776, 136.91842603150315], [150.75648470554967, 142.356828953253  
40637885668613, 190.60885839727706, 281.42515109915473, 287.9957206872196, 261.04199443420066,  
56321586581942, 232.35772625491578, 218.00888975805577, 133.87557880897108, 62.912726305597, 54  
2199878854943, 117.98930161041532, 119.172225393691, 123.86542250804213, 137.46326853411531, 1  
, 223.52492019508966, 194.9020533182287, 125.16906226820872], [70.2156177123351, 104.147793631  
92228, 154.74730700493828, 168.2509157312984, 177.16889257806093, 178.6431939039164, 170.219331  
167.7853499558259, 224.88704239799324, 238.9705069550711, 222.92314313135986, 200.5881596613496  
7783, 278.5683161769592, 238.0799898103727, 195.30864825476027, 193.49486357271124, 189.1893167  
9114959868, 48.204501863046275, 40.96200656453265, 38.50387507339901], [39.8622379915868, 43.26  
, 85.91386887455927, 175.87214424343478, 318.5890096574754, 382.84641170000344, 307.75064206057
```

< [] >

FIGURE 17 – Comparaison des caractéristiques issues des images à $t = 1$ et $t = 99$ pour la norme 2

```
[[0.9877644757845828, 0.9889736427134062, 0.9883023140262206, 0.9913526662803103, 0.99758430515 ^  
22618030645, 0.9986154642773634, 0.9971224852970552, 0.9924574612882805, 0.9889523071928222, 0.  
327, 0.990786177584723, 0.9891260217879184, 0.9874999360547353, 0.9875411034208509, 0.98949561  
, 0.9637496497816389, 0.9594310108170627, 0.964645326399799, 0.9705794385846352, 0.977742706979  
05317412, 0.9915145888674591, 0.9950453915654597, 0.9954896189470539, 0.9936524319314864], [0.9  
0638, 0.9682164902706877, 0.9557107260758764, 0.9599522565942797, 0.9733726847737475, 0.9798588  
.9958990219068237, 0.9950870544602807, 0.9932313613863376, 0.9842029781029404, 0.97251781790396  
664477476642], [0.9898981695286884, 0.9906016319089195, 0.9901181384685966, 0.9908443147940755,  
30780314, 0.9977264818517498, 0.9975439618597365, 0.9976346507821808, 0.9975358045362781, 0.995  
933, 0.9931770976274207, 0.9935901659510923, 0.9911807804341621, 0.9743393271581116, 0.97639083  
, 0.984781968685486, 0.9804823779856542, 0.9759155059716093, 0.974386828556624, 0.961480528066  
649102873709, 0.9946877367104359, 0.9958846582057262, 0.9956168610734973, 0.994587184005972, 0.  
91144, 0.9970918807584047, 0.997124532440271, 0.9967535339470847, 0.9911227624201704, 0.9840930  
0.9828220743725934, 0.9894979619443646, 0.9899474732712663, 0.9895740898464161, 0.9905991333831  
7773468, 0.9529692288476085], [0.9786704423355651, 0.9760132471810214, 0.9606013395013236, 0.96  
837235781420194, 0.9872572803030529, 0.9904137052244012, 0.9900307658645978, 0.9842926369392524  
8015430190412, 0.9961370890589397, 0.9942066059931016, 0.9913776213604854, 0.9902880267512322,  
80984147, 0.9623581068058752, 0.9419402941960527, 0.9353509555460473, 0.9349086125367259, 0.932  
46436, 0.9874678518233859, 0.9911696607739106, 0.991674047195654, 0.9874243148020642, 0.9855095  
, 0.9881320214036678, 0.9861383212384884, 0.9759197098588035, 0.9611240583454399, 0.96184498194  
8609231960732, 0.9852943937439872, 0.9894323938375902, 0.987754696409057, 0.9849203535184463, 0  
8572317, 0.9178656690695548, 0.8943357380201873], [0.9856612478937431, 0.9863218429230532, 0.98  
< ^ >]
```

FIGURE 18 – Comparaison des caractéristiques issues des images à $t = 1$ et $t = 6$ pour la similarité cosinus

```
[[0.7079912648984446, 0.7635972435984036, 0.835340172423371, 0.9147544167499779, 0.936159964710 ^  
0.33202543803823176, 0.47784933972747407, 0.862026166485477, 0.6524095770844611, 0.363375157418  
, 0.532187290056713, 0.5958606511025561, 0.3800522541708323, 0.1630762176997483, 0.102476582701  
83420127387, 0.644022176147734, 0.6396594206884363, 0.6124144341907443, 0.46915178300213956, 0.  
6147397, 0.5393191590721005, 0.5549045741248979, 0.6549879032161403, 0.7024686022939433, 0.6157  
597208578000757, 0.3971527155951776, 0.33303849447666406, 0.32824819099185765, 0.27455424071916  
40550602169949, 0.527167864785367, 0.45108579438239793, 0.3089211159439716, 0.20215838235364403  
9557434003654632, 0.019295238745533055, -0.021942518222756516, 0.05069174239980045, 0.173740078  
910762, 0.31240825438984066, 0.3628482313626, 0.42422900125419427, 0.5358769197009241, 0.538982  
19, 0.615405812360958, 0.6247675690211267, 0.5055481869478974, 0.4914981431366642, 0.4699221137  
0.646911701182732, 0.5154180428298893, 0.4331412281589755, 0.501993132571505, 0.529037534066160  
, 0.09753602982504583, 0.08781114686529937, 0.08307147266373331, 0.057303259083978784, 0.061591  
963721, 0.20774237329123427, 0.19859507552869593, 0.20856832997780897, 0.20898729097203517, 0.2  
.09003508255120825, -0.04157570068532822, 0.3016041867772227, 0.7802748470642094, 0.83127217226  
565554684878, 0.173666604588834, 0.19314371768407038, 0.19160963388416807, 0.21286527276764333,  
9440031, 0.11629355956689266, 0.13681330857245227, 0.09728389612334207, 0.08223141135691239, 0.  
023240496649, 0.5940391421867555, 0.5304625107459567, 0.5024976279815316, 0.4799100654578186, 0  
0006886316785628, 0.09251125457053751, 0.09976722838268014, 0.1274469157893887, 0.1664897356318  
1251194687], [0.1879681548895721, 0.13187198104243425, 0.09045476536389234, 0.17868086936860203  
.08180683897263001, 0.0899884675841606, 0.012712031927215147, -0.023154523877452333, 0.07904415  
0.41428263306659024, 0.2654883759782938, 0.7979667346255613, 0.7876658326851392, 0.713425581638  
9474279, 0.8691802323751584], [0.8616534083596877, 0.8372674951018199, 0.8083096884607571, 0.77  
< ^ >]
```

FIGURE 19 – Comparaison des caractéristiques issues des images à $t = 1$ et $t = 99$ pour la similarité cosinus

ANNEXE B : LIBRAIRIE PSC_UTILS

La taille du code étant conséquente, nous reproduisons seulement les 7 headers.

matrice.hpp

```
#pragma once
#include "psc_utils.hpp"
#include <cstdio>
#include <string>

namespace psc_utils
{
template<typename T>
class raw_matrix
{
protected:
    T* data;
    int number_of_fields;
    int* sizes;
    int total_size;
public:
    raw_matrix();
    raw_matrix( const raw_matrix<T>& other );
    raw_matrix( int n, int s[], const T& d_val );
    raw_matrix( int s_x, int s_y, int s_z, const T& d_val )
        ;
    raw_matrix( int s_x, int s_y, const T& d_val );
    raw_matrix( int s, const T& d_val );
    raw_matrix( const std::string& namefile );
    raw_matrix<T>& operator= ( const raw_matrix<T>& other );
    ~raw_matrix();
    T get_value( int coor[] ) const;
    void set_value( int coor[], const T& value );
    void save_in_file( const std::string& namefile ) const
        ;
    void save_in_file( FILE* fs ) const;
};

class Vecteur : public raw_matrix<float>
{
private:
```

```

        Point position;
        int timestamp;
public:
    Vecteur( Point p, float d_value, int t);
    Vecteur( float* c, Point p, int t);
    ~Vecteur();
    Vecteur( const Vecteur& other);
    void print() const;
    float get_value( int k) const;
    void set_value( int coor, float value);
    Point getPosition() const;
    int getTime() const;
    Vecteur difference( const Vecteur& other);
    float distance( int numDistance, const Vecteur& other)
        const;
    Vecteur operator*( float c) const;
    Vecteur operator+(const Vecteur& other) const;
    Vecteur operator-(const Vecteur& other) const;
    float norm1( const Vecteur& b) const;
    float norm2( const Vecteur& b) const;
    float dist_cos( const Vecteur& b) const;
    float dist_temp( const Vecteur& b) const;
};

class Matrice : public raw_matrix<float>
{
private:
    int timestamp;
public:
    Matrice( int size_x, int size_y, int t, float value);
    Matrice( const std::string& namefile, int t);
    ~Matrice();
    int get_timestamp() const;
    float get_value( int i, int j, int k) const;
    void set_value( int i, int j, int k, float value);
    Vecteur getVector( int i, int j) const;
    void replaceVector( int i, int j, const Vecteur& v);
    void resetTimestamp();
    void save_in_file( const std::string& namefile) const
        ;
    void save_in_file( FILE* fs) const;
};


```

```

}

psc_utils.hpp

#pragma once
#include <list>

namespace psc_utils
{
    class Point
    {
        private:
            int x; // this fields are protected
            int y;
        public:
            Point(int X, int Y);
            Point();
            ~Point();
            Point operator+(const Point& other) const;
            Point operator-(const Point& other) const;
            int getY() const;
            int getX() const;
            std::list<Point> voisins(int r, int largest_x, int largest_y);
    };
}

imagefeatures.hpp

#pragma once
#include "matrice.hpp"
#include <opencv2/opencv.hpp>
namespace psc_utils
{
    class ImageFeatures : public Matrice
    {
        private:
        public:
            ImageFeatures(const std::string& namefile, int time)
                ;
            ImageFeatures(cv::Mat mat, int time);
            ~ImageFeatures();
    };
}

distance.hpp

```

```

tan #include <cmath>
#include "matrice.hpp"
namespace psc_utils
{
    float norm1( const Vecteur& b) const;
    float norm2( const Vecteur& b) const;
    float dist_cos( const Vecteur& b) const;
    float dist_temp( const Vecteur& b) const;
}

kohonen.hpp

#pragma once
#include "psc_utils.hpp"
#include "matrice.hpp"

namespace psc_utils
{
class Kohonen : public Matrice
{
private:
    int typeDistance;
    float alpha;
    float voisinage;
public:
    Kohonen( int numDis, float alphaInit , float r , int timestamp )
        ;
    Kohonen( const std::string& namefile , int timestamp );
    Kohonen();
    ~Kohonen();
    Point plusProchePoint( const Vecteur& v) const;
    void randomInit();
    void modifierVoisin( Vecteur v, Point pointProche );
    void changeParam( float deltaAlpha , float deltaR );
    void evolution( Vecteur v, float deltaAlpha , float deltaR );
    void resetMap();
    void resetTempsMap();
    void save( const std::string& namefile ) const;
    // std::list<Vecteur> voisins( Point& p, int largest_x , int
    largest_y );
};

heatmap.hpp

#pragma once

```

```
#include "psc_utils.hpp"
#include "kohonen.hpp"
#include <string>

namespace psc_utils
{
    class HeatMap : raw_matrix<float>
    {
        public:
            float cooling_rate;
            HeatMap( float c_rate );
            HeatMap();
            HeatMap( const std::string& n, float rate );
            ~HeatMap();
            float get_value( int i, int j ) const;
            void set_value( int i, int j, float v );
            void heat( Vecteur v, const Kohonen & khn );
            void cool( float taux );
            void save( const std::string& namefile ) const;
    };
}
```

macro.hpp Ce fichier permet de modifier tous les paramètres du modèle.

```
#pragma once
#define N_CARAC 512
#define MAX_X 25
#define MAX_Y 25
#define FEATURES_SIZE_X 30
#define FEATURES_SIZE_Y 40
#define DELTA_ALPHA 0.1
#define DELTA_R 0.1
#define ALPHA_INIT 1.0
#define R_INIT 2.0
#define TAUX 0.2
#define LEARNING_RATE 0.1
#define NB_CONTEXTS 3
#define PARAM_DISTRIBUTION_MEAN 0.0
#define PARAM_DISTRIBUTION_SIGMA 0.000005
```

ANNEXE C : NŒUDS ROS

Voici les 6 nœuds ROS que nous avons développés pour ce projet.

Publish_Data_Node.cpp

```
#include "ros/ros.h"
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <string>
#include <list>
#include <chrono>
#include <thread>
#include <sstream>

std::string dirname = "/home/john/data_test/";
std::string namefiles [] = {"sequence_0000001.png.yml",
                           "sequence_0000002.png.yml",
                           "sequence_0000003.png.yml",
                           "sequence_0000004.png.yml",
                           "sequence_0000005.png.yml",
                           "sequence_0000006.png.yml",
                           "sequence_0000007.png.yml",
                           "sequence_0000008.png.yml",
                           "sequence_0000009.png.yml",
                           "sequence_0000010.png.yml"};
};

cv::Mat get_image(const std::string& namefile)
{
    std::cout << "Checkpoint 2" << std::endl;
    cv::FileStorage fs(namefile, cv::FileStorage::READ);
    cv::Mat features;
    fs["deep_features"] >> features;
    return features;
}

int main(int argc, char **argv)
{
```

```

ros::init(argc, argv, "image_publisher");
std::cout << "Checkpoint_0" << std::endl;

ros::NodeHandle n;
image_transport::ImageTransport it(n);
std::cout << "Checkpoint_1" << std::endl;

image_transport::Publisher pub = it.advertise(
    "deep_feature_extractor", 100);
for(int i = 0; i < 2 ; i++)
{
    for(int j = 0; j < 200 ; j++)
    {
        std::string namefile = dirname + namefiles[j];
        int count = 0;
        std::cout << "-----About to publish:" << namefile << "\n"
              << std::endl;
        cv::Mat mat = get_image(namefile);
        sensor_msgs::ImagePtr msg = cv_bridge::CvImage(std_msgs::Header(),
                                                       "bgr8", mat).toImageMsg();
        pub.publish(msg);

        auto start = std::chrono::high_resolution_clock::now();
        std::chrono::duration<int>, std::ratio<1, 100000000>>
            sleep_duration(1000000000);
        std::this_thread::sleep_for(sleep_duration);
        auto end = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double>, std::milli> elapsed = end-start;
        std::cout << "Waited" << elapsed.count() << " ms\n";
    }
}
ros::spin();
return 0;
}

```

Kohonen_Learn_Node.cpp

```

#include "ros/ros.h"
#include "sensor_msgs/Image.h"
#include <cv_bridge/cv_bridge.h>
#include "macro.hpp"
#include "kohonen.hpp"
#include "psc_utils.hpp"
#include "heatmap.hpp"
#include "imagefeatures.hpp"

```

```
#include <fstream>
#include <iostream>

using namespace psc_utils;

int count = 0;
Kohonen map;
cv_bridge::CvImageConstPtr cv_ptr;
float deltaAlpha = 0.1;
float deltaR = 0.1;

void featureCallback( const sensor_msgs::Image::ConstPtr& msg)
{
    std::cout << "Checkpoint 6.1" << std::endl;
    cv_ptr = cv_bridge::toCvShare(msg);
    std::cout << "Checkpoint 6.2" << std::endl;

    ImageFeatures image(cv_ptr->image, cv_ptr->header.stamp.sec)
    ;
    std::cout << "Checkpoint 6.3" << std::endl;

    for ( int i = 0; i < FEATURES_SIZE_X; i++)
    {
        for ( int j = 0; j < FEATURES_SIZE_Y;
j++)
        {
            Vecteur v = image.getVector(
                i , j );
            map.evolution(v, deltaAlpha ,
deltaR);
        }
    }

    map.changeParam(DELTA_ALPHA, DELTA_R);
    std::cout << "Checkpoint 7" << std::endl;
    count++;
    if (count%8 == 0)
    {
        map.save( "myKohonenMap" );
        std::cout << "Checkpoint 8" << std::endl;
    }
}
```

```

int main( int argc , char **argv )
{
    std :: ifstream i file( "myKohonenMap . bv" );
    std :: cout << "Checkpoint_1" << std :: endl ;
    if ( i file )
    {
        Kohonen map( "myKohonenMap . bv" , 0 );
        std :: cout << "Checkpoint_2.1" << std :: endl ;

    }
    else
    {
        Kohonen map( 2 , ALPHA_INIT , R_INIT , 0 );
        map . randomInit();
        // map . remplirCarteRand();
        std :: cout << "Checkpoint_2.2" << std :: endl ;
    }
    ros :: init( argc , argv , "KohonenLearner _ Listener" );
    std :: cout << "Checkpoint_3" << std :: endl ;
    ros :: NodeHandle n;
    std :: cout << "Checkpoint_4" << std :: endl ;
    ros :: Subscriber sub = n . subscribe( "deep _ feature _ extractor" ,
        1 , featureCallback );
    std :: cout << "Checkpoint_5" << std :: endl ;
    ros :: spin();
    return 0;
}

```

Kohonen_Heat_Node.cpp

```

#include "ros / ros . h"
#include "std _ msgs / Float64MultiArray . h"
#include "sensor _ msgs / Image . h"
#include <cv _ bridge / cv _ bridge . h>
#include <fstream>
#include "macro . hpp"
#include "kohonen . hpp"
#include "psc _ utils . hpp"
#include "heatmap . hpp"
#include "imagefeatures . hpp"

using namespace psc _ utils;

int count = 0;

```

```

Kohonen map;
HeatMap heatmap;
cv_bridge::CvImageConstPtr cv_ptr;
ros::Publisher kohonen_pub;

void featureCallback( const sensor_msgs::Image::ConstPtr& msg)
{
    std::cout << "Checkpoint 4" << std::endl;
    cv_ptr = cv_bridge::toCvShare(msg);
    ImageFeatures image(cv_ptr->image, cv_ptr->header.stamp.sec);
    std::cout << "Checkpoint 5" << std::endl;

    for (int i = 0; i < FEATURES_SIZE_X; i++)
    {
        for (int j = 0; j < FEATURES_SIZE_Y; j++)
        {
            Vecteur v = image.getVector(i, j);
            heatmap.heat(v, map);
        }
    }

    heatmap.cool(TAUX);
    std::cout << "Checkpoint 6" << std::endl;

    std_msgs::Float64MultiArray dat;

    dat.layout.dim.push_back(std_msgs::MultiArrayDimension());
    dat.layout.dim.push_back(std_msgs::MultiArrayDimension());
    dat.layout.dim[0].size = MAX_X;
    dat.layout.dim[1].size = MAX_Y;
    dat.layout.data_offset = 0;

    std::cout << "Checkpoint 7" << std::endl;

    for (int i = 0; i < MAX_X; i++)
    {
        for (int j = 0; j < MAX_Y; j++)
        {
            std::cout << "Checkpoint 7.1" << std::endl;
            std::cout << i << ":" << j << std::endl;
            std::cout << "Checkpoint 7.2" << std::endl;
            dat.data.push_back(heatmap.get_value(i, j));
        }
    }
}

```

```

}

kohonen_pub.publish(dat);
std::cout << "Checkpoint8" << std::endl;

count++;
if (count%15 == 0)
{
    count = 0;
    heatmap.save("MyHeatMap");
}
}

int main(int argc, char **argv)
{
    std::ifstream ifile1("myKohonenMap.bv", std::ifstream::in);
    std::cout << "Checkpoint1" << std::endl;

    if (ifile1)
    {
        Kohonen map("myKohonenMap.bv", 0);
        std::ifstream ifile2("myHeatMap.bv", std::ifstream::in);
        std::cout << "Checkpoint2" << std::endl;

        if (ifile2)
        {
            HeatMap heatmap("MyHeatMap.bv", TAUX);
        }
        else
        {
            HeatMap heatmap();
        }

        std::cout << "Checkpoint2" << std::endl;
    }

    ros::init(argc, argv, "KohonenRecog_Listener");

    ros::NodeHandle n;

    ros::Subscriber sub = n.subscribe("deep_feature_extractor", 1,
                                    featureCallback);
}

```

```

kohonen_pub = n.advertise<std_msgs::Float64MultiArray>("heat_map", 1);
std::cout << "Checkpoint_3" << std::endl;

    ros::spin();
}
return 0;
}

Neural_Learner_Node.cpp

#include "ros/ros.h"
#include "std_msgs/Float64MultiArray.h"
#include "macro.hpp"
#include "kohonen.hpp"
#include "psc_utils.hpp"
#include "heatmap.hpp"
#include "imagefeatures.hpp"
#include <iostream>
#include <vector>
#include <cmath>
#include <math.h>
#include "opennn/opennn.h"
using namespace OpenNN;

NeuralNetwork network;
int count = 0;

void heatmapCallback(const std_msgs::Float64MultiArray::ConstPtr&
msg)
{
    std::cout << "Checkpoint_4" << std::endl;
    Matrix<double> inputs;
    std::vector< std::vector<float> > input_data(MAX_X);

    for (int i = 0; i < MAX_X; i++)
    {
        for (int j = 0; j < MAX_Y; j++)
        {
            std::cout << "Checkpoint_4.1" << std::endl;
            input_data.at(i).push_back(msg->data[i*MAX_Y + j]);
        }
    }
    std::cout << "Checkpoint_5" << std::endl;
}

```

```

        for (int i = 0; i < MAX_X; i++)
    {
        std::cout << "Checkpoint_5.1" << std::endl;
        Vector<double> row(input_data.at(i).begin(),
                             input_data.at(i).end());
        inputs.set_row(i, row);
    }

    std::cout << "Checkpoint_6" << std::endl;
    Vector<double> input_vector = inputs.to_vector();
    std::cout << "Checkpoint_6.1" << std::endl;
    Vector<double> param = network.arrange_parameters();
    std::cout << "Checkpoint_7" << std::endl;
    param += modif_vector * LEARNING_RATE;
    network.set_parameters(param);
    std::cout << "Checkpoint_8" << std::endl;

    if (count%15 == 0)
    {
        network.save("myNN.xml");
        count++;
    }
}

int main(int argc, char **argv)
{
    std::ifstream file1("myNN.xml");
    std::string s = "myNN.xml";
    std::cout << "Checkpoint_1" << std::endl;

    if (file1)
    {
        std::cout << "Checkpoint_1.1" << std::endl;
        NeuralNetwork network(s);

        std::cout << "Checkpoint_1.1.1" << std::endl;
    }

    else
    {
        std::cout << "Checkpoint_1.2" << std::endl;
        NeuralNetwork network(FEATURES_SIZE_X * FEATURES_SIZE_Y,
                              FEATURES_SIZE_X * FEATURES_SIZE_Y, NB_CONTEXTS);
    }
}

```

```

// Add a probabilistic layer at the end of the network
network.construct_probabilistic_layer();
ProbabilisticLayer* probabilistic_layer_pointer = network.
    get_probabilistic_layer_pointer();
probabilistic_layer_pointer->set_probabilistic_method(
    ProbabilisticLayer::Probability);
std::cout << "Checkpoint 1.2.1" << std::endl;
}

std::cout << "Checkpoint 2" << std::endl;
ros::init(argc, argv, "NeuralNetworkLearner_Listener");
ros::NodeHandle n;
ros::Subscriber sub = n.subscribe("heat_map", 1,
    heatmapCallback);
std::cout << "Checkpoint 3" << std::endl;
ros::spin();
return 0;
}

```

Neural_Classifier_Node.cpp

```

#include "ros/ros.h"
#include "std_msgs/Float64MultiArray.h"
#include "macro.hpp"
#include "kohonen.hpp"
#include "psc_utils.hpp"
#include "heatmap.hpp"
#include <vector>
#include <fstream>
#include <math.h>
#include <cmath>
#include "opennn/opennn.h"
using namespace OpenNN;

NeuralNetwork network;
int count = 0;

ros::NodeHandle n;
ros::Subscriber sub;
ros::Publisher neuralNetwork_pub;

void heatmapCallback(const std_msgs::Float64MultiArray::ConstPtr&
    msg)
{
    Matrix<double> inputs;

```

```

std :: vector< std :: vector<float> > input_data(MAX_X);

for (int i = 0; i < MAX_X; i++)
{
    for (int j = 0; j < MAX_Y; j++)
    {
        input_data .at( i ) .push_back( msg->data [ i *MAX_Y + j ] );
    }
}

for (int i = 0; i < MAX_X; i++)
{
    Vector<double> row( input_data .at( i ) .begin () , input_data .at( i ) .
        end () );
    inputs .set_row( i , row );
}

Vector<double> input_vector = inputs .to_vector ();
Vector<double> outputs = network .calculate_outputs (input_vector);
std :: vector<float> output_data( outputs .begin () , outputs .end ()) ;

std_msgs :: Float64MultiArray dat;

dat .layout .dim .push_back( std_msgs :: MultiArrayDimension () );
dat .layout .dim [ 0 ] .label = "Contexts ";
dat .layout .dim [ 0 ] .size = NB_CONTEXTS;
dat .layout .data_offset = 0;

for (int i = 0; i < NB_CONTEXTS; i++)
{
    dat .data [ i ] = output_data [ i ];
}
neuralNetwork_pub .publish (dat );
count++;
if ( count %15 == 0 )
{
    count = 0;
    network .save ( "myNN. txt " );
}
}

int main( int argc , char **argv )

```

```
{
    std::ifstream file1( "myNN.xml" );
    if (file1)
    {
        std::string s = "myNN.txt";
        NeuralNetwork network(s);
        ros::init(argc, argv, "NeuralNetworkClassifier_Listener");
        ros::NodeHandle n;
        ros::Subscriber sub = n.subscribe("heat_map", 1, heatmapCallback);
        ros::Publisher neuralNetwork_pub = n.advertise<std_msgs::
            Float64MultiArray>("classification", 1);
        ros::spin();
    }
    return 0;
}
```

Displayer_Node.py

```
#!/usr/bin/env python
import roslib
roslib.load_manifest('std_msgs')
import numpy
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
import rospy
import std_msgs.msg
from std_msgs.msg import String
from std_msgs.msg import Float64MultiArray

class MultiArrayConverter():
    ## Take a Float64 MultiArray message, convert it into a list of
    ## numpy matrices
    def multiArrayToMatrixList(self, ma_msg):
        dim = len(ma_msg.layout.dim)
        offset = ma_msg.layout.data_offset

        if (ma_msg.layout.dim[0].label != "row"):
            print ("Error: dim[1] should be the rows")
        rows = ma_msg.layout.dim[0].size

        if (ma_msg.layout.dim[1].label != "column"):
            print ("Error: dim[2] should be the columns")
        columns = ma_msg.layout.dim[1].size
```

```

mat = numpy.matrix(numpy.empty([rows,columns]))
mat.fill(numpy.nan)

for j in range(0, rows):
    for k in range(0, columns):
        data_index = ma_msg.layout.data_offset + (columns) * j + k
        mat[j,k] = ma_msg.data[data_index]
return mat

def heat_callback(data):
    rospy.loginfo(rospy.get_caller_id() + 'I heard %s', data.data)
    msg=MultiArrayConverter.multiArrayToMatrixList(converter, data)
    print(msg)
    Z = msg
    colors = [(0, 0, 1), (1, 1, 0), (1, 0, 0)] # B->Y->R
    cmap_name = 'my_list'
    cm = LinearSegmentedColormap.from_list(cmap_name, colors, N=50)
    im = plt.imshow(Z, interpolation='nearest', origin='lower', cmap=cm)
    plt.colorbar(im)
    plt.savefig('heat.png')
    plt.show(im)

def class_callback(data):
    rospy.loginfo(rospy.get_caller_id() + 'I heard %s', data.data)
    P=data.data
    lbls=[ "context%s" %i for i in range(len(P))]
    labels=[lbls[i]+": "+str(P[i]) for i in range(len(P))]
    plt.pie(P, labels=labels)
    plt.title('Contexts probabilities')
    plt.savefig('probas.png')
    plt.show()

def listener():
    rospy.init_node('display', anonymous=True)
    rospy.Subscriber('heat_map', Float64MultiArray, heat_callback)
    rospy.Subscriber('classification', Float64MultiArray,
                     class_callback)
    rospy.spin()

if __name__ == '__main__':
    converter=MultiArrayConverter()
    listener()

```

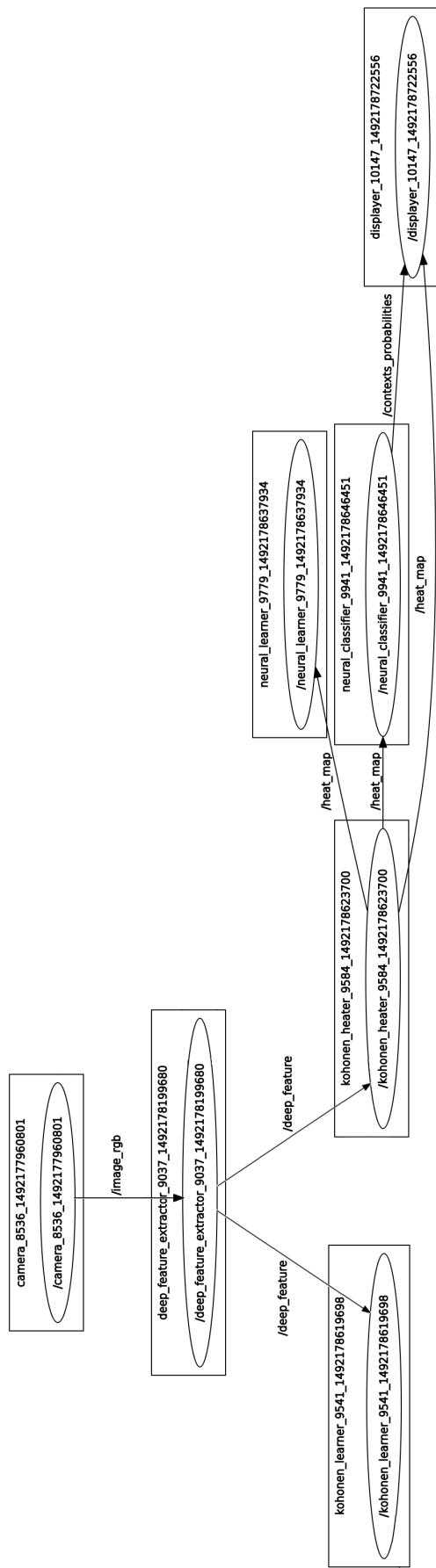


FIGURE 20 – Graphe des nœuds et topics fonctionnant sur le robot. Seule la partie qui concerne directement notre projet est reproduite ici.