Apprentissage pour la navigation robotique parmi les humains : vers des chemins s'adaptant aux dynamiques de l'environnement

Yohan Michelland
Encadré par Laëtitia Matignon, Jacques Saraydaryan et Olivier Simonin

Université Claude Bernard Lyon 1, France

Résumé La navigation robotique en milieu peuplé est un champ de recherche très actif depuis de nombreuses années. De nouvelles perspectives sont offertes grâce à l'essor de l'apprentissage par renforcement profond. On peut retrouver deux types d'approches dans la littérature. D'un côté, une approche réaliste du point de vue robotique, où le robot apprend une politique de contrôle à partir des données bruitées fournies par ses capteurs. De l'autre, une approche symbolique, où le robot possède une perception complète des positions et vitesses des agents de l'environnement. On laisse de côté le problème de vision et de représentation du monde physique, pour se concentrer sur la meilleure manière de convertir la perception en action. Le modèle DS-RNN est un bon exemple de cette dernière approche. Mais ses performances diminuent dans des conditions plus réalistes. La limitation du champs de vision et la présence d'obstacles statiques dans l'environnement affectent grandement son taux de succès. L'indifférenciation entre humain et obstacle semble en être la cause principale, nous avons donc proposé de multiples modifications d'architectures pour palier à ce problème.

Mots-clés: navigation robotique, apprentissage par renforcement profond

Abstract. Robotic navigation in a populated environment has been a very active research field for many years. New perspectives are offered by the development of deep reinforcement learning. Two types of approaches can be found in the literature. On the one hand, a realistic approach from the robotic point of view, where we try to learn a control policy from the noisy data provided by the robot sensors. On the other hand, a symbolic approach, where the robot has a complete perception of the positions and speeds of the agents in the environment. We leave aside the problem of vision and representation of the physical world, to focus on the best way to convert perception into action. The DS-RNN model is a good example of this last approach. But these performances decrease under more realistic conditions. The limitation of the field of view and the presence of static obstacles in the environment greatly affect its success rate. The indistinguishability between human and obstacle seems to be the main cause, so we have proposed multiple architectural modifications to overcome this problem.

Keywords: robotic navigation, deep reinforcement learning

J'adresse mes remerciements à mes encadrants : Laëtitia Matignon, Jacques Saraydaryan et Olivier Simonin qui m'ont fait confiance, et qui m'ont guidé avec bienveillance dans toutes les phases de ce stage. Ils m'ont communiqué avec succès l'esprit de la recherche, et j'ai eu la chance de bénéficier d'un suivi très régulier. Je remercie également les équipes SyCoSMA et CHROMA pour leur accueil, leur écoute et leurs précieux retours

1 Introduction

1.1 Problématique générale

La navigation robotique en milieu peuplé d'humains est un champs de recherche très actif en IA, et un enjeux majeur pour l'acceptabilité des robots autonomes. Il pose de multiples défis, notamment dans la modélisation des comportements humains, la modélisation des flux de personnes, la vision, la représentation de l'environnement, et la stratégie de navigation.

L'apprentissage par renforcement (AR), et en particulier l'AR profond, est une piste prometteuse pour l'apprentissage de stratégies de navigation sûres et efficaces.

1.2 Conditions et objectifs du stage

Il s'agit d'un stage de recherche, d'une durée de 6 mois, effectué au sein du laboratoire LIRIS, encadré par :
- Laëtitia Matignon : enseignante-chercheuse dans l'équipe SyCoSMA, au LIRIS
- Jacques Saraydaryan : enseignant-chercheur dans l'équipe CHROMA, au CITI
- Olivier Simonin : enseignant-chercheur dans l'équipe CHROMA, au CITI

Les objectifs de ce stage peuvent être présentés en deux temps. Tout d'abord, réaliser l'état de l'art sur les approches AR pour la navigation robotique sociale. Ensuite, choisir et adapter un algorithme d'AR profond pour la navigation robotique sociale, dans un milieu peuplé dynamique simulé. Ce stage de recherche ayant une composante exploratoire, les objectifs ont évolué au cours du temps.

1.3 Présentation générale des contributions

La première contribution, est le travail d'état de l'art qui a été effectué pendant le premier mois du stage, sur les différentes approches du problème de navigation robotique. Cela a consisté en la lecture, la compréhension et la synthétisation d'une vingtaine d'articles sur le sujet. Ce premier mois a permis d'aboutir à une vision globale sur les approches existantes, et nous a donné le recul nécessaire au choix des prochains objectifs.

Nous avons ensuite fait le choix de nous concentrer sur un modèle d'apprentissage particulier : DS-RNN, présenté dans l'article de Liu et al. [4]. Il s'agit d'un modèle complexe, qui a demandé un long travail de compréhension. Le simulateur utilisé pour entraîner et tester ce modèle, $CrowdNav$, a également demandé un temps de prise en main. Une fois ceci fait, nous nous sommes lancés dans un travail exploratoire d'adaptation de ce modèle à de nouvelles hypothèses. Plus précisément, nous avons entraîné un grand nombre de modèles personnalisés, dans des conditions variées, afin d'évaluer la capacité du modèle à s'adapter à des conditions plus réalistes. Cette étape nous a permis de mettre en défaut le modèle DS-RNN de base.

Dans un troisième temps, nous avons proposé des modifications d'architecture pour adapter le modèle aux hypothèses testées précédemment. Cette étape consistait pour chaque proposition, à implémenter l'architecture, à entraîner un ou plusieurs modèles, et enfin à évaluer le modèle d'apprentissage en le comparant au modèle DS-RNN de base.

1.4 Plan du rapport

La première partie de ce rapport sera dédiée à l'état de l'art. Après avoir formalisé précisément le problème de navigation robotique, nous présenterons les quatre types d'approches existantes, ainsi qu'un exemple d'article pour chaque approche. Nous conclurons cette partie en présentant l'article de DS-RNN, ainsi que le simulateur $CrowdNav$, utilisé pour entraîner et tester les différents modèles au cours du stage.

Dans la seconde partie, nous détaillerons l'ensemble des contributions. Nous aborderons tout d'abord l'ensemble des modifications effectuées sur le simulateur pour intégrer nos nouvelles hypothèses. Ensuite, nous évoquerons le protocole expérimental choisi pour évaluer les différents modèles d'apprentissage. S'en suivront une présentation détaillée de DS-RNN, ainsi que les résultats de la mise en défaut de ce modèle. Enfin, nous présenterons successivement chaque proposition d'architecture, ainsi que leur évaluation.

La dernière partie nous permettra de prendre du recul sur le travail réalisé, ses apports et les perspectives futures.

2 État de l'art et positionnement

2.1 Formalisation du problème de navigation robotique

On considère un robot mobile, évoluant dans un environnement dont il connaît (éventuellement) la carte. Il est généralement équipé de deux types d'équipement:

- Des senseurs pour la perception de l'environnement (radar laser, caméra de profondeur)
- Des actionneurs pour agir dans l'environnement (servomoteurs)

Le robot doit naviguer jusqu'à une cible, en évitant les collisions avec les obstacles, qui peuvent être d'ordre :

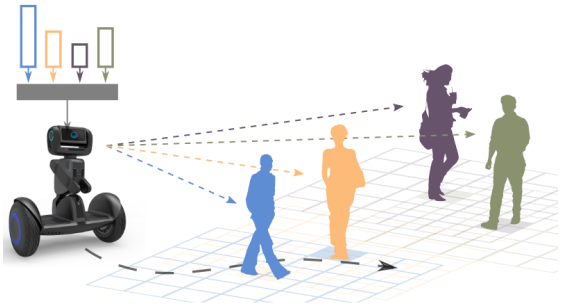

Fig. 1: Chen et al. [4]

– Statiques : cloisons, objets.
– Dynamiques : piétons, foule.

Le problème de navigation robotique consiste à trouver une stratégie de contrôle pour l'agent, c'est- à-dire une fonction qui, à partir de perceptions (senseurs), renvoie une action (consistant souvent à choisir une direction et une vitesse). Le comportement induit par cette stratégie de contrôle peut-être évalué selon une variété de critères, mais les deux objectifs principaux sont de minimiser le risque de collision, et de minimiser le temps ou la distance de trajet. On retrouve également des critères sociaux comme préserver l'espace vital des personnes, ou encore effectuer des mouvements lisses et prévisibles.

2.2 Exemple d'approche réactive : van den Berg et al. [1]

Dans cet article datant d'avril 2011, van den Berg et al. présentent la politique ORCA (Optimal Reciprocal Collision Avoidance). Il s'agit d'une approche réactive pour l'évitement de collision locale entre plusieurs robots mobiles, qui a servi de *baseline* pour l'évaluation des stratégies de navigation jusqu'à aujourd'hui.

La politique ORCA a la particularité de garantir un taux de collisions nul, à condition que l'ensemble des agents dans l'environnement utilisent la même politique. En d'autres termes, plusieurs agents ORCA n'entrent jamais en collision. ORCA ne garantit en revanche pas que tous les agents atteignent leur cible.

On considère donc N robots omni-directionnels en forme de disques dans un espace 2D, utilisant la même politique ORCA, chacun possédant :

– un état externe (i.e visible par tous les agents) : position, rayon, vitesse
– un état interne (i.e non externe) : vitesse maximum, vitesse préférée (celle qui définit la direction de la cible)

A chaque itération, chaque agent A suit cet algorithme :

– perception de l'état externe de l'ensemble des autres agents

- pour chaque autre agent i, calcule de l'ensemble des vitesses garantissant l'évitement de collision avec le robot i : ORCA(A|i). Cet ensemble prend la forme d'un demi-plan dans l'espace des vitesses, comme on peut le voir sur la figure 2, et est calculé en résolvant un problème linéaire.
- calcul de l'intersection de tous les demi-plans, afin d'obtenir l'ensemble des vitesses garantissant l'évitement de collision

$$ORCA_A = \bigcap_{B \neq A} ORCA_{A|B} \tag{1}$$

- choisir dans cet ensemble la vitesse la plus proche de la vitesse préférée

La politique ORCA possède deux limites principales. Tout d'abord, elle fait des hypothèses très fortes sur le comportement des autres agents qui ne seront pas vérifiées dans le cas d'agents humains. Ensuite, cette stratégie rencontre un problème récurrent dans les approches réactives : le *freezing robot problem*. Dans des scénarios à haute densité, l'agent aura tendance à stopper tout mouvement, car aucun ne garantit un évitement de collision, lorsque l'ensemble $ORCA$ est vide par exemple. C'est entre autres pour résoudre ce problème que des approches moins naïves ont été développées par la suite, et que l'apprentissage est rentré en jeu.

2.3 Exemple d'approche basée modèle : Saraydaryan et al. [5]

Nous allons aborder désormais une approche moins naïve du problème de navigation robotique. Dans cet article d'octobre 2018, Saraydaryan et al. proposent une approche basée modèle. Cela signifie qu'une représentation de l'environnement de l'agent est construite, sur laquelle va s'appuyer le calcul d'un chemin minimisant le temps de trajet.

Concernant les hypothèses de travail, on considère un environnement assez vaste (figure 3) composé de deux couloirs, et un flux d'humains circulant dans un sens ou dans l'autre. Il s'agit d'une approche globale, car on ne cherche plus seulement à éviter localement les collisions, mais également à trouver un chemin optimal jusqu'à la cible, en prenant en compte le sens de circulation du flux humain.

La première étape de la stratégie de navigation consiste à construire et mettre à jour une représentation de l'environnement à partir des observations de l'agent. L'espace est représenté dans une grille de cellules, chaque cellule contenant la suite d'observations jusqu'au temps t : Z^t. Une observation $O_{x,y,k}$ de la cellule (x, y) au temps t, pour une direction k vaut 1 si une personne se déplaçant dans la direction k est observée, et 0 sinon. A partir de la suite d'observation associée à une cellule, on peut déduire la probabilité d'observer un humain se déplacer depuis (x, y) dans la direction k : $M_{x,y,k}(Z^t)$. On obtient alors une grille de flux, qui est mise à jour à chaque observation. Un hyper-paramètre permet de contrôler l'importance des informations plus récentes, et l'information des cellules non observées s'atténue avec le temps.

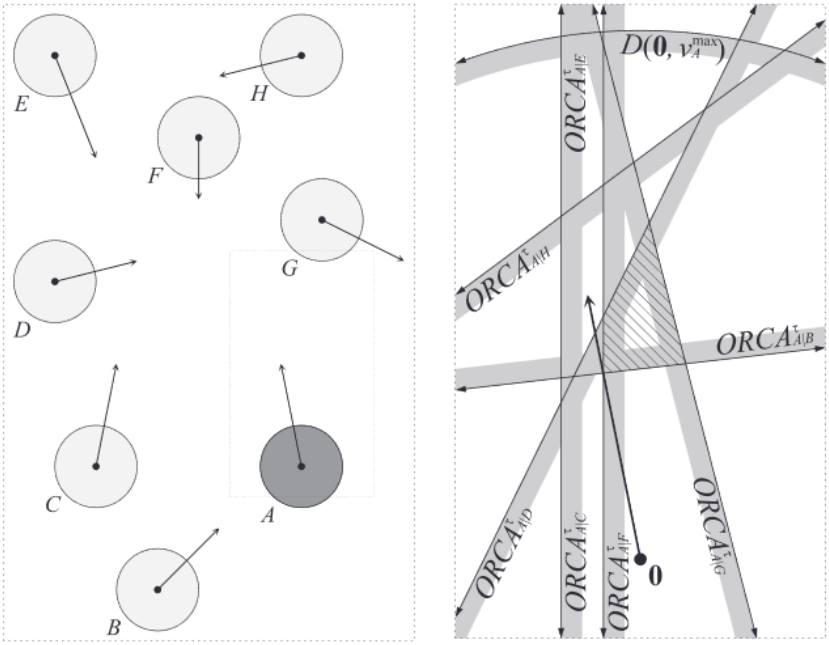

Fig. 2: A gauche : une configuration avec 8 robots. A droite : l'ensemble des demi-plans garantissant l'évitement de collision pour l'agent A. L'intersection des demi-plans est représentée par la zone hachurée.

Dans un second temps, les cellules sans observations sont complétées par des prédictions du mouvement humain, basées sur le modèle de Von Mises.

Enfin, l'algorithme de plus court chemin A* est adapté pour prendre en compte cette grille de flux. Plus précisément, un facteur est ajouté à la fonction de coût, calculé à partir de la grille de flux, et représentant la direction et l'intensité du flux sur les cellules. Ceci a pour effet de doubler le coût dans le cas d'un déplacement dans la direction opposée au flux, et d'avoir un coût nul dans le cas d'un déplacement dans la même direction que le flux.

Cette politique de navigation est évaluée en environnement simulé (figure 3), dans un scénario où le flux d'humain change régulièrement de sens de circulation. L'agent est capable de s'adapter au sens de circulation et de choisir le couloir qui évite d'être à contre-courant.

2.4 Exemple d'approches sur données réelles : Xie et al. [6]

Nous allons maintenant nous pencher sur les approches appliquant le *Machine Learning* à la tâche de navigation robotique. Dans ce premier article, un agent est entraîné à prédire la prochaine action à effectuer, à partir de données issues de ces capteurs : un Lidar, qui est un radar laser, et une caméra. Contrairement

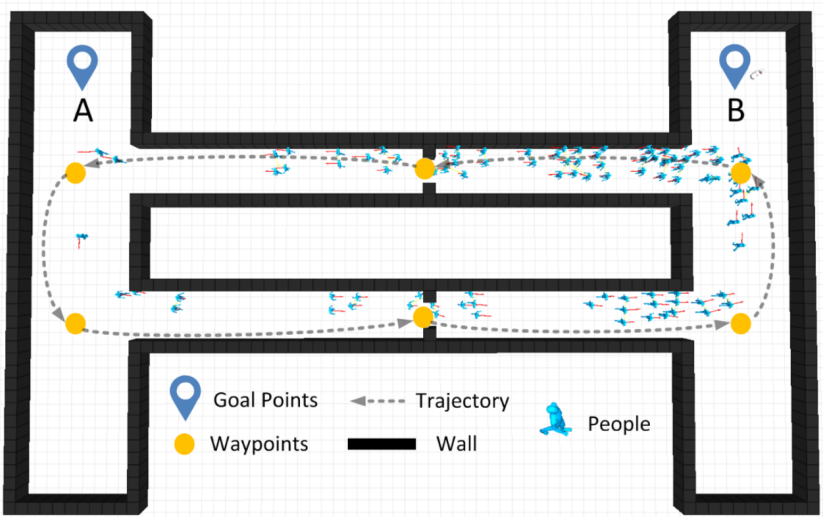

Fig. 3: Exemple de scénario : le robot part du point A et doit rejoindre le point B. Le flux d'humain tourne dans le sens anti-horaire.

à l'approche symbolique, que l'on verra dans la partie suivante, on prend ici en compte la complexité de l'environnement : le robot possède une perception imparfaite et incomplète de son environnement, car il est limité par ses capteurs.

Le simulateur Gazebo est utilisé pour entraîner et tester le modèle. Il est capable de simuler de manière cinématique et dynamique un environnement 3D complexe. Il est complété par le simulateur de foule PedSim, qui utilise un modèle de force sociale. Le robot doit naviguer dans cet environnement pour atteindre sa cible le plus vite possible, en minimisant les collisions (figure 4).

Une première étape de pré-traitement des données a lieu avant le réseau de neurones. Cette étape permet d'obtenir 3 *channels* de 80 par 80 qui seront l'entrée du réseau.

- Le premier est obtenu via un pré-traitement par combinaison de *minimum pooling* et d'*average pooling* sur les données Lidar. Ces données prennent la forme de 10 scans de distances toutes les 0.5 secondes
- Les deux autres représentent les vitesses des piétons, et sont obtenus via un algorithme de détection appliqué aux données caméra, et un algorithme de détection de jambe appliqué aux données Lidar.

Les données précédentes sont exprimées dans les coordonnées locales du robot. Les 3 *channels* sont ensuite donnés en entrée d'un réseau de neurones (figure 5). Plus loin dans le réseau, la position de la cible à atteindre est ajoutée. En sortie de réseau, on obtient la prédiction de la prochaine vitesse à choisir. L'apprentissage s'effectue de manière supervisée via un *dataset* de démonstrations. Celui-ci est construit grâce à une collecte de données à partir de foules humaines réelles.

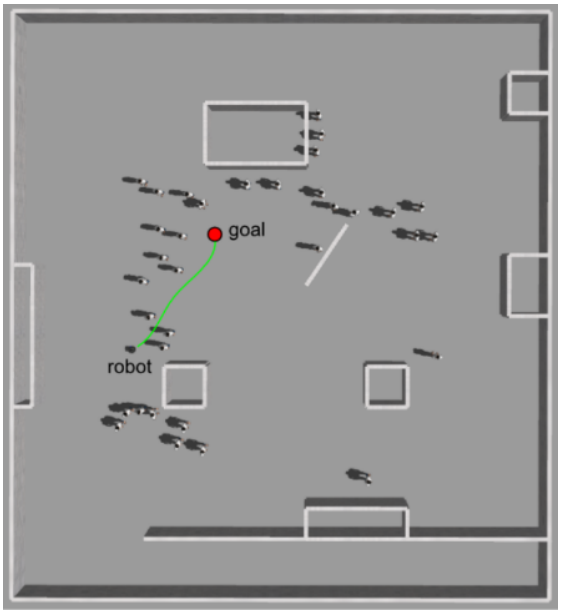

Fig. 4: Exemple de scénario dans le simulateur Gazebo. Le trajet emprunté par le robot est représenté en vert.

2.5 Exemple d'approche symbolique : Chen et al. [2]

Le dernier type d'approche est celui qui va nous intéresser le plus. Contrairement à l'approche sur données réelles, on considère cette fois-ci que l'agent a accès à l'ensemble des positions et vitesses des humains environnants. Le robot possède donc une perception complète et parfaite de son environnement et c'est en ce sens que l'on parle d'approche symbolique.

Cet article de mai 2019 par Chen et al. est un bon exemple de cette approche. Leur idée est d'utiliser les méthodes d'AR profond pour apprendre une politique de navigation.

Le simulateur utilisé pour l'entraînement et l'évaluation des politiques de navigation est *CrowdNav*. C'est en environnement développé par les même auteurs, et conçu spécifiquement pour l'AR profond appliqué à la tache de navigation robotique. Il a été réutilisé par la suite dans beaucoup d'autres travaux similaires, et notamment pour le modèle DS-RNN, que nous aborderons dans ce rapport.

Le scénario typique dans *CrowdNav* est le $CircleCrossing$, et est représenté sur la figure 6. On considère un robot R (en jaune), et N humains (en rouge ou bleu) représentés par des disques. Les humains, contrôlés par la politique ORCA[1] sont positionnés en cercle, et ont pour cible la position diamétralement opposée. Le robot doit rejoindre sa cible (l'étoile) dans le temps imparti en évitant les collisions. Le robot a connaissance de sa cible (i.e la position qu'il

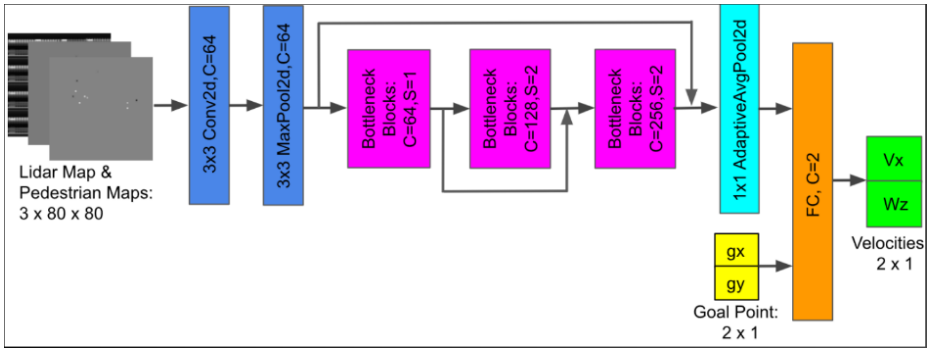

Fig. 5: Architecture du réseau de neurones.

doit atteindre), et de la position relative de tous les agents dans la scène. A chaque itération, le robot doit choisir son prochain vecteur vitesse (V_x, V_y).

L'architecture du modèle SARL, présenté dans l'article, est montrée sur la figure 7, et est composée de 3 modules.

Le module *Interaction* est un perceptron multi-couches qui encode chaque interaction Robot-Humain à partir de l'état de chacun (position, vitesse) et d'une carte locale centrée sur l'humain.

Le module *Pooling* calcule un score d'attention pour chaque interaction, et agrège l'ensemble des interactions dans un vecteur de taille fixe.

Le module *Planning* est un perceptron multi-couches qui prend en entrée le vecteur précédent et prédit la valeur de l'état actuel, c'est-à-dire le gain espéré en terme de récompense depuis cet état. La récompense est fortement positive si l'agent a atteint sa cible et fortement négative si une collision a eu lieu. La politique de navigation est obtenue en choisissant l'action maximisant la valeur de l'état suivant.

2.6 Synthèse des approches et positionnement

Nous avons vu 4 approches différentes pour la tache de navigation robotique. Le stage portant sur l'apprentissage , nous nous sommes concentrés sur les deux dernières.

D'un côté, l'approche sur données réelles est plus réaliste du point de vue de la robotique. En effet le robot perçoit des données bruitées issues de ses senseurs, et les contraintes dynamiques du monde physique limitent sa vision et son champ d'action. Ces aspects sont pris en compte dès l'apprentissage, ce qui facilite la transition entre milieu simulé et milieu physique.

De l'autre, l'approche symbolique utilise des outils d'apprentissage puissants pour aboutir à des politiques de navigation très efficaces, mais dans un environnement 2D simplifié, sans obstacles, et avec une information exacte et complète sur la position des humains.

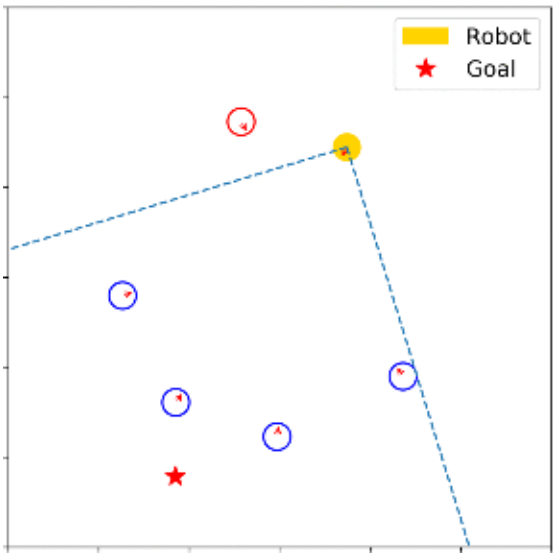

Fig. 6: Exemple de scénario *CircleCrossing* dans le simulateur *CrowdNav*

A partir de ce constat, nous avons trouvé intéressant de partir d'un modèle à approche symbolique, et d'explorer sa robustesse face à de nouvelles contraintes et hypothèses plus réalistes. L'idée est d'évaluer l'évolution des performances du modèle dans des conditions plus difficiles.

Nous avons donc fait le choix de nous concentrer sur DS-RNN, un modèle présenté dans la prochaine partie. Les auteurs étant dans la même optique d'ajout d'hypothèses réalistes, cela nous a semblé être un bon point de départ.

3 Contribution

3.1 Compréhension détaillée du modèle DS-RNN

Présentation de S-RNN : Jain et al. [3] Avant d'aborder le modèle DS-RNN en lui même, il est nécessaire d'évoquer cet article de 2016. En effet, Jain et al. formulent une méthode générique pour convertir un problème pouvant être modélisé sous forme spatio-temporelle en une architecture de réseaux de neurones adaptée. Liu et al. ont appliqué cette méthode au problème de navigation robotique afin d'obtenir l'architecture DS-RNN. La méthode S-RNN peut être appliquée à une variété de problèmes de classification, en particulier lorsqu'un raisonnement spatio-temporel est profitable. Un exemple de problème est montré dans la figure 8. En bas, on a une suite d'image représentant un humain interagissant avec son environnement. L'idée est de modéliser cette situation dans un graphe spatio-temporel (st-graph) tel que :

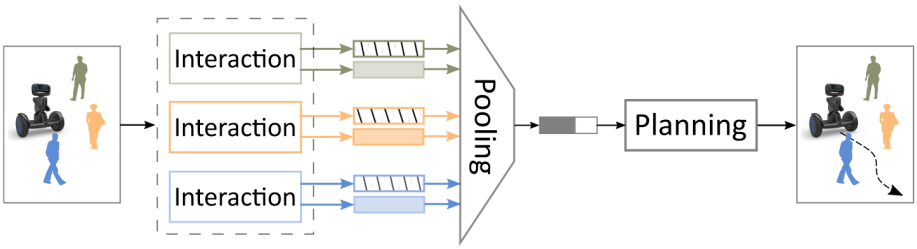

Fig. 7: Architecture du modèle SARL

– les noeuds représentent les objets (en bleu) et agents (en rouge) dans la scène
– les arêtes dites spatiales représentent les interaction entre les noeuds.
– les arêtes dites temporelles représentent le lien entre un noeud au temps t et ce même noeud au temps $t + 1$.

Dans cet exemple particulier, l'objectif est de prédire le label de chaque noeud. Le label du noeud rouge représente l'activité pratiquée par l'humain. Le label des noeuds bleus représente l'affordance de l'objet. La méthode S-RNN permet de convertir automatiquement ce st-graph en une architecture de réseau de neurones profond adaptée, qui permettra un raisonnement spatial et temporel. Cette architecture est composée essentiellement d'une association de réseaux de neurones récurrents (RNN). Elle pourra ensuite être utilisée pour entraîner un modèle d'apprentissage, dans un cadre supervisé, à l'aide des labels réels des noeuds.

Présentation de DS-RNN : Liu et al. [4] Le modèle DS-RNN, sur lequel nous nous sommes basés pendant le stage, est décrit dans cet article de juin 2021. Il s'inscrit dans l'approche symbolique, présentée précédemment. Le simulateur utilisé pour l'apprentissage et l'évaluation de la politique est $CrowdNav$, créé par Chen et al.[4], et le scénario de base est le *circle-crossing* présenté précédemment. Il s'agit d'une architecture de réseau de neurones profond, induite par la méthode S-RNN. Mais à la différence du précédent exemple, l'apprentissage ne s'effectue pas de manière supervisée, mais par renforcement profond.

Nous allons désormais détailler l'architecture de DS-RNN, en commençant par le graphe spatio-temporel associé au scénario (figure 9). Le graphe contient deux types de noeuds : le robot w et les humains u_i . Les arêtes spatiales représentent les positions relatives des humains par rapport au robot. Les arêtes temporelles représentent l'évolution de la position du robot entre une itération et la suivante. Le *Factor Graph* est une version compacte du *St-Graph*, et fait apparaître les trois $factor$. Chaque $factor$ correspond à une entrée du modèle :

– Node Factor : état interne du robot. Il s'agit d'un vecteur $(px, py, radius, gx, gy, vpref, theta)$ contenant la position, le rayon, la position de la cible, la vitesse préférée, et l'orientation.

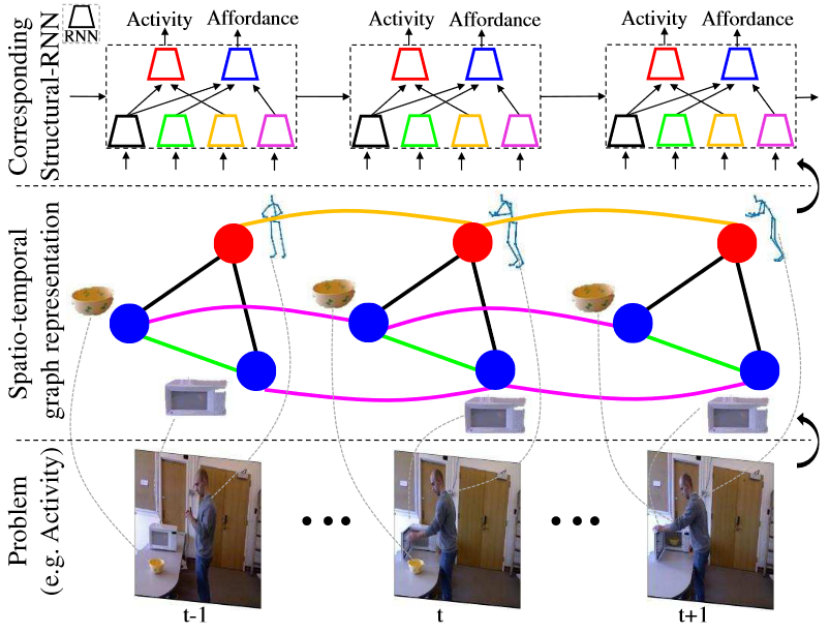

Fig. 8: En bas : Problème initial. Au milieu : Modélisation du problème en st-graph. En haut : Représentation schématique de l'architecture S-RNN dérivée du st-graph

- Spatial Edge Factor : position relative des humains. Il s'agit de la liste des positions (px, py) des humains, dans des coordonnées centrées sur le robot.
- Temporal Edge Factor : évolution de la position du robot. Il s'agit d'un vecteur vitesse (vx, vy)

Ces trois entrées apparaissent sous forme de rectangles dans l'architecture globale de DS-RNN (figure 10).

Les trapèzes représentent les réseaux de neurones récurrents. Ces derniers calculent à la fois un *embedding* des données d'entrée (sortie bleue), et une couche cachée réutilisée lors de la prochaine itération (sortie rouge). Ils contiennent notamment une *Gated Reccurent Unit (GRU)*, qui est comparable à une LSTM et capable de mémoire à court et long terme, grâce à la couche cachée.

Le cercle représente le module d'attention. Il a pour rôle de calculer un poids d'attention pour chaque humain. Ce poids est proportionnel à l'importance qu'il faut accorder à l'humain dans le calcul de la prochaine action. Un humain distant et s'éloignant du robot aura par exemple un poids d'attention très faible.

L'*Actor* prédit le prochain vecteur vitesse, et représente la politique actuelle utilisée par l'agent pour l'entraînement. Le *Critic* prédit la valeur de l'état actuel. L'apprentissage s'effectue grâce à l'algorithme PPO (Proximal Policy Optimisation). Il appartient à la classe des méthodes de *Policy Gradient*, l'idée générale

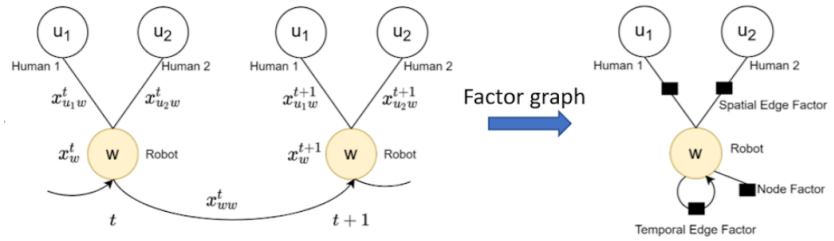

Fig. 9: A gauche : Graphe spatio-temporel du problème de navigation, représenté sur deux itérations. A droite : Factor graph déduit, chaque factor est une entrée du modèle

étant d'effectuer des épisodes, et d'optimiser la politique de l'agent à l'aide d'une descente de gradient. Celle-ci repose sur une fonction de récompense calculée à chaque itération. La récompense est fortement positive si l'agent atteint sa cible, fortement négative si une collision a lieu, et faiblement négative si l'agent est trop proche d'un humain.

Positionnement et objectifs par rapport à DS-RNN Parmi toutes les approches symboliques trouvées en effectuant l'état de l'art, nous avons choisi de nous concentrer sur DS-RNN, pour plusieurs raisons.

Tout d'abord, le modèle DS-RNN a la particularité de faire des hypothèses plus faibles sur la perception de l'environnement. Contrairement à d'autres modèles comparables, DS-RNN considère que la position des humains est connue, mais pas la vitesse. Ceci va dans le sens d'une approche plus réaliste, car dans la pratique il est difficile de détecter la vitesse exacte ou l'orientation actuelle des humains environnants.

Ensuite, les auteurs de l'article ont tenu à prendre plus en compte le caractère partiellement observable de l'environnement, ce qui encore une fois rapproche le problème de conditions plus réalistes de robotique. Ils ont notamment ajouté la possibilité de simuler un bruit sur les positions perçues, ainsi qu'une chance pour les humains de changer de cible aléatoirement. Ils ont également introduit dans le simulateur un moyen de réduire le champs de vision du robot, sous la forme d'un filtre appliqué à l'observation reçue par le robot. La position d'un humain est actualisée si et seulement si celui-ci se trouve dans le champs de vision. Dans le cas contraire, une estimation linéaire est effectuée à partir de la dernière position connue. Modifier la valeur de ce champ de vision est donc un premier moyen de mettre en défaut le modèle DS-RNN.

En s'inspirant de la méthodologie de Liu et al., nous nous sommes donné pour objectif d'explorer d'autre manières de mettre en défaut le modèle DS-RNN de base, en ajoutant de nouvelles hypothèses, l'idée étant de s'approcher des conditions réelles de robotique. Nous nous sommes finalement penchés plus précisément sur l'ajout d'obstacles statiques, et sur l'ajout d'un mécanisme d'occlusion.

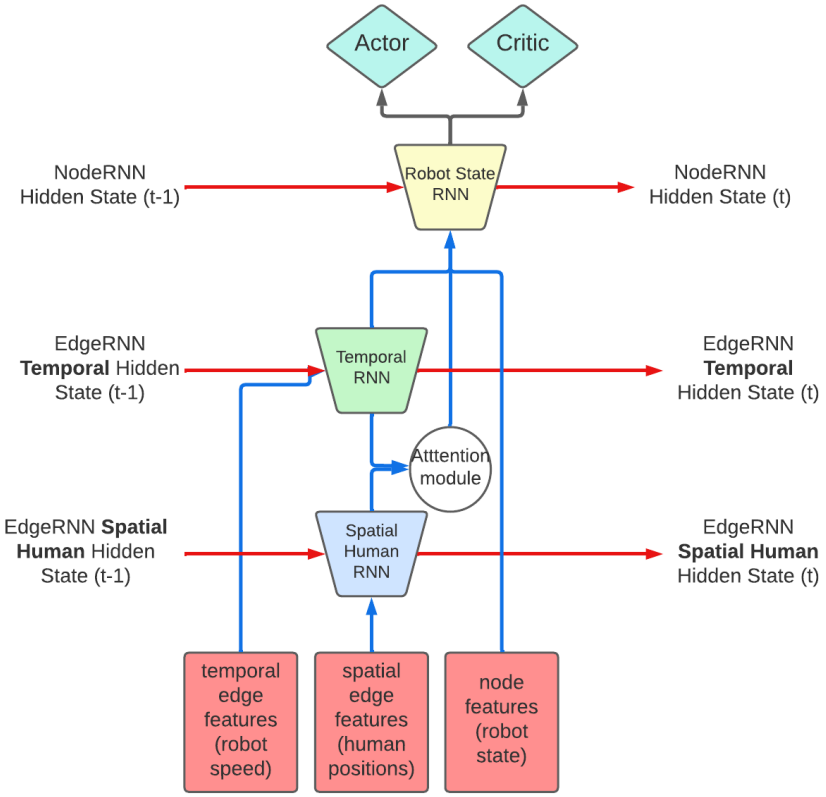

Fig. 10: Architecture globale de DS-RNN.

Dans un premier temps, nous avons donc implémenté les fonctionnalités requises au simulateur pour tester ces nouvelles hypothèses.

Ensuite, nous avons évalué la robustesse du modèle DS-RNN face à ces hypothèses.

Enfin, nous avons proposé plusieurs modifications d'architecture, visant à adapter le modèle aux nouvelles conditions.

3.2 Modification du simulateur CrowdNav

Présentation générale du simulateur. *CrowdNav* est un simulateur open-source développé par Chen et al. [4] et a été conçu spécifiquement pour l'AR appliqué à la navigation robotique. Il fournit un environnement héritant du framework de la librairie *OpenAI Gym*.

La fonction principale de l'environnement, *step* est appelée à chaque itération. Elle prend en entrée l'action choisie par le robot (dans notre cas, le vecteur

vitesse), calcule l'action choisie par chaque humain (en appliquant la politique ORCA) et met à jour l'état du monde. En sortie, elle renvoie au robot l'observation suivante (les trois $factor$ détaillées plus haut), l'évènement (collision, cible atteinte ...) et la récompense associée à cet évènement.

L'environnement fournit également une fonction $render$, appelée à chaque itération, et permettant de visualiser l'épisode. Ceci permet d'ajouter un aspect qualitatif lors de la phase d'évaluation.

Ajout d'un scénario d'obstacles aléatoires L'ajout d'obstacles statiques dans l'environnement est l'hypothèse principale que nous avons exploré. L'idée était d'évaluer la robustesse de DS-RNN face à un environnement plus complexe.

Nous avons donc implémenté la possibilité d'ajouter des obstacles à l'environnement $CrowdNav$. Chaque obstacle possède une position et un rayon, et les collisions sont gérées de la même manière qu'entre agents.

Du point de vue de l'agent DS-RNN, les obstacles sont traités comme des humains. Dans la pratique, l'environnement concatène la liste des positions des humains et des obstacles, avant de renvoyer l'observation. En revanche, à la différence des humains, les obstacles sont toujours visibles, même hors champs de vision. Cela revient à faire l'hypothèse que le robot possède une carte de son environnement.

Du point de vue d'un humain en revanche, l'obstacle est traité différemment, car la politique ORCA prend en compte le caractère immobile d'un obstacle dans le calcul des vitesses possibles.

Il est possible de générer une configuration particulière d'obstacles à partir d'un fichier texte. Mais nous avons plutôt privilégié la génération aléatoire de configuration. A chaque épisode, M obstacles sont placés aléatoirement dans un carré centrale de l'environnement, dont la taille est choisie en paramètre. Nous nous sommes assurés que la cible du robot ne soit jamais positionnée sur un obstacle. La figure 11 montre un exemple de scénario contenant des obstacles.

Ajout d'un mécanisme d'occlusion Toujours dans l'objectif de rendre le problème de navigation plus réaliste, nous avons ajouté un mécanisme d'occlusion. Un humain peut désormais être considéré comme hors champs de vision si il est obstrué par un autre humain, ou un obstacle. En effet, en conditions réelles, les capteurs du robot ne seront pas capables de détecter un humain obstrué.

Plus formellement, comme schématisé sur la figure 12, on considère un robot en position R, un humain en position H dans le champs de vision du robot, et un obstacle O de rayon r. H est obstrué par O selon R si et seulement si ces deux conditions sont réunies :

- La droite (RH) intersecte le disque O, i.e

$$Dist(O, RH) < r$$

- R est plus proche de O que de H, i.e

$$Dist(R, O) < Dist(R, H)$$

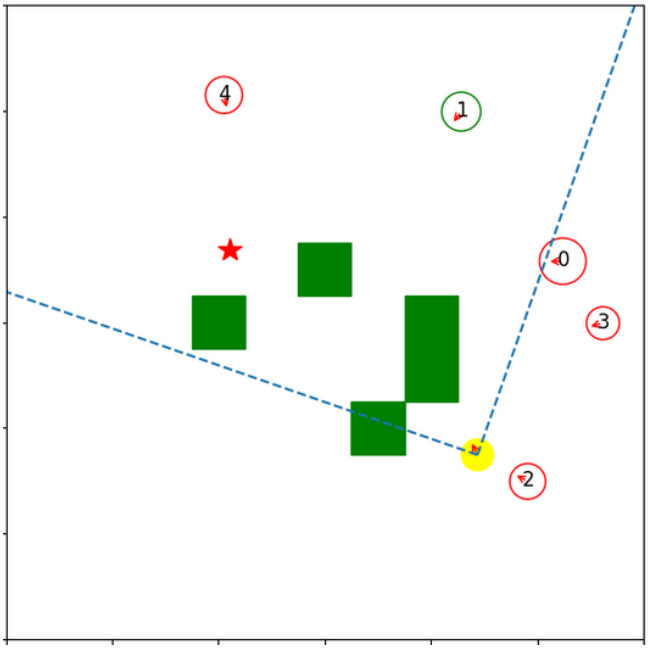

Fig. 11: Exemple de scénario avec obstacles (représentés en vert)

Ajout d'un suivi des poids d'attention Enfin, la dernière modification du simulateur est l'affichage des poids d'attention (figure 13). Il ne s'agit pas d'une modification de l'environnement, mais de l'interface. Cela permet, lors de la phase de test, d'évaluer qualitativement la politique de navigation, en vérifiant que l'attention portée aux agents et obstacles est cohérente.

3.3 Protocole expérimental pour l'évaluation des modèles

Dans cette partie, nous allons décrire la méthodologie utilisée pour l'entraînement et l'évaluation des différents modèles.

Conditions d'apprentissage L'entraînement de l'ensemble des modèles a été effectué sur GPU, à l'aide d'une machine présente au laboratoire LIRIS. Le protocole SSH était utilisé pour y accéder et lancer les apprentissages à distance. Pour la plupart des modèles, l'apprentissage a duré pendant 10 millions d'itérations, ce qui représente environ 11 heures. Une sauvegarde régulière des poids a lieu, ainsi que des données sur l'évolution de la récompense cumulée par épisode. Ceci permet d'afficher une courbe de l'évolution de la récompense, qui peut aider à savoir si le modèle a eu suffisamment de temps pour converger, ou si il est nécessaire de prolonger l'entraînement (figure 14).

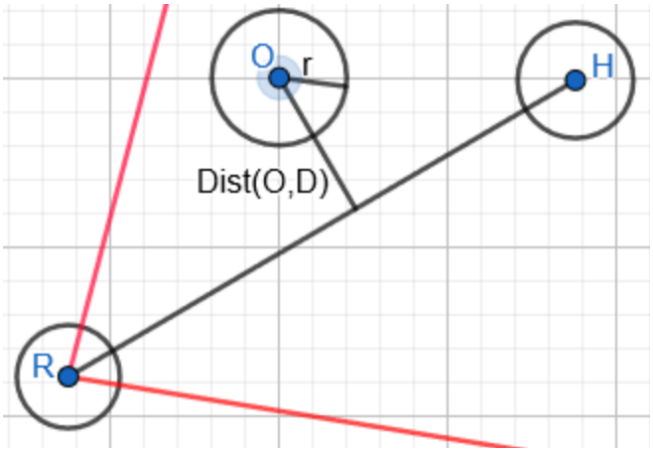

Fig. 12: Représentation géométrique du mécanisme d'occlusion

Condition de test Une fois l'entraînement terminé, le modèle obtenu est évalué sur 500 épisodes. Un système de *seed* permet de générer 500 configurations d'environnement jamais vues auparavant. Plusieurs métriques sont alors utilisées pour évaluer l'efficacité du modèle, et le comparer aux autres :

- Success Rate : le taux de succès
- Col.Hum.Rate : le taux de collisions avec un humain
- Col.Obs.Rate : le taux de collisions avec un obstacle
- Timeout Rate : le taux de *timeout*, lorsque la cible n'est pas atteinte avant la fin du temps imparti
- Mean Reward : la récompense cumulée moyenne par épisode
- Navigation Time : la durée de l'épisode
- Path Length : la distance totale de trajet effectuée par l'agent
- Danger Freq : la proportion de temps pendant laquelle l'agent était trop proche d'un humain
- Avg Danger Dist. : la distance moyenne aux humains lorsque l'agent est trop proche d'un humain

Méthode de compilation des résultats L'ensemble des tests effectués ont été rassemblé dans un tableau (figure 15). La partie *Entrainement* précise le scénario dans lequel l'apprentissage a été effectué. La partie *Test*, de la même manière, précise le scénario utilisé lors de l'évaluation du modèle. Dans les deux cas, sont indiqués le champs de vision de l'agent, l'activation, ou non du mécanisme d'occlusion, et le nombre d'obstacles générés à chaque épisode.

Enfin, la partie *Evaluation* regroupe les métriques citées plus haut. Dans le présent rapport, certaines n'apparaissent pas, par souci de clarté dans la présentation des résultats.

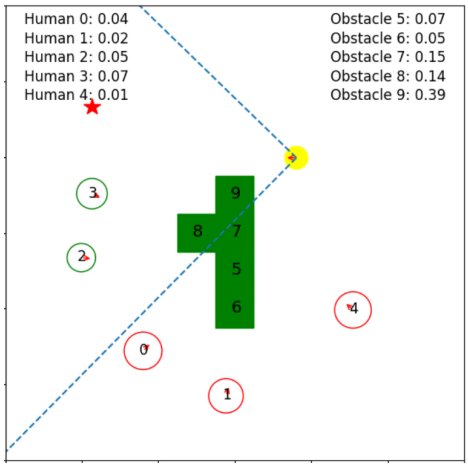

Fig. 13: Affichage des poids d'attention associés aux agents et aux obstacles

3.4 Evaluation du modèle DS-RNN

Réplication des résultats Afin d'avoir un modèle de base, nous avons commencé par reproduire les résultats de l'article de Liu et al. Pour cela, nous avons lancé un premier modèle d'apprentissage dans les mêmes conditions, avec des hyper-paramètres identiques. Comme le montre la première ligne de la figure 15, pour un nombre d'humains de 5, et un champs de vision de 360°, le taux de succès est de 0.96. Le taux de succès descend à 0.82 lorsque l'on réduit le champs de vision à 90° lors de l'apprentissage. Ces résultats correspondent à ceux présentés dans l'article. On remarque également que les performances du modèle chutent drastiquement lorsque le FOV descend à 45°, ce qui n'avait pas été testé par Liu et al.

Le modèle 3 (fig 15) nous servira de base de comparaison dans la suite de ce rapport. Dans les prochains modèles, les seuls paramètres qui changeront seront ceux apparaissant dans le tableau.

Mise en défaut du modèle DS-RNN Nous avons souhaité tester DS-RNN dans les nouvelles conditions, afin d'évaluer l'impact sur les performances.

Nous avons commencé par entraîner un nouveau modèle, dans un scénario ou 5 obstacles sont placés aléatoirement dans le centre de l'environnement. Pour rappel, le modèle DS-RNN n'est pas capable de différencier les humains des obstacles. L'observation envoyée à l'agent par l'environnement comprend donc une liste de 10 positions. Sur la figure 16, on observe que le taux de succès de ce nouveau modèle (modèle 4) chute à 0.66 en raison d'un taux de collisions beaucoup plus élevé. Si l'on se base sur les résultats présentés dans l'article, les performances de DS-RNN sont très peu sensibles au nombre d'humains. Le taux de succès du présent modèle (5 humains, 5 obstacles) est donc particulièrement

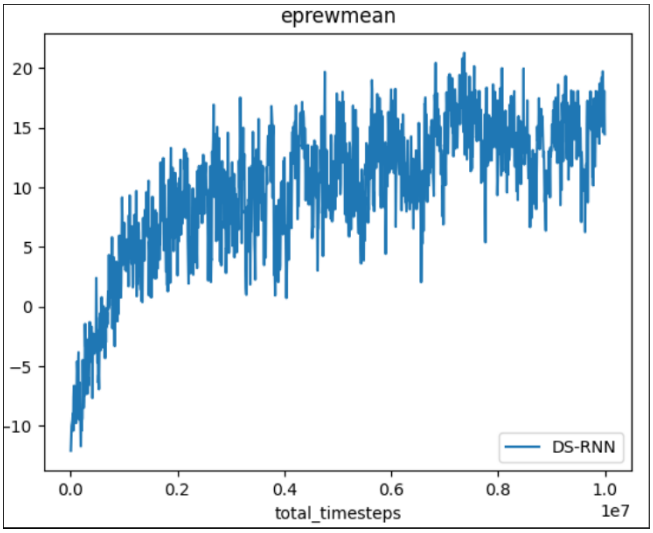

Fig. 14: Exemple de courbe d'évolution moyenne de la récompense en fonction du nombre d'itérations d'entraînement

ENTRAÎNEMENT					TEST			EVALUATION			
ID	Politique	FOV	Occlusion	Obstacles	FOV	Occlusion	Obstacles	Success Rate	Col. Hum. Rate	Timeout Rate	Mean Reward
1	dsrnn	360	No	0	360	No	0	0,96	0.04	0.00	24.67
2	dsrnn	45	No	0	45	No	0	0,24	0.59	0.17	-0.24
3	dsrnn	90	No	0	90	No	0	0,82	0.17	0.01	19.39

Fig. 15: Impact du FOV

bas, si on considère le fait qu'un obstacle statique est en principe moins complexe à éviter qu'un humain.

Nous avons ensuite entraîné un modèle dans les mêmes conditions que le précédent, en activant le mécanisme d'occlusion. L'idée était de comprendre à quel point l'occlusion augmente la difficulté de la tâche, notamment en la présence d'obstacles. La figure 16 (modèle 5) montre que le taux de succès descend cette fois-ci à 0.60, ce qui est significativement moins que lorsque l'occlusion est désactivée.

Pour conclure sur cette partie, on a pu voir que les performances de DS-RNN chutent drastiquement lorsque l'entraînement a lieu dans de nouvelles conditions. Le modèle n'est pas adapté à la présence d'obstacles statiques. Ceci est, d'après nous, causé par le fait que DS-RNN ne fait pas de différence entre un obstacle et un humain, ce qui entraîne l'apprentissage d'une seule et même politique d'évitement de collision pour les obstacles statiques comme dynamiques.

ENTRAÎNEMENT					TEST			EVALUATION			
ID	Politique	FOV	Occlusion	Obstacles	FOV	Occlusion	Obstacles	Success Rate	Collision Rate	Timeout Rate	Mean Reward
3	dsmn	90	No	0	90	No	0	0,82	0.17	0.01	19.39
4	dsmn	90	No	5	90	No	5	0,66	0.29	0.05	13.74
5	dsmn	90	Yes	5	90	Yes	5	0,60	0.32	0.08	12.00

Fig. 16: Impact des obstacles et de l'occlusion

MODEL					SCENARIO			EVALUATION			
ID	Politique	FOV	Occlusion	Obstacles	FOV	Occlusion	Obstacles	Success Rate	Col. Hum. Rate	Col. Obs. Rate	Timeout Rate
3	dsmn	90	No	5	90	No	5	0,69	0,19	0,07	0,05
6	dsmn 2	90	No	5	90	No	5	0,70	0,21	0,04	0,05
7	dsmn 3	90	No	5	90	No	5	0,70	0,20	0,06	0,04
8	dsmn 4	90	No	5	90	No	5	0,70	0,15	0,14	0,01
9	dsmn 5	90	No	5	90	No	5	0,71	0,15	0,10	0,04

Fig. 17: Synthèse des modèles proposés

A partir de ce postulat, l'objectif de la suite du stage a été de proposer une nouvelle architecture, dérivée de DS-RNN, qui permet une différenciation entre obstacle et humain, et donc une politique plus adaptée à chaque situation.

3.5 Conception et évaluation d'architecture adaptées

Dans cette partie, nous allons présenter successivement les quatres architectures qui ont été proposées, et les raisons qui ont motivé nos choix. L'évaluation de ces modèles est synthétisée dans le tableau de la figure 17. Les valeurs ont été obtenues en moyennant les résultats de quatre séries tests de 500 épisodes chacune, effectuées sur des *seed* différentes (2000 épisodes en tout)

DS-RNN2 : Embedding des obstacles dans le RNN Spatial Nous avons tout d'abord essayé de séparer les positions des obstacles et des humains dès l'entrée du RNN Spatial (figure 10, figure 18). L'idée était d'avoir une couche linéaire dédiée aux obstacles, dans le but de donner l'opportunité au modèle d'apprendre des poids différents pour le traitement des obstacles.

Une concaténation permet ensuite de retrouver la dimension que l'on aurait dans DS-RNN à l'entrée du GRU.

Cette première approche, bien que naïve, avait le mérite de ne nécessiter aucun changement hors du RNN Spatial. Le taux de succès sur 2000 épisodes de test est de 0.70 soit légèrement plus que le taux de succès de DS-RNN de 0.69.

DS-RNN3 : Ajout d'un Spatial Obstacle RNN Dans cette seconde approche, nous avons souhaité dédier une plus grande partie de l'architecture aux obstacles. La séparation effectuée dans DS-RNN2 semblait insuffisante, car la politique obtenue était trop proche de celle du modèle de base.

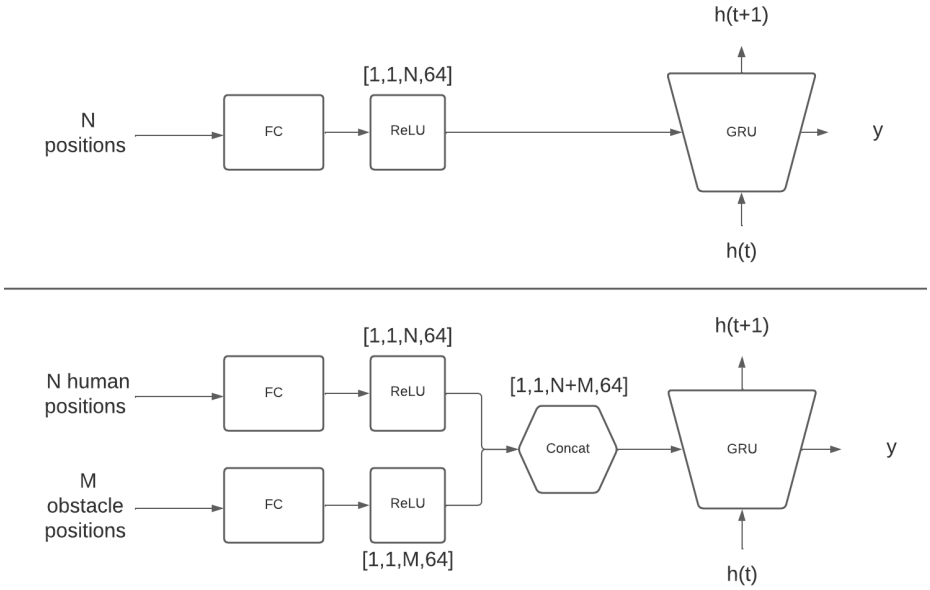

Fig. 18: Architecture du RNN Spatial. En haut : DS-RNN. En bas : DS-RNN2

Comme on peut la voir sur la figure 19, nous avons ajouté un nouveau RNN Spatial : $SpatialObstacleRNN$. Il se comporte comme le RNN Spatial du modèle de base, mais traite uniquement les positions des obstacles. le $SpatialHumanRNN$ s'occupe quand à lui de traiter les positions des humains.

En conséquence, l'*embedding* de la position des obstacles, en sortie de ce nouvel RNN, consistue une nouvelle entrée pour le module d'attention (figure 20). C'est à ce moment qu'on l'on concatène les $features$ des obstacles et celles des humains.

Ce modèle obtient des performances similaires au précédent, avec un taux de succès de 0.70

DS-RNN4 : Embedding des obstacles dans le *RobotStateRNN* Nous avons cherché par la suite à regrouper les $features$ associées aux obstacles et celles associées aux humains plus tard dans l'architecture.

Nous voulions notamment tester si il était possible d'intégrer directement les positions des obstacles juste avant le dernier RNN. Notre hypothèse était que la position des obstacles n'était pas une information assez complexe pour nécessiter un *embedding* via RNN et un calcul de poids d'attention.

La figure 21 montre l'architecture du $RobotStateRNN$. Comme on le voit en bas, nous avons fait le choix d'encoder la position des obstacles dans un vecteur d'*embedding* de taille 32. Ce vecteur est ensuite concaténé avec l'*embedding* de

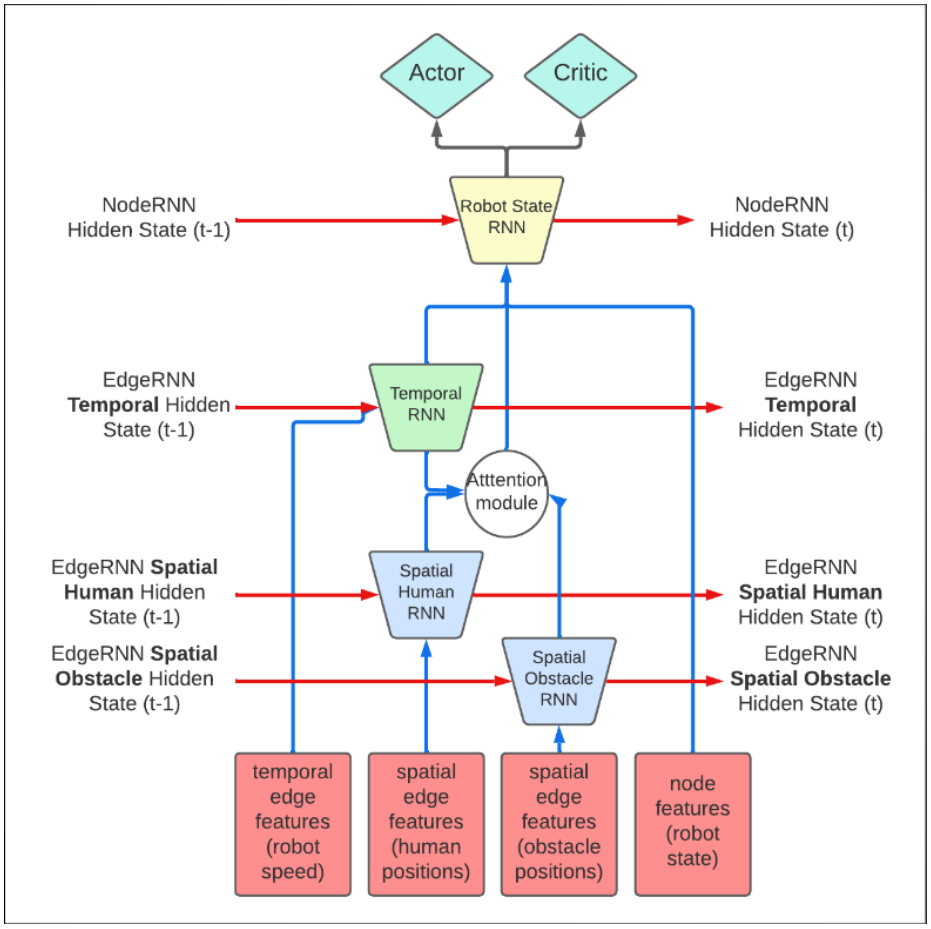

Fig. 19: Architecture de DS-RNN3

l'état du robot, dont l'expressivité a été réduite à 32, afin de maintenir le même nombre de paramètres que dans les modèles précédents.

Les performances, comme pour les précédents modèles restent très proche de DS-RNN.

DS-RNN5 : Ajout d'un Spatial Obstacle RNN et d'un Obstacle Attention Module Pour cette tentative de modification d'architecture, nous avons repris l'idée de DS-RNN3, mais nous avons cette fois-ci dédié un module d'attention aux obstacles. Comme on peut le voir sur la figure 22, la sortie du $SpatialObstacleRNN$ est désormais donnée en entrée du nouveau module d'attention.

Les *embedding* des obstacles et ceux des humains sont ensuite concaténés dans le $RobotStateRNN$. Afin d'obtenir la même dimension en entrée du GRU.

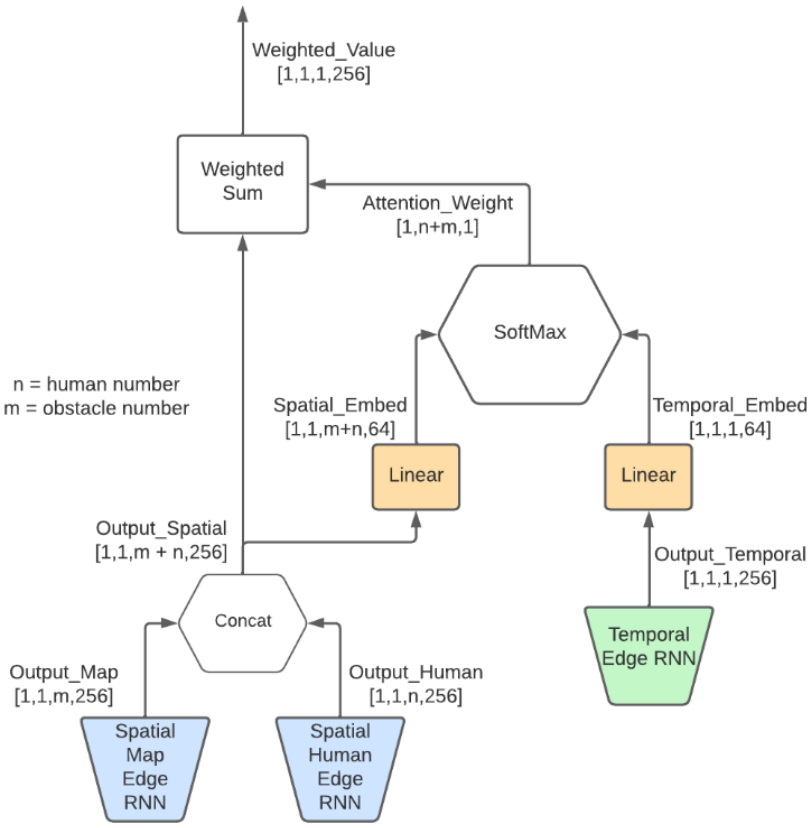

Fig. 20: Module d'attention de DS-RNN3

Nous avons choisi de diviser par deux l'expressivité des *embedding* de positions. Il s'agit d'un vecteur de taille 32 pour les humains comme pour les obstacles, soit un vecteur de taille 64 après concaténation.

Les performances de ce modèle sont légèrement supérieures à celles des modèles précédents, avec un taux de succès de 0.71 dans les même conditions.

DS-RNN6 : Altération de la fonction de récompense selon le type de collision Dans les précédents modèles, l'importance accordée aux obstacles est la même que celle accordée aux humains. Dans DS-RNN5 en particulier, l'expressivité des *embedding* est répartie à parts égales entre les humains et les obstacles. Nous avons souhaité ajouter la possibilité d'altérer l'importance accordée à chaque type de collision. Dans une situation réelle, on peut imaginer qu'il soit préférable de percuter un mur qu'un piéton.

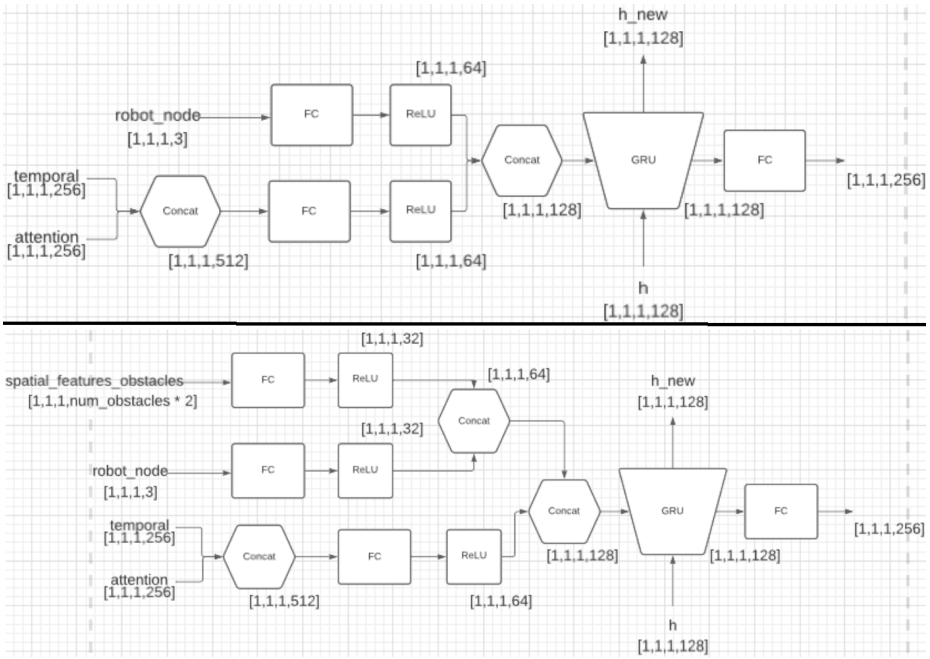

Fig. 21: Architecture du *RobotStateRNN*. En haut : DS-RNN. En bas : DS-RNN4

Nous avons repris DS-RNN5, et avons ajouté un hyper-paramètre permettant de choisir l'importance relative des humain par rapport aux obstacles. Nous avons entraîné un modèle, en choisissant une pénalité de collision dans la fonction de récompense deux fois plus faible dans le cas d'un obstacle.

Les performances ont significativement augmenté, le taux de succès passant de 0.71 à 0.75. Contrairement à ce que l'on aurait pu anticiper, le taux de collisions avec des obstacles n'a pas augmenté, bien que la pénalité associée dans la fonction de récompense soit beaucoup plus faible.

Conclusion sur les architectures proposées Nous avons proposé 4 modifications d'architectures dans l'objectif d'obtenir une politique d'évitement de collision différente selon la nature des obstacles. Au vu de la manière dont nous avons séparé les informations sur les obstacles et celles sur les humains, la politique devrait en théorie être différente selon le type d'obstacle. Mais en pratique, il a été difficile de mettre cela en évidence. On constate en effet une faible amélioration sur les performances des propositions par rapport au modèle de base, mais pas de preuve définitive que les obstacles soient traités différemment.

L'altération de la fonction de récompense semble déclencher une nette amélioration dans les performances du modèle. Une hypothèse serait qu'en accordant

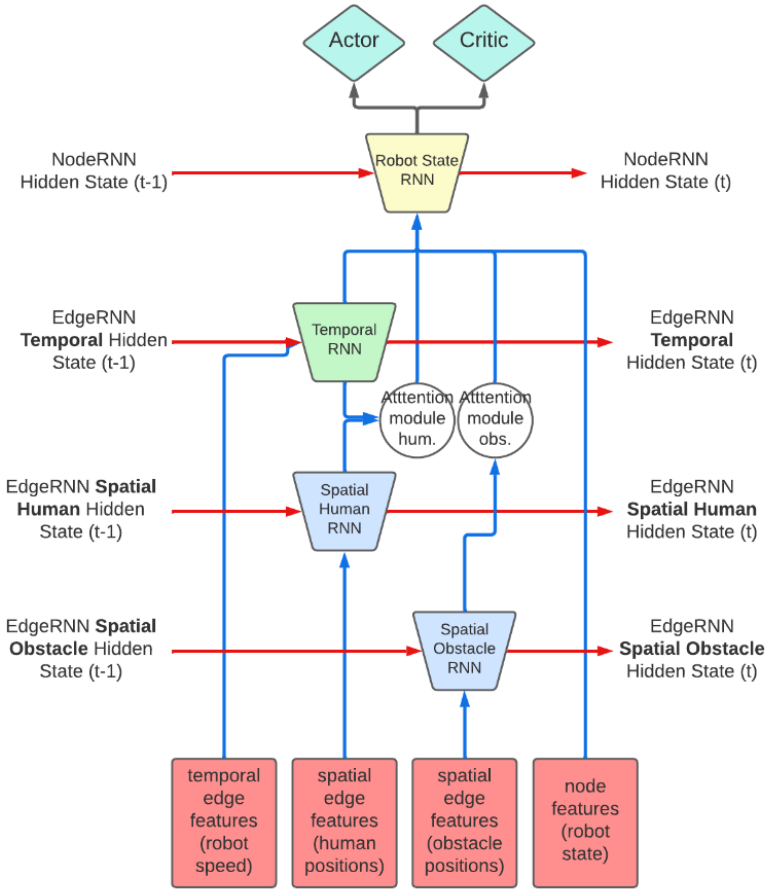

Fig. 22: Architecture de DS-RNN5

explicitement moins d'importance aux obstacles qu'aux humains, on obtient une politique plus différenciée comme on le souhaitait. Les obstacles statiques étant plus simples à éviter, le fait que la politique se concentre plus sur les déplacements humains est profitable pour ses performances. Cela n'explique pas en revanche pourquoi le taux de collisions avec des obstacles reste au même niveau que lorsque la pénalité de collision est toujours la même. D'autres expérimentations semblent nécessaires pour mieux comprendre les conséquences de ce changement, notamment pour savoir si l'altération de la fonction de récompense a le même impact sur les autres modèles.

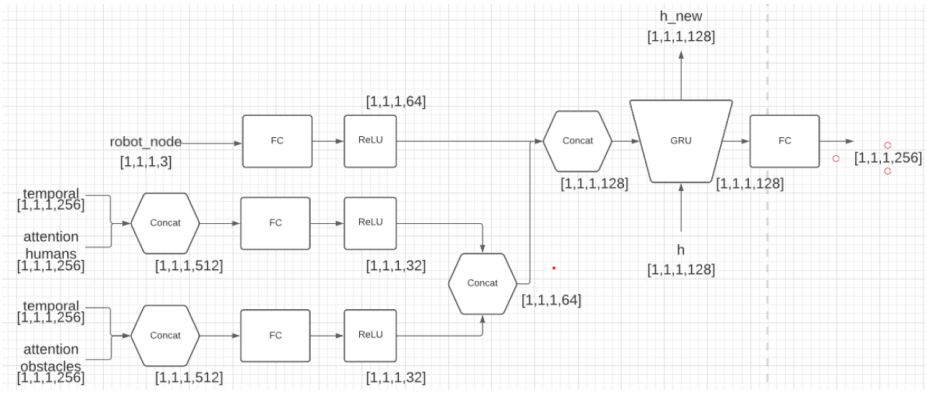

Fig. 23: Architecture du *RobotStateRNN* dans DS-RNN5

4 Discussion, conclusion et perspectives

Ce stage de recherche a eu une forte composante exploratoire. Nous avons commencé par l'état de l'art, qui nous a permis de construire une vision globale des approches existantes en terme de navigation robotique. Nous nous sommes appuyé sur cela pour définir des pistes de recherche possibles, et choisir nos objectifs suivants.

Nous avons décidé de nous baser sur le modèle DS-RNN pour la suite du stage. Ce modèle était assez complexe, et la compréhension détaillée de l'architecture globale et de chacun des modules était nécessaire pour l'étape suivante. Les représentations schématiques que nous avons dessinés (fig. 22, fig. 23) ont été d'une grande aide lorsqu'il s'agissait de proposer et implémenter des modifications d'architecture.

En totalité, nous avons entraîné une vingtaine de modèles différents, et les résultats des tests sur ces modèles sont regroupés dans un tableau d'une soixantaine de lignes, chaque ligne correspond à une évaluation sur 500 épisodes. Ces tests nous ont permis de mieux comprendre l'impact de chaque paramètre sur les performances des modèles, et de comparer les résultats des différents modèles. Les poids des modèles sont sauvegardés, et les expériences reproductibles.

En ce qui concerne les perspectives futures, il serait intéressant d'effectuer plus de tests avec le mécanisme d'occlusion, notamment sur les modèles proposés.

En conclusion ce stage aura eu un intérêt scientifique assez fort. Même si les performances des modèles proposés ne sont pas significativement plus élevées, nous avons réussi à mettre en défaut DS-RNN dans des conditions nouvelles. Nous avons exploré, implémenté et testé des possibilités d'architectures adaptées.

References

[1] Jur van den Berg, Ming Lin, and Dinesh Manocha. "Reciprocal Velocity Obstacles for real-time multi-agent navigation". In: *2008 IEEE International Conference on Robotics and Automation*. 2008 IEEE International Conference on Robotics and Automation. ISSN: 1050-4729. May 2008, pp. 1928–1935. DOI: 10.1109/ROBOT.2008.4543489.

[2] Changan Chen et al. "Crowd-Robot Interaction: Crowd-Aware Robot Navigation With Attention-Based Deep Reinforcement Learning". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019 International Conference on Robotics and Automation (ICRA). ISSN: 2577-087X. May 2019, pp. 6015–6022. DOI: 10.1109/ICRA.2019.8794134.

[3] Ashesh Jain et al. "Structural-RNN: Deep Learning on Spatio-Temporal Graphs". In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016, pp. 5308–5317. URL: https://openaccess.thecvf.com/content_cvpr_2016/html/Jain_Structural-RNN_Deep_Learning_CVPR_2016_paper.html (visited on 05/03/2022).

[4] Shuijing Liu et al. "Decentralized Structural-RNN for Robot Crowd Navigation with Deep Reinforcement Learning". In: *arXiv:2011.04820 [cs]* (June 3, 2021). arXiv: 2011.04820. URL: http://arxiv.org/abs/2011.04820 (visited on 02/21/2022).

[5] Jacques Saraydaryan, Fabrice Jumel, and Olivier Simonin. "Navigation in Human Flows : Planning with Adaptive Motion Grid". In: *IROS Workshop CrowdNav*. Paris, France, Oct. 2018. URL: https://hal.archives-ouvertes.fr/hal-03196208 (visited on 02/02/2022).

[6] Zhanteng Xie, Pujie Xin, and Philip Dames. "Towards Safe Navigation Through Crowded Dynamic Environments". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). ISSN: 2153-0866. Sept. 2021, pp. 4934–4940. DOI: 10.1109/IROS51168.2021.9636102.