École Nationale Supérieure d'Ulm

Rapport du stage de L3

Repérage spatial d'un robot mobile par filtrage particulaire et contrôle par recherche arborescente Monte-Carlo

Gabriel Belouze

sous la direction de
Olivier Buffet et Vincent Thomas

Centre de recherche INRIA Nancy Grand-Est
615 Rue du Jardin-Botanique, 54600 Villers-lès-Nancy

Juin-Juillet 2019

Je remercie l'équipe Larsen du laboratoire de l'INRIA Nancy Grand Est pour son accueil. Je remercie tout particulièrement Francis Colas pour m'avoir aidé à déchiffrer et résoudre des obscurs messages d'erreur de CMake, et pour son expertise Python. Merci enfin et surtout à mes maîtres de stage, Vincent Thomas et Olivier Buffet, non seulement pour leur aide précise, mais aussi pour les réunions hebdomadaires où ils n'ont pas hésité à discuter des ramifications des problèmes qui dépassaient le cadre du stage.

Table des matières

1 Abréviations

- **POMDP** : processus décisionnels de Markov partiellement observables *Partially Observable Markov Decision Process*
- **MCTS** : Recherche arborescente Monte-Carlo *Monte Carlo Tree Search*
- **HMM** : modèle de Markov caché *Hidden Markov Model*
- **SIS** : Sequential Importance Sampling
- **MDP** : Processus de Décision Markovien *Markov Decision Process*
- **PJA** : Problème-Jouet de l'Avion
- **UCB** : Upper Confidence Bound
- **UCT** : Upper Confidence bound applied to Trees
- **POMCP** : Partially Observable Monte-Carlo Planning
- **PFT-DPW** : Particle Filter Tree with Double Progressive Widening

2 Introduction

Dans le cadre de ma troisième année de license d'informatique à l'Ecole Nationale Supérieure d'Ulm, j'ai réalisé un stage de deux mois dans l'équipe Larsen de l'INRIA Nancy Grand Est. Larsen est une équipe de recherche à l'interface entre robotique et intelligence artificielle. Elle est spécialisée dans les interactions robot-humain et la planification en environnement incertain.

Mon stage s'inscrit dans le cadre des processus décisionnels de Markov partiellement observables (POMDP), pour lesquels un agent intelligent est amené à choisir une action à chaque pas de temps dans le but de maximiser sur le long terme une récompense. Il est intéresssant de développer des algorithmes efficaces de résolution de tels problèmes car les applications sont nombreuses : robotique, médecine, jeux ([1]). Ces problèmes sont complexes car l'agent ne connaît pas directement l'état du monde dans lequel il se trouve, mais accède seulement à des observations, éventuellement bruitées, qui en dépendent.

Le stage possède une dimension applicative : développer deux algorithmes pour la résolution de la localisation respectivement passive et active de Thymio, un robot mobile éducatif. Ces deux problèmes se définissent dans le même contexte : Thymio est plongé dans un monde dont il possède la carte, sans connaître ses propres coordonnées. Cependant il possède des capteurs qui lui apportent des informations sur son environnement proche, et peut se déplacer dans le monde.

Dans le problème de localisation passive, Thymio est contrôlé par l'utilisateur et doit utiliser les données capteurs pour déterminer la distribution de probabilité de ses coordonnées (x, y, θ) dans leur espace de valeurs. Ce problème se formalise facilement dans le cadre des modèles de Markov cachés (HMM), qui généralise les chaînes de Markov lorsque l'état courant n'est pas directement accessible, mais est observé selon un second processus stochastique.

Le problème de localisation active est plus difficile : Thymio doit déterminer lui-même la trajectoire à suivre pour "se localiser" le plus rapidement possible, c'est-à-dire restreindre la distribution de probabilité de ses coordonnées autour d'un ou quelques points. Ce problème se formalise bien dans le cadre POMDP, où l'ensemble d'états correspond à l'ensemble des coordonnées possibles pour Thymio, l'ensemble d'observation est l'ensemble des valeurs que les capteurs peuvent prendre, et l'ensemble d'actions correspond aux différentes commandes moteur possibles. L'objectif d'un algorithme efficace pour ce problème consiste à déterminer une trajectoire quasi-optimale pour la localisation.

Les algorithmes de type MCTS constituent une classe populaire d'algorithmes pour résoudre des POMDP. Les caractéristiques des ensembles d'états, d'observations et d'actions sont déterminantes de la difficulté du problème associé : le cadre fini est bien étudié ([5], [11]) et des approches récentes ont proposé des variantes pour POMDP avec un ensemble d'états continu ([12]) ou lorsque la récompense dépend de la connaissance du monde ([13]). Un des objets de

mon stage est d'étudier ces deux derniers algorithmes, puis de développer et d'implémenter un algorithme dérivé, destiné à être appliqué au problème de localisation active pour Thymio. Un tel algorithme doit être capable de gérer d'une part un ensemble d'états continu, un ensemble d'observations continu, éventuellement un ensemble d'action continu, et d'autre part une récompense dépendant de la connaissance du monde.

Au moment de mon arrivée à INRIA Nancy Grand Est, trois étudiants de M1 avaient déjà travaillé sur les méthodes de communication avec Thymio, et sur un algorithme de type filtrage particulaire pour la localisation passive de Thymio. Leur travail n'étant pas terminé, l'objectif de mon stage était double : compléter le travail déjà abordé pour obtenir un algorithme pour la localisation passive de Thymio, et développer un algorithme de contrôle pour la localisation active. L'algorithme de contrôle doit être générique, c'est-à-dire qu'il doit être adaptable à n'importe quel POMDP présentant des caractéristiques compatibles. Je l'ai donc implémenté sous forme d'une bibliothèque Python. Celle-ci peut être utilisée entre autres pour la localisation du Thymio. Le choix du langage de programmation est contraint par deux facteurs : d'une part la programmation de Thymio se fait via le méta système d'exploitation ROS qui attend des programmes C++ ou Python, d'autre part, parmi ces deux langages, je ne maîtrise que Python. Par ailleurs, les étudiants de M1 ayant travaillé en C++, j'ai dû de fait reprendre depuis le début l'algorithme de filtrage particulaire, également implémenté pour être générique, c'est-à-dire adaptable à n'importe quel HMM continu.

La section 3 présente un état de l'art sur les algorithmes de Monte-Carlo pour le filtrage et le contrôle. La section 4 présente le cadre applicatif : le robot Thymio et les méthodes de programmation robotique. Les sections 5 et 6 exposent le travail réalisé, respectivement pour le problème de localisation passive et pour le problème de localisation active. La section 7 détaille la mise en oeuvre des algorithmes développés sur Thymio. Enfin la section 8 présente des perspectives intéressantes à développer pour poursuivre le travail réalisé.

3 État de l'art

3.1 Modèle de Markov Caché - Filtrage Particulaire

Definition

Le formalisme Modèle de Markov Caché (Hidden Markov Model - HMM) permet de modéliser une évolution stochastique d'un système partiellement observable. Un agent suit un processus markovien et se retrouve successivement dans les états $s_0, s_1, \ldots, s_n, \ldots$, où $s_i \in \mathcal{S}$ désigne la valeur de l'état pris à l'instant i. On note S_i la variable aléatoire associée au i-ième état pris par l'agent. S_0 suit la distribution intiale b_0. À chaque pas de temps, le nouvel état est uniquement dépendant de l'état courant de l'agent selon la loi conditionnelle $T(s_{t+1}, s_t) = f(s_{t+1}|s_t)$. Je précise ici que cette dernière notation $f(s_{t+1}|s_t)$ exprime que $T(\cdot, s_t)$ est une loi de probabilité sur $\mathcal{S}$. De plus, pour rendre compte de l'erreur des capteurs, l'agent ne peut pas accéder directement aux états, mais il observe $o_0, o_1, \ldots, o_n, \ldots$, valeurs d'observation prises dans $\mathcal{O}$. On note O_i les variables aléatoires correspondantes. o_t dépend uniquement de s_t l'état de l'agent au temps t, selon $\mathcal{Z}(o_t, s_t) = f(o_t|s_t)$. Si les observations sont déterministes, c.-à-d. $O_t = f(S_t)$, avec f injective, on retrouve les conditions d'une chaîne de Markov où les observations ne sont jamais plus qu'un alias de l'état antécédent par f. HMM est donc une généralisation des chaînes de Markov.

Classiquement, l'étude d'un HMM $\lambda = \langle \mathcal{T}, \mathcal{Z}, b_0 \rangle$ se centre autour de trois objectifs :

1. Problème 1 : étant donnés les paramètres de λ, déterminer $\mathbb{P}(O|\lambda)$ la probabilité d'observer une séquence $O = o_1, o_2, \ldots$.
2. Problème 2 : étant donné O, trouver une séquence d'état de λ optimale pour O, dans un sens à préciser. Souvent, il s'agit de trouver $s_0, \ldots, s_n$ qui maximisent $\mathbb{P}(O \mid s_0, \ldots, s_n)$.
3. Problème 3 : étant donnés $\mathcal{S}$, $\mathcal{O}$ et une séquence d'observations O, trouver les paramètres de λ qui maximisent $\mathbb{P}(O|\lambda)$.

Le modèle HMM de la localisation passive de Thymio correspond à

$$\mathcal{S} = \{(x, y, \theta)\}$$

l'ensemble des coordonnées possibles de Thymio (x, y sont les coordonnées cartésiennes et θ représente l'orientation du robot), et

$$\mathcal{O} = \textit{ensemble des données capteur.}$$

Le problème de localisation à résoudre s'exprime comme la marginale du Problème 2, c'est-à-dire qu'étant donné $O = o_1, \ldots, o_n$, il s'agit de trouver la distribution $\mathbb{P}(S_n|o_1, \ldots, o_n)$, soit la distribution sur l'état de l'agent au temps n. À noter que dans ce problème la fonction de transition T dépend du temps t, et plus précisément dépend de la vitesse des roues imposée parl'utilisateur au temps t.

Filtre particulaire

HMM à ensemble d'états fini est bien étudié. Une première approche consiste donc à discrétiser l'espace d'états de Thymio. Alors $\mathcal{S} = \{q_1, \ldots, q_m\}$ et on peut envisager au temps n de faire un calcul exact des $\gamma_t(i) = \mathbb{P}(S_t = q_i|o_0, \ldots, o_t)$ pour $1 \leq i \leq m$, $0 \leq t \leq n$. Ces variables peuvent être déterminées à partir des variables "forward"

$$\alpha_t(i) = \mathbb{P}(O_0 = o_0, O_1 = o_1, \ldots, O_t = o_t, S_t = q_i)$$

et "backward"

$$\beta_t(i) = \mathbb{P}(O_{t+1} = o_{t+1}, \ldots, O_n = o_n \mid S_t = q_i)$$

qui sont calculées en $O(m^2 n)$ ([7], [9]). La complexité augmente vite si l'on cherche à augmenter la précision lors de la discrétisation de l'espace.

On préfère utiliser un filtrage particulaire, lequel fournit une estimation de la densité de probabilité $\mathbb{P}(S_t|o_0, \ldots, o_t)$. L'objectif du filtrage particulaire est d'estimer une distribution en générant des particules selon une autre distribution, puis en leur attribuant des poids correctifs. [3], [2]

Supposons que l'on puisse échantillonner N particules indépendantes $s_t^{(1)}, \ldots, s_t^{(N)}$ selon la distribution $\mathbb{P}(S_t|o_0, \ldots, o_t)$. Une estimation empirique de cette distribution est alors fournie par

$$\frac{1}{N} \sum_{i=1}^{N} \delta_{s_t^{(i)}}$$

où $\delta_{s_t^{(i)}}$ désigne la fonction de Dirac localisée en $s_t^{(i)}$. Cependant, il n'est généralement pas aisé de générer selon $\mathbb{P}(S_t|o_0, \ldots, o_t)$.

Une alternative classique est *Sequential Importance Sampling* (SIS). Générons N particules $s_t^{(1)}, \ldots, s_t^{(N)}$ selon la loi de S_t, et considérons les poids $\omega_t^{(1)}, \ldots, \omega_t^{(n)}$ définis par

$$\omega_t^{(i)} = \frac{\mathbb{P}(S_t|O_t)}{\mathbb{P}(S_t)}$$

Alors une bonne estimation de $\mathbb{P}(S_t|o_0, \ldots, o_t)$ est donnée par

$$\widehat{\mathbb{P}}_t = \sum_{i=1}^{N} \hat{w}_t^{(i)} \delta_{s_t^{(i)}}$$

où $\hat{w}$ est le poids normalisé. Cette estimation est bonne dans le sens où pour toute fonction f_t intégrable par rapport à $\mathbb{P}(S_t|o_0, \ldots, o_t)$, on a (voir [2])

$$\int f_t(S_t) \widehat{\mathbb{P}}_t \, dS_t \underset{N \to +\infty}{\to} \int f_t(S_t) \mathbb{P}(S_t|o_0, \ldots, o_t) \, dS_t$$

La génération des N particules reste problématique. Fort heureusement, la loi de Bayes nous permet de formuler une méthode récursive utilisable en pratique : étant donnés une nouvelle observation o_{t+1}, les particules et les poids au temps t, $s_{t+1}^{(i)}$ peut être générée à partir de $T(\,\cdot\,, s_t^{(i)})$, et

$$w_{t+1}^{(i)} \sim w_t^{(i)} \mathbb{P}(o_{t+1} \mid s_{t+1}^{(i)})$$

Un problème que rencontre SIS est que les poids des particules ont tendance à se concentrer, si bien qu'après quelques pas de temps, tout le poids est concentré sur une particule. *Bootstrap Filter* palie ce problème en ajoutant une étape de sélection à chaque pas de temps. Formellement, on remplace l'estimation de la distribution $\widehat{\mathbb{P}}_t$ par une distribution sans poids

$$\widehat{\mathbb{P}'}_t = \frac{1}{N} \sum_{i=1}^{N} N_t^{(i)} \delta_{s_t^{(i)}}$$

où les $N_t^{(i)}$ correspondent au nombre de descendants de la i-ième particule. Les $N_t^{(i)}$ sont déterminés en échantillonnant N fois selon la distribution discrète $\widehat{\mathbb{P}}_t$.

3.2 Processus de Décision Markovien - UCT

La notion de contrôle du robot n'est pas intégrée aux HMM, il faut donc changer de modèle.

Définition

Le modèle des processus de décision markoviens (Markov Decision Process - MDP) permet de modéliser un agent qui effectue des choix d'action dans $\mathcal{A}$ en vue de maximiser un critère prédéfini. À chaque pas de temps t, l'agent se trouve dans l'état s_t et choisit l'action a_t. L'agent se déplace alors vers l'état s_{t+1} selon $T(s_{t+1}, a_t,\ s_t) = f(s_{t+1} \mid s_t, a_t)$. Ce faisant il reçoit la récompense $r_t = r(s_t, a_t) \in \mathbb{R}$. r peut être une fonction déterministe ou probabiliste, peut également dépendre de s_{t+1}. Assez logiquement, je désignerai par A_t et R_t les variables aléatoires associées respectivement à a_t et r_t. Plus formellement, un MDP est un tuple $\langle T, r, b_0 \rangle$. [8] [6]

L'action prise par l'agent à chaque pas de temps est déterminée par une politique π dépendante de l'état courant : $A_t = \pi(S_t)$.

Résoudre un MDP consiste à déterminer une politique optimale π^* qui maximise typiquement l'espérance de la récompense totale

$$v^\pi = \mathbb{E}^\pi \left(\sum_{t=1}^{N} R_t \right) = \mathbb{E} \left(\sum_{t=1}^{N} r(S_t, \pi(S_t)) \right)$$

Lorsque le processus est infini, on utilise plutôt le critère γ-atténué

$$v_\gamma^\pi = \mathbb{E}^\pi \left(\sum_{t=1}^{+\infty} \gamma^t R_t \right)$$

avec $0 < \gamma < 1$. On suppose généralement que r est bornée, pour garantir que $v_\gamma(\pi)$ est finie. Dans la suite, je me restreins à cette hypothèse.

On note $v^* = \sup_\pi v^\pi$ ($= v^{\pi^*}$ lorsque π^* existe). On définit v_γ^* de même.

Il est aussi utile de définir la fonction de valeur $V^\pi : \mathcal{S} \to \mathbb{R}$ associée à une politique. Cette fonction associe à un état s l'espérance de la récompense totale lorsque le processus commence en s

$$V^\pi : s \longmapsto \mathbb{E}^\pi \left(\sum_{t=0}^{N} R_t \mid S_0 = s \right)$$

De même est définie V_γ^π. On parle de la *valeur* d'un état s pour désigner $V^{\pi^*}(s)$. Par concision on utilisera plutôt la notation $V(s)$, ou $V_\gamma(s)$ pour le critère γ-atténué. De façon naturelle, on

parle également de la valeur d'une action a depuis un état s, et on la note $V(s,a)$. Celle-ci correspond à l'espérance de la récompense totale si a est choisie depuis s, puis que π^* dirige le reste du processus (intuitivement, cela correspond à la meilleure valeur moyenne que l'on peut espérer obtenir si l'on choisit a en s).

Une action a est sous-optimale si $V(s,a) < \sup_{a' \in \mathcal{A}} V(s,a')$.

Upper Confidence bound applied to Trees (UCT)

Lorsque les ensembles d'états et d'actions sont finis, il est clair qu'une politique optimale existe (l'ensemble des politiques est fini). Le cas continu est moins clair, mais l'on se contente de l'existence d'une politique ε-optimale pour tout $\varepsilon > 0$, qui elle est acquise (une politique π est ε-optimale si $v(\pi) > v^* - \varepsilon$). UCT est un algorithme de type MCTS (Monte Carlo Tree Search) avec simulation qui permet d'approcher une politique optimale. Il présente deux qualités intéressantes : si l'algorithme est arrêté prématurément, la politique obtenue est proche de la politique optimale $v(\pi) \sim v^*$, et sous certaines hypothèses, l'algorithme converge vers une politique optimale si assez de temps lui est laissé [5].

Les algorithmes de type MCTS construisent l'arbre des futurs possibles à partir d'un état de façon incrémentale. Dans l'application à la résolution d'un MDP, les noeuds correspondent à des états, et les arêtes à des actions. Chaque itération ajoute une arête et un noeud à l'arbre, et se décompose en 4 étapes :

i **Descente** : Un chemin partant de la racine est échantillonné selon une heuristique à déterminer, jusqu'à ce qu'un noeud qui n'appartient pas à l'arbre soit atteint.

ii **Expansion** : Ce nouveau noeud est ajouté à l'arbre.

iii **Simulation** : La valeur du noeud est estimée. Ceci est fait en échantillonnant une trajectoire depuis ce noeud selon une stratégie à déterminer.

iv **Remontée** : La valeur estimée est utilisée pour mettre à jour l'estimation de la valeur des noeuds rencontrés lors de la descente.

Pour choisir une action optimale, il suffit de connaître la valeur des actions en l'état donné. Puisque ces valeurs dépendent des valeurs de l'état successeur, le problème se ramène à déterminer rapidement une bonne estimation de la valeur d'un état. Pour atteindre cet objectif, un algorithme doit à la fois explorer l'action qui semble pour l'instant la meilleure pour affiner l'estimation de la valeur d'un état, et explorer les alternatives qui semblent sous-optimales pour être sûr que de meilleures actions ne sont pas ignorées à cause d'erreur d'estimation. Ces deux besoins sont de toute évidence contradictoire, l'enjeu est donc de déterminer une bonne heuristique d'équilibre. L'idée d'UCT est d'utiliser le critère *Upper Confidence Bound* (UCB) pour réaliser cet équilibre lors de la phase descente de MCTS.

UCB attribue à chaque action un mérite, défini par

$$\widehat{V}(s,a) + C_p \sqrt{\frac{\log\ N(s)}{N(s,a)}}$$

où $\hat{V}(s,a)$ est l'estimation courante de la valeur de l'action a prise au noeud s (0 si elle n'a jamais été prise), C_p est la constante d'exploration à choisir correctement, $N(s)$ est le nombre de fois que le noeud s a été visité et $N(s,a)$ est le nombre de fois où l'action a a été choisie depuis ce noeud. Puis UCB choisit l'action qui maximise le mérite. Cette heuristique favorise les actions qui ont une bonne valeur estimée, mais aussi qui ont été peu explorées.

3.3 Processus de Décision Markovien Partiellement Observable - POMCP

Le formalisme MDP ne permet pas de modéliser directement le problème de localisation active de Thymio puisque ce dernier n'a pas accès directement à son état. Le formalisme suivant modifie MDP pour intégrer la notion d'observation, de la même façon que l'on a modifié les chaînes de Markov dans le formalisme HMM.

Définition

Les processus de décision markoviens partiellement observables (Partially Observable Markov Decision Process - POMDP) sont semblables aux MDP, mais l'agent n'a plus accès à son état. En revanche, à chaque pas de temps t, l'agent reçoit une observation o_t de son état s_t, selon la distribution $Z(o_t, s_t) = f(o_t \mid s_t)$. Formellement, un POMDP est un tuple $\langle T, Z, r, b_0 \rangle$.

La notion de politique π développée pour le formalisme MDP se généralise naturellement au POMDP, simplement π est maintenant une fonction de $\mathcal{O}$ dans $\mathcal{A}$. Les critères de performance v^π et v^π_γ sont également naturellement définis pour un POMDP. De façon un peu surprenante, mais très utile, un POMDP peut être considéré comme un MDP sur un espace d'états bien choisi.

États de croyance et MDP de croyance Formellement, un état de croyance est une distribution de probabilité sur l'ensemble d'états $\mathcal{S}$. On note $\mathcal{B}$ l'ensemble des états de croyance. Informellement, il correspond à la probabilité d'être dans tel ou tel état du point de vue de l'agent. À chaque pas de temps, l'agent peut calculer l'état de croyance dans lequel il se trouve après l'historique d'observations reçues et d'actions prises. Il se trouve que du point de vue de l'agent, la situation est strictement équivalente à celle d'un MDP où l'espace d'états est $\mathcal{B}$ [10] : après t pas de temps, l'agent se retrouve dans l'état de croyance $b_t = \mathbb{P}(S_t | o_0, a_0, \ldots, a_{t-1}, o_t)$. La loi de Bayes permet d'établir une relation de récurrence sur b_t. En particulier, l'agent n'a pas besoin de garder en mémoire l'historique des actions prises et des observations reçues. Remarquons que le problème de garder à jour l'état de croyance courant est esesntiellement similaire au problème de calculer la marginale $\mathbb{P}(S_t | o_0, \ldots, o_t)$ dans un HMM 3.1. On appelle MDP de croyance le MDP à espace d'états $\mathcal{B}$ associé à un POMDP.

POMCP

L'application directe de UCT au MDP de croyance associé peut paraître attractive, mais le coût computationnel des mises à jour exactes de l'état de croyance rend impossible cette approche. L'algorithme *Partially Observable Monte-Carlo Planning* (POMCP) [11] contourne cette difficulté en échantillonnant des trajectoires d'états plutôt que d'états de croyance. Conceptuellement, les noeuds de l'arbre de recherche correspondent donc plutôt à un historique de paires action-observation. La phase descente ne requiert donc plus qu'un simulateur fonctionnant en boîte noire qui, étant donné un état s et une action a, génère un état suivant s', une observation o et une récompense r : le premier état est échantillonné selon b_0, puis le simulateur est utilisé pour le reste de la trajectoire. Naturellement, on attend que le simulateur génère selon le modèle sous-jacent correspondant au POMDP. Mise à part cette subtilité, le fonctionnement de POMCP ressemble fortement à UCT. En particulier, l'heuristique de choix d'action est UCB.

Par ailleurs, POMCP fournit une estimation de l'état de croyance associé à chaque historique en gardant en mémoire les états pris pour un historique donné lors des descentes. En particulier, cela permet d'utiliser POMCP comme algorithme en ligne : une fois une action effectivement prise et une observation effectivement reçue, POMCP utilise l'estimation de l'état de croyance correspondante comme nouveau b_0.

4 Thymio

4.1 Description

Thymio est un robot mobile à but éducatif. Il dispose de 9 capteurs de proximité laser, d'une portée de 10cm environ, cinq à l'avant disposés en arc de cercle, deux à l'arrière, et deux en dessous dirigés vers le sol (figure 1). Thymio possède deux roues indépendantes, chacune contrôlée par un moteur. Thymio est associé à un langage basique de programmation, Aseba, et à un logiciel de simulation, Aseba Playground.

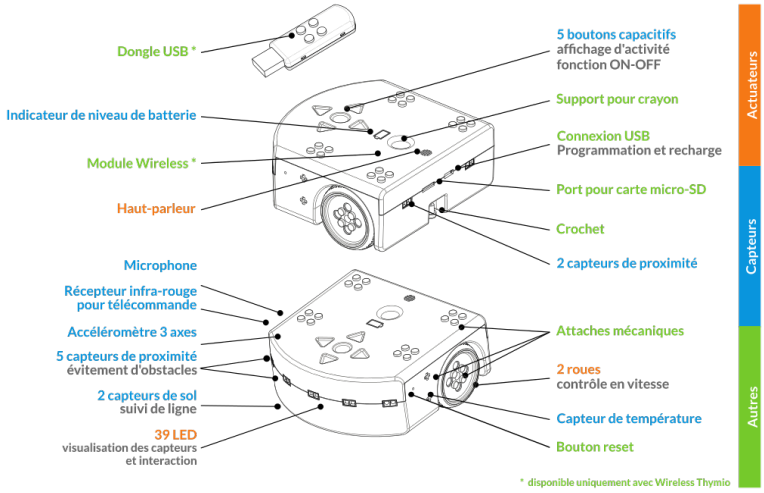


FIGURE 1 – Composants de Thymio

4.2 Communication avec Thymio

Aseba est trop limité pour implémenter ne serait-ce que la localisation passive de Thymio. Mes maîtres de stage m'incitent donc à utiliser ROS.

4.2.1 ROS

ROS (Robot Operating System) est un méta système d'exploitation utilisé pour la mise en place de programme robotique. ROS a pour but de permettre facilement la mise en communication de différents processus. Les processus forment les noeuds du graphe orienté de communication, et les services et topics en forment les arêtes. Les topics peuvent être considérés comme des bus de communication nommés et fortement typés. Ils sont adaptés pour les communications *many-to-many* et permettent conceptuellement de découpler la production d'un message de son utilisation. Les services sont adaptés aux communication *one-to-one* de type requête-réponse. Les services sont également nommés et typés.

4.2.2 Asebaros

Un pont Aseba-ROS, sous forme d'un package ROS appelé Asebaros, a été prévu par les développeurs de Thymio pour accéder à quelques informations des capteurs Thymio et pouvoir lui instruire des commandes moteurs. Un tel pont prend la forme d'un noeud ROS qui se connecte au logiciel Aseba pour accéder aux capteurs et moteurs de Thymio. Mes maîtres de stage ont modifié ce pont pour qu'il fournisse toutes les informations nécessaires, à savoir les données des 7 capteurs horizontaux. L'utilisation d'Asebaros constitue une difficulté non négligeable, mais avait déjà été mise en place par mes maîtres de stage et les trois stagiaires de M1.

5 Localisation passive

Le problème de localisation passive correspond à l'estimation de la position de Thymio à partir des données reçues de ses capteurs. Plus précisément, Thymio est plongé dans un monde dont il connait la carte, mais il n'a accès ni à son orientation, ni à ses coordonnées cartésiennes. Cependant, il reçoit des données de ses capteurs lattéraux donnant une distance bruitée des obstacles de son entourage proche. Un utilisateur contrôle Thymio (en lui envoyant des commandes moteurs), qui se déplace donc dans son environnement. Thymio a également

accès aux vitesses de rotation de ses roues. À partir de la séquence temporelle des informations mentionnées, Thymio doit déterminer la distribution du triplet $x, y, \theta)$.

5.1 Formalisation HMM

Soit $\mathcal{P}$ le monde dans lequel évolue Thymio, vu comme un sous ensemble de $\mathbb{R}^2$. On considère le problème de localisation passive comme un HMM où

- $\mathcal{S}$ l'ensemble des états est $\{(x, y, \theta) \mid (x, y) \in \mathcal{P},\ \theta \in [0, 2\pi[\}$
- $\mathcal{O}$ l'ensemble des observations est $\mathbb{R}_+^7$ où une coordonnée représente le taux de stimulation du capteur lattéral correspondant.
- b_0 est uniforme sur $\mathcal{S}$
- T est à déterminer selon la dynamique du robot. Rappelons qu'ici, T au temps t dépend de l'action choisie par l'utilisateur au temps t.

5.2 Bootstrap filter

J'ai implémenté un algorithme de type *Bootstrap Filter* générique sous forme d'une bibliothèque Python.

5.3 Calibration de l'algorithme

5.3.1 Le Problème-Jouet de l'Avion (PJA)

On introduit le Problème-Jouet de l'Avion (PJA) comme une version simplifiée de la situation du robot. Un avion vole en ligne droite et à vitesse constante (moyennant un certain bruit) au-dessus de montagnes plongées dans le brouillard. L'avion connaît la carte topographique de la zone qu'il survole, connaît sa hauteur absolue (par rapport au niveau de la mer) et souhaite déterminer sa position. Il a pour cela à sa disposition un capteur qui lui donne sa hauteur immédiate au sol (moyennant un certain bruit gaussien). La situation est donc un problème de repérage à une dimension (abscisse de l'avion) modélisable par un HMM dont l'ensemble des états est $\mathbb{R}$, c.-à-d. l'ensemble des abscisses de l'avion possibles, et l'ensemble d'observation est $\mathbb{R}$, soit l'ensemble des mesures de hauteurs au sol possibles. Pour simplifier le problème davantage et avoir un ensemble d'états borné, on considère que l'avion se déplace sur un cylindre, c'est-à-dire que la carte topographique est 1-périodique. L'ensemble des états devient donc $[0, 1[$.

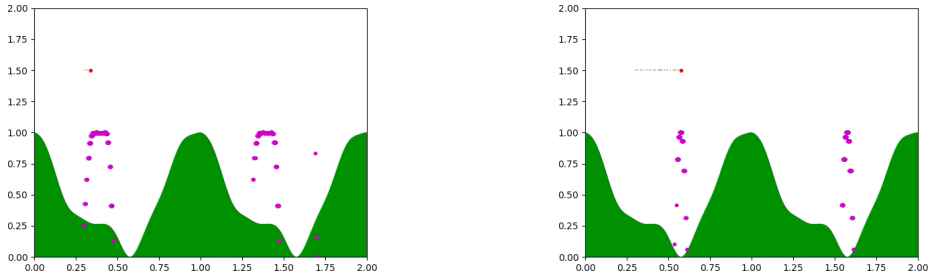
La génération de la carte topographique est faite de façon aléatoire à partir de bruit de Perlin.

5.3.2 Calcul à partir des variables forward et backward

Dans un soucis de vérification de la correction du bootstrap filter d'une part, et de comparaison de performances d'autre part, j'ai implémenté un calcul exact offline des variables $\gamma_t(i)$ (voir 3.1) après discrétisation de l'ensemble d'états de l'avion.

5.3.3 Résultats

Le bootstrap filter et le calcul exact de la distribution de l'état de l'avion donnent des résultats sensiblement identiques. J'ai constaté également que cette distribution s'applatit lorsque l'avion survole des "plateaux" (zones de hauteur quasi constante), ce qui confirme l'intuition que la mesure de nouvelles hauteurs apporte peu d'information, et que par conséquent les bruits de déplacement s'ajoutent 2a. J'ai vérifié par ailleurs que sans l'étape de sélection du *bootstrap filter*, le poids des particules s'effondre pour ne laisser plus qu'une particule dominante, ce qui était annoncé par la littérature.



(a) Autour d'une zone à faible variation de hauteur, la distribution s'élargit.

(b) Au niveau d'une vallée, la distribution a une variance faible.

FIGURE 2 – Différents stades du filtrage particulaire. Les montagnes sont exactement 1-périodiques et donc la distribution estimée l'est aussi. Le point rouge montre la vraie position de l'avion. Les abscisses des points violets correspondent à des particules et les ordonnées sont proportionnelles au poids de la particule correspondante.

6 Localisation active

Le contexte de la localisation active est sensiblement le même que celui de la localisation passive, mais l'objectif est différent. Thymio est placé dans une position aléatoire dans un monde dont il connaît la carte. Cette fois-ci, il n'y plus d'utilisateur. Thymio contrôle lui-même ses actions moteurs. Son but est de suivre une trajectoire qui lui permet de "se localiser" le plus rapidement possible. La notion de "se localiser" est à préciser, mais elle traduit en essence le fait que la distribution de probabilité de (x, y, θ) est focalisée autour d'un seul point. Une traduction possible de cette idée est que Thymio doit chercher à minimiser l'entropie de cette distribution. Précisons que Thymio a toujours accès aux vitesses de rotation de ses roues et aux données fournies par ses capteurs latéraux.

6.1 Formalisation POMDP

On considère le problème de localisation active comme un POMDP où

- $\mathcal{S}$ l'ensemble des états est $\{(x, y, \theta) \mid (x, y) \in \mathcal{P},\ \theta \in [0, 2\pi[\}$
- $\mathcal{A}$ l'ensemble des actions est $[-1, 1] \times [-1, 1]$. Une action (x, y) correspond à imposer une vitesse linéaire de x et une vitesse angulaire de y.
- $\mathcal{O}$ l'ensemble des observations est $\mathbb{R}_+^7$ où une coordonnée représente le taux de stimulation du capteur lattéral correspondant.
- b_0 est uniforme sur $\mathcal{S}$.
- r la fonction de récompense est une fonction à définir sur l'ensemble des états de croyance $\mathcal{B}$. Elle peut par exemple être égale à la fonction entropie, ou à la fonction variance.
- T et Z les fonctions de transition et d'observation sont à déterminer selon la dynamique de Thymio.

6.2 ρ-POMCP, PFT-DPW

POMCP n'est pas adapté au problème de localisation active pour deux raisons.

Difficulté de la récompense dépendante de la croyance D'une part, la fonction récompense n'est pas définie sur $\mathcal{S}$ mais sur $\mathcal{B}$. Il est donc crucial d'avoir une bonne approximation des états de croyance, ce qui n'est pas le cas lors des premières itérations de POMCP. Les POMDP à récompense sur les états de croyance, ou ρ-POMDP, ont été étudiés par Buffet, Thomas et Hutin [13] qui ont développé un nouvel algorithme ρ-POMCP. Lors des phases de descente de ρ-POMCP, un algorithme de filtrage particulaire est utilisé et les particules sont ajoutées dans

chaque noeud rencontré pour obtenir rapidement une bonne approximation de l'état de croyance correspondant.

Difficulté de la continuité des espaces D'autre part, l'ensemble d'observation est continu : la probabilité d'obtenir deux fois la même observation par le simulateur est 0. L'arbre d'exploration construit par POMCP est donc de profondeur 1 et sa largeur explose. Remarquons également que le critère UCB favorise avant tout les actions qui n'ont jamais été choisies. Si l'espace d'action est grand (ou pire, continu), une nouvelle action est choisie à chaque passe de l'algorithme. Sunberg et Kochenderfer ont proposé un algorithme *Particle Filter Tree with Double Progressive Widening* (PFT-DPW) qui permet de traiter des espaces d'état, d'action et d'observation continus à l'aide d'un double élargissement progressif [12]. L'élargissement progressif consiste à limiter artificiellement le nombre d'enfants d'un noeud à kN^α où N est le nombre de fois que le noeud a été visité, et k et α sont des méta-paramètres d'élargissement à déterminer. Dans PFT-DPW, l'élargissement progressif est utilisé à la fois pour l'espace d'états et l'espace d'actions. Une autre caractéristique intéressante est que PFT-DPW échantillonnent des trajectoires d'états de croyance plutôt que d'états. Comme nous allons le voir, PFT-DPW permet déjà de traiter des ρ-POMDP et est de fait déjà adapté au problème de la localisation active.

Algorithme utilisé Décrivons plus précisément le fonctionnement de PFT-DPW. PFT-DPW est de type MCTS et son schéma global ressemble beaucoup à celui POMCP, en ce que chaque itération se décompose en une phase de descente, d'expansion, de simulation et de remontée. La différence fondamentale réside en ce que les trajectoires échantillonnées sont directement sur les états de croyance. Pour générer ces trajectoires, PFT-DPW utilise un filtre particulaire G_{PF} à m particules, où m est un hyper-paramètre à fixer. Le besoin de fixer m peut sembler limitant, mais en pratique il semble qu'il n'est pas difficile de trouver une valeur convenable (une valeur typique est $m = 20$) [12].

Seul le déroulement de la phase de descente mérite plus de précision. L'arbre d'exploration se compose comme une succession de noeuds état de croyance, et de noeuds croyance-action. Lorsque la trajectoire se trouve dans un noeud état de croyance b, une action a est choisie par critère d'élargissement progressif : selon le $N(b)$ le nombre de visites du noeud, une action est tirée parmi $C(b)$ les actions déjà utilisées depuis ce noeud, ou bien une action jamais rencontrée est échantillonnée. Une fois l'action a choisie, la descente progresse vers le noeud état-action b, a correspondant (ou bien un nouveau noeud état-action est créé si l'action n'avait jamais été rencontrée). Lorsque la trajectoire se trouve dans un noeud état-action b, a, un critère d'élargissement pour l'état de croyance suivant est utilisé. Si un nouveau noeud doit être créé, $G_{PF}(b, a)$ fournit b' le prochain noeud. Sinon b' est échantillonné uniformément parmi les enfants de b, a.

La procédure PLAN renvoie l'action estimée comme optimale après n itérations de ONEPASS. On peut alors effectuer l'action réelle et recevoir une observation. Un filtre particulaire doit alors être utilisé pour estimer la nouvelle distribution à la racine (qui joue le rôle de b_0). Il est important qu'un grand nombre de particules soit utilisé à ce stade car c'est à partir de cette distribution que seront échantillonnés des paquets de particules lors des futures phases de descente.

6.3 Calibration de l'algorithme

Pour pouvoir calibrer les méta-paramètres de l'algorithme d'une part, et mesurer la performance de notre implémentation d'autre part, on utilise un problème connu de la littérature *LightDark* ([12]). Un agent se déplace dans $\mathbb{N}$. À chaque étape, il choisit une action parmi $\{-10, -1, 0, 1, 10\}$ et se déplace en conséquence : $S_{t+1} = S_t + A_t$. Si l'action choisie est 0, il reçoit 100 comme récompense s'il se trouve dans l'état 0 et -100 sinon, et le processus s'arrête. Sinon, il reçoit -1 comme récompense et continue. Le bruit de l'observation est proportionnelle à la distance à 10 de l'agent. Plus précisément, O_t suit une loi gaussienne centrée en S_t et de

Algorithm 1 PFT-DPW

procedure ActionProgWiden(b)
  if $|C(b)| \leq k_a N(b)^{\alpha_a}$ **then**
    $a \leftarrow NextAction(b)$
    $C(h) \leftarrow C(b) \cup \{a\}$
  end if
  return $\arg\max_{a \in C(b)} \hat{V}(b,a) + C_p \sqrt{\frac{\log N(b)}{N(b,a)}}$
end procedure

procedure Plan(b)
  for $i = 1, \ldots, i = n$ **do**
    OnePass($b, d_{\max}$)
  end for
  return $\arg\max_a Q(b,a)$
end procedure

procedure OnePass(b, d)
  if $d = 0$ **then**
    return 0
  end if
  $a \leftarrow$ ActionProgWiden(b)
  if $|C(b,a)| \leq k_o N(b,a)^{\alpha_o}$ **then**
    $b', r \leftarrow G_{PF}(b,a)$
    $C(b,a) \leftarrow C(b,a) \cup \{(b',r)\}$
    $total \leftarrow r + \gamma$Simulation$(b', d-1)$
  else
    $b', r \leftarrow$ échantillon uniforme sur $C(b,a)$
    $total \leftarrow r + \gamma$Simulate$(v', d-1)$
  end if
  incrémente $N(b)$
  incrémente $N(b,a)$
  $\hat{V}(b,a) \leftarrow \hat{V}(b,a) + \frac{total - \hat{V}(b,a)}{N(b,a)}$
end procedure

variance $|S_t - 10|$. La stratégie optimale consiste à se rapprocher de l'état 10 (la zone "claire") pour se localiser suffisamment précisément, puis à aller en 0 et effectuer l'action 0.

L'intérêt de ce problème comme problème-témoin est qu'il requiert de façon plus ou moins cachée que l'agent sache bien se localiser, et introduit des objectifs contradictoires à équilibrer : aller en 0 sans perdre trop de temps, et aller dans la zone claire pour se localiser et à terme faire l'action 0 avec bonne probabilité de réussite.

7 Mise en oeuvre sur Thymio

7.1 Communication

La mise en place d'une communication ROS-Thymio est faite en suivant le travail conjoint de mes maîtres de stage et des trois précédents stagiaires de M1. *In fine*, un topic */sensors* est utilisé pour communiquer les données des 5 capteurs frontaux, un topic */odom* pour communiquer les vitesses de rotation instantanée de chacune des roues, et un topic */cmd_vel* pour communiquer, indirectement, les commandes moteurs. Les commandes effectives données sont une vitesse linéaire et une vitesse angulaire.

7.2 Simulation du Thymio

Enki Les algorithmes développés lors de ce stage repose en partie sur la génération de nombreuses particules capables de simuler des trajectoires d'états. Il faut donc disposer d'un moyen de simuler facilement les déplacements de Thymio dans son environnement. Aseba Playground est un simulateur temps réel et n'est donc pas une solution viable. Sur conseil de mes tuteurs, j'ai recours au simulateur Enki, utilisé par Aseba Playground, qui permet de faire évoluer des Thymio en temps simulé.

Simulation des capteurs laser En plus d'un générateur *black box* de mouvements de particules, les algorithmes utilisés supposent disposer d'une fonction de vraissemblance qui fournisse $\mathbb{P}(\text{observation} \mid \text{état})$. J'ai donc regardé le code source d'Enki pour voir le modèle des capteurs laser utilisé et déterminer la fonction de vraissemblance associée.

Difficultés Par manque de temps d'une part, et des difficultés d'utilisation d'Enki (prévu pour C++) sous Python d'autre part, je n'ai pas pu intégrer Enki aux algorithmes développés. La conséquence décevante est que ces derniers n'ont pas pu être testés sur Thymio.

7.3 Résultats

Du fait des difficultés sus-mentionnées, les algorithmes n'ont pas pu être testés dans le cadre applicatif Thymio.

8 Perspectives à développer

8.1 ρ-PFT-DPW

8.2 Estimation de l'entropie

Une des difficultés de la localisation active, à laquelle je n'ai pas eu le temps de me confronter, est la manière d'estimer l'entropie d'un état de croyance $H(b_t) = -\int_{s\in\mathcal{S}} b_t(s)\log b_t(s)ds$ à partir d'une estimation particulaire de b_t. Une idée naturelle peut être de discrétiser le problème : on partitionne l'espace $\mathcal{S}$ en P ensembles $q_1, \ldots, q_P$ de volume à peu près égaux et on définit les

poids de q_i comme étant $w(q_i) = \sum_{j=1}^{N} w_t^{(j)} \delta_{s_t^{(j)} \in q_i}$. On estime alors

$$\widehat{H}(b_t) = -\sum_{i=1}^{P} w(i) \log w(i)$$

. Le choix de P est crucial : s'il est trop petit, toutes les particules se retrouveront dans le même ensemble et l'entropie estimée est toujours 0, s'il est trop grand, toutes les particules sont seules dans leur ensemble et l'entropie estimée est celle des poids des particules.

8.3 Mise à jour max ou mise à jour espérance

Les algorithmes que j'ai rencontrés dans la littérature utilisent tous une mise à jour des valeurs d'état de type "espérance", c'est-à-dire que la valeur de l'état estimée est la moyenne des valeurs remontées vers ce noeud. Puisque le chemin optimal est censé être suivi infiniment plus souvent dans le sous arbre issu d'un noeud donné, la valeur estimée doit converger vers la valeur effective de l'état correspondant. Une alternative naturelle existe : la mise à jour peut être faite en reprenant la définition de valeur d'un état et prend alors la forme d'un max.

Dans le cas de DFT-DPW, j'ai implémenté une variation de la mise à jour où la valeur d'un noeud action est estimée comme étant la moyenne sur les états directement sous-jacents de la valeur estimée de ces états, et la valeur d'un noeud état est estimée comme étant le max sur les actions partant de ce noeud de la valeur estimée de ces actions.

$$\widehat{V}(s) \leftarrow \max_{a \in \mathcal{A}} r(s,a) + \gamma \widehat{V}(s,a)$$
$$\widehat{V}(sa) \leftarrow \frac{1}{N(sa)} \sum_{s' \text{ enfant de } sa} \widehat{V}(s')$$

L'algorithme obtenu est appelé max-PFT-DPW. Les algorithmes max-PFT-DPW et PFT-DPW se comportent de façon équivalente sur LightDark. [4] présente une comparaison de variantes d'UCT, dont MaxUCT, qui diffère uniquement par cette différence de mise à jour. Dans 6 problèmes sur 8, MaxUCT apportent de meilleurs résultats qu'UCT. Il reste cependant difficile d'estimer à l'avance quelle mise à jour est la plus adaptée pour le problème de localisation active.

8.4 Garanties théoriques

La convergence, et le cas échéant les conditions de convergence, de l'algorithme PFT-DPW sont encore à déterminer. En l'état, les tailles des paquets de particules étant fixes dans PFT-DPW, il paraît impossible de pouvoir assurer la convergence de l'algorithme vers une politique optimale. On peut envisager utiliser des techniques de grossissement progressif des paquets de particules, développées dans [13], pour obtenir un algorithme avec de bonnes garanties théoriques.

8.5 À propos de la localisation

Si je n'ai pas eu le temps de mettre les algorithmes en pratique, il reste néanmoins intéressant de réfléchir aux environnements sur lesquels tester les algorithmes de localisation. Le "coin" est sans doute le plus simple et doit permettre de tester le bon fonctionnement des algorithmes. La salle carrée présente déjà plus de difficultés. En effet, la carte présente une symétrie de rotation d'angle $\frac{\pi}{4}$. Il en suit que Thymio ne peut jamais que connaître sa position à une rotation près, en particulier on attend que la distribution estimée après quelques pas de temps ressemble à exactement 4 pics (sauf cas pathologique). Deux problèmes risquent d'être soulevés : d'une part, il est possible qu'un ou plusieurs pics soient aléatoirement perdus au cours de l'exploration, d'autre part, la fonction de récompense peut s'avérer mauvaise. La variance de la distribution estimée, qui est un choix par ailleurs naturel, est typiquement très mal adaptée aux carte à symétrie, puisque la distribution la plus "localisée" possède une grande variance.

Par ailleurs, les tests peuvent être conduits sur simulateur ou sur un Thymio réel. Des tests sur simulateur présentent l'intérêt d'être facilement répétables et contrôlables mais les conditions d'évolution sont peut être trop idéales pour modéliser le réel. Des difficultés lors du passage de tests sur simulateur à des tests sur un Thymio réel sont à prévoir.

Enfin, la façon dont les hyper-paramètres de PFT-DPW peuvent être calibrés est un sujet à développer.

Références

[1] A.R. Cassandra. A survey of POMDP applications. In *AAAI Fall Symposium*, 1998.

[2] Arnaud Doucet, Nando de Freitas, and Neil Gordon. An introduction to sequential Monte Carlo methods. *Sequential Monte Carlo Methods in Practice. Springer, Berlin*, 01 2001.

[3] Arnaud Doucet and Adam M. Johansen. A tutorial on particle filtering and smoothing : fifteen years later. 2011.

[4] Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 135–143, 2013.

[5] L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning. In *Proceedings of the Sixteenth European Conference on Machine Learning*, 2006.

[6] Martin L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Progamming*. John Wiley and Sons, Inc, 1994.

[7] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSp Magazine*, 1986.

[8] Stuart Russell and Peter Norvig. *Artificial Intelligence : A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[9] Dawei Shen. Some mathematics for HMM. 2008.

[10] Olivier Sigaud and Olivier Buffet. *Markov Decision Processes in Artificial Intelligence*. Wiley-IEEE Press, 2010.

[11] David Silver and Joel Veness. Monte-carlo planning in large pomdps. pages 2164–2172, 2010.

[12] Zachary Sunberg and Mykel J. Kochenderfer. POMCPOW : an online algorithm for POMDPs with continuous state, action, and observation spaces. *CoRR*, abs/1709.06196, 2017.

[13] O. Buffet V. Thomas, G. Hutin. Planification Monte Carlo orientée information. 2019.